Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Dissertation

Petri Net-based Combination and Integration of Agents and Workflows

by

Thomas Wagner

from Schleswig

for the attainment of the degree

Doctor of Natural Sciences
(Dr. rer. nat)

Hamburg, 2017

Submitted at the

University of Hamburg

Faculty of Mathematics, Informatics and Natural Sciences

Department of Informatics

Date of oral disputation: 19.12.2017

Members of the examination board:

Dr. Daniel Moldt
University of Hamburg
(*Supervisor*)

Prof. Dr.-Ing. Norbert Ritter
University of Hamburg
(*Supervisor*)

Prof. Dr. Maciej Koutny
Newcastle University
(*Reviewer*)

Prof. Dr.-Ing. Wolfgang Menzel
University of Hamburg
(*Head of examination board*)

For my grandmother

Acknowledgements and Thanks

First of all, I would like to thank my supervisors Dr. Daniel Moldt and Prof. Dr.-Ing Norbert Ritter for guiding me through this thesis. Especially Dr. Moldt provided significant support with much appreciated and helpful advice and probably hundreds of hours of fruitful discussions. In addition, I would like to thank Prof. Dr.-Ing Ritter for his continued support throughout the thesis. Furthermore, I would like to thank Prof. Dr. Maciej Koutny for reviewing this thesis and Prof. Dr.-Ing. Wolfgang Menzel for serving as the head of the examination board.

I would also like to thank past and present colleagues at the TGI research group at the University of Hamburg, as well as students I have had the pleasure of working with. These are (in no particular order): Dennis Schmitz, Dr. Lawrence Cabac, David Mosteller, Michael Haustermann, Julian Mosteller, Dr. Christine Reese, Dr. Sofiane Bendoukha, Dr. Frank Heitmann, Dr. Matthias Wester-Ebbinghaus, Prof. Dr. Michael Köhler-Bußmeier, Prof. Dr. Rüdiger Valk, Dr. Michael Duvigneau, Gila Dinter, Christian Röder, Dr. Kolja Markwardt, Margit Wichmann, Frederick Thomssen, Jan Henrik Röwekamp, Joanna Sieranski and Max Friedrich. All of them provided helpful feedback, collaborations and discussions. Special thanks go out to Dennis Schmitz, who collaborated on some parts of the practical implementations of the Paffin-System. I would also like to thank Ulf Steuding and Henning Husmann, who made it possible to work part time in the final phase of finishing this thesis.

Finally, I would like to thank my parents Reinhard and Christa, my grandmother Anneliese and my uncle Achi for their continued support throughout school, university and this dissertation. I would like to thank Marie, whose feedback helped in many regards, as well as my friends and extended family, including Catharina, Martin, Michi, Janina, Denis and Jack.

All of the above provided valuable feedback, support and assistance that helped me in writing this thesis.

Hamburg, 19.12.2017

Thomas Wagner

Abstract

Agents, on the one hand, represent autonomous components, which means they emphasise the structure of a software system. Workflows, on the other hand, represent processes, meaning that they emphasise the behaviour of a software system. These emphases enable certain strengths in both agents and workflows. Agents excel at representing and modelling structural aspects, while workflows excel at representing and modelling behavioural aspects. This thesis researches, develops and provides a combination and integration of agents and workflows. This combination and integration enables the strengths for *both* structure and behaviour simultaneously and creates novel and beneficial synergies.

The main result of this thesis is a combined and integrated model for the desired integration. The conceptual model is called the Agent-Activity integration approach. This approach is practically implemented in a technical proof-of-concept, called the **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System). Both the Agent-Activity integration approach and the Paffin-System are modelled with reference nets, a Petri net formalism following the nets-within-nets principles. Using Petri nets allows for a clear specification of all models and also facilitates the integration by providing a shared foundation for modelling both structure and behaviour. That foundation is complemented by the use of previously-existing, well-established individual Petri net models for agents and workflows.

The core of the Agent-Activity integration approach is the Agent-Activity modelling construct. The modelling construct combines a set of agent actions and workflow operations into an internal, ordered process describing an abstract, complex task. That task is executed by a so-called integrated entity. An integrated entity can act as and be interacted with as agent, workflow, both or a hybrid in between, depending on the set and order of the agent actions and workflow operations within the Agent-Activities it executes. Agent-Activities containing only agent actions implement entities as agents, Agent-Activities containing only workflow operations implement workflows and Agent-Activities containing both agent actions and workflow operations implement hybrids of agents and workflows. Finally, if multiple Agent-Activities are executed concurrently the integrated entity can be both agent and workflow at the same time. The unification of the different concepts into one integrated entity concept allows for various synergies of the integration to be realised and utilised.

A system featuring an integration based on Agent-Activities requires a specific form and construction. That form and construction, including the required management facilities, is captured in a reference architecture. The Paffin-System prototype implements that reference architecture. It is technically based on the Capa agent framework and adapts that framework for the integration mechanisms by adding workflow and hybrid management functionality. Ultimately, the Paffin-System prototype represents a full-fledged development framework for the creation of applications utilising agents, workflows and hybrids of both.

The conceptual and technical results of the thesis are complemented by a number of application prototypes, as well as extensive and detailed discussions. These provide a basis for the evaluation and assessment of the Agent-Activity integration approach and Paffin-System prototype.

Zusammenfassung

Agenten repräsentieren selbstständige, autonome Komponenten, welche die Struktur eines Softwaresystems hervorheben. Workflows repräsentieren Prozesse, welche das Verhalten eines Softwaresystems betonen. Diese beiden Schwerpunkte ermöglichen bestimmte Stärken, sowohl in Agenten als auch in Workflows. Agenten zeichnen sich durch die Möglichkeiten zur Darstellung und Modellierung von Struktur aus, während Workflows sich durch die Möglichkeiten zur Darstellung und Modellierung von Verhalten auszeichnen. In dieser Dissertation wird eine Kombination und Integration von Agenten und Workflows erforscht, entwickelt und bereitgestellt. Dies geschieht, um sowohl die Stärken für die Struktur, als auch die für das Verhalten, gleichzeitig zu nutzen und dadurch entstehende, vorteilhafte Synergien zu ermöglichen.

Das Hauptergebnis dieser Dissertation ist ein kombiniertes und integriertes Modell der angestrebten Integration. Dieses Modell wird auf der konzeptionellen Ebene durch den Agent-Activity Integrationsansatz realisiert. Dieser Ansatz ist praktisch in einem technischen Konzeptnachweis implementiert, dem sogenannten **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System). Sowohl die Konzepte, als auch die praktischen Systeme, sind mit Referenznetzen modelliert. Referenznetze sind ein erweiterter Petrinetzformalismus, welcher das Netze-in-Netzen Prinzip verfolgt. Die Petrinetzbasis erlaubt eine klare Spezifikation aller Modelle und unterstützt die Integration durch die Bereitstellung einer gemeinsamen Grundlage für die Modellierung von Struktur und Verhalten. Diese Grundlage wird weiterhin durch die Verwendung von existierenden und bewährten, individuellen Petrinetzmodellen für Agenten und Workflows ergänzt.

Der Kern des Agent-Activity Integrationsansatzes ist das gleichnamige Modellierungskonstrukt. Dieses Modellierungskonstrukt kombiniert eine Menge von Agentenaktionen und Workflowoperationen in einem internen, geordneten Prozess einer abstrakten und komplexen Aufgabe. Diese Aufgabe wird von einer sogenannten integrierten Einheit ausgeführt. Je nachdem was für Aktionen und Operationen in einer Agent-Activity ausgeführt werden, kann die integrierte Einheit als Agent, Workflow, beides oder ein Hybrid agieren und interagieren. Eine Agent-Activity, die nur Agentenaktionen enthält, realisiert eine Einheit als Agent, eine Agent-Activity, die nur Workflowoperationen enthält, realisiert einen Workflow und eine Agent-Activity, die sowohl Agentenaktionen als auch Workflowoperationen enthält, realisiert einen Hybrid zwischen Agent und Workflow. Falls Agent-Activities gleichzeitig oder nebenläufig ausgeführt werden, ist die ausführende integrierte Einheit sowohl Agent, als auch Workflow. Die Vereinheitlichung von den verschiedenen Konzepten in einem Konzept für integrierte Einheiten, ermöglicht viele Synergien der Integration zu realisieren und zu nutzen.

Ein System, welches dem Agent-Activity Integrationsansatz folgt, benötigt einen bestimmten Aufbau. Dieser Aufbau, inklusive der benötigten Managementfunktionalität, ist in einer Referenzarchitektur abgebildet. Das Paffin-System implementiert diese Referenzarchitektur prototypisch. Technisch basiert das Paffin-System auf dem Capa-Agentenframework. Dieses wurde für die Verwendung von Integrationsmechanismen adaptiert und um die benötigte Workflow- und Hybrid-Managementfunktionalität erweitert. Schlussendlich repräsentiert das Paffin-System ein vollständiges Entwicklungsframework für die Erstellung von Anwendungen, welche Agenten, Workflows und Hybride von beiden verwenden können.

Die konzeptionellen und technischen Ergebnisse der Dissertation werden durch weitere Anwendungsprototypen, sowie ausführliche und detaillierte Diskussionen ergänzt. Diese stellen die Basis der Evaluierung und Einschätzung des Agent-Activity Integrationsansatzes und des Paffin-Systems dar.

Contents

List of Figures

List of Tables

Part A

Introduction and Background

Part A serves as the introduction to the topics and context of this thesis. It contains three chapters. Chapter 1 introduces and motivates the proposed research and presents the research questions and goal. Chapter 2 then contributes the basic understanding of the involved research areas, emphasising Petri nets, agent-orientation and workflow management. Finally, Chapter 3 provides an overview of the state-of-the-art of related agent and workflow research areas.

The purpose of this part is to define the direction of the research conducted for this thesis and to substantiate the foundation of basic, previous and related work for the involved research areas.

1 Introduction

Modern software systems are continuously becoming larger and more complex. There is a need for systems to capture *all* essential aspects of real-world scenarios involving large groups of human participants and other resources. Instead of single or small groups of users, a system may have to incorporate and support an entire organisation, including all of the users and diverse (hardware) resources within it. Instead of small-scale, self-contained processes, a system may need to deal with constantly running, adaptive hierarchies of inter-dependent processes that span multiple, distributed subsystems. As a consequence realising and implementing these systems becomes more challenging and complex.

A good example for this development is the inter-organisational context. These systems, especially inter-organisational workflow systems, are systems involving multiple organisations interacting and cooperating with one another. Complex production processes, in which different organisations are responsible for specific manufacturing steps from raw materials to finished goods, are often captured and supported by inter-organisational systems. These kinds of processes can be very extensive, long-running and may feature complex inter-dependencies between tasks, resources, subworkflows, etc. Furthermore, modellers of inter-organisational systems need to handle additional challenges as the conditions and autonomy of the individual organisations need to be taken into account. Questions of, for example, encapsulation, security and responsibility therefore continue to become even more important.

Given these kinds of challenges, a need for more comprehensive and suitable modelling techniques arises. Such techniques need to provide refined, extensive and appropriate constructs to modellers. They must be able to sufficiently capture all of the different aspects of a software system in order to comprehensively produce the required target functionality.

In this thesis, the focus lies on the structure and behaviour of and within software systems. As is described early on, structure and behaviour are orthogonal perspectives that capture the entirety of a software system together. A focus on structure and behaviour is supported by the Paose development approach (Petri Net-Based Agent- and Organisation-oriented Software Engineering, [Cabac, 2007]), which is related to the work throughout the thesis.

The structure of a software system is its inherent construction and configuration. It includes, for example, the division into functional units, their individual interfaces and the (possibly distributed) infrastructure for execution. The behaviour of a software system is the accumulation of all the possible actions within it. It relates these actions to one another into hierarchies represented as processes and (sub-)systems of processes.

Current modelling techniques in general only emphasise one aspect of a software system. A modelling technique's main modelling construct is the primary unit of a technique and defines how a system is modelled at the highest level of abstraction. Main constructs are, for example, objects in object-orientation and services in service-oriented architectures. In agent-oriented systems the main construct is the agent, while in workflow systems it is the workflow.

Everything in a modelled system is dependent on the main construct and also set into a relationship to that construct. In agent systems behaviour is always regarded as the processes happening within and between agents. In workflow systems the (structural)

resources and users executing tasks are only considered within the context of the workflows they participate in. Herein lies the issue. The main construct itself is a quality of one specific aspect of the software system. Agents and objects represent functional units of the structure of a system. Services and workflows represent (parts of) processes within a system's behaviour.

Consequently, if everything is related to and dependent on the main modelling construct, then everything is also set into relation with the main modelling constructs' emphasised system aspect. In this way the main construct determines and defines a particular modelling perspective for the system. That perspective has a fixed focus on the system aspect associated with the main modelling construct. In other words, by utilising only one modelling construct a technique tends to emphasise only one specific aspect. Other aspects are subordinated, devalued or possibly even neglected.

This results in the technique making itself more difficult and convoluted. Instead of modelling aspects with fitting and natural constructs the modellers are forced to work around the main construct. In practice, this does not necessarily restrict a modeller in a technique. Behaviour, for example, is easily represented and implemented in a modelling technique emphasising structure. However, as explained above, all elements of the behaviour (and other aspects of the system) are dependent on the main modelling construct and thus the structure of the system.

In small-scale systems this has hardly any effect. The relatively small size of any modelling artefact ensures a comparatively easy manageability. This means that all aspects captured using the main construct can still be easily understood by the modeller. In large-scale systems, though, it becomes more difficult to adequately capture and model any aspect not associated with the main construct. The mapping and relation between the main construct and other aspects of the system is usually indirect and may utilise an auxiliary construct (e.g. an ontology) or even additional tools (e.g. databases). With the increased size and scope of a system, these kinds of approaches become inefficient, may fail to capture important properties or generally become more difficult to realise. It also becomes very hard or even impossible for modellers to directly discern the different aspects directly from the larger modelling artefacts.

For example, it is relatively easy to model and understand the behaviour between two or three structural modelling constructs like agents. Behaviour involving hundreds of structural units with a high degree of interconnection between them is, however, very difficult to work with if the representation is built around a structural main construct. In such cases a separate and direct representation of the behaviour would be useful. This kind of representation, though, is not easily available within a technique that uses a structural main modelling construct.

This does *not* mean that current modelling techniques are, in a practical sense, limited by these issues. Even though it can be cumbersome or difficult to capture aspects not associated with the main construct it is nonetheless possible. As mentioned above effects of these issues mostly apply to modelling large-scale systems. In most systems modellers can overcome them due to the size of modelling artefacts. In other cases these issues can be side-stepped by reducing the complexity of systems through the division into subsystems or the like.

The issues introduced by the focussed modelling construct and perspective do, however, still represent a weakness in the modelling techniques. Such a weakness is opposed to the strengths of a technique which directly result from utilising one main construct. These strengths are properties and facets of the aspect associated with the main construct that can be captured directly, naturally and overall especially well. The modelling perspective

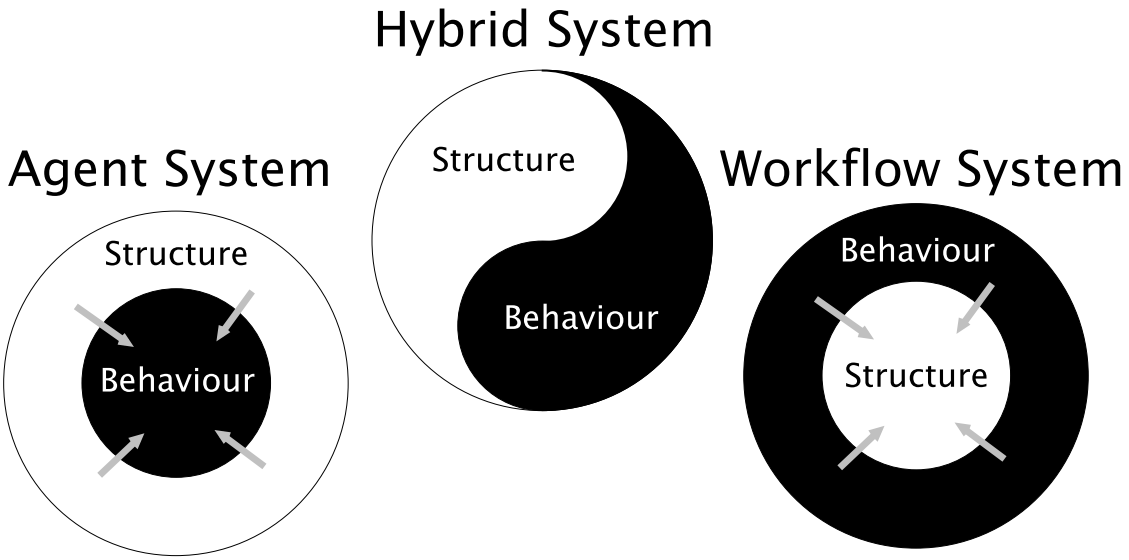

Figure 1.1: Conceptual view of agent, workflow and hybrid systems

with its focus strongly supports the modeller here. Even in very large and complex systems it continues to be (relatively) easy for modellers to handle challenges and issues related to the strengths.

The emphasis created by introducing a focussed and sole modelling construct represents part of the general context of problems examined in this thesis. There is a duality to this effect on an abstract level. On the one hand, this emphasis supports the modelling of one aspect directly and overall very well. On the other hand, it subordinates other aspects to this one focused aspect.

To be clear, the motivation of this thesis is *not* that certain aspects can't be modelled well in current, traditional techniques. Rather, the motivation is that certain aspects can't be modelled *as well as* other ones. Modelling techniques exhibit a particular strength for the aspect directly related to the main construct. This strength is not available for other aspects that are not directly related to the main construct. Therefore, modelling those other constructs is not as well supported as the emphasised one. Behaviour, for example, can't be modelled as well with a structural main construct as with a directly behavioural one. The thesis deals with combining and integrating the strengths of different modelling perspectives, thereby ensuring that the different aspects of a system are all well supported.

As stated above, this thesis considers the structure and behaviour of a software system as the two main system aspects. The chosen modelling constructs and abstractions for these two aspects are agents for structure and workflows for behaviour. This selection was made because the concepts of agents and workflows are well-researched and well-established. Additionally, they are capable of modelling and representing their related aspects in a comprehensive way.

Figure 1.1 illustrates the main motivation of the thesis. It shows a representation of the modelling perspectives in an agent system and a workflow system on the left-hand and right-hand sides respectively. In an agent-system the structure is the determining aspect. Everything else is, in some way, related to the individual agents (e.g. data, knowledge) or sets of agents (e.g. distribution to platforms). Behaviour, especially, is always dependent on and considered only for *specific* (sets of) agents. In this way the behaviour of a system is governed by the agents and consequently by the structure. In workflow systems this

situation is reversed, as the behaviour is the determining aspect. Every other aspect is connected to the workflows or tasks within the workflows (e.g. required/invoked functionality, security/access issues). Here, in particular, the structure of a system is dependent on the behaviour. Resources, users and engines (as structural units) are always regarded in the context of (sets of) specific workflows. This means that the structure in workflow systems is governed by the workflows and consequently the behaviour.

The middle part of Figure 1.1 illustrates the hypothesis and main goal. The hypothesis of the thesis is that two individual modelling techniques and their related perspectives can be united to provide the strengths of both, as well as additional beneficial synergies, by integrating and combining their main modelling constructs. More specifically, the two orthogonal aspects of structure and behaviour of a software system are combined via agents and workflows in order to create an enhanced modelling perspective.

This perspective is indicated in Figure 1.1 as the *hybrid system*. The illustration borrows from the symbol of the philosophical Yin and Yang concept. It represents the two aspects structure and behaviour in a kind of balance. They should be of equivalent importance and on the same level of abstraction without one of the two governing the other one.

In this thesis the two concepts agents and workflows are combined and integrated in order to create the hybrid system. The research, development and provision of this combination and integration are the overarching *goal* of the thesis. On a conceptual level this goal is constituted by a conceptual model and approach for the motivated combination and integration. This model and approach are supplemented by a set of prototypes constituting a working proof-of-concept realisation and implementation. This proof-of-concept is the goal of this thesis on the practical level.

The research in this thesis aims to provide a general solution to the desired combination and integration. However, a general solution may impede concrete and practical considerations. Therefore, this thesis adopts an approach, in which the research is oriented on specific agent and workflow models, yet kept as general as possible until concrete and practical models are required. The principles, insights and results gained with these specific models in mind are then evaluated and examined for more general contexts.

For this thesis, the agent model is based on the Mulan agent model and architecture [Rölke, 2004]. Mulan agents represent intelligent software agents according to the established understanding from [Wooldridge, 2009, pp. 26–27]. They do not feature cognitive or artificial intelligence concepts, though, meaning that BDI [Rao and Georgeff, 1991] or other related agent fields are outside of the scope of this thesis. Such concepts are only discussed as related work and in the context of the general applicability of results.

For workflows, this thesis utilises workflow Petri nets [van der Aalst et al., 1994] as the basic model. Workflow Petri nets graphically model (business) processes and enable the simulation and execution of these processes. They represent a general and powerful modelling formalism. Other workflow modelling formalisms, like BPMN [von Rosing et al., 2015], are outside of the scope of this thesis, but are treated as related work.

In order to facilitate the integration of agents and workflows, this thesis utilises reference Petri nets [Kummer, 2002] as a shared basis. Through that shared basis, an integration can be described more easily and the concepts from both sides can be incorporated more directly and understandably. Reference Petri nets fully support Java as an inscription language, meaning that there are no restrictions or limitations to their modelling[1] capabilities. This makes them a true extension of Java for modelling concurrency, conflict, synchronisation

[1] Please note that the term *modelling* in this thesis in relation to reference nets always refers to an executable model. Therefore, the terms *modelling*, *realising* and *implementing* are synonymous for the practical systems in this thesis. This is elaborated on in Section 2.1.2.

and unification. Reference Petri nets also provide a number of additional benefits, including a formal basis, a well-understood and well-understandable graphical representation and the means to visualise the execution. Additionally, reference Petri net models for MULAN agents and workflows are established, well-researched and available for conceptual use and practical deployment. Other integration bases, like databases, would have been feasible choices as well, but the already existing models and further benefits of reference models motivated their selection so that other bases are outside of the scope of this thesis.

Relating to the utilisation of (reference) Petri nets for modelling, this thesis also provides relevant contributions. There are established Petri net models, for example, for objects [Moldt, 1996], agents [Moldt and Wienberg, 1997, Rölke, 2004], and workflows [van der Aalst et al., 1994]. However, outside of preliminary work from unit theory [Moldt, 2005, Tell and Moldt, 2005], there is not yet a concrete and fully realised Petri net model for a full hybrid of agent and workflow available. The conceptual model and approach of the integration referred to in the goal of this thesis provides such a net model through the utilisation of (reference) Petri nets as a modelling basis.

Please note that the restrictions about an orientation on specific agent and workflow models, as well as a shared technological basis, do not mean that the results obtained for this thesis are only applicable in these particular contexts. Discussions in later chapters of the thesis show that the results are viable for more than just reference Petri net-based agents and workflows.

Related to the overarching goal there are two major research questions that serve to guide the research of this thesis. The first major research question is:

How can the concepts "agent" and "workflow" be combined and integrated in a reasonable, conducive and beneficial way?

This research questions deals with researching and examining the generally possible solutions for the aspired combination and integration. To answer this question the very first thing to do is to clearly define (for the context and extent of this thesis) the structural and behavioural perspectives. Important aspects that need to be established are the exact contents of the perspectives, their possibilities and limitations. Following this it becomes possible to examine the different *reasonable* possibilities of a combination and integration and evaluate which ones are better suited for the proposed problem. From the remaining possibilities the conceptual model can be created, which constitutes the answer to the research question. The open practical questions are addressed by the second major research question:

Which beneficial effects can be achieved through a combination and integration of agents and workflows, potentially and substantiated through a technical proof-of-concept, and in which scenarios are they best applicable?

The beneficial effects achieved by combining and integrating agents and workflows become clearer through the creation of practical prototypes of the conceptual model. As such, this research question focuses on the need to develop technical prototypes and a proof-of-concept, in order to observe and substantiate the actual practical advantages and disadvantages of the combination and integration. This research question defines the technical and practical orientation of this thesis and also involves aspects for evaluation. The actual benefits are derived from the evaluation of both the conceptual results from the first research question, as well as the technical results. Together they provide the answer to this research question.

Outline The thesis is partitioned into six parts. Part A serves as the introduction of the thesis. Besides this introduction, it contains the overview of the basics of Petri nets, agents and workflows, as well as a chapter describing the current state-of-the-art in the related research fields. Part B then deals with how to approach an integration of structure and behaviour based on agents and workflows. This part defines the terms structure and behaviour, refines them for agents and workflows and then establishes a specification and vision of an integration. Based on the results of this part, the requirements of this thesis are defined. Next, Part C proposes multiple abstract integration approaches and evaluates them, resulting in the selection of an approach which is further developed. The developed concrete approach, called the Agent-Activity integration approach, is then presented and constitutes the main conceptual result of this thesis. W.r.t. the goal of the thesis, it represents the conceptual model and approach for the motivated combination and integration. Agent-Activities are described in both concept and through prototypes implementing the second result prescribed by the goal, the technical proof-of-concept. That proof-of-concept is called the **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System). The Paffin-System represents the practical and technical main result of this thesis. Following this, Part D applies, discusses and evaluates the results of the thesis. Here, practical use cases are also presented which serve as the basis for the discussions. This part answers the research questions laid out in this introduction. Part E concludes the thesis with a summary and outlook. Finally, Part F contains the thesis appendices, including a key term glossary and a list of acronyms. Note that each part of the thesis starts with an outline of the part, which provides an overview over the individual chapters.

2 Basics

This chapter presents the theoretical, conceptual and technical foundations for the research in this thesis. It is divided into four sections.

Section 2.1 presents the basics of Petri nets and the utilised reference net formalism. It also describes the Renew tool used for the creation of most of the models in the thesis. Section 2.2 represents a basic introduction to the field of agent-orientation. Important concepts and properties related to software agents and their execution are presented here. It also details the specific agent model (Mulan), implementation (Capa) and development approach (Paose) used in this thesis. Section 2.3 provides an introduction to the field of workflow management. This section emphasises workflow management systems and the principles and standards behind them. Workflow Petri nets and the field of inter-organisational workflows are also discussed here. Finally, Section 2.4 discusses the term *process* and how it is used in this thesis.

2.1 Petri Nets

Petri nets are a formalism to represent and model sets of related actions. Their theoretical foundation, capabilities in representing concurrent behaviour and easy-to-understand graphical representation are just some of the qualities which make them a versatile and useful tool. Since the original groundwork was laid in 1962 they have been extensively studied and extended in various ways. Today, many specialised formalisms exist sharing the same fundamental concepts laid out in 1962 by Carl Adam Petri in [Petri, 1962].

One such formalism, reference nets, is used in the architectures, concepts, and prototypes throughout this thesis. Reference nets are directly used to model and implement agent, workflow and integration systems. Please note that this section only covers the very basics of Petri nets and the advanced topics that are used in the remainder of this thesis. This section does not provide a complete introduction to Petri nets and related concepts. Such an introduction can be found, for example, in [Girault and Valk, 2003] or [Reisig, 2013].

This section contains three subsections. Section 2.1.1 gives a general introduction to Petri nets. Next, Section 2.1.2 describes the reference net formalism and the Renew tool.

2.1.1 Basic Petri Nets

The P/T net (place/transition net) formalism is one of the simplest Petri net formalisms. P/T nets are defined in the following way (adapted from [Girault and Valk, 2003, p. 41, Def. 4.1.2]):

Definition 2.1 (P/T Net)**.** *A place/transition net (P/T net) is defined by a tuple* $N = (P, T, F, W)$*, where*

- P *is a finite set of places (graphical representation: rounded elements like circles, ellipses, ...)*
- T *is a finite set of transitions (graphical representation: edged elements like boxes, rectangles, ...), disjoint from* P*, and*

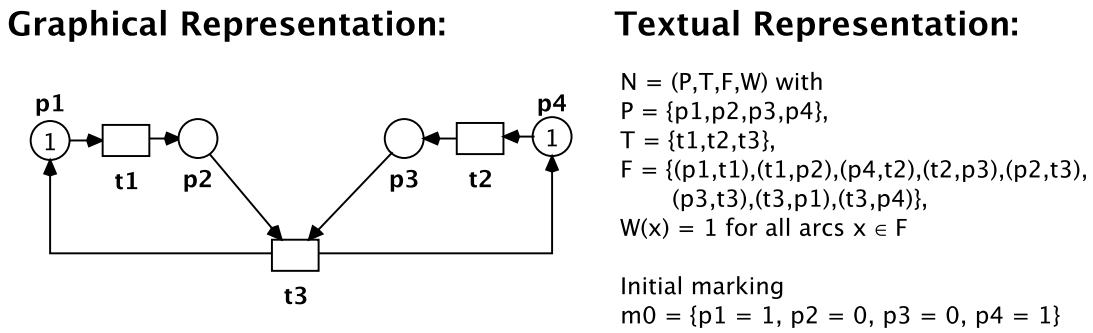

Figure 2.1: A simple P/T net

- *F is a flow relation $F \subseteq (P \times T) \cup (T \times P)$ for the set of arcs (graphical representation: arrows).*
- *$W : F \to \mathbb{N} \setminus \{0\}$ is a function (weight function).*

An example of a P/T net in both graphical and algebraic representation is shown in Figure 2.1. The following is adapted from and based on the descriptions of [Girault and Valk, 2003, p. 42], especially Def. 4.1.4. The state of a net is determined by the distribution of tokens onto the places. This distribution is called a *marking*. A marking of a P/T net $N = (P, T, F, W)$ is a vector $\mathbb{N}^{|P|}$. A net N together with an initial marking m_0 is called a *net system*. A transition t is *enabled* (or *activated*) in a marking m if all of its preconditions are fulfilled. This means that there are enough tokens on all places connected to the transition with incoming arcs to satisfy the weights for those arcs. An enabled transition can *fire*. If the transition is fired tokens are removed from the precondition places according to the arc weight. Firing creates tokens on the places connected from the transition with outgoing arcs. The amount of tokens created in each place is again determined by the weight of the arcs. Creating tokens in these places satisfies the postconditions of the transition. The firing process is considered to be instantaneous and atomic. Firing only affects the locality (i.e. pre- and postconditions) of a transition. Transitions with disjoint localities can fire concurrently. Concurrency and locality are two important principles of all Petri net formalisms [Girault and Valk, 2003, p. 12]. Another interesting feature of P/T nets and Petri nets in general is that, if multiple transitions are enabled, the choice of which fires (first) is non-deterministic and random.

Petri nets are executed/simulated[1] by providing an initial marking and letting the transitions fire. To illustrate the execution of a P/T net Figure 2.2 shows the firing sequences for the net from Figure 2.1. A firing sequence (or occurrence sequence) is "*a chain of transition firings*" [Verbeek et al., 1999, p. 5]. The exact definition is given in the following (adapted from [Verbeek et al., 1999, p. 5, Def. 2.14]):

Definition 2.2 (Occurrence Sequence (Firing Sequence))**.** *Let $S = (P, T, F, W, M_0)$ be a system (P/T net plus initial marking), let $M_1, ..., M_n$, for some natural number n, be the markings of $N = (P, T, F)$, and let $t_0, t_1, ..., t_{n-1}$ be transitions in T. Sequence $s = M_0 t_0 M_1 ... t_{n-1} M_n$ is an occurrence sequence of S iff $\forall i, 0 \leq i \leq n : M_i \xrightarrow{t_i} M_{i+1}$*

[1] The terms *executing* a Petri net and *simulating* a Petri net are both used in this thesis and refer to the same activity (unless explicitly stated otherwise.)

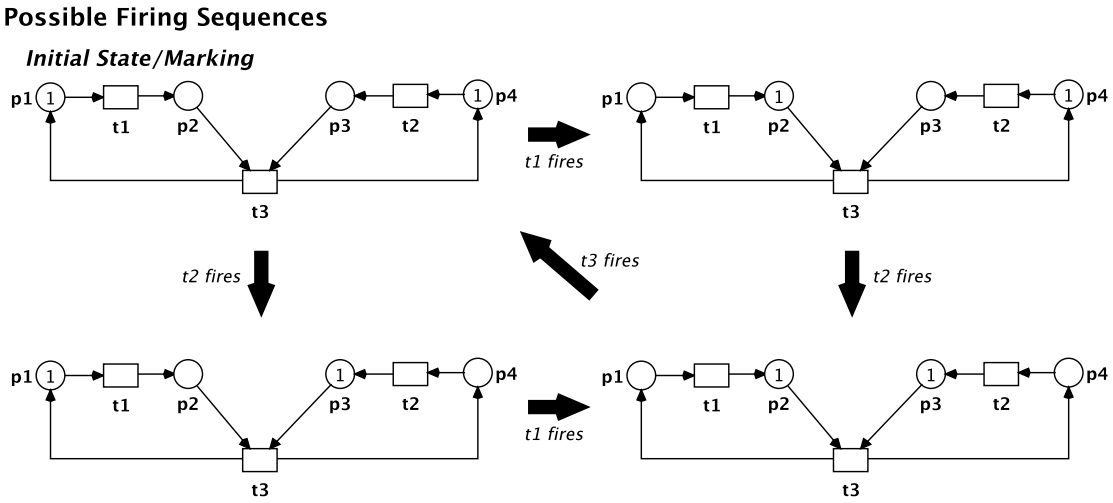

Figure 2.2: Possible firing sequences of a P/T net

Firing sequences are one way to describe the behaviour of, or process within, a Petri net system. How the term process is used in this thesis is expanded upon later in Section 2.4.

2.1.2 Reference Nets and Renew

Simple P/T nets already feature many important characteristics of Petri nets. They feature concurrent, non-deterministic behaviour and possess an easy-to-understand graphical representation. However, they are limited w.r.t. the software engineering potential. Limitations are caused, e.g., by the indistinguishability of the tokens and the lack of net hierarchies.

Over the years Petri nets have been extensively expanded, resulting in advanced formalisms, mechanisms and principles. Coloured Petri nets [Jensen, 1981, Jensen, 1987, Jensen and Kristensen, 2009], for example, feature distinguishable tokens of different types, called colours. Coloured Petri nets were extended in [Christensen and Hansen, 1994] with synchronous channels, which enabled synchronous, bi-directional communication between net transitions. Object Petri nets [Valk, 1996, Valk, 1998, Valk, 2004] feature a hierarchy in which system nets can contain other object nets as tokens, emphasising the so-called nets-within-nets principle. Object-oriented Petri nets [Becker and Moldt, 1993, Moldt, 1996] introduced concepts from object-oriented software engineering, such as classes, methods and objects, for coloured Petri nets.

Ideas from all of these formalism have also been incorporated into the reference Petri net (or short reference net) formalism [Kummer, 2002]. Additional mechanics and principles, like reference semantics, full Java support for inscriptions and modelling tools such as different arc types and virtual places, are also featured. All in all, the reference net formalism is, due to its extensive range of functionality, well suited for advanced software engineering and modelling. It, and the related modelling and execution tool Renew, are described in the following subsections. For a complete view please refer to [Kummer, 2002] and the Renew user guide [Kummer et al., 2016]. [Kummer, 2001] also provides a good overview, showcasing a stepwise explanation from P/T nets to reference nets.

Reference Net Formalism

Informally[2], reference nets are a Petri net formalism following the nets-within-nets principle. nets-within-nets means that tokens in a net may again be other nets. The idea behind nets-within-nets originated in [Valk, 1987] and [Jessen and Valk, 1987, p. 77], where so-called task/flow-nets (German: *Auftrags/Verkehrs-Netze*) supported task-systems as tokens in other nets. nets-within-nets-based formalisms are relatively rare. Examples include cooperative nets [Sibertin-Blanc, 1994], object Petri nets [Lakos, 1995] and nested nets [Lomazova and Schnoebelen, 2000]. Reference nets originate from the object Petri nets[3] from [Valk, 1996, Valk, 1998, Valk, 2004].

The nets-within-nets idea within reference nets exhibits reference semantics. This means that tokens within reference nets are references to the elements they represent. The only exceptions to this are the indistinguishable simple tokens (indicated by [] in Renew), as well as simple data types such as integers. With distinguishable tokens the same element may be referenced in multiple places and even multiple nets by different token instances. Reference semantics are opposed to value semantics, in which a token actually *is* the element it represents. Reference and value semantics are examined in [Valk, 2000, Köhler and Rölke, 2005, Kummer, 2002].

Reference nets (and Renew) are implemented in Java[4]. Tokens in reference nets can not only be references to other nets, but can be references to any Java object. Objects of the class `NetInstance` represent net instances as tokens. Renew distinguishes between net schemas and net instances. A net or net schema is the basic definition of a reference net that is modelled (or "drawn") in Renew during design time. Net schemas are implemented in the class `Net` and saved as net drawings in *.rnw* files in the file system. A net instance is a concrete, executed version of a previously defined net schema. There can be arbitrarily many net instances of any given net schema, all with different states. Descriptions in the thesis adopt the Renew distinction between schema and instance. It is also applied to other concepts, such as agents and workflows[5]. If only the general term, e.g. net, agent or workflow, is given, the context clarifies if a schema or instance is referred to.

Firing transitions within reference nets is handled according to general Petri net principles. Only the locality of the transition is affected by firing and firing is instantaneous, non-deterministic and can happen concurrently. Since tokens can be arbitrary Java objects the correct binding of tokens to a firing is more complicated than with indistinguishable tokens. Available tokens on places must be matched to variables of inscriptions on arcs going to and from the transition, as well as with the inscriptions on the transition itself (see paragraph about inscription language below). For this purpose, reference nets and Renew utilise a complex unification algorithm, described in [Kummer, 2002, pp. 297–322].

Reference nets can be used to model complex systems. These models are *fully* executable in the Renew environment. Since reference nets support Java as an inscription language (see next paragraph), this means that they can also be used to model complex and sophisticated executable software systems. These systems can utilise the full expressiveness of Java, extended by Petri net aspects for modelling concurrency, conflicts, synchronisation and

[2] Reference nets are formally defined in [Kummer, 2002, p. 218, Def. 10.17]. For the purposes of this thesis and this basic introduction, the formal definition is unnecessary. It would require extensive formal groundwork to be covered, which would inflate this section needlessly. Rather, descriptions in this section focus on the practical aspects and mechanisms of reference, i.e. how to use them for system modelling and software engineering.

[3] Though the object Petri nets in [Lakos, 1995] and [Valk, 1996] share a name, they represent different strands of research.

[4] `https://www.java.com` (last accessed May 28th, 2017)

[5] Which are both, in the context of this thesis, reference nets with special form and structure.

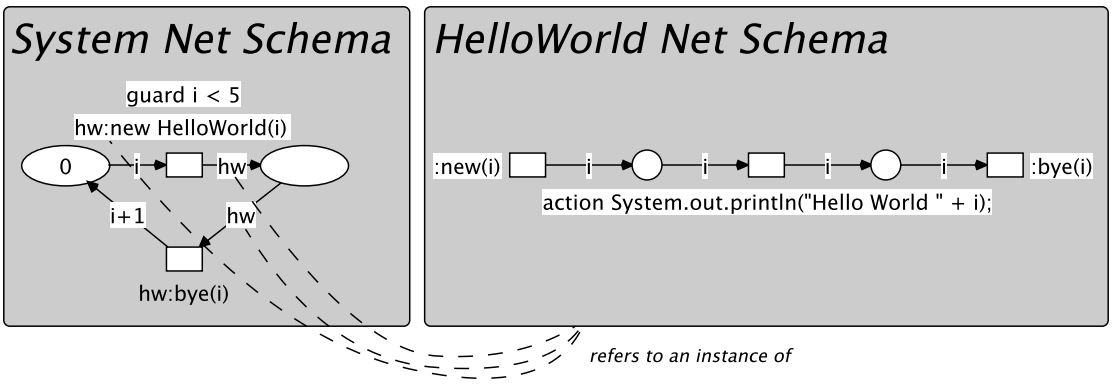

Figure 2.3: Guards, actions and synchronous channels in reference nets

unification. Concerning terminology, the term *modelling* associated with Petri net models and the terms *realising* and *implementing* usually associated with software systems are interchangeable in the context of reference Petri nets. They are therefore used synonymously in this thesis.

Inscription Language The inscription language for reference nets in Renew is Java-based. There are some differences between standard Java code and net inscriptions for arcs and transitions.

With some restrictions, e.g. binary logical operators cannot be directly used, simple Java statement can be inscribed to arcs. An important aspect of arc inscriptions regards the assignment of variables. Variable names are only valid for the locality of a transition and assignments are only valid for one firing. This means that the same token can be assigned to different variables in different localities and, more importantly, that different tokens can be assigned to the same variable in consecutive firings.

More complex Java expressions are possible on transitions. Action inscriptions describe actions that are executed during the firing of a transition. Each individual action inscription is prepended by the keyword `action`, yet there can be an arbitrary number of these action inscriptions. Transition inscriptions can include and combine assignments of variables and Java method calls. Another kind of transition inscription is a guard. Prepended by the keyword `guard`, a guard is a logical expression inscribed to a transition that must be evaluated to true before that transition can fire. Guards are useful for managing the control flow of reference net systems.

More details about the inscription language for Renew can be found in [Kummer, 2002, pp. 247–265] and [Kummer et al., 2016, pp. 42–46].

Net Communication An important aspect for nets-within-nets is net communication. In Renew that communication is handled by synchronous channels, a mechanism for synchronising the firing of and transmitting data between transitions in nets. They were introduced for coloured Petri nets in [Christensen and Hansen, 1994].

The basic idea behind synchronous channels is to connect two transitions via a specific named channel. That channel enforces a synchronous (simultaneous) firing of the transitions and allows bidirectional communication and transmission of data between them. Data transmitted is inscribed in the channel and relates to an incoming arc of one of the

transitions. During firing the data is available to both transitions and their localities. Note that transitions connected via synchronous channels can't fire individually.

Reference nets in Renew distinguish between initiating and receiving transitions. An initiating transition is inscribed with a so-called *downlink*, while a receiving transition contains a corresponding *uplink*. A downlink has the form *net_instance_variable:channel_name(data)*. The *net_instance_variable* must be bound to a net instance in the locality of the transition[6]. That net instance must contain an activated transition with an uplink of the form *:channel_name(data)*. The data expression may contain references to multiple tokens and other objects, e.g. objects created in action inscriptions on the firing transitions, originating from either or both of the initiating or receiving transition localities. Synchronous channels in Renew are described in [Kummer et al., 2016, pp. 48–51].

Figure 2.3 shows an example of a simple reference net system, referencing classic *Hello World* examples. The left-hand net schema is the *system* net. The upper transition contains a guard inscription that ensures that the transition can only fire if there is a token on the precondition place and if that token is an integer with a value less than five. The inscription `hw:new HelloWorld(i)` on the transition is a special inscription that creates a new net instance of the net schema *HelloWorld* (right-hand net schema), binds it to the variable `hw` and transmits the integer `i` via a special synchronous channel `new`. After firing, a token representing the new net instance of *HelloWorld* is put on the right-hand place in the system net instance. The *HelloWorld* net instance then executes independently from the system net. It receives the integer `i` via synchronous channel during instantiation, then uses the standard Java mechanism to print a *Hello World* String onto the console, and finally synchronises with the lower transition of the system net by returning `i` via the `bye` channel. The output arc in the system net increments the value of `i`, reenabling the previous cycle until `i` is increased to five. This simple example showcases the nets-within-nets in reference nets, the inscription language, as well as synchronous channels.

Renew

Renew the ***Re**ference **Ne**t **W**orkshop* is the modelling and execution environment for reference nets. It was developed alongside the reference net formalism in [Kummer, 2002]. A more concise overview can be found in [Kummer et al., 2003] or [Kummer et al., 2004]. Renew was used for the creation of all net models for this thesis.

Renew is freely available from its website[7]. Most of Renew is available as open-source under the GNU Lesser General Public License. The currently (May 2017) released version of Renew is 2.5.

The general graphical interface of Renew can be seen in Figure 2.4. It consists of the main window, as well as individual windows for net schemas (white background) and executing net instances (blue background). The main window contains tool palettes. Figure 2.4 shows the default palettes for drawing reference nets and geometric forms.

Internally, Renew employs an extensible plugin architecture. Plugins allow the modular and flexible expansion of functionality for Renew. Most prototypes developed in the context of this thesis are also realised as Renew plugins. For more information on Renew plugins, see [Duvigneau, 2010].

Shadow Nets Renew differentiates between data models for its editor (representation) and simulator (execution). After modelling/drawing a net schema in the editor, it is

[6] For a synchronous channel call within the same net instance, it is also possible to use the `this` keyword for a net to reference itself.

[7] `http://renew.de` (last accessed May 28th, 2017)

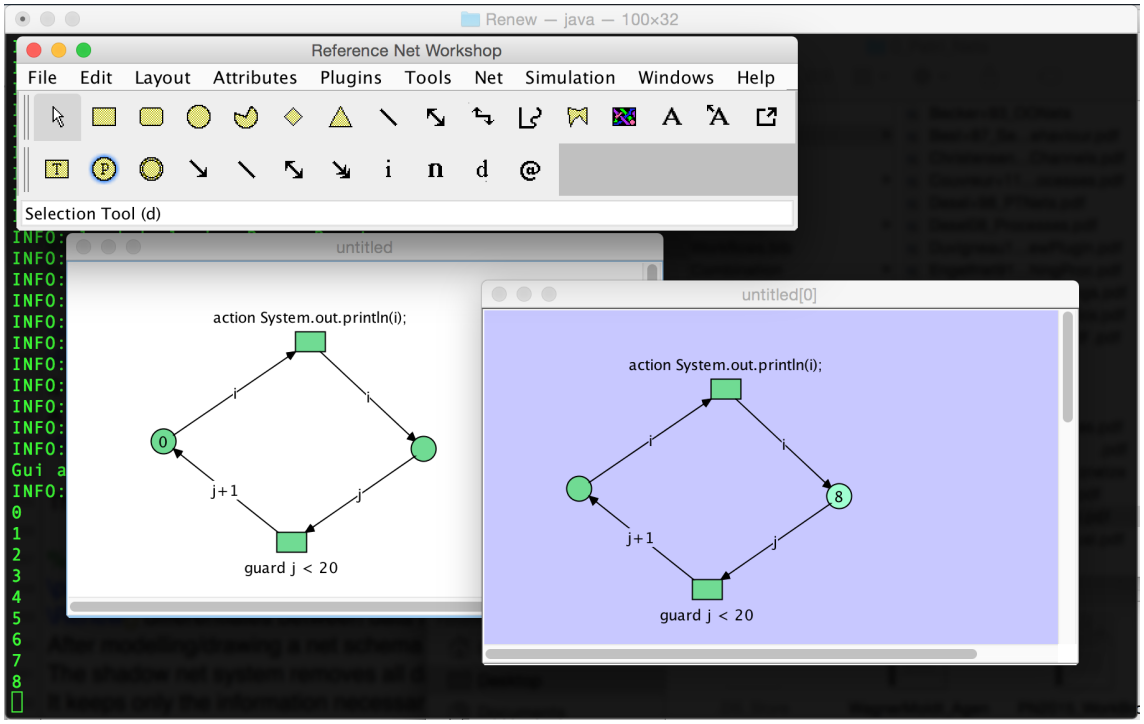

Figure 2.4: Renew graphical interface with running simulation (screenshot)

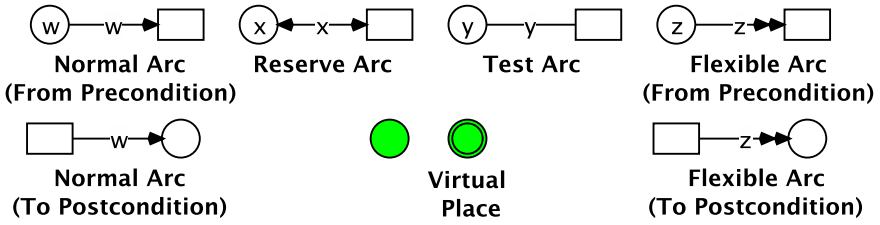

Figure 2.5: Modelling mechanisms supported in reference nets in Renew

compiled into a so-called *shadow net system*. The shadow net system removes all data used only for representation purposes, e.g. positions of net elements, colours, fonts. It keeps only the information necessary for correctly executing the system. In some cases, e.g. the task-transition for workflow nets in Renew (see Section 2.3.2), the shadow net mechanism is also used to hide technical details of complex net mechanisms.

Modelling Mechanisms Reference nets in Renew support multiple arc types. The main arc types are illustrated in Figure 2.5. All arcs can be inscribed with variables or constants to determine what tokens need to be removed from or put on places. Arcs without an inscription assume one simple token.

Normal Arcs remove the inscribed token from a precondition place or put the inscribed token on a postcondition place. They are drawn in Renew with a single arrowhead on one side.

Reserve Arcs remove the inscribed token during the firing but put it back directly after the firing. They "reserve" the inscribed token so that it is not available for concurrent behaviour. They are drawn in Renew with arrowheads on both sides.

Test Arcs assure (or "test") that the inscribed token exists when firing starts. The token is not removed during firing and is still accessible for concurrent behaviour. They are drawn in Renew without arrowheads.

Flexible Arcs allow for a variable amount of tokens to be moved in one firing step. They are drawn in Renew with double arrowheads on one side.

Another modelling mechanism supported by Renew is the *virtual place*. Virtual places are references to other places in the same net. Virtual places are used to improve readability, especially in large nets. If a place is accessed in multiple locations, virtual places eliminate the need for long arcs cutting through a net. They are indicated by double lined places and (by code convention) share a colour in the representation (see Figure 2.5).

2.2 Agent-Orientation

Agent-orientation emphasises the concept of software agents. The domain of software agents are distributed systems, in which autonomy of computational units is very relevant. Other properties often associated with agents are, for example, intelligence, asynchronous interaction and mobility.

This section is divided into three subsections. Section 2.2.1 provides a general introduction to the research field of agent-orientation. It mostly deals with the definition of the term *agent*, as well as common agent properties. Next, Section 2.2.2 gives an overview of the FIPA (Foundation for Intelligent Physical Agents) standards for agent-based systems, which are used in many well-known agent systems. Finally, Section 2.2.3 describes the specific agent context for this thesis, given by the Mulan agent architecture, the Capa agent implementation and the Paose development approach.

For a more detailed, general introduction see [Wooldridge, 2009], which is the de-facto standard work on agent-orientation. [Braubach, 2007] and [Pokahr, 2007] also provide a comprehensive description of the field of research.

2.2.1 Agents and Multi-Agent Systems

Historically, the concept of software agents originates in the field of artificial intelligence[8]. According to [Kay, 1984] the "*idea of an agent originated with John McCarthy in the mid-1950's, and the term was coined by Oliver G. Selfridge a few years later, when they were both at the Massachusetts Institute of Technology. They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An agent would be a "soft robot" living and doing its business within the computer's world.*" [Kay, 1984, p. 6] One of the first works that signifies the move from artificial intelligence towards modern agent-orientation is [Hewitt, 1977]. The actor model described in that contribution emphasises a society of actors and the communication between them. Since these early days agent-orientation has evolved to become a separate, individual field of research.

In order to further discuss agent-orientation the term *agent* itself needs to be clearly defined. There are a number of definitions of the term agent available. The following lists some examples.

[8] This short passage on the history of agents is partially based on the overview given in [Rölke, 2004, pp. 13–18]

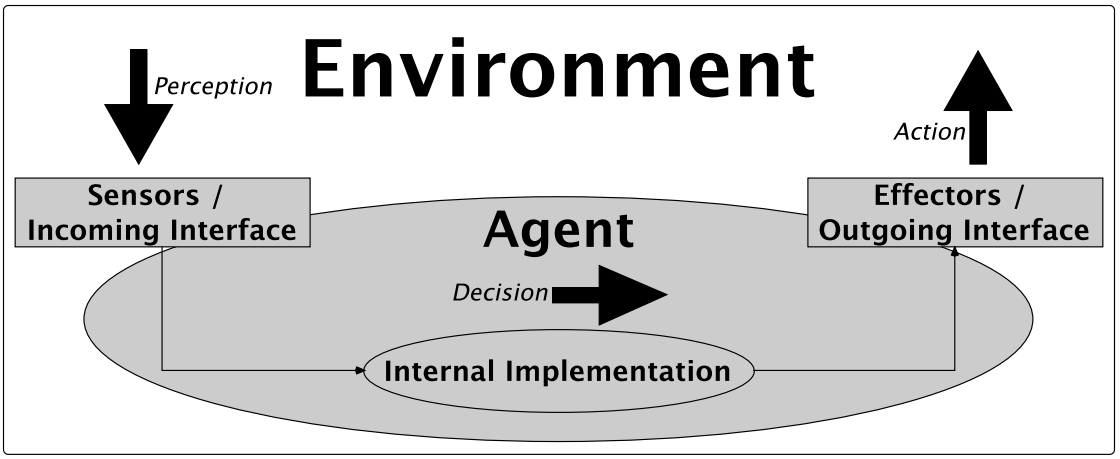

Figure 2.6: An agent in its environment (adapted from [Wooldridge, 2009, p. 22], after [Russel and Norvig, 1995, p. 35] and [Rölke, 2004, p. 24])

Definition 2.3 (Agent (Variant I))**.** *An agent is referred to as* "an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist." *[Shoham, 1993, p. 52]*

Definition 2.4 (Agent (Variant II))**.** "An agent is a computational process that implements the autonomous, communicating functionality of an application." *[FIPA01L, 2002, p. 22]*

Definition 2.5 (Agent (Variant III))**.** "An agent is an autonomous computational individual or object with particular properties and actions." *[Wilensky and Rand, 2015, p. 1]*

Definition 2.6 (Agent (Variant IV))**.** "An Agent is an atomic autonomous entity that is capable of performing some (potentially) useful function." *[Evans et al., 2001, p. 15]*

Definition 2.7 (Agent (Variant V))**.** *An agent* "is a concrete specialized autonomous entity representing a self-contained entity that is capable of autonomous behavior within its environment. An agent is a special object having at least the following additional features: autonomy, i.e. control over its own state and behavior, based on external (reactivity) or internal (proactivity) stimuli, and ability to interact, i.e. the capability to interact with its environment, including perceptions, effecting actions, and speech act based interactions." *(adapted) [Červenka and Trenčanský, 2007, p. 40]*

Key Term Definition A.1 (Agent)**.** "*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.*" [Wooldridge, 2009, p. 21]

Note that the last definition is a key term definition. Key terms are used heavily throughout the thesis and warrant special attention. Key term definitions are not more important than "normal" definitions, they are only used more often. Many later key term definitions also define terms that are newly introduced in this thesis. For easier reference they are repeated at the end of the thesis in the key term glossary in Appendix II.

There are a number of common properties found in these definitions. All of the definitions emphasise the autonomy of agents[9].

Autonomy: Autonomy relates to the ability of agents to decide for themselves without outside interference. In other words, "*the quality of autonomy means that an agent's actions are not solely dictated by external events or interactions, but also by its own motivation.*" [Evans et al., 2001, p. 15] Autonomy is also the property that most clearly distinguishes agents from objects in object-oriented software development. The following quote from Michael Wooldridge sums this aspect up quite nicely: "*Objects do it for free, agents do it because they want to.*" [Wooldridge, 2009, p. 29]

Another property that is found explicitly in most of these definitions is that agents are situated or executed in some form of environment, in which they are capable of actions. Many of these definitions also stress that agents possess a purpose, function or some form of objective.

Encapsulation: A direct consequence of the autonomy is that agents encapsulate their purpose. They are an independent component within their environment that has (autonomous) control over its internal state, data and functionality.

Figure 2.6 illustrates these properties of the different definitions. An agent, situated in an environment, perceives stimuli from the environments with its sensors, uses its internal implementation to decide on actions and performs actions in the environment using its effectors.

There are further agent definitions that don't emphasise the same properties. Some are too general for the purpose of this thesis (e.g. "*An agent is just something that perceives and acts.*" [Russel and Norvig, 1995, p. 7]), while others are too specific with further concrete agent properties (e.g. "*An agent is an autonomous, adaptive and interactive element that has a mental state.*" [Silva et al., 2003, p. 8]). Very few leave out the autonomy aspect explicitly (e.g. an agent is "*an entity that performs a specific activity in an environment of which it is aware and that can respond to changes.*" [Sterling and Taveter, 2009, p. 7]), which makes them unsuitable for this thesis.

For the purposes of this thesis Definition A.1 from [Wooldridge, 2009] is best fitting. It deals with agents on a high enough abstraction level and captures all of the aforementioned properties. It is also the most widespread and adopted. Many agent contributions use it or variations of it, e.g. [Jennings, 2000, Luck et al., 2003].

The general approach in agent-orientation is to partition the functionality of the overall system onto a number of agents. The common term for a system consisting of agents is *multi-agent system.*

The purpose of a developing multi-agent system is to pursue a divide and conquer strategy in the partitioning of the functionality. Consequently, agents require additional properties. These properties are included within the term *intelligent agent.* The term is used in [Wooldridge, 2009, pp. 26–27] to further enhance the definition of *agent.* Intelligent agents are agents that exhibit reactivity, proactiveness and social ability:

Reactivity: Reactivity relates to an agent's ability to observe its environment and react to stimuli it perceives within the environment. These stimuli can be direct (e.g. a message for the agent) or indirect (e.g. the agent's sensors perceive a change in a data repository).

[9] Agent properties in this section are presented in a uniform way as itemised descriptions. This is done to highlight them and make their identification for later discussions in this thesis easier.

Proactiveness: Proactiveness describes an agent's ability to initiate actions themselves. Proactive behaviour is usually related to the agent's function or objective. If the agent internally determines that a certain action should be taken it can execute that action without any outside stimulus.

Social Ability: Social ability describes that an agent can communicate and interact with other agents within the environment in order to fulfil its objective. Social ability also includes the ability of an agent to communicate and interact with humans. This property is very important in multi-agent systems as it is the basis for the advantageous partitioning of functionality.

A concept strongly related to social ability is emergence. Emergence is "*the arising of novel and coherent structure, patterns, and properties through the interactions of multiple distributed elements.*" [Wilensky and Rand, 2015, p. 6] Multi-agent systems can utilise the emergence of interacting agents to increase beneficial aspects such as, for example, efficiency and security. Emergence is one of the key qualities of multi-agent systems and their internal partitioning of functionality and wouldn't be possible without social ability.

Regarding Figure 2.6 the properties of intelligent agents fit as well. Reactivity is the entire process between perceiving a stimulus from the sensors/incoming interface, making an internal decision and then performing an action via the effectors/outgoing interface. Proactiveness is similar, with the only difference being the missing external stimulus. Social ability refines the actions an agent performs in the environment into communication and interaction with other agents, components or humans in the environment.

The basic properties of intelligent agents further substantiate the understanding of the concept agent and make them more suitable for the development of complex multi-agent systems. For the context of this thesis, the terms agent and intelligent agent are synonymous. All agents that are considered, created and discussed in the remainder of this thesis are entities that, in some form or another, are intelligent agents in the sense of [Wooldridge, 2009, pp. 26–27].

In addition to the properties of intelligent agents [Jennings and Wooldridge, 1995] also identifies some potential and desirable properties of agents. These are adaptability, mobility, veracity and rationality.

The properties discussed in this section make agents a powerful and expressive tool for software developers. The domain of multi-agent systems are distributed systems. "*A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.*" [Coulouris et al., 2001, p. 1] Distributed systems emphasise the relative independence of their components as well as the interactions between them. This is why intelligent agent properties, especially autonomy and social ability, make agents perfectly suited for this field.

2.2.2 FIPA Standards

The ***F**oundation for **I**ntelligent **P**hysical **A**gents*[10] (FIPA) is an IEEE[11] Computer Society standards organisation. The FIPA was founded in 1996 with the goal of providing standards for software agents that promote interoperability and reusability. FIPA maintains more than 20 documents classified as standards dealing with different aspects of multi-agent systems.

[10] `http://www.fipa.org` (last accessed May 28th, 2017)
[11] Institute of Electrical and Electronics Engineers, `https://www.ieee.org` (last accessed May 28th, 2017)

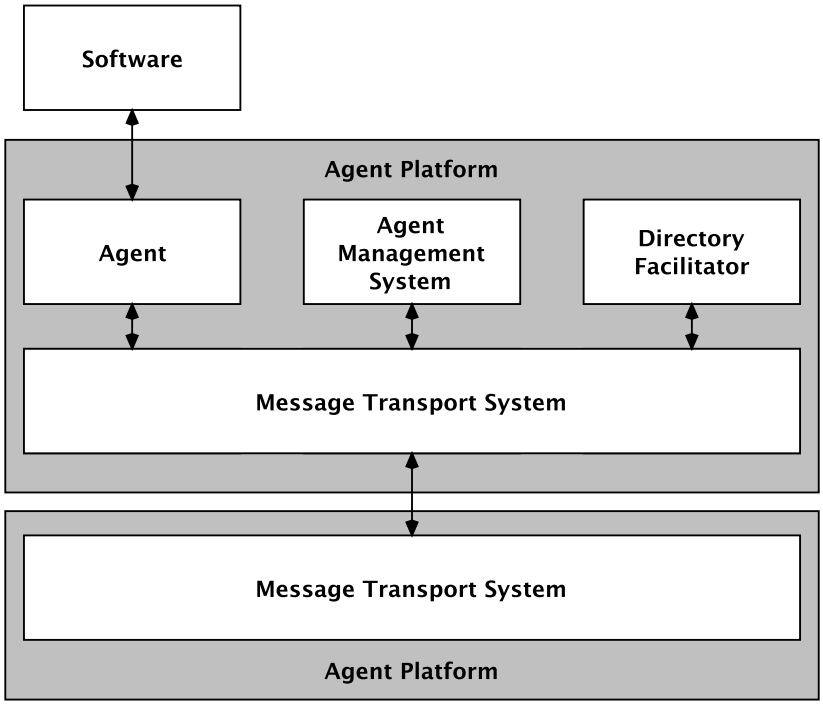

Figure 2.7: FIPA agent management reference model (adapted from [FIPA23K, 2004, p. 5])

The following shortly describes the most relevant ones for the context of this thesis. The selection features those standards that deal with agent-orientation in general and especially with the interoperability focus. These standards are relevant for this thesis, as they provide widely recognised views on agents, which are useful for the deliberations on agents later on in the thesis. Other FIPA standards deal with, for example, technical implementation details or specific protocol definitions.

FIPA Abstract Architecture Specification SC00001L [FIPA01L, 2002]

This standard defines the FIPA abstract architecture, which describes the architectural components of an agent system, as well as the relations between them. The focus of the abstract architecture is to describe the different components in such a way that it promotes interoperability of agent systems. By defining what components function abstractly in which way certain assumptions can be made that make interactions between otherwise heterogeneous systems possible. This is a paramount concern especially in distributed systems. The standard itself states that the "*[...] FIPA Abstract Architecture defines at an abstract level how two agents can locate and communicate with each other by registering themselves and exchanging messages.*" (adapted) [FIPA01L, 2002, p. 11]

FIPA Agent Management Specification SC00023K [FIPA23K, 2004]

The FIPA Agent Management Specification describes agent management, i.e. "*the normative framework within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents.*" [FIPA23K, 2004, p. 5] Agent management specifies which services are required from an *agent platform*, i.e. the execution environment of and

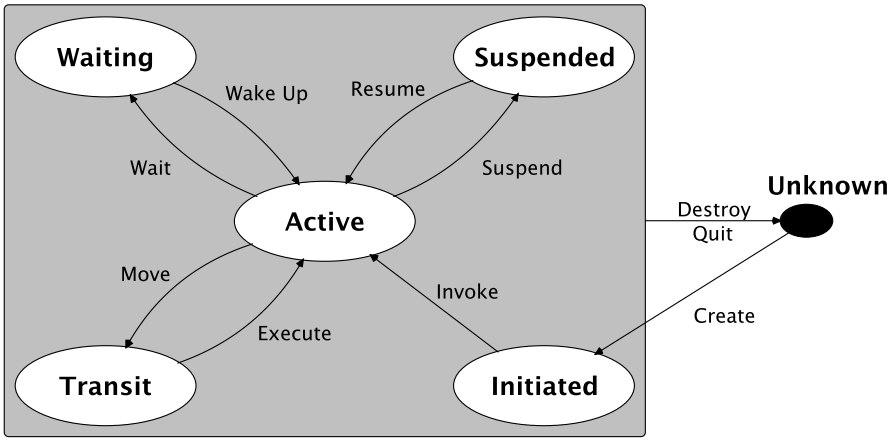

Figure 2.8: FIPA agent life cycle (adapted from [FIPA23K, 2004, p. 14])

infrastructure for agents, and how these services interact with other, external services. This issue is represented in the agent management reference model show in Figure 2.7.

The individual components of an agent platform are shortly described in the following. *Agents* publish their capabilities in the optional *directory facilitator* (DF), which provides a yellow-pages like service on an agent platform. The *agent management system* (AMS) supervises, manages and controls access to the agent platform. Communication infrastructure is handled by the *message transport system*.

The standard also deals with the life cycle of agents, shown in Figure 2.8. The life cycle covers the states *unknown*, *initiated*, *active*, *waiting/suspended* and *transit* (regarding agent mobility).

FIPA ACL Message Structure Specification SC00061G [FIPA61G, 2002]

The abstract architecture defines that agents communicate using the agent communication language (ACL). ACL messages consist of a set of message parameters that describe the content and function of a message. The FIPA ACL Message Structure Specification contains the specifications for these parameters.

FIPA SL Content Language Specification SC00008I [FIPA08I, 2002]

The FIPA Semantic Language (SL) is the standardised content language for agent communication within the FIPA. In other words, the content of ACL messages is written in SL. This standard document describes the syntax and partially the semantics of SL. The expressiveness of SL often isn't necessary in ordinary agent systems. For that purpose the standard defines three less expressive subset of SL: SL0, SL1, SL2. The minimal subset SL0, for example, is used heavily in Capa (see Section 2.2.3).

FIPA Communicative Act Library Specification SC00037J [FIPA37H, 2002] This standard describes the communicative acts used in agent communication in FIPA. Communicative acts originate from the speech acts theory [Austin, 1962]. The communicative acts for agents in FIPA describe how certain communicative exchanges, e.g. confirmations, calls for proposals, have to be handled by agents. The standard defines the formal models for ACL messages realising these communicative acts.

2.2.3 Mulan, Capa and Paose

The previous section described agent-orientation in general. The current section now focuses on the specific agent context for this thesis: Petri-net based software agents. There are three major areas of interest in this section: Mulan, Capa and Paose.

Mulan is the general agent model. It is based completely on reference nets and provides the general view on agents and complete multi-agent systems assumed in this thesis. Capa, an implementation of the concepts of Mulan, adds more technical and practical considerations, as well as compliance to the FIPA standards (see Section 2.2.2). Capa serves as an implementation basis for integration prototypes in later chapters. Finally, Paose proposes a way to structure the development process of agent systems in a way that considers the different developers as agents themselves.

Mulan

Mulan, short for Multi-Agent Nets, describes multi-agent systems as systems of reference nets. Mulan is described in [Rölke, 2004]. Every aspect of a system, from the overall system infrastructure to the individual agent behaviours, is implemented as reference nets with Java inscriptions.

The core of Mulan is the Mulan reference architecture, shown in Figure 2.9. It describes multi-agent systems on four elemental levels: protocols[12], agents, platforms, systems. Each level is nested (and executed) in the one directly above it: Protocols are executed by agents, which are managed by platforms, which interact via the system infrastructure. Consequently, the *zoom* relation in Figure 2.9 does not represent a place refinement. *Zoom* refers to the net that is represented by a token in an upper level of the architecture. Communication between the different levels is handled via synchronous channels. For example, the **Out** transition in the protocol net in Figure 2.9 synchronises with the **Protocol Out** transition in the agent net. In the next step the **Out** transition in the agent net synchronises via one of the two communication transitions, **Internal Communication** and **External Communication** in the platform net. That firing also binds and synchronises an additional agent net and **In** transition in that net.

The following describes the four levels of the Mulan reference architecture. For more details, please refer to [Rölke, 2004, Chapter 7, pp. 153–193].

Protocol Level (Agent) protocols describe the behaviour of agents in Mulan. They can be regarded as instructions for agents. These instructions consist of actions including, for example, sending messages, waiting for and receiving messages, data processing or knowledge access. Each protocol describes and defines one specific ordered set of actions that an agent takes in a given context. That context is, usually, a larger interaction between a group of agents. Interactions consist of multiple protocols executed by multiple agents, where each protocol describes the connected behaviour of one agent (role[13]) involved in the interaction.

Agent Level Mulan agents are the main modelling construct in Mulan. Mulan agents fulfil Definition A.1 in Section 2.2.1 and are also intelligent agents in the sense of the

[12] In the context of this dissertation the term protocol does not possess any relation to technical communication protocols such as TCP or IP. Protocols in Mulan, Capa and the prototypes in later chapters describe executed behaviour or behaviour templates for one agent or entity.

[13] Modellers don't design each agent separately, but rather design agent roles. These agent roles define behaviour and knowledge for each (instantiated) agent that is assigned that role. An agent in Mulan can have multiple agent roles and an agent role can be assigned to multiple agents.

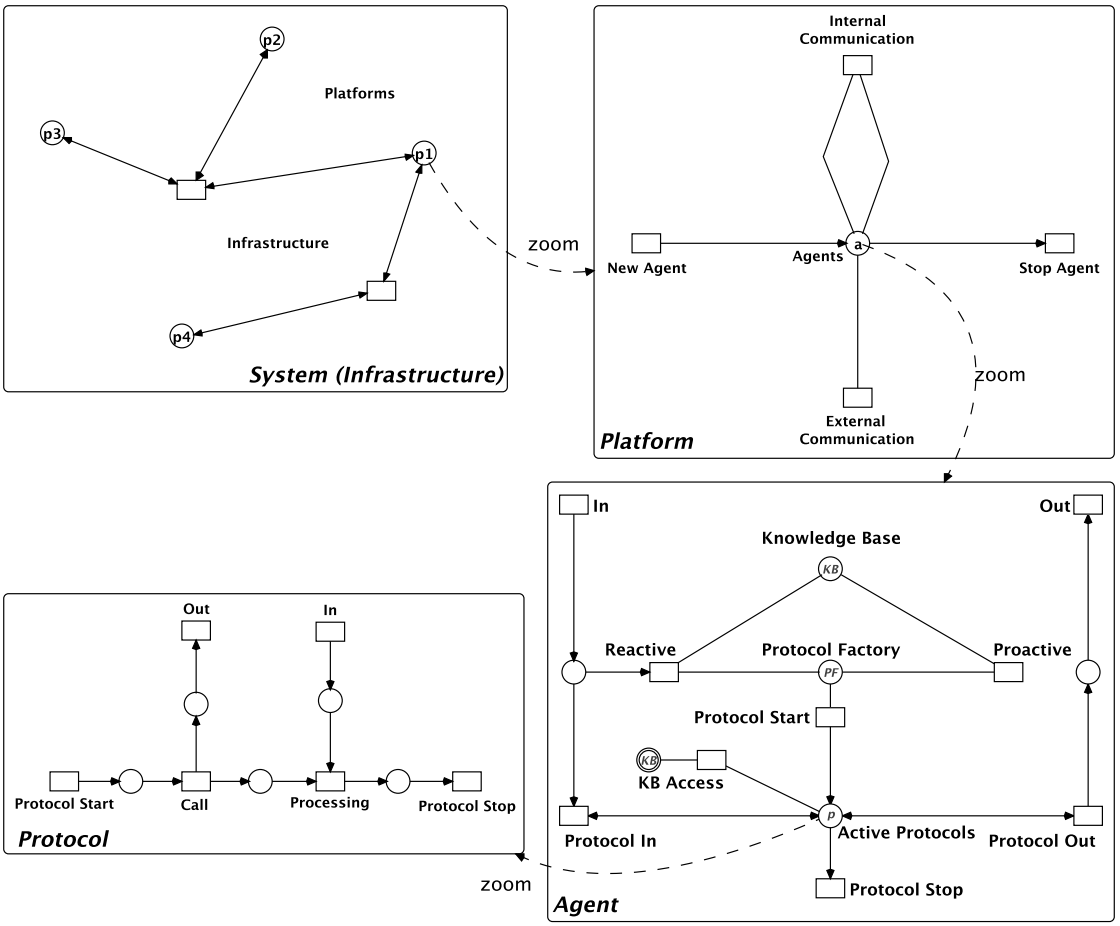

Figure 2.9: MULAN reference architecture
(edited and translated from [Rölke, 2004, p. 157]. Depiction of agent net edited, translated and simplified from [Rölke, 2004, p. 169])

definition in [Wooldridge, 2009, pp. 26–27]. They maintain autonomy over their own actions, meaning that neither the platform, other agents nor other components of the system can control their behaviour. They also maintain their own state and data, meaning that encapsulation is also fulfilled. Regarding social ability, MULAN agents possess two interfaces to the environment. These interfaces are the transitions **In** and **Out** in the agent net in the lower right corner of Figure 2.9. Via the **In** transition they can receive asynchronous messages, while they can send asynchronous messages via the **Out** transition.

MULAN agents also support proactive and reactive behaviour. Within the agent net, proactive and reactive behaviour refers to starting new protocols proactively or reactively. Other proactive and reactive behaviour can be modelled directly within the protocols.

Proactive behaviour is triggered from the knowledge base. The knowledge base in MULAN is another reference net that stores and manages the internal data, or knowledge, of one agent. Changes within the knowledge, e.g. through actions executed in a protocol, can trigger the proactive behaviour via the **Proactive** transition in the agent net. This causes the protocol factory to start a new protocol for the agent. The protocol factory is another internal reference net of the agent. It is, as the name suggests, the component responsible for creating and starting protocols.

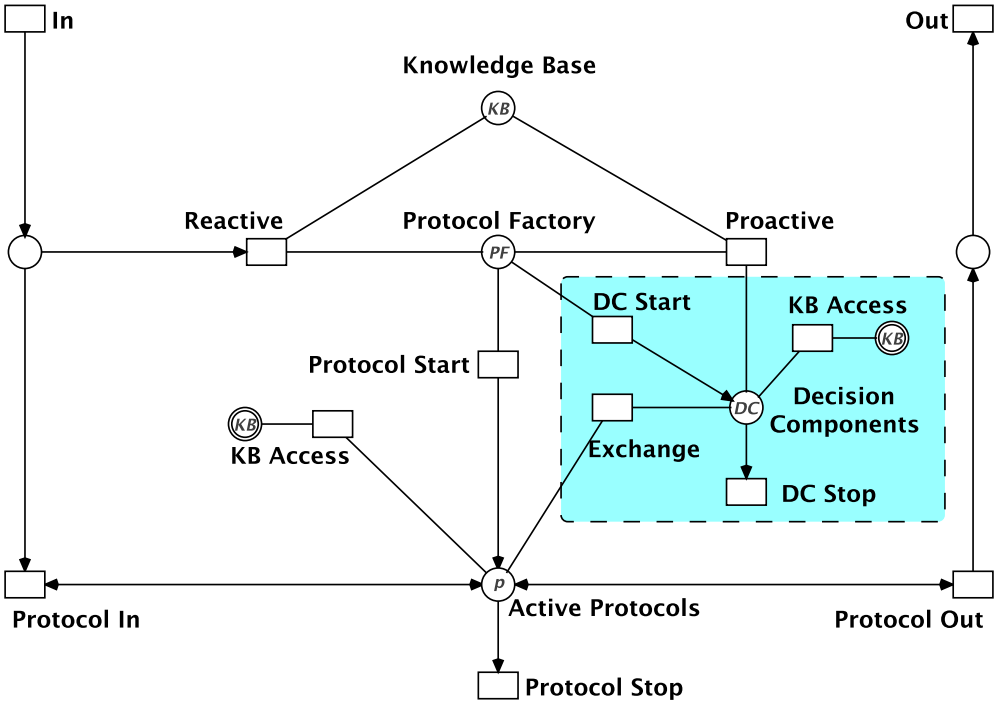

Figure 2.10: Extended Mulan agent model
(edited and simplified from [Cabac, 2010, p. 51])

Reactive behaviour is implemented similarly to proactive behaviour. When a message arrives via the **In** transition it is routed within the agent net either to a running protocol or to the **Reactive** transition. That transition looks up the correct protocol associated with the message in the knowledge base and then causes the protocol factory to start that protocol.

Mobility is a property that is conceptually maintained in [Rölke, 2004]. Even though not represented in Figure 2.9, Mulan supports the migration of agents between platforms.

Figure 2.10 shows an extension to the standard Mulan agent, incorporating the decision component (DC) mechanism. Decision components implement internal behaviour of agents. In contrast to protocols, DCs do not possess the ability to directly communicate with the environment of the agent, but need to pass and receive data to protocols for that. DCs "*can be interpreted as internal services offered to protocol nets.*" [Cabac, 2010, p. 52] While not part of the original Mulan architecture, DCs are heavily utilised in the practical Mulan contexts, especially in Capa applications.

Platform Level Agent platforms in Mulan serve as the direct execution environment for agents. They also manage the life-cycle of agents. Figure 2.9 highlights four mechanisms of the platform. Platforms can start new agents and stop the execution of running agents. They also support internal (i.e. on the same platform) and external (i.e. on different platforms) communication between agents. These mechanisms are realised as transitions in the platform net that bind one or (in the case of internal communication) two agents during firing. Further mechanisms, like the support of mobility, are removed from the architecture overview for readability. Additional management mechanisms are implemented as special agents of the platform. These include the FIPA compliant AMS and DF services (see Section 2.2.2).

System Level The system level of the MULAN architecture describes the overall infrastructure between the different agent platforms. It defines how the individual platforms are related and interconnected within the system. These connections define which platforms can interact with which other platforms. This also defines, which agents can directly communicate with one another. Basically, the system layer defines the topology of the system, which provides a global, abstract perspective.

Before moving on, the modelling aspects of MULAN are important. Most of the MULAN components are standardised, i.e. they are used without changes in each application built with MULAN. The application-specific modelling content include the protocols (and interactions), the application-specific DCs and the initial agent knowledge. These three areas need to be defined for each agent role of an application. In addition to that, the system level needs to be fully modelled to define the overall communication infrastructure.

Another important aspect of application-specific modelling is the ontology. The ontology defines concepts and objects within the system. It is very important for any system, since it describes the common vocabulary for agents necessary for them to interact and understand one another.

Capa

The CONCURRENT AGENT PLATFORM ARCHITECTURE (CAPA) is a technical implementation of MULAN described in [Duvigneau, 2002, Duvigneau et al., 2002, Duvigneau et al., 2003]. Within the TGI[14] research group it is the de-facto standard framework for the realisation and implementation of agent applications.

MULAN's reference architecture focuses on the concepts of agents and relies exclusively on reference nets. One of the issues with this approach is the limitations regarding distributed execution of reference nets system. CAPA performs the step moving from the model of a distributed system to an actually, physically distributed system [Duvigneau, 2002, p. 69]. For this reason the CAPA implementation provides extensive enhancements to the platform level of MULAN.

These extensions have made the system level of MULAN obsolete from a modelling point of view. Instead of modelling the system manually, the system is automatically defined by the actual, physical network topology. Considering the system level as implicit can still yield useful observations of the system though, which is why it is maintained on the conceptual level throughout this thesis.

Another focus of the CAPA implementation is the compliance with the FIPA standards (see Section 2.2.2). While MULAN is oriented around FIPA, it is not fully compliant and thus cannot ensure interaction with other, heterogeneous agent systems. CAPA fully realises the FIPA standards for interoperability of agent systems. The interoperability of CAPA was confirmed by its incorporation into the Agentcities project [Reese, 2003].

With the exception of the system level, CAPA maintains the principles of MULAN. Agents are executed on and managed by agent platforms. Their behaviour is defined by their protocols in the context of larger interactions. Consequently, the application-specific modelling remains identical with the omission of the system level.

Agent properties are also maintained and have been the focus of research themselves. For example, mobility is a focus in the MAPA (**M**obile **A**gent **P**latform **A**rchitecture, [Cabac et al., 2009]) extension of CAPA. For the context of this thesis, CAPA serves as the technical

[14] *Theoretische Grundlagen der Informatik* (engl. *Theoretical Foundations of Computer Science*) `http://www.informatik.uni-hamburg.de/TGI/` (last accessed May 28th, 2017)

basis for the developed prototypes in Chapter 10. The remainder of this section illustrates some of the most important, w.r.t. this thesis, aspects and mechanisms of Capa. This only represents an excerpt of the overall framework though. Many technical details are also omitted at this point of the thesis, because they only become relevant later on. These details are addressed, wherever necessary, in the later chapters.

Capa Agent Net Figure 2.11 shows the Capa standard agent net with all technical inscriptions. The basic Mulan agent structure is still recognisable. The agent possesses two interfaces to the environment and the realisation of reactive and proactive behaviour is unchanged. DC interaction is refined by providing three data exchange transitions (exchange with new id, exchange with existing id, exchange with no id) each for the communication between DCs and protocols and between DCs themselves. The *Initialization*[15] component initialises the agent by creating the knowledge base and protocol factory nets and setting the initial state of the agent. The *Idle Counters* component measures the activity status of the agent by counting incoming, not yet processed messages, outgoing, not yet sent messages and active protocols. These counters are used to determine if an agent may be terminated or moved in the platform.

Agent Role Model The *Agent Role Model* (ARM, [Mosteller, 2010]) is used to describe the different agent roles of an application. Within the ARM the initial knowledge of an agent role, including its data, DCs and reactive and proactive behaviour, is defined. Furthermore, the ARM describes the deployment of agent roles. It maps agent roles to individual agents, which are then compiled into knowledge base content.

WebGateway The WebGateway is a specialised platform agent. The WebGateway provides services that allow other agents to interact with web services and also provide their own functionality as web services. A web service is defined by the W3C[16] as "*a software system designed to support interoperable machine-to-machine interaction over a network.*" [Booth et al., 2004, p. 7]. A general introduction to web services and service-oriented architectures can be found in [McGovern et al., 2006]. The Capa WebGateway is described in [Betz, 2011, Betz et al., 2013, Betz et al., 2014]. The WebGateway is an important, yet relatively recent addition to the Capa framework. Its most common use is to provide a graphical user interface within a web browser.

Persistent Ontology The persistent ontology plugin [Mosteller, 2016] enables storing ontology objects in an object-oriented database. It supports the Db4o[17] and MongoDB[18] databases, but can be easily extended through a standardised interface. It is implemented in Capa as a number of standardised agent roles, which possess DCs that can connect, via auxiliary Java helper classes, to connected databases.

Paose

The Paose approach (Petri Net-Based Agent- and Organisation-oriented Software Engineering, [Cabac et al., 2008, Cabac, 2010]) is a software developing approach

[15] This thesis is written in British English. American English, German or other language terms are, however, used when they refer to or quote sources (papers, source code, illustrations, etc.) that are not written in British English.
[16] World Wide Web Consortium, `https://www.w3.org/` (last accessed May 28th, 2017)
[17] `https://sourceforge.net/projects/db4o/` (last accessed May 28th, 2017)
[18] `https://www.mongodb.com/` (last accessed May 28th, 2017)

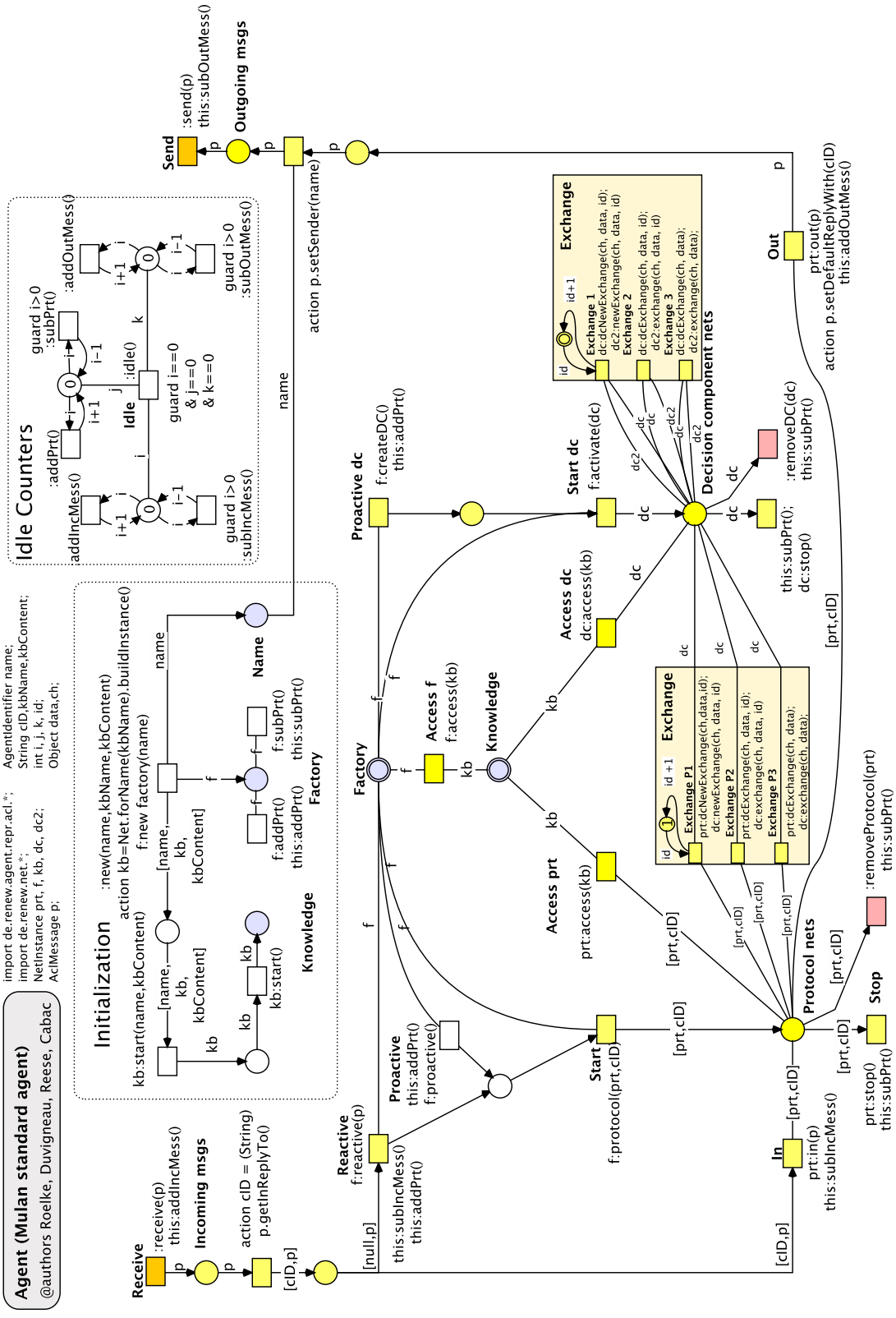

Figure 2.11: Capa standard agent net
(from the developer branch of the Mulan repository, 22.09.2016)

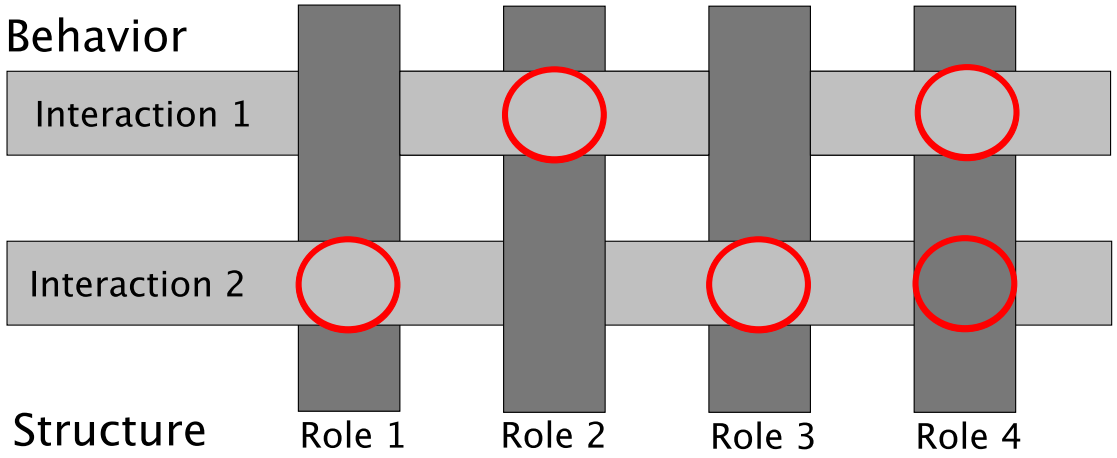

Figure 2.12: PAOSE matrix (from [Cabac, 2010, p. 123])

for multi-agent systems. It is specially tailored for the development of multi-agent systems in CAPA, but is general enough to be applicable in wider contexts.

PAOSE not only develops multi-agent systems, but also uses the concept of a multi-agent system as an overall guiding metaphor (German: *Leitbild*, [Cabac, 2007]) The system under development is a multi-agent system and the development team itself is also regarded as a multi-agent system. Developers are individual agents that cooperate and interact with one another to create the actual software multi-agent system. Using multi-agent systems as a guiding metaphor this way ensures a development completely aligned to the system being developed. Terminology (ontology), procedure and orientation of the development team and the software system are in total compliance to one another.

Figure 2.12 illustrates the basic perspective on multi-agent systems in PAOSE. A multi-agent system consists of three perspectives, two of which are represented directly in the so-called PAOSE matrix in Figure 2.12. The first perspective is the structure of system. The structure consists of the different agent roles that provide individual functionality in the system. The second perspective is the behaviour of the system. The behaviour consists of the interactions between the different agent roles. The connection points (red circles in Figure 2.12) between structure and behaviour determine which roles are involved in which interactions. Such a connection determines that, in the implementation, an agent role possesses a protocol net realising that agent roles specific part of the interaction. At the connection points between roles and interactions, the third perspective of a multi-agent system comes into play: the ontology. In order to realise the relation between different agent roles via an interaction, the agent roles need to be able to communicate via a set of concepts known and understood by all involved agents. That set of concepts is provided by the ontology. In other words, a (multi-agent) system in PAOSE is completely described by its structure (agent roles) and behaviour (interactions) connected through shared concepts (ontology).

The general perspective of PAOSE governs the steps of, models and design artefacts created in the approach. Figure 2.13 illustrates this. First off, a requirements engineering has to determine the requirements of the system under development. From these requirements a coarse design of the system is developed and recorded in a coarse design diagram (CDD). The CDD defines which agent roles and which interactions exist in the system. It is an alternative representation of the PAOSE matrix. The next step, design, separates and

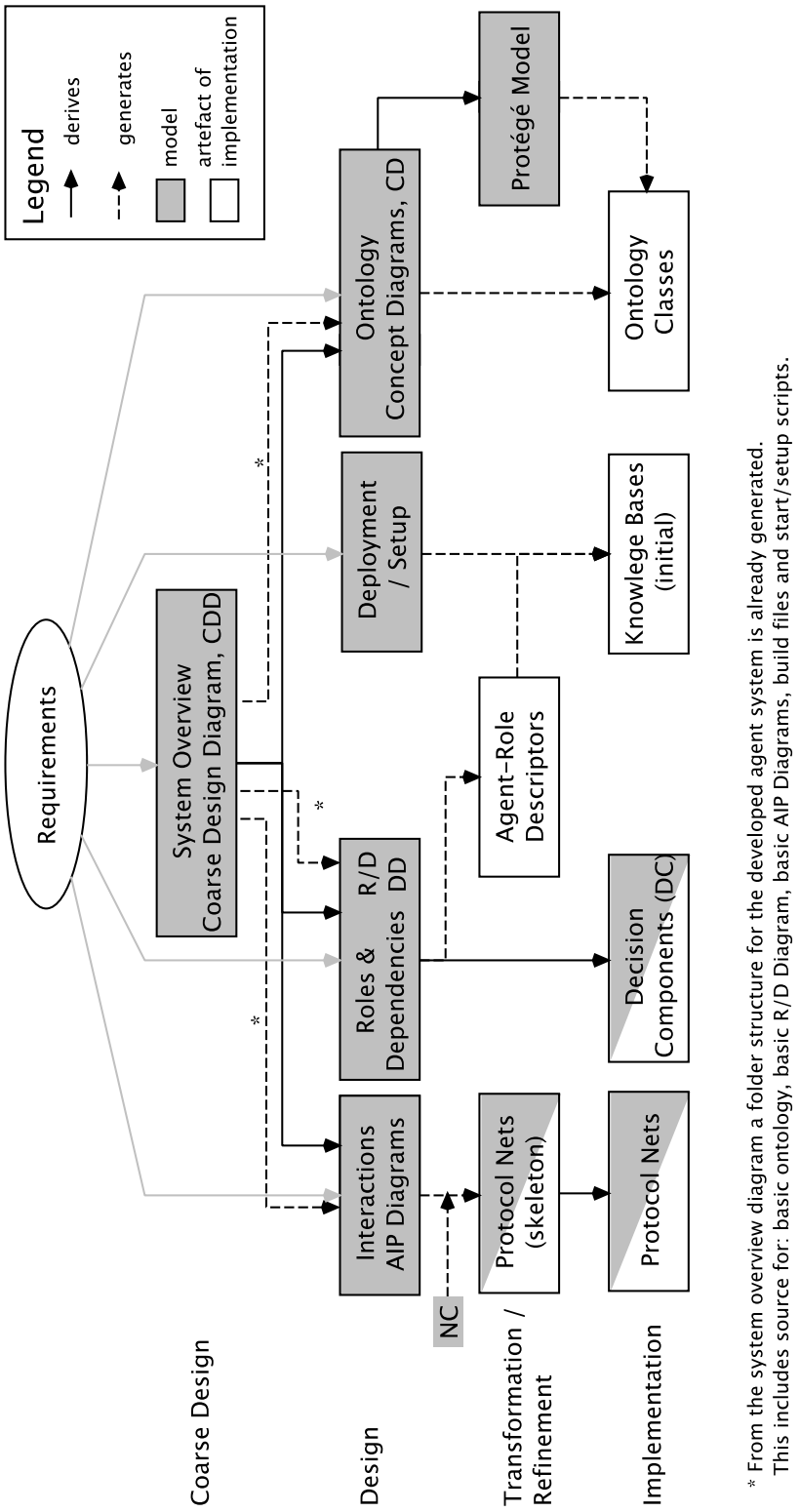

Figure 2.13: PAOSE models and artefacts (from [Cabac, 2010, p. 134])

refines the individual aspects of interactions, roles, deployment and ontology. The result of this step is a set of different design artefacts. Interactions are described in agent interaction protocol (AIP) diagrams[19], roles and deployment in the ARM and the ontology in a concept diagram. The following step transforms these design artefacts into the basis for an implementation. AIP diagrams are transformed into skeleton nets using predefined net components [Cabac, 2002, Cabac, 2009], the ARM generates empty DCs and initial knowledge bases and the concept diagram automatically generates Java classes for each ontology concept. Using these artefacts as a basis, the remainder of the functionality can be implemented. After all aspects are implemented they are integrated and tested in combination. Paose follows agile and iterative ideas, so that the different steps of the approach described above are repeated often in order to realise the system under development.

Tool support for Paose is provided directly in Renew. The MulanViewer and MulanSniffer tools [Cabac and Dörges, 2007] provide monitoring and debugging functionality. The MulanViewer provides a monitor for a running multi-agent system, showcasing all platforms, agents and protocols active at any given time. The MulanSniffer tracks messages sent between agents and continuously updates a diagram of the message exchanges during the execution of a multi-agent system. Verification support was examined in [Hewelt et al., 2011]. A Renew plugin provides an interface for the external Lola verification tool [Schmidt, 2000a], though the extent functionality supported in Renew is still limited.

2.3 Workflows

The day-to-day operations of modern organisations employ a large number of complex processes. Supporting the execution of these (business) processes and the tasks within them in a computerised way is the topic of workflow management.

This section is separated into three subsections. Section 2.3.1 discusses workflow management in general and the basic terminology. Next, Section 2.3.2 describes workflow Petri nets, the specific context for workflows in this thesis. Finally, Section 2.3.3 presents the field of inter-organisational workflows, an important application area.

2.3.1 Workflows and Workflow Management

Workflow Management is generally seen as a part of the overall business process management (BPM) of an organisation. Workflow management is concerned with the modelling, execution, facilitation, automation and management of the individual business processes within an organisation. BPM goes beyond the individual processes and considers the interconnections, relations and dependencies of all processes in an organisation. "*BPM can be seen as an extension of Workflow Management [...]. [Workflow Management] primarily focuses on the automation of business processes [...], whereas BPM has a broader scope: from process automation and process analysis to operations management and the organization of work.*" (adapted) [van der Aalst, 2013, p. 1]

Figure 2.14 shows the so-called BPM lifecycle. The BPM lifecycle illustrates the different aspects of BPM, from initial process identification and discovery, over analyses, redesign, implementation and monitoring/control. Workflow management aspects can be found throughout the BPM life cycle, wherever the management of individual processes is concerned. More information on general BPM can be found, e.g., in [Dumas et al., 2013, vom Brocke and Rosemann, 2010a, vom Brocke and Rosemann, 2010b, Weske, 2012].

[19] AIP diagrams are similar to AML interaction diagrams [Červenka and Trenčanský, 2007].

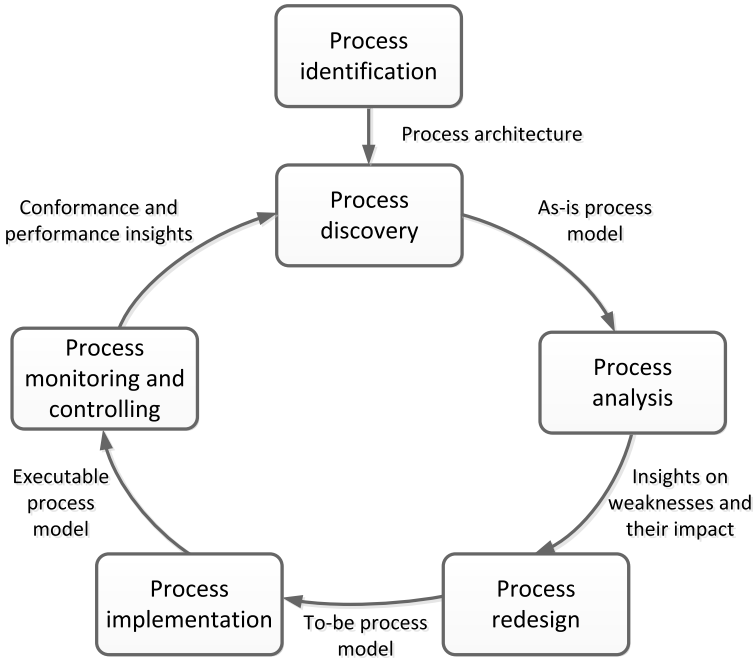

Figure 2.14: BPM lifecycle (from [Dumas et al., 2013, p. 21])

The basic understanding of workflows is defined by the workflow reference model of the *Workflow Management Coalition* (WfMC[20]). The WfMC created the workflow reference model as a standard in 1995 [Hollingsworth, 1995]. That workflow reference model represents a detailed description of workflow management systems (WFMS) and their context. The definitions from the WfMC are provided in two documents: The workflow reference model [Hollingsworth, 1995] and the glossary [Wfmc99, 1999]. Both documents contain definitions that do not always completely match. For the purposes of this introduction and thesis, the more relevant (w.r.t. level of detail) is chosen. Some definitions and terms are adapted in anticipation of terminology necessary for workflow nets and the specific context of this thesis. These cases are explicitly marked and discussed.

As stated above, workflows deal with business processes (BP). BP are, basically, any kind of procedure (in the sense of the German word *"Ablauf"*) in an organisation that is, directly or indirectly, related to creating a value for the organisation. The WfMC defines BP in the following way:

Definition 2.8 (Business Process (BP, Variant I))**.** *"*A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.*" [Wfmc99, 1999, p. 10]*

Another definition, from the field of business process reengineering, is the following:

Definition 2.9 (Business Process (BP, Variant II))**.** *"*We define a business process as a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer.*" [Hammer and Champy, 1993, p. 38]*

[20] `http://www.wfmc.org/` (last accessed May 28th, 2017)

According to [van der Aalst and van Hee, 2002, pp. 9–10] BP can be categorised as primary (production), secondary (support) or tertiary (managerial). Workflows are concerned with the computerised support of all types of BP. The definitions of workflow provided by the WfMC in [Hollingsworth, 1995] and [Wfmc99, 1999] differ slightly:

Definition 2.10 (Workflow (Version A))**.** *"The computerised facilitation or automation of a business process, in whole or part." [Hollingsworth, 1995, p. 6]*

Definition 2.11 (Workflow (Version B))**.** *"The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [Wfmc99, 1999, p. 8]*

Definition 2.10 contains the facilitation aspects left out of Definition 2.11. Definition 2.11, on the other hand, contains the explicit aspect of passing data between participants for action. Distinguishing facilitation from automation is an important aspect, since it helps to distinguish manual (human) parts and automatic (machine) parts of a workflow. The passing of data between the different participants is equally important, since it is one of the key features for the support of BP.

Additionally, there is an issue with Definitions 2.10 and 2.11 regarding the context of this thesis. This thesis is concerned with the behaviour of a software system. By limiting the term workflow to BP, the scope of the behaviour of a software system would also be restricted. Consequently, for the context of this thesis, it is more suitable to widen the definition in this regard and consider general processes. This includes all BP, but also allows for workflows to model other processes as well. The exact definition of the term process for this thesis is provided in Section 2.4.

Taking these points into account the following Definition combines both Definition 2.10 and Definition 2.11 from the WfMC and expands the scope to general processes. This is the definition of workflow used in this thesis:

Key Term Definition A.2 (Workflow)**.** The automation and facilitation of a process, in whole or part, during which data and tasks are passed from one participant to another for action, according to a set of procedural rules.

For workflows, it is important to distinguish between modelling- and runtime. The WfMC provides the following distinction:

Definition 2.12 (Process Definition)**.** *"The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc." [Wfmc99, 1999, p. 11]*

Definition 2.13 (Process Instance (Case))**.** *"The representation of a single enactment of a process." [Wfmc99, 1999, p. 16]*

This distinction between definition and instance/case[21] is applied in this thesis. The definition is created at modelling time and is purely a modelling artefact. A process instance is enacted at runtime from one definition.

Human users or automatic resources that actually perform the work described in a workflow are subsumed into the term workflow participant.

[21] Process instance and case are interchangeable, yet emphasise different aspects. *Case* emphasises the application viewpoint of the process set within the real-world, while *process instance* emphasises the technical viewpoint of the WFMS.

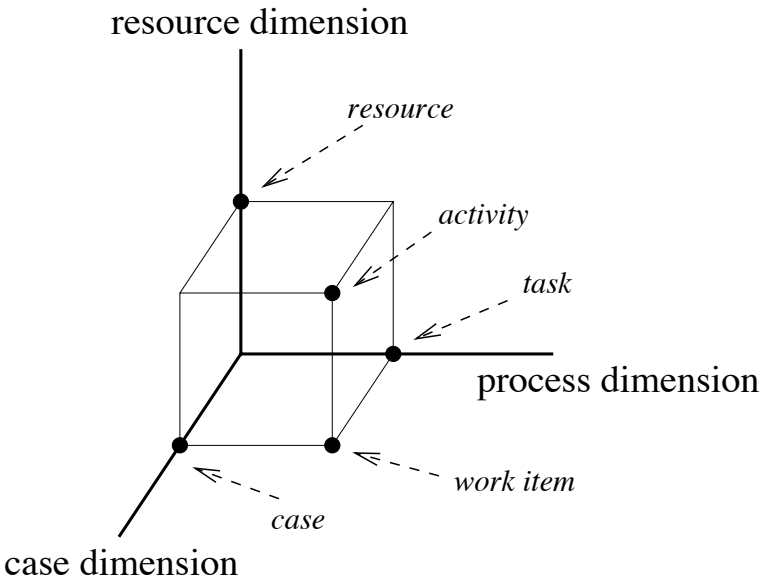

Figure 2.15: Dimensions of a workflow (from [van der Aalst, 1998b, p. 6])

Key Term Definition A.3 (Workflow Participant/Resource)**.** "*A resource which performs partially, or in full, the work represented by a workflow activity instance.*" [Hollingsworth, 1995, p. 54]

For this thesis, the distinction between automated resource and human user is often important. Both of these terms refer to specialised (w.r.t. human or machine) variants of a workflow participant. Still, if the distinction is irrelevant, the general term resource is often used to describe a non-specified workflow participant. The individual building blocks of workflows are called activities in the WfMC reference model:

Definition 2.14 (Activity (WfMC))**.** *"*A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources(s) to support process execution; where human resource is required an activity is allocated to a workflow participant.*" [Wfmc99, 1999, p. 13]*

When executed within a process instance the actual work performed in the context of activities is referred to as work items:

Definition 2.15 (Work Item (WfMC))**.** *"*The representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance.*" [Wfmc99, 1999, p. 19]*

In this point the terminology of the WfMC is insufficient. There is not distinction between the states of an activity during the execution of the workflow. Also, the relationship between work item and activity is ambiguous. Work items are somehow part of or a representation of the work within an activity. For these reasons, this thesis employs another terminology regarding building blocks of workflows, distinguishing between *task*, *workitem* and *activity*. This terminology is based on [van der Aalst and van Hee, 2002] and originates from the context of workflow Petri nets.

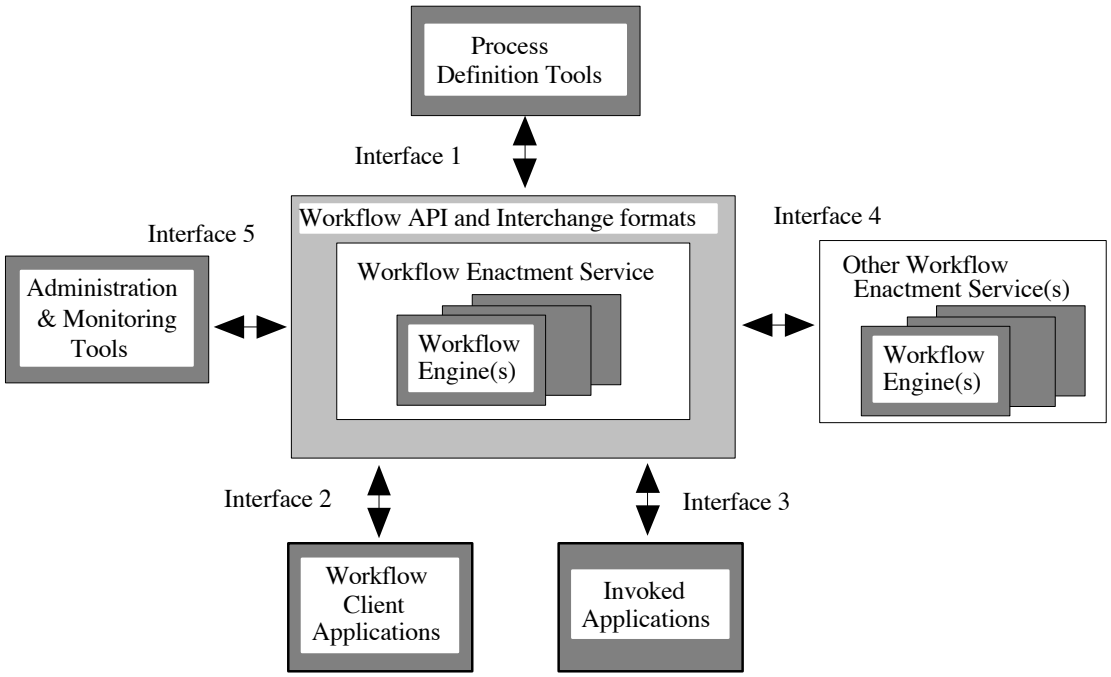

Figure 2.16: Workflow Management Coalition - Workflow Reference Model (from [Wfmc99, 1999, p. 23])

Key Term Definition A.4 (Task)**.** "*A task is a logical unit of work. It is indivisible and is thus always carried out in full. If anything goes wrong during the performance of a task, then we must return to the beginning of the entire task.*" [van der Aalst and van Hee, 2002, p. 32]

Key Term Definition A.5 (Workitem)**.** "*A workitem is the combination of a case and a task which is just about to be carried out. A workitem is created as soon as the state of a case allows it. We can thus regard a workitem as an actual piece of work which may be carried out.*" (adapted) [van der Aalst and van Hee, 2002, p. 33]

Key Term Definition A.6 (Activity)**.** "*The term activity refers to the actual performance of a workitem. As soon as work begins upon the workitem, it becomes an activity.*" (adapted) [van der Aalst and van Hee, 2002, p. 33]

Workflows are executed and managed in a WFMS.

Definition 2.16 (Workflow Management System (WFMS))**.** *"*A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.*" [Wfmc99, 1999, p. 9]*

Figure 2.16 shows the workflow reference model of the WfMC, which describes the different components of a WFMS. The workflow enactment service and workflow engines are at the centre of the workflow reference model:

Definition 2.17 (Workflow Enactment Service (WFES))**.** *"*A software service that may consist of one or more workflow engines in order to create, manage and execute workflow instances. Applications may interface to this service via the workflow application programming interface (WAPI).*" [Hollingsworth, 1995, p. 54]*

Key Term Definition A.7 (Workflow Engine)**.** "*A software service or "engine" that provides the run time execution environment for a workflow instance.*" [Hollingsworth, 1995, p. 22]

While the WFES and the different engines are the core of the workflow reference model, the five interfaces of the workflow application programming interface (WAPI) enable the incorporation of further, essential functionality. The five interfaces connect to process definition tools (Interface 1), workflow client applications that allow workflow participants to work on workflows (Interface 2), other applications invoked during the execution of workflows (Interface 3), other WFES for interoperability beyond a single WFMS (Interface 4) and administration and monitoring tools (Interface 5).

A WFMS is a software system built to execute and manage workflows. The workflows themselves capture some kind of process in some application context. All of the workflows related by the same application context can be considered as the system constituting that application. In other words, a WFMS represents a framework for the execution of workflow applications. Since the WfMC uses the term workflow application to refer to some application that handles the "*processing required to support a particular activity*" [Wfmc99, 1999, p. 41], this thesis uses the term workflow system to distinguish the application provided by a set of workflows from the framework given by the WFMS.

Definition 2.18 (Workflow System)**.** *A workflow system is a set of workflows related by a common application context. These workflows are designed, modelled and implemented for a particular WFMS.*

2.3.2 Workflow Nets

Workflow Petri nets, or short workflow nets, are Petri nets designed to model and represent workflows. They require specialised WFMS that support both the workflow, as well as the Petri nets aspects.

Using Petri nets for workflow management and more general, overall BPM enables the utilisation of formal methods associated with Petri nets in these fields. A survey [Recker and Mendling, 2016] about the articles presented at the main BPM conference (International Conference on Business Process Management[22] found evidence of BPM as a formal science: "*In the BPM conference series, we observe a strong tradition of research that acknowledges BPM as formal science.*" [Recker and Mendling, 2016, p. 65] Moreover, the survey also explicitly mentioned Petri nets: "*The results of our analysis suggest that BPM as a formal science is well-represented in the BPM conference series and that it is well-understood by its key contributors. This is, for instance, reflected in the extensive reference to formal Petri net concepts, algebraic definitions and utilization of formal logics in many papers.*" [Recker and Mendling, 2016, p. 65] [van der Aalst, 2015, p. 3] identifies three reasons for using Petri nets for BPM, namely the formal semantics coupled with a graphical representation, the state-based nature and the large amount of available analysis techniques. Even more so, Petri net concepts have had an essential impact on BPM: "*Petri nets play an even more prominent role in BPM as they are graphical and able to model*

[22] `http://bpm-conference.org` (last accessed May 28th, 2017)

concurrency. In fact, most of the contemporary BPM notations and systems use token-based semantics adopted from Petri nets." [van der Aalst, 2013, p. 5]

Workflow Nets in General

Workflow nets were introduced by Wil van der Aalst in the mid-1990s [van der Aalst et al., 1994, van der Aalst, 1996, van der Aalst, 1997, van der Aalst, 1998b]. [van der Aalst and van Hee, 2002] provides a thorough description of the overall context of workflow management with workflow nets. Workflow nets are defined in the following:

Definition 2.19 (Workflow Net)**.** *"A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if: (i) PN has two special places: i and o. Place i is a source place: $\bullet i = \emptyset$. Place o is a sink place: $o\bullet = \emptyset$. (ii) If we add a transition t^* to PN which connects place o with i (i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$), then the resulting Petri net is strongly connected." (adapted) [van der Aalst, 1997, p. 7]*[23]

In addition to the above given definition, there are some additional requirements to workflow nets in order for them to be suitable for representing business processes. "*For any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place o and all the other places are empty. Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary task by following the appropriate route though the WF-net.*" [van der Aalst, 1997, p. 7] These requirements translate to the so-called soundness property of workflow nets.

Definition 2.20 (Soundness of Workflow Nets)**.** *"A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if: (i) For every state M reachable from state i, there exists a firing sequence from state M to state o. Formally: $\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$ (ii) State o is the only state reachable from state i with at least one token in place o. Formally: $\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$ (iii) There are no dead transitions in (PN, i). Formally: $\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M'$ " (adapted) [van der Aalst, 1997, p. 8]*

Figure 2.17 shows a sound workflow net[24]. This net fulfils the conditions of a sound workflow net. First off, it is a workflow net. It has places i and o and if t^* is added the net is strongly connected Place o is also reachable from every state and there can only be one token in o and no token elsewhere once the task *archive* was executed. Finally, all transition can fire at least once, meaning none of them is dead.

Workflow Nets for Renew

[Jacob, 2002] describes the implementation of a workflow management system for Renew. The *Agenda* WFMS implements basic workflow management functionality. The system was directly incorporated into the Renew environment and recognised tasks from workflow nets executed in the local simulator.

In general, the definition of workflow nets needs to be slightly adapted to reference nets. This adaption is described in [van der Aalst et al., 1999].

[23] "*A Petri net is strongly connected iff, for every pair of nodes (i.e. places and transitions) x and y, there is a directed path leading from x to y.*" [van der Aalst, 1997, p. 4]

[24] Figure 2.17 was modelled in Renew using the special task-transition (transition with two thick vertical borders). The semantics of the task-transition are described in the following. For the purposes of this current examination of soundness please please regard them as regular Petri net transitions. The inscription only indicates the name of the task to be executed.

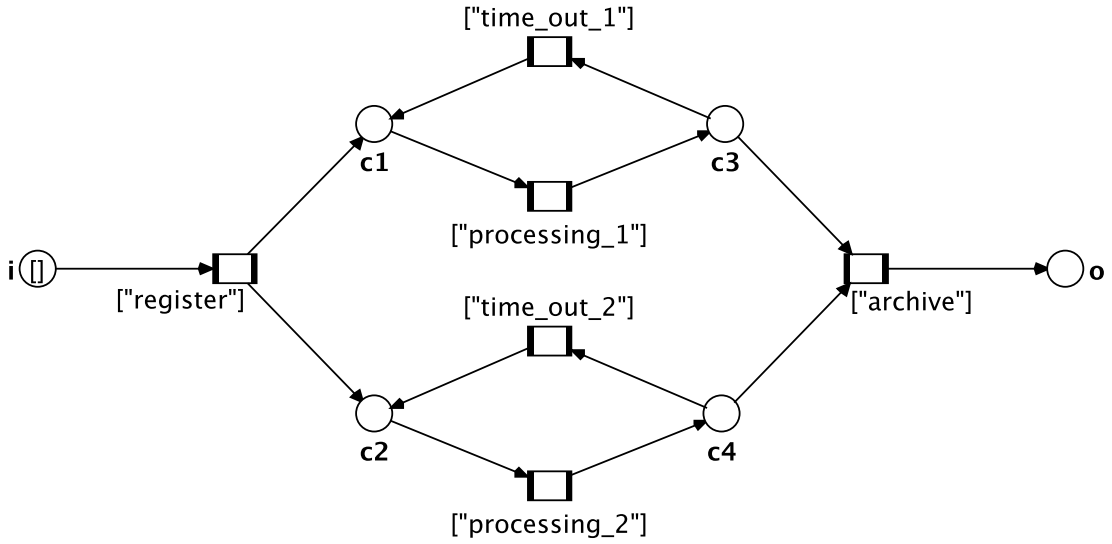

Figure 2.17: A sound workflow net using the Renew task-transition (adapted to Renew from [van der Aalst, 1998b, p. 36])

Definition 2.21 (Workflow net (Reference net))**.** *"A workflow net (WF-net) is a reference net with one source transition and one sink transition. The source transition has a synchronous channel named* `new()` *and the sink transition has one named* `done()`*. Every node of the net is on a path from the source to the sink. Except for synchronous channels the net inscriptions are not included in this definition." [van der Aalst et al., 1999, p. 7]*

The names of the synchronous channels are arbitrary. They depend on the management system executing the workflow net. For example, in the Renew and Capa WFMS from [Wagner, 2009a, Wagner, 2009b], the channels are called `startwf()` and `stopwf()`.

Workflow soundness is also slightly adapted for reference nets:

Definition 2.22 (Workflow soundness (Reference net))**.** *"For sound behavior of workflow nets it is necessary that after instantiation (i.e., the firing of the source transition* `new()`*) proper termination is guaranteed: From all possible reachable states termination (i.e., the firing of the sink transition* `done()`*) must be possible and after termination no part is allowed to be active anymore and all places have to be empty or may only contain references to inactive parts, e.g., to the initial data of the net instance." [van der Aalst et al., 1999, p. 7]*

As indicated in the previous descriptions, a task is considered atomic, yet is not executed instantaneously. This means that modelling a task in a workflow net with a single, regular Petri net transition is unsuitable. Consequently, [van der Aalst et al., 1999] introduces the realisation of a task with three transitions and a place. The concept is illustrated on the right-hand side of Figure 2.18. One transition models the request of the workitem, which creates an activity. The activity is stored on the place until it is either successfully confirmed or unsuccessfully cancelled. These two operations are modelled by the other two transitions. The former enables the continuation of the workflow net, while the latter enforces a rollback in the locality of the request transition. The rollback returns any tokens that were originally removed from input places during the firing of the request transition.

The solution of using a complex net structure for tasks complies with the operative execution of workflow tasks: Firing takes time but happens fully or not at all. However,

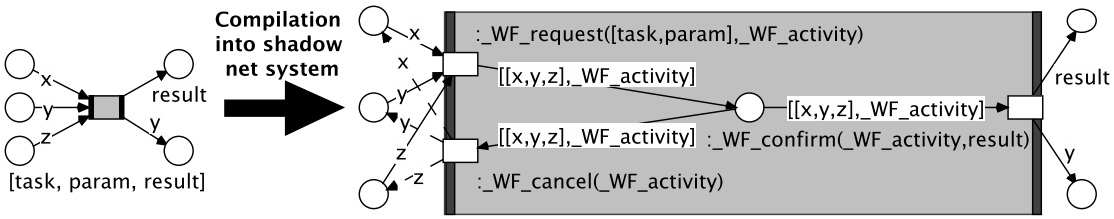

Figure 2.18: The Renew task-transition (adapted from [Jacob, 2002, p. 96])

from a modelling point of view using multiple transitions is counter-intuitive. It suggests a subdivision in the modelling of the task where there is none. The task is atomic and should be represented as such. For this reason [Jacob, 2002] introduced the task-transition.

The task-transition utilises the Renew shadow net mechanism (see Section 2.1.2) and is illustrated in Figure 2.18. During modelling time the task-transition appears as a single transition with two thick vertical bars to distinguish it from regular transitions (left-hand side of Figure 2.18). It has a special inscription containing the task name and optionally task parameters and a variable to bind any result to. During the compilation into the shadow net system, the task-transition is handled differently. The workflow net compiler checks every element of a net. Any regular transition, place or arc is forwarded to the regular reference net compiler. Any task-transition, however is translated into the net structure described above (right-hand side of Figure 2.18).

The shadow net is not graphically represented during execution. Rather, Renew uses the graphical representation of the task-transition during execution as well. This means that for modellers/users executing the system, the task-transition appears as a singular transition with a different firing behaviour. In the representation the task-transition appears to fire as long as the activity is being executed and stored on the central place. The task-transition enables both the conceptual, modelling view of a task as an atomic unit of work, as well as the operative necessity of using a net structure to realise the task and its different operations. In other words, the task-transition abstracts from internal and standardised composition in order to provide a clearer perspective on the concept.

2.3.3 Inter-organisational Workflows

Traditionally, workflows facilitate the BP of a single organisation. However, the application area of inter-organisational workflows enables workflows to cross organisational borders. Inter-organisational workflows support inter-organisational BP, which are defined in the following way:

Definition 2.23 (Inter-organisational Business Process)**.** *"An interorganizational business process is a business process the process logic of which is enacted by two or more autonomous organizations, of which at least one organization exposes a non-black box projection of the explicit control flow structure of a process to the other organization(s)." [Grefen, 2009, p. 405]*

The limitation of exposing (parts of) the control flow in an inter-organisational BP distinguishes the concept from simple service invocation collaborations. Analogous to regular workflows, inter-organisational workflows facilitate and automate inter-organisational BP. Since Definition A.2 already applies the more general *process* term, inter-organisational BP are already contained in it and represent only a specific application of the established workflow definition.

Inter-organisational workflows can arise for different reasons. Organisations may collaborate to, e.g., increase output or expand operations with new products or in new areas. Often the desired goal is economically unfeasible (e.g. associated with too high costs) for one organisation to achieve alone. Other types of inter-organisational workflows can arise from outsourcing or when new organisations are created as corporate spin-offs. Corporate spin-offs may still closely interact and collaborate with the parent organisation, yet are often separate, independent entities which should be treated as such.

[van der Aalst, 1999b, pp. 3–7] identifies six different forms of interoperability between organisations in inter-organisational workflows.

- *Capacity sharing* assumes a centralised control flow. Task instances are distributed among the participating organisations.
- *Chained execution* divides the overall workflow into sequential parts that are executed by the participating organisations.
- *Subcontracting* utilises an overall workflow, the tasks of which are subcontracted as subprocesses to other, participating organisations.
- *Case transfer* shares the same workflow specification among all organisations. Cases are then distributed among the organisations to distribute the workload.
- *Extended case transfer* is similar to normal case transfer with the exception that the workflow specifications don't need to be identical in the participating organisations.
- *Loosely coupled* inter-organisational collaboration divides the workflow into different parts with each being executed by one participating organisation. Only the interfaces are shared between organisations.

Modelling and supporting inter-organisational workflows faces a number of challenges due to the requirements introduced by the multi-organisational nature. [Legner and Wende, 2007, p. 109] classifies the challenges of inter-organisational BPM into the following groups: The representation of inter-organisational business processes, the allocation of tasks to actors, the alignment of semantics, the decoupling of internal and external processes, the selective visibility of internal processes to external partners, the formal specification of process interfaces and the support for alignment with multiple partners. In addition to these inter-organisational challenges, the challenges of traditional mono-organisation business process management need to be dealt with as well. However, the traditional challenges are often made more complex in the inter-organisational setting, e.g. confidential data access.

More general information about inter-organisational workflows can be found in, e.g., [Legner and Wende, 2007, Lin, 2008, Grefen, 2009, Engel et al., 2016]. Inter-organisational workflows in the direct context of (workflow) Petri nets have also been comprehensively researched, e.g. in [van der Aalst, 1998a, van der Aalst, 1999a, van der Aalst, 1999b, van der Aalst et al., 1999, van der Aalst et al., 2010, van der Aalst and Weske, 2013, Lenz and Oberweis, 2001, Prisecaru and Jucan, 2008].

One of the particularities of inter-organisational workflows is that organisational entities in one organisation need to be able to act in different subcontexts. An organisational entity may, e.g., be involved in an internal process while at the same time interacting across the organisation border for the inter-organisational collaboration. The organisational entity may even be the element of the organisation that links the internal process with the inter-organisational collaboration. Conceptually, the organisational entity is, at that time, **both** (part of) a workflow (internal process) **and** an autonomous agent (inter-organisational

collaboration). This duality is what makes inter-organisational workflows a very interesting application area for the research described in this thesis. Organisations act as and are interacted as behavioural elements (workflows) and structural elements (agents).

2.4 Regarding the Term Process

In the context of workflows, business processes and Petri nets the term *process* is overloaded with different meanings. The sentiment is described in following quote: "*Unfortunately, the term "process" was and is used in the Petri net research community in an ambiguous way. Since more than 30 years, a process net is known to be an occurrence net representing a concurrent run of a net representing a system. This naming does not nicely match processes in the sense of business processes [...]*" (adapted) [Desel, 2008, p. 19]. This section shortly addresses the process-related terminology and different definitions.

The term process is defined in variations in many research fields associated with Petri nets. Examples include net unfoldings [McMillan, 1992, Esparza and Heljanko, 2008], occurrence sequences/nets of infinite Petri nets [Best and Devillers, 1987], traces [Mazurkiewicz, 1987, Hoogers et al., 1995], application to nets-within-nets [Valk, 1996, Köhler and Rölke, 2005] and in general and true concurrency research [Janicki and Koutny, 1987, Priese and Wimmel, 1998]. For the purposes of this thesis, these definitions are too specific to their formal context.

A more general, yet still formal, definition of process concerns the process of a Petri net. The process of a Petri net is defined in [Desel and Reisig, 1998] as another Petri net, called process net, satisfying the properties of a causal net:

Definition 2.24 (Causal Net)**.** *"*Formally, a **causal net** is a net with conditions (places) and events (transitions) satisfying the following properties: **(1)** the flow relation is acyclic, i.e. no path with at least two elements leads from an element to itself; **(2)** conditions are not branched, i.e. each condition b satisfies $|{}^\bullet b| \leq 1, |b^\bullet| \leq 1$; **(3)** only finitely many conditions b have an empty pre-set ${}^\bullet b$; **(4)** for each event e, both sets ${}^\bullet e$ and $e^\bullet$ are finite and nonempty; **(5)** for each element x, only finitely many different paths lead to x. *" (adapted) [Desel and Reisig, 1998, p. 159]*

Definition 2.25 (Process Net)**.** *"*Since a causal net K represents a concurrent run of a marked net N, its conditions correspond to places of N and its events correspond to transitions of N. This relation is formalized by a **labeling function** $\pi : (B_K \cup E_K) \to (S_N \cup T_N)$. A labeled causal net K is a **process net** of a net N with initial marking m_0 if the labeling function π enjoys the following properties: **(6)** for each condition b in B_K, $\pi(b) \in S_N$; **(7)** for each event e in E_K, $\pi(e) \in T_N$ **(8)** for each event e in E_K, π generates bijections $\pi_e^- :{}^\bullet e \to {}^\bullet(\pi(e))$ and $\pi_e^+ : e^\bullet \to (\pi(e))^\bullet$, defined by $\pi_e^-(b) = \pi(b)$ for each b in ${}^\bullet e$ and $\pi_e^+(b) = \pi(b)$ for each b in $e^\bullet$; **(9)** for each place s in S_N, $m_0(s) = |\{b \in B_K | {}^\bullet b = \emptyset \wedge \pi(b) = s\}|$ *" (adapted) [Desel and Reisig, 1998, p. 163]*

Another process-related term for Petri nets is that of branching processes [Engelfriet, 1991, Khomenko and Koutny, 2003, Couvreur et al., 2011].

Definition 2.26 (Branching process)**.** *"*Let N be a net. A *branching process of* N, abbreviated *b-process of* N, is a pair (N', π), where $N' = (Pl, Tr, pre, post, In)$ is an occurrence net and π is a homomorphism from N' to N, such that for every $t_1, t_2 \in Tr$, if $pre(t_1) = pre(t_2)$ and $\pi(t_1) = \pi(t_2)$ then $t_1 = t_2$. *" (adapted) [Engelfriet, 1991, p. 580]*

Note that the term occurrence net here refers to a causal net as defined in Definition 2.24. The condition for branching processes basically means that "*in any particular situation,*

a transition of N can be chosen at most once in N'" [Engelfriet, 1991, p. 581]. In other words, alternatives in net execution are translated in the branching process as different branches.

These definition is suitable for purely Petri net-concerned aspects. This thesis, however, is more concerned with processes in the real-world or, more concretely, processes of and within software systems and applications. These systems are often in the context of workflows and BP. The following definitions originate from the fields of BPM and workflow nets:

Definition 2.27 (Process (Variant I))**.** *"*A formalised view of a business process, represented as a co-ordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal.*" [Wfmc99, 1999, p. 26]*

Definition 2.28 (Process Model)**.** *"*A **process model** aims to capture the different ways in which a case (i.e., process instance) can be handled. A plethora of notations exists to model operational business processes (e.g., Petri nets, BPMN, UML, and EPCs). These notations have in common that processes are described in terms of activities (and possibly subprocesses). The ordering of these activities is modeled by describing causal dependencies. Moreover, the process model may also describe temporal properties, specify the creation and use of data, e.g., to model decisions, and stipulate the way that resources interact with the process (e.g., roles, allocation rules, and priorities).*" [van der Aalst, 2013, p. 2]*

These definitions are more suitable for the application focus of this thesis, but still relate explicitly to BP. This is a semantic restriction that is not desired here. While the term workflow is historically associated with BP, one of the intentions of the desired integration of structure and behaviour is to represent the behaviour of and between the structural elements of a software system. If that software system is regarded as an organisation, its internal processes can easily be regarded as the BP of that organisation. However, this consideration is rather sluggish and may lead to semantic inconsistencies, especially if the software system in question deals with the actual BP of an organisation. Rather, it is more reasonable to regard processes in an equivalent way, yet without the explicit link to organisations and their operation. A process should just be any kind of procedure, in the sense of the German word *"Ablauf"*, without any predefined context. The following definitions still originate from the field of workflows and BP, yet mostly avoid the explicit connection between process and BP:

Definition 2.29 (Process (Variant II))**.** *"*A co-ordinated (parallel and/or serial) set of process activity(s) that are connected in order to achieve a common goal. Such activities may consist of manual activity(s) and/or workflow activity(s).*" [Hollingsworth, 1995, p. 52]*

Definition 2.30 (Process (Variant III))**.** *"*A *process* is a collection of tasks, conditions, subprocesses, and their relationships with one another. As we have seen, we can describe a process using a Petri net. Conditions are depicted using places and tasks using transitions.*" [van der Aalst and van Hee, 2002, p. 61]*

Key Term Definition A.8 (Process)**.** "*A **process** consists of **tasks** that have to be executed. These tasks can be in some order (**sequentially**), stating that one task can only be executed after the execution of another task is finished. If two tasks are not ordered, then they can be executed **concurrently**. Tasks can also be alternative, that is, if one task is executed, then the other task is not executed and vice versa. Tasks can be executed more*

than once in general. A process can be in different states. A process starts with an initial state (which is not necessarily unique) and might end with a final state (which is also not necessarily unique). Usually, it passes through several intermediate states." [Desel, 2005, p. 157]

The common theme of these definitions is that a process consists of tasks that are related to one another in some order. Although the connotation of the term *task* implies a connection to BP, the term actually only refers to a logical unit of work (see Definition A.4). Each of the three definitions contains a different focus: Definition 2.29 relates the process to the term workflow, Definition 2.30 relates the process to its representation as a Petri net and Definition A.8 emphasises the ordering of tasks and state of the process. For the context of this thesis, each of these three definitions is viable and suitable. They do not contradict each other and are compatible they can be used synonymously. However, to create a clear discussion basis and one should be chosen. Definition A.8 is the most explicit and detailed. It captures the spirit of individual tasks (as units of work) in a process and is general enough to cover the requirements for this thesis. When necessary, the other definitions are explicitly referred to.

3 State-of-the-Art

This chapter presents the state-of-the-art of the research fields relevant to the context of this thesis. Please note that the emphasis of this chapter lies on research concerned with an integration of agents and workflows for system modelling and development. Traditional agent-oriented modelling and workflow management are included in this chapter for completeness and as basic reference points for later discussions. Therefore, those depictions are kept purposefully short.

The chapter consists of three sections. Section 3.1 presents the state-of-the-art for traditional agent-orientation. Section 3.2, likewise, presents the state-of-the-art for traditional workflow management. Section 3.3 then examines previous and related research in the context of an integration of agents and workflows.

3.1 Software Agents

This section provides a basic overview of the current state-of-the-art in the field of agent-oriented modelling by highlighting a selection of research efforts. Since this thesis is concerned with agents as a means for modelling and software engineering only research in that area is examined. Other related fields such as game theory [Neumann and Morgenstern, 1944, Parsons et al., 2002], artificial intelligence [McCarthy et al., 1955, Russell and Norvig, 2009], agents for robotics [Murphy, 2000, Bekey, 2005] or emotional agents [Fix, 2012] are outside of the scope of this thesis. Areas such as mobility, e.g. [Perez-Diaz et al., 2015, Leppänen et al., 2017], self-adaption, e.g. [Preisler, 2013, Puviani et al., 2015, Ayala et al., 2016], or agent applications, e.g. [Lützenberger et al., 2016, Premm and Kirn, 2015, Poveda and Schumann, 2016], are also only considered in the context of agent architecture, models and frameworks. Note that the current section does *not* feature any contributions combining agents with aspects and technology from the field of workflow management. That area of research is presented individually in Section 3.3. Also note that agent methodologies, like Prometheus [Padgham and Winikoff, 2002, Khallouf and Winikoff, 2009, Padgham et al., 2014], Passi [Cossentino and Potts, 2002, Chella et al., 2006], Tropos [Bresciani et al., 2004, Morandini et al., 2014], Zeus [Nwana et al., 1999, Glanzer et al., 2001] or Gaia [Wooldridge et al., 2000, Cernuzzi et al., 2014] are also not considered further. The Paose methodology (see Section 2.2.3) is used throughout this thesis as a tool and application area, but agent methodologies themself are not directly part of the research agenda and therefore outside of the scope of this thesis.

Agent-orientation is an established and very active field. There are a number of large, international annual conferences and workshops concerned with the field. Prominent examples include AAMAS (International Conference on Autonomous Agents and Multiagent System[1]) sponsored by the IFAAMAAS (International Foundation for Autonomous Agents and Multiagent Systems[2]), EUMAS (European Conference on Multi-Agent Systems[3]), MATES (German Conference on Multiagent System Technologies[4]), PAAMS (International

[1] Most recent edition: `http://www.aamas2017.org/` (last accessed May 28th, 2017)
[2] `http://www.aamas-conference.org/` (last accessed May 28th, 2017)
[3] Most recent edition: `http://eumas-at2016.webs.upv.es/` (last accessed May 28th, 2017)
[4] Most recent edition: `http://www.dfki.de/mates2016/` (last accessed May 28th, 2017)

Conference on Practical Applications of Agents and Multi-Agent Systems[5]) and CARE (Collaborative Agents Research & Development[6]). A website at WikiCfP[7] provides an overview of current conferences and workshops in the field of agent-orientation.

The following presents a selection of agent architecture, models and frameworks ordered by date of the first major publication. The selection highlights the most well-known and established contributions in the field of agent-oriented modelling. That selection serves as the relevant related work for this thesis in the context of agent-orientation. A relatively recent and general overview over agent-oriented software engineering can be found in [Sturm and Shehory, 2014].

Subsumption (General) The subsumption agent architecture assumes a hierarchical layer approach to agent behaviour. The subsumption architecture originates from controlling robots [Brooks, 1986]. All behaviour is defined between input and output interfaces of agents. Behaviour in lower layers of the hierarchy describes basic behaviour, while higher layers add specialisations and sophistication. That way, the basic behaviour can be maintained and influenced by the specialisation from higher layers. Current research in subsumption architecture is relatively sparse, with most contributions still emphasising the robotic origins of subsumption, e.g. [Oland et al., 2016], or presenting applications emphasising collaboration and coordination [Krzywinski et al., 2008, Heckel and Youngblood, 2010, Linson et al., 2015].

Soar Soar [Laird et al., 1987] is a general cognitive architecture for intelligent behaviour in software. Soar agents exist in a set of spaces. Based on their current state, Soar agents decide on an operator to get to a new state. Problem solving to get to specific states is key in the Soar architecture. Soar is still being actively developed, with the current version 9.5.0 beta available from the Soar website[8]. Current research emphasises learning agents [Mohan et al., 2012, Mininger and Laird, 2016, Kirk et al., 2016] and language processing [Lindes et al., 2015, Lindes and Laird, 2016].

BDI (General) The Beliefs-Desires-Intentions (BDI) model [Rao and Georgeff, 1991, Rao and Georgeff, 1995, Fischer et al., 1995] ascribes software agents with the titular concepts. BDI agents execute plans. Depending on the state of their knowledge about their environments (beliefs), they deliberate upon their long-term goals (desires) to choose plans to achieve more short-term goals (intentions). The interplay between deliberation and executing plans leads to BDI agents being able to flexibly achieve their design goal in dynamic environments.

There is still a lot of active research concerning BDI models. That research not only concerns practical frameworks using BDI, the amount of which is also considerable. A number of BDI frameworks are discussed as individual related work in separate paragraphs. Current research in BDI not directly linked to specific frameworks is concerned with, for example, improving deliberation, reasoning and plan execution [Meneguzzi and de Silva, 2015, Waters et al., 2015, Yao and Logan, 2016, Visser et al., 2016, Harland et al., 2017], testing [Cunha et al., 2015, Winikoff, 2016], BDI application to games [Rivera-Villicana et al., 2016, Luong et al., 2017], coordination [Chen et al., 2016b, Dominguez et al., 2016]

[5]Most recent edition: `http://www.paams.net/` (last accessed May 28th, 2017)
[6]Most recent edition: `http://www.care-workshops.org/` (last accessed May 28th, 2017)
[7]`http://www.wikicfp.com/cfp/call?conference=multi-agent%20systems` (last accessed May 28th, 2017)
[8]`http://soar.eecs.umich.edu/` (last accessed May 28th, 2017)

or using BDI agents in and for simulations [Adam et al., 2016, Badica et al., 2016, Coelho et al., 2016, Singh et al., 2016].

JADE The Java Agent DEvelopment Framework (JADE) [Bellifemine et al., 1999, Bellifemine et al., 2000] is a FIPA compliant agent framework. The current version JADE 4.4.0 is available from its website[9]. JADE agents are purposefully kept generic, making them extensible and versatile. Extensions have added, for example, BDI features [Nunes et al., 2011, Jørgensen et al., 2015] and support for mobile devices [Bergenti et al., 2014, Naoui et al., 2014]. A workflow extension Wade is discussed later on with agent/workflow integration efforts.

3APL The "An Abstract Agent Programming Language" (3APL) agent programming language [Hindriks et al., 1999, Dastani et al., 2005] implements BDI agents. It is available in a Java and Haskell implementation from the projects website[10]. The implementation is not actively being developed further, the last update dating November 2007.

NetLogo NetLogo [Wilensky, 1999, Badham, 2015] is a framework for developing and simulating agent models with an emphasis on examining behaviour and emergent behaviour on micro and macro levels. Development and research in NetLogo are still quite active. Current research contributions are concerned with, for example, models simulating social behaviour [Izquierdo et al., 2015, Scott et al., 2016], banking [Monett and Navarro-Barrientos, 2016] or agent cognition [Head et al., 2015, Dzikowski and Hood, 2015]. Its current version 6.0.1 from March 2017 is available from its website[11].

MaDKit The **M**ulti**a**gent **D**evelopment **Kit** (MaDKit) [Gutknecht and Ferber, 2000, Ferber et al., 2003] is a Java-based multi-agent system development framework. Agents in MaDKit utilise the agent-group-role model [Ferber and Gutknecht, 1998], wherein agents are parts of groups and have roles within those groups that describe their function. The agent-group-role model makes MaDKit very organisation-centric and purposefully leaves the agent-internals open, so that system modellers have a degree of freedom on how to implement agents. MaDKit is still being actively developed, though there are no direct, current scientific contributions available. Its current release 5.1.1 dated March 2017 is available from its website[12].

Jadex Jadex [Pokahr et al., 2003, Pokahr et al., 2005, Pokahr and Braubach, 2009] is an agent framework for BDI agents maintained by the distributed systems group of the University of Hamburg. Its current release 3.0.43 is available from its website[13]. Jadex started out as an extension to the JADE framework (see above), providing a BDI realisation for the internal workings of agents, details of which JADE intentionally left open. By now, Jadex has moved beyond the JADE framework and is an independent project. Current work and research related to Jadex deals with the so-called active components, which are discussed later on in more detail.

[9] `http://jade.tilab.com/` (last accessed May 28th, 2017)
[10] `http://www.cs.uu.nl/3apl/` (last accessed May 28th, 2017)
[11] `https://ccl.northwestern.edu/netlogo/` (last accessed May 28th, 2017)
[12] `http://www.madkit.net/` (last accessed May 28th, 2017)
[13] `https://www.activecomponents.org/` (last accessed May 28th, 2017)

Jack Intelligent Agents Jack Intelligent Agents [Winikoff, 2005] is a Java-based BDI agent framework. It is developed and maintained by Agent Oriented Software Inc.[14] and available commercially in its current version 5.6c. Its commercial focus leads to a good technical maturity. Research on applications deals with, for example, deception detection [Merritts, 2013], robotic assembly automation [Maeda et al., 2007] or mobile robots [Gascuena and Fernandez-Caballero, 2011].

Spade The Smart Python multi-Agent Development Environment (Spade) [Gregori et al., 2006] is a Python-based agent framework. Agents in Spade follow the FIPA standards (see Section 2.2.2) and XMPP messaging server[15] for secure communication. Version 2.2.1 is available from multiple sources, including the projects Python website[16]. The project is no longer active, with the last commit on its GitHub page[17] dating from December 2013.

Jason and Agentspeak Jason [Bordini et al., 2007], originally an acronym for "Java-based Agentspeak interpreter used with Saci for multi-agent distribution Over the Net", is a Java based interpreter and development/execution platform for systems using a variant of the Agentspeak(L) [Rao, 1996] agent programming language. Agentspeak(L) follows a BDI agent model, meaning Jason agents are also BDI agents. Jason version 2.0 is available from its website[18]. Jason is still actively used for research, with a focus being applications, for example software development [Croatti and Ricci, 2016], robotics [Pantoja et al., 2016], teaching support [Bouchet et al., 2016] and price negotiation [Pan and Choi, 2016].

GOAL GOAL [de Boer et al., 2007] is an agent framework for cognitive agents. GOAL agents have knowledge about themselves and their environment (beliefs), as well as certain goals. From those two concepts they derive a set of actions, from which one is chosen. This approach is similar to BDI, yet does not distinguish between the long- and short-term goals. The GOAL project is still active and available as a downloadable Eclipse plugin from the project website [19].

Jiac Java Intelligent Agent Componentware (Jiac) [Lützenberger et al., 2013, Lützenberger et al., 2014] is a Java-based agent framework. Its current version 5.2.0 is available from its website[20]. Jiac emphasises a service-orientation by providing its agents with modules (AgentBeans) that serve specific behaviour purposes. Current research in the context of Jiac is concerned with, e.g., model-driven development [Kuster et al., 2014]. Process-orientation and BPM are also examined, which is discussed further later on in Section 3.3.

SARL SARL [Rodriguez et al., 2014] is a relatively recent agent programming language. The development environment for SARL is available at its website[21]. SARL is linked to the JANUS[22]. project, which fully supports SARL as an execution environment. SARL makes little assumptions about the nature of the individual agents. It provides the basis for system modellers to create so-called capacities. These capacities can implement different

[14] `http://aosgrp.com/` (last accessed May 28th, 2017)
[15] Extensible Messaging and Presence Protocol, `https://xmpp.org/` (last accessed May 28th, 2017)
[16] `https://pypi.python.org/pypi/SPADE` (last accessed May 28th, 2017)
[17] `https://github.com/javipalanca/spade` (last accessed May 28th, 2017)
[18] `http://jason.sourceforge.net/wp/` (last accessed May 28th, 2017)
[19] `https://goalapl.atlassian.net/wiki/` (last accessed May 28th, 2017)
[20] `http://www.jiac.de/agent-frameworks/jiac-v/` (last accessed May 28th, 2017)
[21] `http://www.sarl.io/` (last accessed May 28th, 2017)
[22] `http://www.janusproject.io/` (last accessed May 28th, 2017)

agent models, such as BDI agents. Another interesting feature of SARL is its support of holonistic and recursive agents, meaning agents containing yet more agents.

3.2 Workflows

This section gives an overview of the state-of-the-art for workflows. This thesis utilises workflows, and especially workflow Petri nets (see Section 2.3.2 and [van der Aalst et al., 1994, van der Aalst, 1996, van der Aalst, 1997]), for modelling and executing the behaviour of a software system. Therefore, related fields, such as business process reengineering [Hammer and Champy, 1993], as well as purely economic and administrative aspects of BPM, are outside of the scope of this thesis. Other research issues, such as process mining or verification, are only discussed in the context of the main fields of modelling and execution as additional information. Note that, like Section 3.1, the current section does *not* feature any contributions combining workflows with aspects and technology from the field of agent-orientation. That area of research is presented individually in Section 3.3.

Workflow management and the wider area of BPM are a very active research field. Prominent conferences and events that deal primarily with workflows and BPM include the BPM conference (International Conference on Business Process Management[23]), the Gartner Business Transformation & Process Management Summit[24] and BPM Europe (Business Process Management Conference Europe[25]).

This section consists of two subsections. Section 3.2.1 presents approaches towards workflow modelling. Section 3.2.2 then proceeds to examine the management workflows.

3.2.1 Workflow and Process Modelling

For the context of this thesis, the main topic of workflow and process modelling is that of workflow nets. Concepts from workflow nets form the basis for many observations and concepts in the remainder of this thesis. Other, major modelling approaches are also considered, albeit, with the exception of the highly established BPMN, with a reduced level of detail due to limited relevance to this thesis. The order of the following paragraphs is based on the date of the first major publication.

EPC Event-driven process chains (EPC) [Keller et al., 1992, Scheer and Nüttgens, 2000] are an early way of modelling and representing business processes. EPC describe processes by connecting (passive) events via (active) functions with additional information like input, output, involved systems, organisational units etc. There is little current research happening in the context of EPC, yet some work is being done on standardisation efforts [Jannaber et al., 2016, Karhof et al., 2016].

Workflow Nets As described in Section 2.3.2, workflow nets originated in the work of Wil van der Aalst [van der Aalst et al., 1994, van der Aalst, 1996, van der Aalst, 1997]. Since then they have been researched extensively. Current research emphasises workflow soundness (see Definition 2.20). A relatively recent overview of the soundness property can be found in [van der Aalst et al., 2011]. State-of-the-art research deals with soundness under special circumstances. Examples of such research include soundness in

[23]Most recent edition: `https://bpm2017.cs.upc.edu/` (last accessed May 28th, 2017)
[24]Most recent edition: `http://www.gartner.com/events/emea/business-process-management#` (last accessed May 28th, 2017)
[25]Most recent edition: `http://www.irmuk.co.uk/bpm2016/` (last accessed May 28th, 2017)

nets with reset arcs [van der Aalst et al., 2009, Wynn et al., 2009, Clempner, 2017], in time workflow nets [Boucheneb and Barkaoui, 2015], timed-arc workflow nets [Mateo et al., 2015], workflow nets extended with resources [Sidorova and Stahl, 2013], acyclic workflow nets [Tiplea et al., 2015], Petri nets in the reversible released form [Tiplea and Leahu, 2016], free choice probabilistic workflow nets [Esparza et al., 2016] or bridge-less workflow nets [Yamaguchi and Ahmadon, 2016]. Soundness of inter-organisational workflow nets has also been examined [Passos and Julia, 2013]. Providing reductions that maintain soundness of workflow nets is also part of current research, e.g. [Esparza and Hoffmann, 2016, Bride et al., 2017], as well as providing different new techniques, e.g. based on branching processes [Liu et al., 2016], and examining further properties [Gou and Yamaguchi, 2017].

Apart from the research focussing on soundness and verification, other research examines the complexity of workflow net models [Lassen and van der Aalst, 2009]. Workflow nets are also used in process mining, where they can be generated or aggregated from event logs or other sources, e.g. [Weijters and van der Aalst, 2001, van der Aalst et al., 2004b, van Dongen et al., 2012, van der Aalst and van Dongen, 2013, Bergenthum et al., 2015].

YAWL YAWL[26] (Yet Another Workflow Language) is a workflow modelling language originating from research on workflow patterns [van der Aalst et al., 2003, Russell et al., 2006, Russell et al., 2016]. YAWL and its support environment have been described, e.g. in [van der Aalst et al., 2004a, van der Aalst and ter Hofstede, 2005, Russell and ter Hofstede, 2009b, ter Hofstede et al., 2010]. The goal of YAWL was to use (workflow) Petri nets as a starting point and then extend them in such a way as to fully support all possible workflow patterns. To supplement the modelling language, an execution and management system for YAWL was developed as a support environment. An extension, known as newYAWL, has also been developed [Russell and ter Hofstede, 2009a, Russell et al., 2009], which focuses on supporting perspectives beyond the original control-flow perspective.

UML Activity Diagrams Another way of modelling processes are UML[27] Activity diagrams [Booch et al., 2005]. Activity diagrams capture the behaviour of or within a system by describing the control flow between different activities. They model processes on a relatively abstract level, relying on other UML diagram types to provide additional information. Activity diagrams are not intended specifically for business processes, but are often used to model any kind of general processes within computer systems.

BPEL Web Services Business Process Execution Language[28] (WS-BPEL or short BPEL) [Barreto et al., 2007] is a language for specifying and executing business processes with web services. It represents an orchestration of these web services in order to achieve the purpose of the specified business process. An overview and introduction to web services orchestration with BPEL can be found in, for example, [Mazzara and Govoni, 2005, Ouyang et al., 2007a, Albreshne et al., 2009]. BPEL as a standard is maintained by OASIS[29]. BPEL itself does not feature a standard graphical notation, but research exists utilising BPMN models [Schumm et al., 2009]. Translating Petri nets into BPEL has also been researched [van der Aalst et al., 2005, Lassen and van der Aalst, 2006], as well as using

[26] `http://www.yawlfoundation.org/` (last accessed May 28th, 2017)
[27] Unified Modeling Language, `http://www.uml.org/` (last accessed May 28th, 2017)
[28] `https://www.oasis-open.org/committees/wsbpel/` (last accessed May 28th, 2017)
[29] Organization for the Advancement of Structured Information Standards, `https://www.oasis-open.org/` (last accessed May 28th, 2017)

Petri nets as a basis for verification and analysis of BPEL, e.g. [Hinz et al., 2005, van der Aalst et al., 2006, Lohmann, 2007, Ouyang et al., 2007b, Lohmann et al., 2008].

BPMN The *Business Process Model and Notation*[30] (BPMN) [BPMN10a, 2010] is a widely established way of modelling business processes. It is maintained by the Object Management Group[31]. A general overview of BPMN can be found, for example, in [von Rosing et al., 2015], while [Völzer, 2010, Yan et al., 2010] feature some general discussions and observations. BPMN is roughly based around events and connected activities. Control flow can be defined via different gateways for, e.g., parallel, concurrent or looped activities. Elements of the control flow are grouped into pools (which can partitioned into lanes), which are often used to indicate organisational entities or actors.

Research in the area of BPMN is quite active, with research emphases in different areas, such as analysis and verification, e.g. in [Mallek et al., 2015, Hichami et al., 2015, Rachdi et al., 2016a, Rachdi et al., 2016b, Nazaruka et al., 2016], verification with explicitly Petri nets, e.g. in [Wang et al., 2010, Meghzili et al., 2016], testing, e.g. in [Lübke and van Lessen, 2017, Kurz, 2016], extensions to the basic notation, e.g. supporting time aspects [Arevalo et al., 2016], resource aspects [Bocciarelli et al., 2016] or inter-organisational aspects [Jankovic et al., 2015]. Process mining in the context of BPMN is also being researched, e.g. in [Kalenkova et al., 2015, Conforti et al., 2016].

XPDL The XML Process Definition Language[32] (XPDL) [Wfmc12, 2012] is a standardised XML-based format for the representation of business processes. It is maintained by the Workflow Management Coalition (see Section 2.3.1). XPDL is broadly used as a standardised serialisation of BPMN model that facilitates exchanging such models between different modelling environments.

3.2.2 Workflow and Process Management

Workflow management is relevant for this thesis, as any integration of agents and workflows is not only to be modelled, but also controlled and managed in its execution. A reference to workflow management systems and their capabilities is set here.

Commercial BPM is quite active, with workflow and business process management solutions and tools being available and continuously improved in extensive software suites such as IBM Websphere[33], Signavio Process Manager[34], Alfresco Process Services[35] or Atlassian Jira Core[36]. Due to the scientific nature and conceptual focus of this thesis, commercial BPM suites and tools are outside the scope of this thesis. Instead, concepts are emphasised that improve upon workflow modelling and management fundamentally. These concepts may, in future work, be incorporated into commercial solutions. For now though, commercial BPM is not considered further in this thesis.

[30] `http://www.bpmn.org/` (last accessed May 28th, 2017)
[31] `http://www.omg.org/` (last accessed May 28th, 2017)
[32] `http://www.xpdl.org/` (last accessed May 28th, 2017)
[33] `http://www.ibm.com/software/products/appserv-was` (last accessed May 28th, 2017)
[34] `https://www.signavio.com/products/process-manager/` (last accessed May 28th, 2017)
[35] `https://www.alfresco.com/` (last accessed May 28th, 2017), enterprise BPM utilising Activiti BPMN 2.0 Platform `https://www.activiti.org/` (last accessed May 28th, 2017)
[36] `https://www.atlassian.com/` (last accessed May 28th, 2017)

The field of workflow and business process management is vast. Comprehensive overviews and surveys of the field can be found, for example, in [Sidorova and Isik, 2010, van der Aalst, 2013, Roeser and Kern, 2015, Harmon and Wolf, 2016, Recker and Mendling, 2016]. Examples of general research directions and emphases are:

Flexibility and adaptivity: During runtime, circumstances may invalidate a predefined process. The tasks and parameters originally provided may potentially become unsuitable for the current case. When that happens adaptive and flexible processes are needed. The principle allows for changing business processes at runtime. This can include tasks, subworkflows, parameters, etc. Examples of current research in this field are [Arunagiri and Ramachandran, 2015, Kaes et al., 2015, Sackmann and Kittel, 2015, Borrego et al., 2015, Said et al., 2016].

Resource handling: Workflow resources are the participants in workflows (see Definition A.3). Handling, representing, managing and efficiently assigning resources is therefore an important part of BPM. Examples of research contributions in this area include [Kumar et al., 2013, Liu et al., 2013, Schönig and Zeising, 2015, Arias et al., 2016, Havur et al., 2016].

Simulation: Simulating business processes can yield important results on how a process may perform when it is actually executed and managed. Simulation is used to test ideas and find issues before they occur or in the real-world. Examples of research concerned with business process simulation are [van der Aalst, 2010, Cimino and Vaglini, 2014, Liu and Iijima, 2015, Marquard et al., 2015, Bisogno et al., 2016, Vasilecas et al., 2016].

Analysis, verification and validation: Simulation can only identify specific problem cases and otherwise approximate a validation of correctness, compliance or freedom of errors. Therefore analysis, verification and validation, often based on formal methods, are another research area for BPM. Some examples can be found in [Accorsi and Lehmann, 2012, Monakova and Leymann, 2013, Kheldoun et al., 2015, Corradini et al., 2015, van Dongen et al., 2016, Rogge-Solti et al., 2016, Anseeuw et al., 2016, Roa et al., 2016, De Masellis et al., 2017].

The following presents three major research directions of BPM, which, due to their special relevance and importance, warrant individual considerations:

Process-aware Information Systems A Process-aware information system (PAIS) is defined as "*a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models.*" [Dumas et al., 2005, p. 7] PAIS represent a research direction of systems that emphasise the processes they support over other perspectives such as data or individual tasks. WFMS are generally seen as a subset of PAIS, with PAIS also including collaboration, project management, issue tracking and enterprise resource planning tools in their definition [Dumas et al., 2005, p. 10]. A noteworthy distinction is also that the definition of a PAIS does not specifically refer to *business* processes, but rather to general processes. This indicates a more generic process support emphasis of PAIS.

A current research focus of PAIS is flexibility and change support [Reichert et al., 2009c, Reichert and Weber, 2012, Reuter and Dadam, 2013, Weber et al., 2013, Lohrmann and Reichert, 2016]. Much of that research originates from the ADEPT2[37] and ARISTAFLOW[38]

[37] `http://www.uni-ulm.de/in/iui-dbis/forschung/abgeschlossene-projekte/adept2/` (last accessed May 28th, 2017)

[38] `http://www.uni-ulm.de/in/iui-dbis/forschung/abgeschlossene-projekte/aristaflow/`

research projects from Ulm University [Reichert, 2000, Rinderle, 2004, Reichert et al., 2005, Rinderle-Ma et al., 2006, Rinderle et al., 2007, Reichert et al., 2009a, Reichert et al., 2009b, Dadam et al., 2009]. Other research emphases are security [Strembeck and Rinderle-Ma, 2013, Leitner et al., 2015] and the resource perspective [Stroppi et al., 2015, Cabanillas, 2016]. Monitoring and ensuring compliance with different kinds of regulations and rules [Ly et al., 2015, Indiono et al., 2016, Knuplesch et al., 2017], as well as general evaluation of PAIS [Kriglstein et al., 2016] are also being actively researched.

Adaptive Case Management Adaptive case management (ACM) [Swenson, 2010] is another way of supporting organisations in their processes. Instead of emphasising the processes, as BPM does, ACM emphasises the data and individual cases. Processes still play an important role, but the circumstances and data surrounding the processes are prioritised. Users and organisational entities enjoy a higher degree of freedom and adaptability of actions at runtime as a consequence, since they are not as extensively guided and supported by predefined processes. A comparison of BPM and ACM is discussed in [Goesmann et al., 2015] and a general overview of the research challenges in ACM given in [Nezhad and Swenson, 2013, Hauder et al., 2014b]. Regarding graphical representation, there is a standard of the Object Management Group[39] called the Case Management Model And Notation (CMMN) [CMMN16, 2016]. It is similar in nature to BPMN and was designed to complement and be consistent with BPMN models. Current research in ACM is concerned with, for example, questions of complexity [Huber et al., 2013], compliance while maintaining flexibility [Czepa et al., 2016, Kim et al., 2016], applications of ACM to enterprise architecture [Hauder et al., 2014a] and extensions based on different concepts such as speech acts [Tenschert and Lenz, 2016] or the semantic web [Benner-Wickner et al., 2015].

BPM in the Cloud Moving BPM into cloud environments is another research focus, due to the accessibility, availability and pricing of cloud services. General observations about the topic can be found, e.g., in [Baeyens, 2013, Riemann, 2015]. Research in this area is often concerned with providing architecture and concepts, e.g. [Fang and Yin, 2010, Bendoukha, 2016], as well as execution engines and systems, e.g. [Mangler and Rinderle-Ma, 2014, Marozzo et al., 2015, Esteves and Veiga, 2016, Deelman et al., 2016]. Additionally, efficient resource allocation [Schulte et al., 2015], verification [Klai and Ochi, 2015, Boubaker et al., 2016] and moving applications to the cloud using workflow concepts [Leymann et al., 2011, Leymann, 2012] are also being examined.

3.3 Integration of Agents and Workflows

This section presents an overview of previous and related work concerning an integration or combination of agents and workflows. Included in this overview are research efforts that utilise properties, features or mechanisms of agents for workflow management and execution, as well as, more rarely, efforts that utilise workflows for agent-orientation.

The section is structured around that division. Section 3.3.1 examines agent-based workflow management systems. Here, agents have been used to build workflow management systems so that their feature set and properties may be utilised. Section 3.3.2 examines workflow-based agent management systems. This includes research efforts where workflows are used to build or enhance agents in their execution and management. Finally,

(last accessed May 28th, 2017)

[39] `http://www.omg.org/` (last accessed May 28th, 2017)

Section 3.3.3 presents research contributions that go beyond the previous two areas. In these contributions the distinction between agent and workflow management becomes less clear. These "advanced" integration efforts are the ones most closely related to this thesis.

3.3.1 Agent-based Workflow Management

Agent-based workflow management systems (AgWFMS) utilise agent technology to implement workflow management functionality. These kinds of systems realise an integration of sorts, yet this integration is only unidirectional. Users of the created systems usually aren't presented with agents or any agent-oriented modelling. Only workflows are emphasised. These integrations are therefore only partial, a term which is discussed further in contrast to "full" integrations in the later evaluations and discussions.

[Delias et al., 2011] describes, in depth, the different roles agents can play in workflow management. It represents an extensive survey, classifying over one hundred research contributions into seven areas of functions for agents in workflow management. This classification is roughly based on the WfMC workflow reference model (see Section 2.3.1) and is used in this section to structure the descriptions. Each classification group corresponds to an interface of the reference model and contains sub-groups describing different functionalities of agents in the context. Where available, more recent research contributions are cited. Recent contributions are discussed separately at the end of each functionality, while examples of citations from the survey in [Delias et al., 2011] are used in the text and marked with a *. Note that citations may appear in multiple functionalities, since agents are used extensively in some research.

Process Definition Tools The process definition tools interface of the WfMC reference model deals with providing, creating and editing workflow definitions. Agents can be used in a variety of fashions here, from actually modelling and encapsulating the process definitions and instances to providing and making these definitions and instances available for execution. There are three functionalities for agents identified:

Analyse, model, compose, describe and document a BP: In this functionality agents and their properties are used for realising a BP. There is a large variety of options available here. Agents can be used to represent and encapsulate the process or parts of/elements within it [Ooi, 2003]*, agent properties like mobility can be utilised to enable advanced, active behaviour of the process [Budimac et al., 1999, Fakas and Karakostas, 1999]*, agents can be used to orchestrate services into an overall process [Barker and Mann, 2006]* or agents can support a specific process as part of an application-specific solution [Jennings et al., 1998, Chen et al., 2000]*.

Recent Contributions:

- [Hsieh and Lin, 2016] uses a Petri net representation of an overall, adaptive workflow that is shared between agents involved in the workflow execution. Whenever changes occur/are necessary these changes are communicated between the agents to keep the process definition valid.
- [Bouzguenda and Abdelkafi, 2015] uses agent concepts and especially performatives from the FIPA agent communication language to mine organisational structures within workflow models.
- [Venero, 2014] uses mobile agent concepts in Petri net models to analyse inter-organisational workflows.

Process definition write/edit: This functionality enables agents to write and edit process definitions. Research in this area deals with agents autonomously managing the process definitions [Ehrler et al., 2006]*, managing adaptivity of the process definition in process instances [Wang et al., 2005c]* or modifying themselves if they encapsulate the process [Ooi, 2003]*.

Recent Contributions:

- [Rantrua et al., 2013] considers emergent behaviour from a group of cooperating agents to realise flexible workflows.
- [Saleem et al., 2014] uses agent concept to generate workflow definitions to share between different organisations.

Definition Retrieval: The final functionality identified for agents is the retrieval of process definitions from some persistent source for further usage [Zeng et al., 2001, Ehrler et al., 2006]*.

Recent Contributions:

- No prominent recent and/or new contributions known

Client Applications The client application interface emphasises human user support. Here, the interactions between the actual workflow management and the applications used by human users are described. Agents may be used to facilitate these interactions, handle data for the users or take control of certain aspects of process execution.

Worklist Handling: A worklist contains the available workitems for a user. Agents may be used to manage, fetch, display/provide or process worklists on behalf of human users [Manmin and Huaicheng, 1999, Cao et al., 2004, Trappey et al., 2009]*. Interpretation of different formats [Ricci et al., 2002]* or rules [Kappel et al., 2000]* are also an application of agents in the context of worklists.

Recent Contributions:

- No prominent recent and/or new contributions known

Process Control: In this functionality, agents are used to control the workflow instances on behalf and at the behest of the users. They act as process supervisors and may control operations like start, stop, pause or resume on instances, as well as control and enforce other state changes. Examples of encapsulating the process control interface for human user clients in agents are [Chen et al., 2000, Suh et al., 2001, He et al., 2006]*.

Recent Contributions:

- [López-Mellado and Flores-Badillo, 2013] proposes a WFMS as an agent management system that features mobile agents that guide and supervise the processes.
- [Delias et al., 2012] utilises agents to realise a manual intervention from human users to address robustness and flexibility issues.

Data Handling: This functionality utilises agents to handle, transport and make available the different data objects required for workflow execution and management. Often agents are used as a gateway to access different persistent data storages or repositories. Examples of agents being used in these ways include [Chang and Scott, 1996, Aye and Tun, 2005, Zhao et al., 2007]*.

Recent Contributions:

- [Zytniewski, 2016] proposes agents and full multi-agent systems as knowledge stores for process modelling and execution. The data provided by the agents is used to enrich the process definition.

User Interface: This functionality includes those research efforts, where agents directly provide the user interface and also interact in some form with the human user. They may offer limited support to users in their workflow execution [Hawryszkiewycz and Debenham, 1998]*, or may actively support users in their process [Ricci et al., 2002]*. User interface functionality may not require dedicated agents, but may be contained in more general agents within the system as well [Ehrler et al., 2006]*.

Recent Contributions:

- [Cranshaw et al., 2017] discusses the Calender.help system for scheduling meetings using workflows. Here, agents are interacted with as if they were personal assistants that automate as many workflow tasks as possible and only escalate to human users when necessary.
- [Zhao et al., 2016] enables incorporating human users into the workflow execution next to automatic agents executing tasks. Humans are represented as special "human agents" within the system that provide a web interface and control the life-cycle of human user tasks.

Invoked Applications The invoked applications interface serves a similar function in the context of agents as did the client application interface. However, instead of interacting with human users, agents here interact with other (automatic) applications, especially other agents. Due to the similarity in nature to the previous group, the functionalities are partially analogous.

Worklist Handling: Worklist handling is analogous to the client application functionality of the same name, only that other applications or agents are now the ones that are being provided with the workflows. The autonomy of agents in their execution can be utilised in this functionality [Stormer and Knorr, 2001]*.

Recent Contributions:

- No prominent recent and/or new contributions known

Process Control: Process control for invoked applications distinguishes two directions. Either agents control the invoked applications (and the processes therein) [Korhonen et al., 2002, Jingjing et al., 2004]*, possibly using web services orchestration [Buhler and Vidal, 2005]*, or agents are the invoked applications themselves [Jennings et al., 1998, Cao et al., 2004, Blake and Gomaa, 2005]*. In the latter case, the agents provide distinct functionality required to advance the work on the process instances.

Recent Contributions:

- [Hsieh and Lin, 2016] encapsulates tasks as individual task-agents that provide functionality and are orchestrated using a Petri net model to execute an overarching workflow.
- [Dam et al., 2015] translates BPMN models into agent concepts and artefacts to allow the execution of business processes in a multi-agent system context.
- [Liu et al., 2014] uses intelligent agents for two consecutive planning stages and subsequent control of the execution of manufacturing processes.

- [Rojek, 2016] employs agent simulation with historic experience to improve adaptive parameters in a production process.

Data Handling: Data handling between agents involved in invoked applications is, in concept, equal to data handling between agents involved in the client applications. The only difference is that human users are not involved as source or target of data handling operations. However, no further citations are given in [Delias et al., 2011].

Recent Contributions:

- [Feng et al., 2015] uses storage agents as centralised data stores that receive case data as input from survey agents and provide access to analysis agents.

Service Discovery: A specialised, yet widespread, use of agents in the invoked applications interface is service discovery. Here, agents utilise their functionality to gather data from service directories about available services (web or otherwise) that may be used to facilitate the workflow execution in some way. Examples of contributions in this field include [Madhusudan, 2005, Buhler and Vidal, 2005, Zhao et al., 2007]*.

Recent Contributions:

- [Hsieh and Lin, 2016] utilises agent intelligence and cognitive concepts to discover available actor and task agents that provide the services necessary for complex, overarching goals. These agents are then selected and orchestrated to execute a workflow that achieves that goal.
- [Narock and Yoon, 2015] uses agents to gather and aggregate provenance data, i.e. data concerning service creation and specification, about services that are candidates to be orchestrated in larger workflows.
- [Coria et al., 2014] describes an agent-based approach of automatically discovering web services and composing them into larger orchestrations.

Other Workflow Enactment Services The interface to other workflow enactment services is the focus point for interoperability of different WFMS. Therefore, ensuring that different systems can understand one another and the exchange data itself are a focus of the functionalities for agents in this group.

Common Interpretation of Process Definition: Different WFMS may use different representations or formats for the process definitions. In this functionality agents are responsible for ensuring that the different involved WFMS are able to understand each other's process definitions. Most often, this is handled via a common format or ontology. Solutions may feature agents acting as definition servers [Chang and Scott, 1996]* or encapsulating the representation [Merz and Lamersdorf, 1999]* or the process context [Ooi, 2003]*.

Recent Contributions:

- [Kravari et al., 2016] uses semantic web functionality in agents to allow for exchanging rules and policies relating to e-Commerce contracts without using a common syntax.
- [Benmerzoug, 2015] aggregates cloud services into agent interaction protocols to realise workflows. One focus here lies on bridging heterogeneous cloud environments through the use of agents.

Workflow Data Interchange: Exchanging workflow data between different WFMS is similar to the previously discussed workflow data handling functionalities. Yet additional challenges arise due to the more open, heterogeneous nature of the current interface. Agents for workflow data interchange may, for example, act as gateways that transmit the data [Omicini et al., 2006, Barker and Mann, 2006]* or realise data, e.g. process definitions, synchronisation between different involved workflow enactment services [Merz and Lamersdorf, 1999, Jennings et al., 2000]*.

Recent Contributions:

- [Santos-Pereira et al., 2013] utilises mobile agents to encapsulate and securely provide patient data used in healthcare processes.

Administration and Monitoring Tools The administration and monitoring tools interface deals with all aspects of organising and "managing" the workflow management itself. Here, auxiliary, organisational data is handled, edited, maintained and operated on. In addition to that, monitoring of the workflow system is also a focus.

User/Role Management: This functionality refers to agents representing human users as proxies. By utilising agents in such a way the agent becomes the virtual avatar of the human within the system, facilitating the system's interaction with the user technically. Such personal agents are widespread [Budimac et al., 1999, Barker and Mann, 2006]*. Another similar function is to have agents represent roles rather than individual users [Blake and Gomaa, 2005]*. Actual management functions regarding users and roles are rarely realised using agent concepts, yet some work exists, e.g. [Trappey et al., 2009]*.

Recent Contributions:

- [Kadar et al., 2014] proposes an approach in which agents representing human workflow users automatically negotiate with one another in order to create a consensus for all involved participants.
- [Samiri et al., 2017] realises all participants, including external resources like suppliers and customers, of a self-adaptive workflow systems as collaborative agents.

Audit Management: Audit management is concerned with managing, analysing and possibly reacting to workflow execution logs. Logs can be maintained by agents in a variety of ways [Wang et al., 2005a, Ehrler et al., 2006]* and different aspects can be evaluated through agent concepts [Chang and Scott, 1996, Kuo, 2004]*. Agents reacting to and handling exceptions is also possible [Wang et al., 2005b, Jennings et al., 2000]*

Recent Contributions:

- [Alfonso-Cendón et al., 2016] utilises agents for knowledge management in context-aware workflows. Agents monitor the workflow execution and autonomously decide on actions in order to ensure workflows reach desirable states.
- [Topçu and Yilmaz, 2014] uses an agent-based simulation to discover process models and evaluate workflow variants.
- [Wei and Blake, 2013] employs a number of monitoring agents in order to evaluate the system and workflow instance states.

- [Queiroz and Leitão, 2016] uses cooperating agents to analyse industrial workflows dynamically.
- [Feng et al., 2015] utilises analysis and recommendation agents working on a base of historic case data to determine the risks of process execution and recommendations for improvements.

Resource Control: In this functionality agents monitor, supervise and influence issues of process execution relating to resources and resource workload. Agents may, for example, analyse resource availability and workitem priority rules [Hongchen and Meilin, 1999]* or may use a process plan to facilitate the supervision [Jain et al., 1999]*.

Recent Contributions:

- [Yan et al., 2015] mines goals from BDI agents executing tasks in workflow execution. Monitoring these tools allows for the system to better support resource assignment.

Process Monitoring: The process monitoring functionality is concerned with keeping logs to supervise processes. It is distinguished from the audit management by the fact that no interpretation, e.g. analysis or reaction, is involved. Usually, special agents are used to monitor the system in part or in whole. Examples of research contributions in this area include [Manmin and Huaicheng, 1999, Biegus and Branki, 2004, Ehrler et al., 2006]*.

Recent Contributions:

- [Jhu et al., 2015] uses a so-called workflow agent to monitor and analyse scientific workflows executed in a cloud environment.

Workflow Enactment Service The workflow enactment service is the central component of the WfMC reference model that represents the runtime environment of workflow engines and the workflow instances executed within. Here, agents are used to facilitate the actual, technical execution of the workflows, rather than the interfaces around it.

Runtime Control Environment: This functionality encompasses agents as the controllers for the workflow instances in their entirety. Basically, it deals with agents as workflow enactment or workflow engine providers. Agents may serve directly as the workflow engine [Korhonen et al., 2002]*, managing and coordinating agents besides workflows [Chang and Scott, 1996]*, encapsulate workflows in special agent-specific formats [Jain et al., 1999]*, use communication mechanisms to coordinate workflow enactment [Marin and Brena, 2005]* or may utilise web services orchestration [Blake and Gomaa, 2005]*.

Recent Contributions:

- [Zhao et al., 2016] uses agents as centralised workflow engines that control and orchestrate tasks executed by other agents to achieve the overall workflow goal.

Definition Interpretation: Agents may be used to interpret process definitions and act according to the results of the interpretation. Such interpretations are for example used with BDI agents to determine goals achievable through the execution of a process [Merz and Lamersdorf, 1999, Ooi, 2003]*.

Recent Contributions:

- [Both et al., 2012] uses agent intelligence to reason about workflow progress in definitions in which that progress cannot be completely observed by the agents.

The goal of that reasoning is to still be able to support the user in the tasks he or she might possibly need to execute next.

Execution of Tasks: Execution of tasks functionality is concerned with agents performing and executing parts of a process. This includes the automatic execution of work associated with the tasks of the process. Agents may, for example, serve as the automatic resources directly [Stormer and Knorr, 2001, Marin and Brena, 2005]*, may encapsulate them [Zeng et al., 2001, Blake and Gomaa, 2005]* or may actually be the subject of the process that is being executed [Cai et al., 1996, Zhuge et al., 2002]*.

Recent Contributions:

- [Cranshaw et al., 2017] presents the Calender.help application in which agents are used to automatically execute simple tasks for scheduling meetings of human users.
- [Steckel et al., 2013] proposes an agent model for controlling robots executing work within agricultural workflows.

Scheduling: Scheduling deals with optimising the priorities, deadlines and constraints in order to make the workflow execution more efficient. Agents can utilise negotiation mechanisms or other communications for these purposes [O'Brien and Wiegard, 1998, Barker and Mann, 2006]*. They may also use rule sets [Joeris, 2000]* or their own mobility [Budimac et al., 2006]*.

Recent Contributions:

- [Chen et al., 2016a] uses negotiating agents to schedule task (agent) execution in cloud-based data-intensive workflows.
- [Werner-Stark et al., 2014] represents (user) resources as cooperating agents that negotiate in order to schedule the workflow execution to optimise time or cost.

Data Functions: Agents for data functions in workflow enactment are quite similar to data handling in client applications and invoked applications. Often, special agents are used to provide the workflow execution components with the relevant data functions they require [Xu et al., 2003, Ehrler et al., 2006]*.

Recent Contributions:

- No prominent recent and/or new contributions known

Task Assignment: Agents responsible for task assignment utilise their intelligence or other decision mechanisms to determine the resources to which tasks are assigned. There are two basic approach to this, either using agent negotiation [O'Brien and Wiegard, 1998, Jingjing et al., 2004]* or a hierarchical structure [Xu et al., 2003, Marin and Brena, 2005]*. In both approaches the requirements of the tasks are usually compared to the roles and abilities the resources (or the agents representing the resources) possess [Madhusudan, 2005, He et al., 2006]*.

Recent Contributions:

- [Wei and Blake, 2013] uses algorithms in agents to have them decide the optimal assignment of resources in cloud-based workflows.
- [Chen et al., 2016a] has different kinds of task, resource, manage and director agents schedule and allocate cloud resources for task execution in data-intensive workflows.

Resource Allocation: Similar to task assignment and scheduling, the resource allocation functionality allows agents to decide how resources are allocated in the system generally and with a larger scale than individual tasks. Here, again, negotiation is often used [Savarimuthu et al., 2005, Debenham and Simoff, 2006]* to deal with the allocation optimisation problem.

Recent Contributions:

– [Merdan et al., 2013] uses agent simulation to optimise flexibility and robustness of the dynamic execution schedule within a system.

Clearly, the above listing shows that agents can be utilised in workflow management in a variety of ways. This particular research field is also still quite active. Current work emphasises problems of workflow automation, orchestration and evaluation. Little to no prominent current work deals with worklist management and other data handling. It is feasible to presume that research of these problems is not stagnant, but is conducted in separate, independent contexts which can then be readily applied to agents, workflows and their integrations.

3.3.2 Workflow-based Agent Management

While using agents to build WFMS is fairly common (see previous section), the opposite direction of using workflows to build agent systems is rather rare. Reasons for this may lie in the fact that agents are an established tool for building software systems, which WFMS clearly are, while workflows are more established for modelling business contexts within other software systems. Still, some work exists that uses workflows to enhance or even build agents and agent management systems. All noteworthy research efforts known to the author are presented in the following, ordered by date of publication.

[Korhonen et al., 2002] aims at improving flexibility in agent interactions by avoiding hard-wired behaviours in favour of dynamic workflows. The approach is to have agents provide basic workflows describing their behaviour, functionality and capabilities, information about which is provided in the system as a kind of service directory maintained in a special workflow management agent. Each agent possesses an internal workflow engine, which then dynamically combines individual basic workflows of different agents into a master workflow that describes a complete interaction. Much of this approach, including questions about where the master workflow is executed is unclear and without any newer publications it appears that this approach has been abandoned.

[Kotb, 2011, Kotb et al., 2012] realise workflow net-based cooperating agents, especially in the context of agents controlling robots. They use algorithms to determine similarities in goals and capabilities of agents, which are used to propose cooperation plans for the considered agents. These plans are then transformed into complex cooperation workflows, which the agents follow to achieve their collaborative goal. A particularity of this work is that the authors utilise workflow soundness (see Definition 2.20) to ensure that agent plans are achievable.

[Mislevics and Grundspenkis, 2012] proposes a framework with which to model the behaviour of mobile agents using workflows. At runtime the mobile agents directly execute the modelled workflows featuring their behaviour. The authors argue that their approach is user friendly and that does not require much programming expert knowledge in order to

create applications using mobile agents. The graphical modelling capabilities of workflows are highlighted as a major reason for this.

[Küster et al., 2015] describes a formal mapping from BPMN (see Section 3.2.1) to agent models. They implement that mapping in different variants for the JIAC V (see Section 3.1) agent framework, with each implementation emphasising different aspects, such as versatility, performance or expressiveness. The mapping and implementation of the mapping allow for the specification of multi-agent system through workflow models, emphasising behaviour and processes.

[Nouzri and Fazziki, 2015] describes a transformation from BPMN models (see Section 3.2.1) to AML (agent modelling language, [Červenka and Trenčanský, 2007]) agent models and finally to the JADEX agent framework (see Section 3.1). The BPMN models are intended to capture business needs for an application, which are then transformed to incorporate agent concepts in the AML model, from which JADEX agent code is generated. The authors in [Nouzri and Fazziki, 2015] emphasise their work in the context of supporting BPM, yet there appear to be no actual limitations w.r.t. general system modelling.

[Gomes et al., 2016] models Extract-Transform-Load (ETL) patterns as agents using the YAWL workflow language. These simple, yet workflow-oriented agents are then managed in an overarching YAWL workflow to realise more complex ETL workflows. The agent metaphor is used here to specify a model of a system to be implemented at some later point. Whether the later implementation uses agents as well is unknown.

From the descriptions of these research efforts it becomes clear that, when workflows are used for agent management, their focus lies in the behaviour and cooperation. Workflows are used as a specification of agent behaviour and as execution templates. This aligns with some of the functionalities from agent-based workflow management, especially the invoked applications interface group. However, the focus in the contributions discussed in this current section is actually on executing agents, rather than executing workflows with the help of agents.

3.3.3 Advanced Integration Efforts

The distinguishing factor between the research efforts presented in Section 3.3.1 and those in Section 3.3.2 is the modelling and execution focus. The agent-based WFMS from Section 3.3.1 emphasise modelling and executing workflow and BPM, while the agent-based agent management systems emphasise agents. While there is some alignment and some similarities, the focus in those efforts is still clear.

The research efforts presented in this current section feature more sophisticated and extended integrations of agents and workflows. Here, that integration is advanced enough that a clear modelling and execution focus can't be determined anymore. In other words, these research efforts model and execute systems in which the line between agent and workflow can't be distinguished as clearly. In fact, these are the research efforts most closely related to this thesis' goal and motivation.

Regarding approaches with a more organisation-centric nature, only SONAR and ORGAN [Wester-Ebbinghaus, 2010] are considered here, due to the closeness of that research to

reference nets and the agent models used in this thesis. Other organisation-centric multi-agent system research [Ferber and Gutknecht, 1998, Ferber and Gutknecht, 1999, Gutknecht and Ferber, 2000, Ferber et al., 2003, Boissier et al., 2013] are not discussed here.

The following lists the advanced integration efforts ordered by date.

Agile, Goal-oriented BPM [Burmeister et al., 2006, Burmeister et al., 2008] present work focussed on endowing processes with explicit goals and a representation of context in order to support agile and flexible process execution. In order to realise the goals and handle/process the context representation, the authors utilise BDI agent models. For each process the goals are modelled first. The result of that is a set of goals and sub-goals. Next, context variables are defined, which can later be used to determine state and circumstances of process execution. Finally, plans describing how to achieve the different goals are defined and linked to contexts via conditions. These plans contain the behaviour executed and/or initiated by software agents, included automated behaviour concerning the processes and user interfaces. Prototypes have been implemented with JADEX (see Section 3.1) and a predecessor of LSPS (see below).

What qualifies this research effort as an advanced integration is the fact that the process is clearly and on a primary level endowed with cognitive agent concepts. However, the research effort still emphasises processes as the main modelling construct, even though those processes are very much like agents. By enabling the process to have goals, be aware of its context and autonomously react according to the two, the distinction between agent and workflow is less pronounced though.

Living Systems Process Suite The Living Systems Process Suite (LSPS) is a commercial business process management system developed by Whitestein Technologies[40]. Because of its commercial nature, relatively little information can be found about its origins. It appears to be a continuation of the Living Systems Technology Suite [Rimassa et al., 2006], an agent framework supporting BDI concepts, and the Living Systems Autonomic BPM platform [Greenwood and Rimassa, 2007, Greenwood, 2008], which already applied BDI concepts to workflow management. Processes in LSPS are defined with business goals in mind. "*Goals allow process management responsibility to drive general project planning and tracking, focusing on what is to be achieved before drilling down to consider the details of how to achieve it.*" [Whitestein15, 2015, p. 4] Plans are attached to specific goals and sub-goals and describe how the goal is to be achieved. Furthermore, processes are designed to utilise all agent mechanisms, including communication facilities for all involved parties, including the processes themselves and user groups working in the context of the processes.

WADE The Workflows and Agents Development Environment (WADE, [Caire et al., 2008a, Bergenti et al., 2012a, Bergenti et al., 2012b, Bergenti et al., 2013]) is an agent and workflow framework built on top of the JADE agent framework (see Section 3.1). Each agent in WADE has a set of workflows that describe different behaviours. The agents then strive to enact those workflows correctly according to the dynamic circumstances at runtime. The original focus of WADE was to use workflows to target "*the implementation of the internal behaviour of each single system*" [Caire et al., 2008a, p. 32]. More recently, that focus has shifted somewhat to include user-centric business processes through *interactive workflows*, i.e. workflows that are modelled around the central role of actors and users [Bergenti et al., 2012a, Bergenti et al., 2012b]. This means that WADE supports workflows for both the interactions of users and agents, making it an advanced integration effort.

[40] `https://www.whitestein.com/lsps-platform/lsps-overview` (last accessed May 28th, 2017)

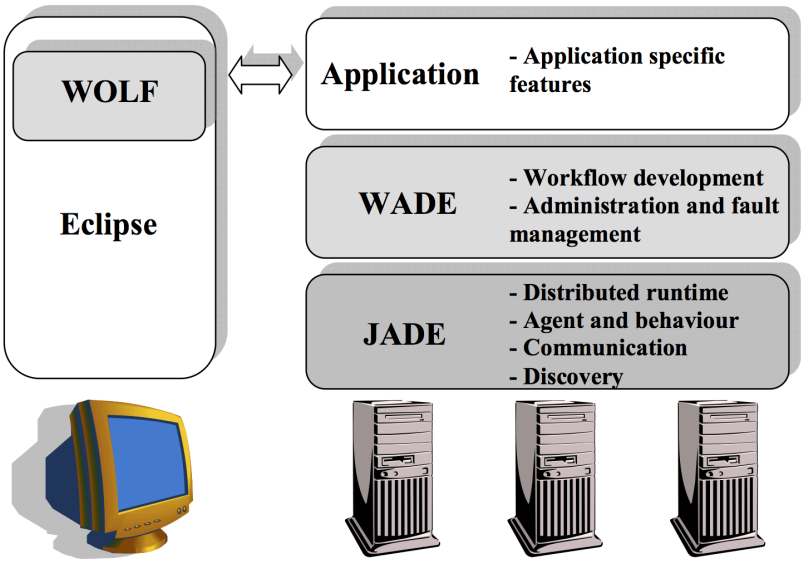

Figure 3.1: The WADE platform (from [Caire et al., 2008a, p. 32])

Considering the implementation, WADE adds two elements to the JADE platform, as illustrated in Figure 3.1. While the basic JADE platform takes care of the general agent runtime environment, the WADE extension allows for agent behaviour to be defined and executed as workflow tasks. Finally, the WOLF plugin [Caire et al., 2008b] for the eclipse development environment[41] allows for the graphical modelling of the basic workflows, which are then translated into Java classes, which can be extended to incorporate technical details not visible or available to the workflow view.

Sonar and Organ Both the *Self-Organizing Net Architecture* (Sonar, [Köhler-Bußmeier et al., 2009, Köhler-Bußmeier, 2009b]) and the "*Organisations-Architektur mit Netzen*" (Organisation-Architecture with Nets, Organ, [Wester-Ebbinghaus et al., 2009, Wester-Ebbinghaus et al., 2010, Wester-Ebbinghaus, 2010]) are concerned with modelling organisations with and through agents. Due to their high thematic proximity to reference nets, Mulan, Capa and Paose, they are considered here as related work, even though other organisation-centric approaches have been determined as out of scope for this thesis. Both Sonar and Organ utilise reference nets. Sonar provides a theoretical model for organisations, while Organ provides concrete models for organisational units.

Sonar describes an organisation as a specially formed Petri net, called a role/delegation net. In a role/delegation net, each task of an organisation is modelled as a place and each task implementation as a transition. Places are furthermore inscribed with organisational roles, while transitions are inscribed with distributed workflow nets supporting multiple roles involved in the workflow. Places and transitions are partitioned according to organisational positions. These positions determine who/what is responsible for the actual execution or delegation of executed tasks. Overall, organisation models in Sonar yield hierarchic, tree-like structures that suitably model the relationships including organisational structure, functionalities and interactions.

[41] `https://eclipse.org/` (last accessed May 28th, 2017)

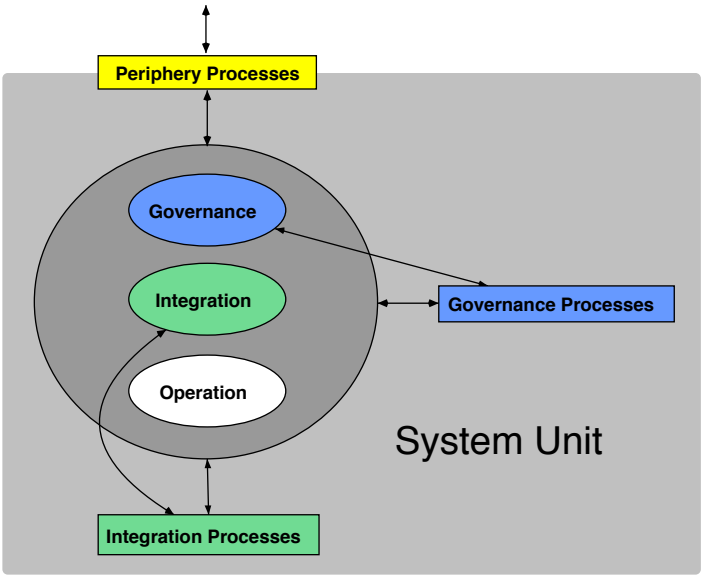

Figure 3.2: ORGAN organisational system unit (from [Wester-Ebbinghaus et al., 2009, p. 4])

The core component of ORGAN is a concrete and universal model for an organisational unit. That model can be seen in Figure 3.2 and it is applied, following reference net semantics of nets-within-nets, to all system elements and levels. Each organisational unit can be embedded into other organisational units and can itself be the environment into which other organisational units are embedded. Operation units function on the current level, integration units and processes function with and are embedded into the current level and governance units and processes represent and interact with the units into which the current level is embedded.

With this universal model of an organisational unit, ORGAN also defines a reference architecture consisting of four specific types of organisational units, namely society, organisational field, organisation and department. Each of these types is specified according to its content and organisational role. For the direct organisation type of organisational unit, SONAR is intended as the basic model. The organisational unit types of the reference architecture are all embedded into one another according to the integration and governance units defined in each (main) unit. For practical implementation of these organisational units, software agents and full multi-agent systems are intended to take the specific, technical places of units within the overarching organisational units.

The work on ORGAN and SONAR is considered as an integration effort for agents and workflows because the organisation concept here is used as a bridge between the two. In SONAR, which is used in ORGAN, organisational roles interact in some functional way. The organisational roles are filled by agents in an implementation, while the interactions can be considered as (distributed) workflows. Therefore, the organisation metaphor serves as the abstraction from its internal agent and workflow elements. Going further and considering the organisation as a collective actor interacting with other organisations as collective actors results in entities interacting with one another that are consolidated as both internal agents and workflows.

Process-Infrastructure for Agent Applications (PIA) The research presented in [Reese, 2010] envisioned and roughly sketched the idea of an integration of agents and workflows, which provides access to the strengths of both. The results of [Reese, 2010] provided part of the motivation and inspiration for this thesis. Implementing and realising that idea practically is part of the ambition of this thesis.

The main part of [Reese, 2010] is concerned with what is called a *process-infrastructure for agent applications* (PIA). PIA describes a system in which the external processes between agents (i.e. their interactions) are modelled and implemented as workflows and executed and supported by a workflow management system. The concept of PIA was developed in an integration architecture that described five different tiers of integration between agents and workflows. PIA is part of the fourth tier, which partially integrates agents and workflows by utilising workflows for agent interactions. The fifth and final tier of that integration architecture represents the full integration of agents and workflows, which is comparable to the core idea and motivation this current thesis pursues.

In other words, [Reese, 2010] and this current thesis share a common idea but the practical emphases diverge significantly. [Reese, 2010] focuses on PIA, which does not fully integrate and combine workflows, and only envisions a full integration. This thesis takes that vision as an inspiration and a starting point.

The following descriptions focus on the five tier integration architecture, as it is the most relevant related work to this current thesis. Afterwards, technical implementations of the fourth tier are shortly discussed.

The integration architecture can be seen in Figure 3.3. The starting point, with no integration of agents and workflows, is tier I with regular, traditional agent and workflow management systems.

In the second tier the respective opposite concept is used to realise management of the other. These are the agent-based WFMS and workflow-based agent management systems discussed in Sections 3.3.1 and 3.3.2 respectively. Here, traditional agent and workflow management is enhanced by concepts, mechanisms and ideas from the respective other concept. For application modelling, however, nothing changes as the main modelling constructs are always exclusively either agent or workflow.

The third tier changes that. Here, the capabilities of both agent and workflow management are provided for application modelling. There are no restrictions, limitations or guidances for modellers how to combine concepts and mechanisms from agents and workflows. This full combination of agents and workflows is immensely powerful, yet due to its unstructured and unguided approach almost unusable.

Structure and guidance are added to the approach on the fourth tier. Here, the capabilities of both agent and workflow management systems are restricted in an integration layer before being made available for application modelling. This results in multi-agent systems with workflow parts on the one hand and workflow systems with agent parts on the other hand. Utilisation of those secondary parts goes beyond what is done in the second tier. In the second tier agents were used to implement traditional workflow management and vice versa. This allowed the exploitation of some mechanisms in the background without changing the fundamental way agent or workflow management worked. In this fourth tier, however, agents are not (just) used to implement traditional workflow management, but rather improve it directly and combine it with agent management and vice versa for workflows. That way the traditional ways of managing agents or workflows can be radically changed. In fact, PIA is part of this fourth tier. Still, the fourth tier restricts the options for application modelling to exclusively agents or workflows, even if these agents and workflows strongly utilise the respective other concept.

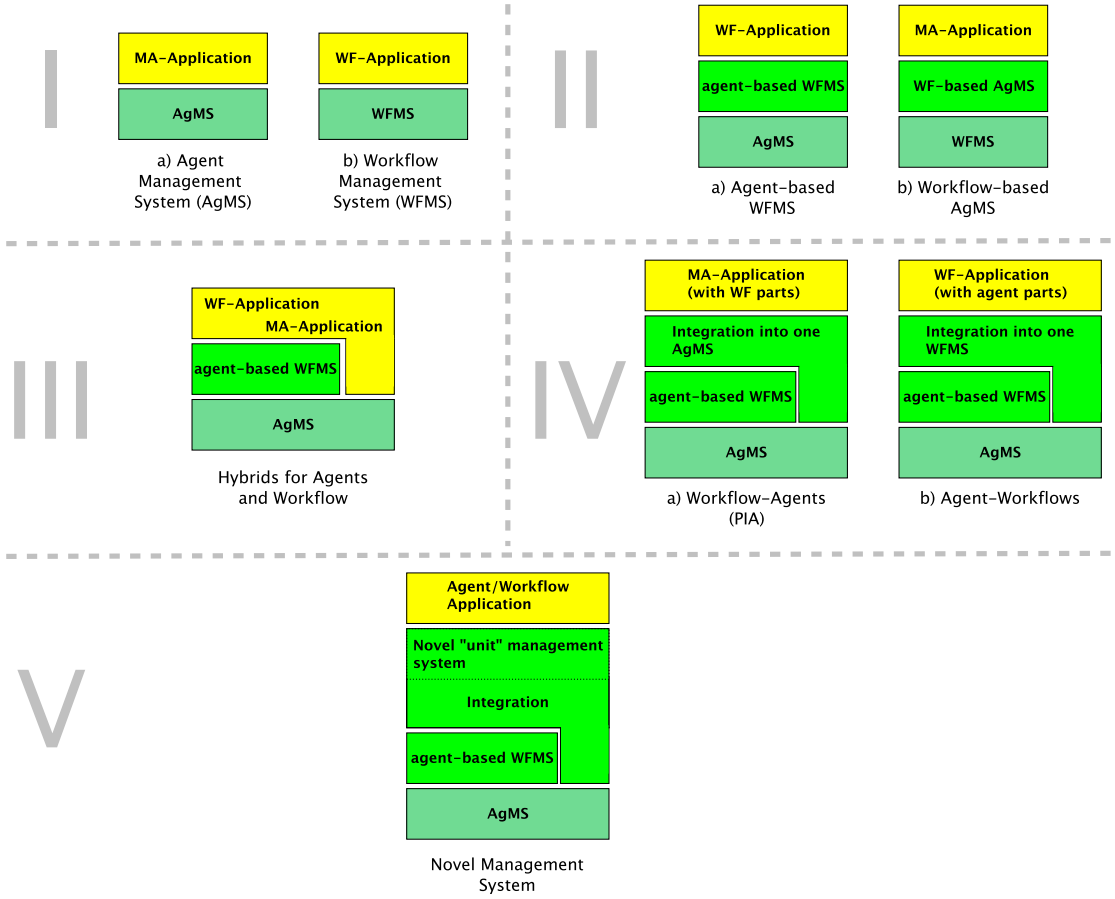

Figure 3.3: Tiered integration architecture
(modified and translated from [Reese, 2010, p. 119])

The fifth and final tier combines the two variants of the fourth tier into one completely integrated system. This system no longer uses traditional agents or workflows, but rather a novel "unit" which can be considered in both agent and workflow perspectives whenever it is useful. Being only envisioned and roughly sketched out in [Reese, 2010], few details are provided except that this tier utilises workflows for all behavioural facets of a system and agents for all structural ones.

The overall integration architecture describes a scale between no integration at all and a complete integration. Practically, [Reese, 2010] was concerned with Pia on the fourth tier of the integration architecture. A basic prototypical implementation of Pia was discussed in [Reese, 2010] as well. [Wagner, 2009c], the author's diploma thesis, described extended prototypes for both variants of the fourth tier. Due to the enhanced scope of those prototypes these are discussed here instead of the one from [Reese, 2010].

The focus of [Wagner, 2009c] lay on the provision of the second variant of the fourth tier of the integration architecture, the agent-workflows. Here, workflows are provided for application modelling which strongly utilises agents and agent concepts to improve workflow management. Out of multiple approaches the so-called structure-agentworkflow (S-AgWf) was chosen for further realisation. The basic operation of the S-AgWf is illustrated in Figure 3.4. Basically, a so-called structure-agent executes an overall workflow. That workflow consists of tasks which represent subworkflows that are executed by other agents

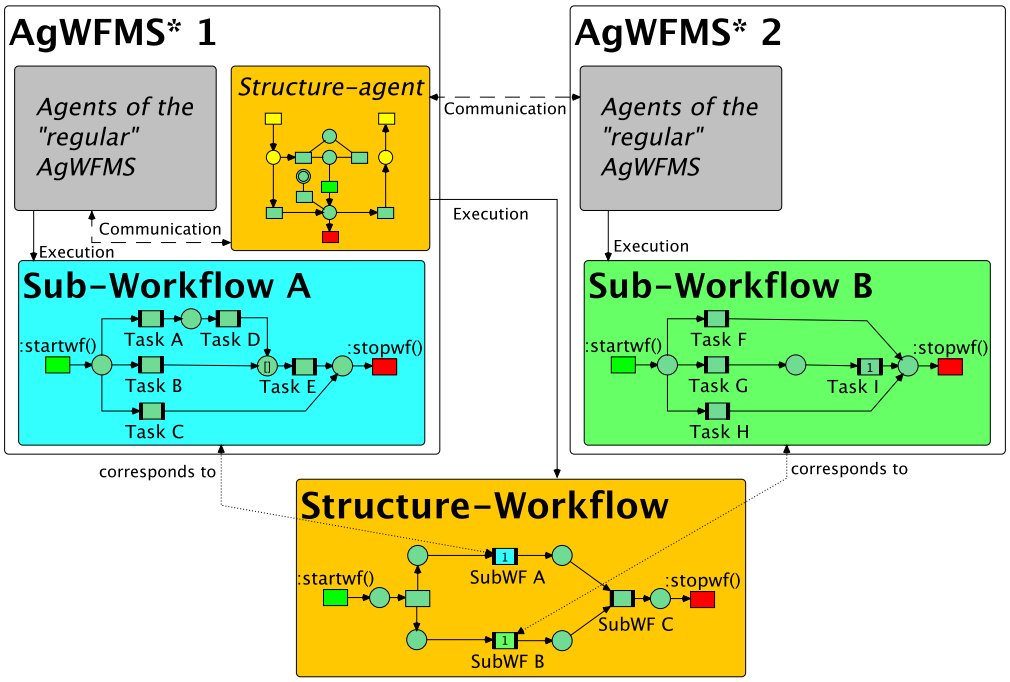

Figure 3.4: Structure-Agentworkflow principle (from [Moldt et al., 2010, p. 78])

of the management system. In this approach, agents control the execution of workflows, subworkflows and tasks allowing for the utilisation of agent features like mobility and intelligence.

For the other variant of the fourth tier of the integration architecture, the workflow-agents, [Wagner, 2009c] also described a realisation. This realisation is called the Protocol-Dispatcher (ProDi). Basically, a so-called interaction-workflow is used to control and manage the interactions between the different agents of the system. Each task in that interaction-workflow corresponds to a protocol in one of the agents involved in the interaction. This is a realisation of the PIA principle and allows for workflow features to be utilised for the interactions between agents.

More information about the overall integration approach and the prototypes can be found in, for example, [Reese et al., 2008, Wagner, 2010, Moldt et al., 2010, Wagner, 2012]. Of special relevance for this thesis is [Wagner et al., 2012], a revised journal version of [Moldt et al., 2010]. Thematically, it can be seen as beginning the transition between the research of [Reese, 2010, Wagner, 2009c] and this current thesis.

Jadex Active Components Jadex active components [Pokahr et al., 2010, Pokahr and Braubach, 2011, Pokahr and Braubach, 2013, Pokahr et al., 2013, Braubach et al., 2013] are the current continuation of the Jadex BDI agent framework (see Section 3.1). The current Jadex release 3.1 contains the active components extension and is available at the project website[42].

Active components represent an extension of the agent concept, incorporating concepts and mechanisms from other research areas, especially components. "*An active component is an autonomous, managed, hierarchical software entity that exposes and uses functionalities*

[42] `https://www.activecomponents.org/` (last accessed May 28th, 2017)

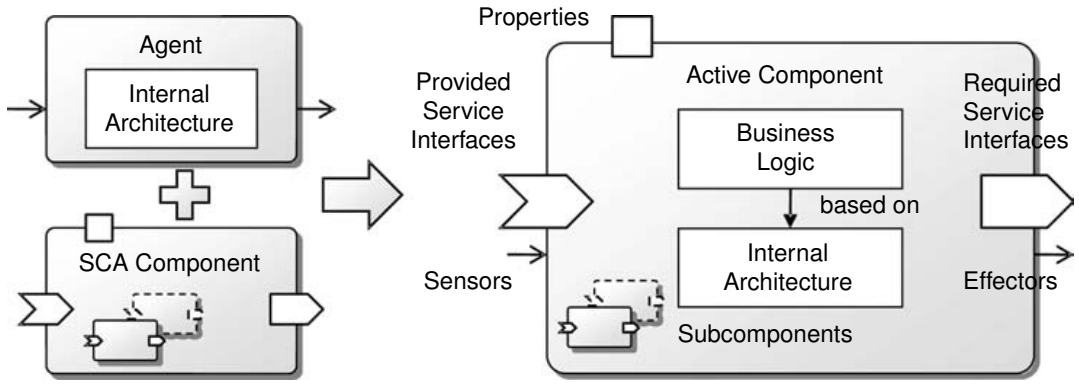

Figure 3.5: Jadex Active Component structure (from [Pokahr and Braubach, 2013, p. 14])

via services and whose behaviour is defined in terms of an internal architecture." [Pokahr and Braubach, 2013, p. 13] The motivation behind active components is similar to the one in this current thesis, arguing that different concepts have different strengths that should be combined and used together.

Figure 3.5 illustrates the previous definition and how the agent and component concepts have been integrated into active components. Active components adopt the internal architecture and message/external interfaces (sensors and effectors) from agents and the required/provided service interfaces and properties from service components. While both sides contribute to the overall active component, the emphasis is that the agent concepts support the active component internals, while the service concepts support the active component externals, i.e. interfaces. The internal architecture of an active component determines its basic execution mode, characterising a specific active component as a particular type of agent or another concept such as a workflow. Different internal architectures and the interplay between them are what establish Jadex active components as an advanced integration effort, which is discussed further below. Regardless of the specific internal architecture, the external interfaces of an active component remains unchanged, providing the means for interoperability between different software concepts.

The implementation of Jadex active components consists of four major areas. The *infrastructure* realises the execution environment and platform for active components. It provides three kinds of services: basic (e.g. component life-cycle management), remote (e.g. message service) and support (e.g. security and persistence). The *abstract interpreter* realises the standardised parts and interfaces of active components. Through the abstract interpreter, which is the same for all active components, it becomes possible for active components with different internal architectures to interact in a common way. *Kernels* realise the different internal architectures for active components. "*A kernel encapsulates the internal behaviour definition of a specific active component type.*" [Braubach et al., 2013, p. 4] There are a number of kernels already developed for active components, including, for example, BDI agents, microagents and BPMN (see Section 3.2.1). Additionally, another kernel exists for the goal-oriented business process notation (GPMN, [Jander et al., 2011]), a combination of BPMN and cognitive agent concepts. GPMN is discussed further below. The final area of the implementation concerns *tools*. Here modelling and implementation plugins for the eclipse development environment are provided.

W.r.t. the integration of agents and workflows pursued in this current thesis, there are two areas of Jadex active components that need to be discussed and considered in more detail. The first concerns the active component concept itself, while the second revolves around the special GPMN kernel.

The active component concept directly allows for an integration of agents and workflows. As described above, active components are completely interoperable, regardless of their internal architecture and kernel. That means that an active component using the BDI kernel can easily exchange data and interact with an active component using the BPMN kernel. For an integration that means that agents and workflows are being handled on the same execution and modelling level, doing their (application) part and role for the system and being able to communicate and influence each other's execution. This does not yet mean that the agent and workflow concepts are somehow combined. However, putting them next to one another already opens up a number of possibilities, e.g. agents participating in workflows or providing data for workflow execution.

The next feasible degree of an integration is to incorporate agents and workflows into one active component. An active component can only use kernel, and therefore only one internal architecture. This means that it is not directly possible to define an active component as employing both an agent and a workflow kernel simultaneously. However, each active component can have an arbitrary amount of sub-components with different kernels. This means that agent and workflow components can be subsumed by one active component acting as the intermediary for both concepts. In other words, a complex subsystem of agents and workflows can act as and be interacted with as a single entity in the overarching system This would integrate agents and workflows from an external and execution point of view, but not yet from a direct modelling point of view. Agent and workflow components would be incorporated into the same active component, but would be modelled and treated separately.

Still, this last issue can also be addressed. Active components are, by design, extendable. The concept itself only defines a container with standardised interfaces into which an internal architecture is embedded via a kernel implementation. Therefore, it is feasible that a new kernel representing an agent/workflow hybrid can be developed. Such a kernel is already provided by the GPMN kernel, which leads directly to the second point of integration discussion.

As mentioned above, GPMN [Jander et al., 2011, Jander and Lamersdorf, 2013, Jander, 2016, Jander et al., 2016] can be seen as a combination of BPMN with cognitive agent concepts, more specifically BDI. It was developed in the context of the Go4Flex project [Braubach et al., 2010]. One of the goals of GPMN is to distinguish technical and business aspects for modelling workflows more clearly. Business aspects emphasise the business value, performance indicators and business goals, while technical aspects emphasise concrete activities, control and data flow, etc. Both aspects are important for modelling, yet have different target user groups when it comes to modelling. By both distinguishing them while also maintaining them in a uniform, precise model or notation both user groups and their requirements can be addressed.

The core concept of GPMN is to imbue workflows with a hierarchy of goals. At the top, the abstract, overarching goal of a process is defined. Below it, several sub-goals refine the overall goal by partitioning it into smaller, more concrete parts. It is recommended, that the sub-goals are complete, i.e. that if all sub-goals are fulfilled the overall goal is also fulfilled. Beyond the sub-goals, any number of further sub-goal hierarchy levels can be added as necessary for a particular process. Concrete goals can be achieved by attaching plans to them, with multiple plans representing alternatives to achieving the

goal. Dynamic (runtime) circumstances of a workflow are captured by the *process context*, which is the basis for any reasoning about goals and plans. There are some more elements of a GPMN workflow model, like activation plans and subprocesses, available to further refine a workflow and capture more complex relations between workflow elements.

GPMN workflows are, for all intents and purposes, BDI agents representing workflows and are executed as such. They try to achieve their overall goal by reasoning about sub-goals and plans. Through the decomposition of the process into sub-goals, it becomes possible to flexibly adapt the execution of the workflow. If the BDI interpreter can detect that a goal is not achievable at a time it can reprioritise another goal for execution first. Accomplishing such an agility and flexibility in workflow execution was another focus of GPMN and the Go4Flex project.

GPMN workflows can be executed with the corresponding active component kernel. Furthermore, workflow management facilities, including user interfaces, are available. That way, workflow execution is fully supported.

W.r.t. an integration of agents and workflows, a GPMN active component is both a workflow and an agent. One advantageous result is that the workflow is able to reason about the process context and its goals and adapt its own execution accordingly. It can also communicate, via the standardised active component interfaces, with any other active components. While, for GPMN, workflows are still emphasised, the previously discussed integration aspects of active components in general also contribute to make Jadex active components and GPMN an advanced and comprehensive integration effort.

Hera and Potato [Markwardt, 2012] describes the Hera (HElper and Resource Agents) and Potato (Process-Oriented Tool Agents for Team Organization) systems. The Potato system is intended as a framework to realise a distributed software development environment. Similar to the work in this thesis, it emphasises structure and behaviour as key perspectives and dimensions of software systems and their development. One of the core aspects of Potato is that the tools of the development environment are implemented as interacting agents. Depending on the needs of the development, agents can be added or removed from the environment at any time. These agents are provided by the Hera system. Hera distinguishes user agents, helper/tool agents and resource agents. User agents represent the interface to human users. In order to provide users with the functionality they desire, user agents interact with the helper/tool and resource/material agents. The helper/tool agents implement the desired functionality, while the resource/material agents are passive (data) resources. Interactions between agents in Potato are defined as workflows using the Process-Infrastructure for Agent Applications (Pia) from [Reese, 2010] (see above). The concrete connection between Potato and Pia was examined more closely, in collaboration with the author of this current thesis, in [Markwardt et al., 2009b]. Pia and Potato have been developed in parallel and cooperation, meaning there is a strong conceptual alignment between the two. In fact, the reference architecture for Potato proposed in [Markwardt, 2012] can be regarded as a specialised version of the overall integration architecture from [Reese, 2010]. For Potato the generic agents from the integration were replaced in the Potato reference architecture with the helper/tool agents. As a consequence, Potato features an integration of agents and workflows, on top of which it provides a framework for the creation of systems designed to support distributed software development. Workflows in Potato are also intended to be flexible, work on which has been presented in [Markwardt et al., 2009a].

Conclusion Advanced Integration Efforts All of the research efforts presented in this section feature an integration of agents and workflows that goes beyond the ideas of agent-based workflow management or workflow-based agent management. From examining the different research contributions it becomes clear that the advanced integration is a desirable feature, yet how that integration is characterised and which mechanisms and concepts are emphasised is varied. Something that appears in many contributions is the introduction of cognitive concepts, especially goals, into workflows. While cognitive concepts are outside of the scope of this thesis, they are picked up again late in the thesis for the discussion and outlook.

Part B

Approaching an Integration

Part B presents the first results and contains four chapters. Chapter 4 discusses and clarifies the essential terms of structure and behaviour of a software system. Building on that, Chapter 5 refines and concretely defines the term structure for software agents, while Chapter 6 does the same for the term behaviour for workflows. The definitions from those chapters are central to the further discussions throughout this thesis. With the fundamentals defined for this thesis, Chapter 7 approaches the key ambition of an integration of agents and workflows. It provides the specification and vision of an integration and refines that vision into concrete integration criteria with which to evaluate possible integration models later. The chapter also specifies the concrete requirements to this thesis.

The purpose of this part is to lay out the basis for the conceptual and practical integration efforts in later parts of this thesis. It elaborates on, introduces and defines the key terms for the integration and then proceeds to flesh out the vision of the integration itself. This part contains analytical research, results of which are used throughout the remaining parts.

4 The Aspects of Structure and Behaviour

The terms *"structure"* and *"behaviour"* are, on their own, rather wide-ranging and vague. Their application to systems can vary wildly depending on the context and concrete research field. That understanding is required before the challenge of combining and integrating them can be addressed.

This chapter aims to establish an understanding of what exactly is meant by "structure of a software system" and "behaviour of a software system" in the context of this thesis. That understanding is required before the challenge of combining and integrating them can be addressed. It does not yet assume agents or workflows as the specific modelling constructs, but considers the structure and behaviour as aspects of a software system in general. The concrete definitions, which take the specific models and modelling techniques into account, follow later in Chapters 5 and 6 respectively.

The chapter consists of three sections. Section 4.1 examines definitions of the architecture of software systems with an emphasis on how the aspects of a system are defined and related. These definitions illustrate that the terms are not universally clear or uniformly defined. Section 4.2 then takes Petri net systems into account and examines them w.r.t. structure and behaviour. The result of that section is a clear understanding and distinction of structure and behaviour in net systems. This context also examines and affirms why structure and behaviour are suitable choices for an integration. Finally, Section 4.3 applies the view gained for net systems in the previous section to general software systems. This yields an understanding of structure and behaviour for software systems in the context of this thesis.

4.1 Examining Software Systems

Before the aspects of structure and behaviour of a software system can be more closely examined and understood, they have to be generally captured, understood and distinguished. In order to distinguish such specific aspects of software systems it is useful to first consider and examine the software system as a whole. A term often used for this purpose is *architecture* of a software system. There are many different definitions of the term coming from various research fields and contexts and emphasising different properties. The following paragraphs examine a selection of some of these definitions. The selection is partially based on the definitions provided by the website of the Software Engineering Institute of Carnegie Mellon University dedicated to software architecture[1]. That website contains a large number of definitions of architecture and provides an excellent overview of how the term is used. The following selection highlights different, general views on architecture as well as some additional definitions related to specific research fields. Each of the following definitions is examined w.r.t. how structure and behaviour are regarded.

– [Taylor et al., 2009] gives the following definition: "*A software system's architecture is the set of principal design decisions made about the system*" [Taylor et al., 2009, p. 58]. These design decisions "*encompass every aspect of the system under development*"

[1] `http://www.sei.cmu.edu/architecture/index.cfm` (last accessed May 28th, 2017)

[Taylor et al., 2009, p. 58]. The authors explicitly name the system structure, functional behaviour, interaction, non-functional properties and implementation as examples of those aspects [Taylor et al., 2009, p. 58]. An important facet of this definition is that only the "principal" design decisions are taken into account [Taylor et al., 2009, pp. 58–59]. This implies that the details are not considered part of the architecture. The selection of principal design decisions is left to the people involved in the creation of the software system.

An interesting and for this context relevant distinction is made about static and dynamic aspects of a system. "*Static aspects of a system are those that do not involve the system's behavior while it is executing. Dynamic aspects of a system involve the system's runtime behavior.*" [Taylor et al., 2009, p. 190] This distinction is, however, not always clear. Structure, for example, may be considered as a static aspect of a system but can evolve dynamically during the runtime of that system [Taylor et al., 2009, p. 190].

The view on the architecture expressed in these definitions is relatively general. It distinguishes between aspects of software systems and explicitly names structure and behaviour as two of them. It does, however, not really qualify what these aspects constitute, except for a vague grouping of certain design decisions. Consequently, it is not well suited for the purposes of this thesis.

– [Bass et al., 2012] defines architecture in the following way: "*The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*" [Bass et al., 2012, p. 4] At first glance this definition is very structure-centric: It identifies structures as the main elements in a software system. However, the authors examine three kinds of structures [Bass et al., 2012, pp. 10–11]: *Module structures* describe the static, functional elements (called modules) and their relations, *Component-and-connector structures* describe the runtime behaviour of elements and the dynamics between interacting elements and *Allocation structures* describe the connections to elements outside of the software system. Behaviour is consequently captured in the second kind of structure. The authors state that "*The behavior of each element is part of the architecture insofar as that behavior can be used to reason about the system. This behavior embodies how elements interact with each other, which is clearly part of our definition of architecture.*" [Bass et al., 2012, p. 6] This definition is also cited in [Tsui and Karam, 2010].

  For the purposes of this thesis this definition is too oriented towards the structure of a system. While behaviour is explicitly included within the definition as a special kind of structure it is treated secondary to the other structures. Therefore it does, on a conceptual level, exactly what this thesis has motivated to eliminate: using a main construct which governs every other aspect. This is partially addressed by the introduction of the concept of *views*. "*A view is a representation of a coherent set of architectural elements as written by and read by the system stakeholders. It consists of a representation of a set of elements and the relations among them. [...] In short, a view is a representation of a structure.*" (adapted) [Bass et al., 2012, p. 10] By combining the architectural elements dealing with structure into a view, a representation of the aspect of behaviour can be achieved. Nonetheless, this kind of helper construct does not change the issue of behaviour not being conceptually represented on the same level as structure.

– [Shaw and Garlan, 1996] states that "*The architecture of a software system defines that system in terms of computational components and interactions among those components.*" [Shaw and Garlan, 1996, p. 5] Components can be any kind of element in the architecture and the interactions are also not restricted in any way. This definition captures the structure of a system with the computational components and the behaviour in the interactions between them.

  One issue with this definition is apparent from the following: "*Structural issues include the organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives.*" [Shaw and Garlan, 1996, p. 1] This listing of structural issues shows the vagueness of the structural dimension. It includes aspects which could be easily argued as behaviour, e.g. communication protocols, control structures.

– [Kruchten, 2004] gives the following definition in the context of the Rational Unified Process: "*Architecture encompasses significant decisions about the following: The organization of a software system; The selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaboration among those elements; The composition of these elements into progressively larger subsystems; The architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.*" [Kruchten, 2004, p. 84] The author himself states that this definition has evolved from [Shaw and Garlan, 1996]. It covers software systems with a more concrete and detailed perspective. For the context of this thesis the explicit inclusion of the interfaces of structural elements and the behaviour are most relevant.

  The author elaborates on the description of an architecture by describing the five *views* included in the Rational Unified Process [Kruchten, 2004, pp. 87-89]. These views are *the logical view*, *the implementation view*, *the process view*, *the deployment view* and *use case view*. These views give insight into and capture facets of structure and behaviour for particular groups of individuals involved with the system (e.g. end users, testers, programmers).

  This definition of architecture and the corresponding views is also viable in the context of the Unified Modeling Language (UML). [Booch et al., 2005, p. 32] cites this definition and describes how UML can be used to capture the different aspects expressed in the views. Phillipe Kruchten later gives another, similar definition: "*Software architecture involves the structure and organization by which modern system components and subsystems interact to form systems, and the properties of systems that can best be designed and analyzed at the system level.*" [Kruchten et al., 2006, p. 2] This definition has less explicit emphasis on the behaviour, but includes general properties for design and analysis.

  The definition of architecture given in [Kruchten, 2004] is already quite suitable for the purposes of this thesis. It emphasises both structure and behaviour and distinguishes them through the use of the architectural views. However, the definition also includes many more aspects. The author states that "*Software architecture is not only concerned with structure and behaviour but also context: usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and trade-offs, and aesthetics.*" [Kruchten, 2004, p. 84] The explicitness of structure and behaviour is weakened by the inclusion of these other aspects, as some can be argued to be part

of structure, e.g. functionality (w.r.t. what element is providing what functionality), reuse, resilience, and behaviour, e.g. functionality (at runtime), usage.

– [Perry and Wolf, 1992, p. 5] defines software architecture as a tuple of a set of elements with a form and rationale. The architectural elements can be processing elements "*that supply the transformation on the data elements*" [Perry and Wolf, 1992, p. 5], data elements that "*are those that contain the information that is used and transformed*" [Perry and Wolf, 1992, p. 5] or connecting elements that "*are the glue that holds the different pieces of the architecture together*" [Perry and Wolf, 1992, p. 5] and "*which at times my either be processing or data elements, or both*" [Perry and Wolf, 1992, p. 5]. The architectural form of these elements "*consists of weighted properties and relationships*" [Perry and Wolf, 1992, p. 5] while the rationale "*captures the motivation for the choice of architectural style, the choice of elements, and the form.*" [Perry and Wolf, 1992, p. 6]

  This definition is again very structure-centric. The emphasis is on the elements and their relations. Behaviour (through processes) is only concretely considered in specific views on the architecture [Perry and Wolf, 1992, p. 6]. It is, in a way, contained in the connecting elements as they are exemplified with procedure calls and exchanged messages between other elements [Perry and Wolf, 1992, p. 5].

  The authors also note an interdependence between process and data similar to the interdependence between static and dynamic aspects observed in [Taylor et al., 2009]. The state that "*there are some properties that distinguish one state of the data from another; and those properties are the result of some transformation produced by some processing element.*" [Perry and Wolf, 1992, p. 6] They conclude that the data and process views are intertwined.

– The Institute of Electrical and Electronics Engineers (IEEE) provides standards dealing with software architecture. The original standard *IEEE Std 1471-2000 - IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* [IEEE00, 2000] was refined in 2007 to *ISO/IEC 42010 IEEE Std 1471-2000 - ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems* [IEEE07, 2007] and finally superseded by *ISO/IEC/IEEE 42010:2011(E) - ISO/IEC/IEEE Systems and software engineering – Architecture description* [IEEE11, 2011] in 2011.

  The current standard defines architecture as the "*fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*" [IEEE11, 2011, p. 8].

  Identifying basic aspects of a system, like structure and behaviour, is difficult in this definition. The notes on the terms state that "*In some cases, the fundamental elements are physical or structural components of the system and their relationships. Sometimes, the fundamental elements are functional or logical elements. In other cases, what is fundamental or essential to the understanding of a system-of-interest are its overarching principles or patterns.*" [IEEE11, 2011, p. 25] (similar statements are made in [IEEE07, 2007, p. 24]). This means that aspects of software systems can be captured with different scopes in different software architectures. Aspects could be captured with the notion of concerns, however the relationship between the two terms is unclear. Concerns of the architecture are captured by views. An architecture view is a "*work product expressing the architecture of a system from the perspective of specific system concerns*" [IEEE11, 2011, p. 8]. Concerns are identified as "*the*

purposes of the system; the suitability of the architecture for achieving the system's purposes; the feasibility of constructing and deploying the system; the potential risks and impacts of the system to its stakeholders throughout its life cycle; maintainability and evolvability of the system." [IEEE11, 2011, p. 18]. They are noted as "*any topic of interest pertaining to the system. The stakeholders of a system hold these concerns.*" [IEEE11, 2011, p. 26]. This means that anything might be a concern, which isn't helpful when trying to determine specific aspects. Consequently the definition is too vague for the purposes of this thesis.

– [Eeles and Cripps, 2010] uses the IEEE definition given in [IEEE07, 2007]. The authors do, however, elaborate on structure and behaviour in that definition. They emphasise structure and behaviour as key concerns of the architecture [Eeles and Cripps, 2010, Sec. 2][2]. The authors stress that structure can consist "*not only [of] the structural elements themselves, but also the composition of structural elements, the relationships [...], their interfaces, and their partitioning*" (adapted) [Eeles and Cripps, 2010, Sec. 2]. W.r.t. the behaviour the authors state that "*As well as defining structural elements, an architecture defines the interactions among these structural elements. These interactions provide the desired system behaviour.*" [Eeles and Cripps, 2010, Sec. 2]

  This definition addresses some of the issues present in the original IEEE definition. It explicitly examines both the structure and behaviour and lists them as two integral aspects within the architecture. However, the behaviour is still restricted to the interactions between the structural elements.

– [Jansen and Bosch, 2005] defines "*software architecture as the composition of a set of architectural design decisions.*" [Jansen and Bosch, 2005, p. 1] While this is similar to other definitions (e.g. [Taylor et al., 2009, Kruchten, 2004]) it elaborates further on the design decisions. A design decision is defined as "*A description of the set of architectural additions, subtractions and modifications to the software architecture, the rationale, and the design rules, design constraints and additional requirements that (partially) realize one or more requirements on a given architecture.*" [Jansen and Bosch, 2005, p. 2] This means the authors do not explicitly consider certain aspects like structure and behaviour but only look at the set of changes in the architecture. One of the reasons given for this view is that the design decisions are usually interconnected and involve multiple aspects of a system.

– [Gacek et al., 1995] gives the following definition of architecture: "*A software system architecture comprises: A collection of software and system components, connections, and constraints. A collection of system stakeholders-need statements. A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholder need statements.*" [Gacek et al., 1995, p. 2] This definition focuses more on the realisation of the needs of the stakeholders through the different components and their interconnections. Behaviour of the system, even interconnections between the components, is not captured here.

– [Rozanski and Woods, 2005] uses the following definition: "*The architecture of a software-intensive system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*" [Rozanski and Woods, 2005, p. 12]. This is similar to other

[2] [Eeles and Cripps, 2010] was only available as an Ebook edition without page numbers.

definitions, especially [Bass et al., 2012]. The authors though elaborate on the different structures in the system and identify two different types: "*The static structures of a software system define its internal design-time elements and their arrangement.*" [Rozanski and Woods, 2005, p. 12] "*The dynamic structures of a software system define its runtime elements and their interactions*" [Rozanski and Woods, 2005, p. 12] This distinction is useful for the purposes of this thesis as static structures can be considered as the structure, while dynamic structures can be considered as the behaviour. The definition, however, fails to capture the interdependency between static and dynamic structures, as runtime behaviour can have an effect on the static elements of the system.

The previous definitions were all very general. They defined the architecture of a software systems independent from any specific research context or modelling technique. The following definitions also define the architecture of systems, but do so in a specific context:

– [Dumas et al., 2013] explicitly defines the term *process architecture* as "*an organized overview of the processes that exist within an organizational context, which is often accompanied with guidelines on how they should be organized.*" [Dumas et al., 2013, p. 38] It "*is a conceptual model that shows the processes of a company and makes their relationships explicit.*" [Dumas et al., 2013, p. 42] The definition of process architecture is similarly made in [Malinova et al., 2013]. There, the authors elaborate that "*A structure for a process model collection is often referred to as process architecture.*" [Malinova et al., 2013, p. 1]

  These BPM-centric definitions are in contrast to the other ones described in this Section. Here, the behaviour of the system, given by its processes, is the clear focus. The architectural elements, which are otherwise often equated with functional modules, are the processes. Connections and relations between these processes are captured quite nicely, while the classical structural elements, like resources and users, are not.

– [van der Aalst et al., 1999] provides Figure 4.1 describing the architecture of a workflow following nets-within-nets ideas. They identify five perspectives that are combined by the enactment service to cover the entirety of the workflow execution. The five perspectives are the control-flow (contains workflow process definitions), resources (contains organisational structure and entities), data (contains workflow routing and other data), task (contains instructions/definitions for individual tasks) and operation (contains the mapping between operations, data and applications).

  Obviously, this definition emphasises the behaviour, as it originates in workflow management. While structure is contained in the resource and data perspectives, the elements of that perspective exist to enable the execution of a (workflow) process. Also, while structure and behaviour can be identified in the perspectives, a clear separation and application is not explicitly made. Consequently, the definition is unsuitable for use in this thesis.

– [Hayes-Roth, 1995] gives a definition of architecture in the context of agents. It states that "*By 'architecture' we mean the abstract design of a class of agents: the set of structural components in which perception, reasoning, and action occur, the specific functionality and interface of each component, and the interconnection topology among components.*" [Hayes-Roth, 1995, p. 3] This definition differentiates between agents and other structural components that serve as the agent environment. It emphasises the structure through structural elements (agents and components), functionality and

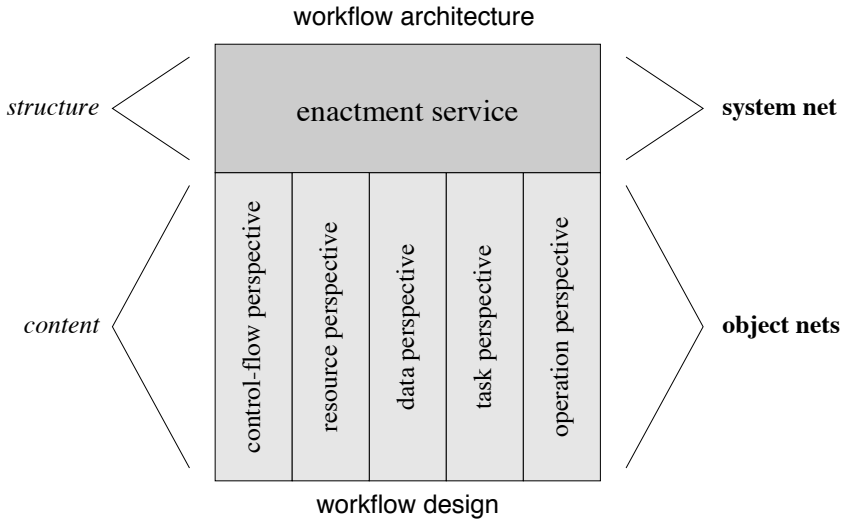

Figure 4.1: Workflow architecture and related perspectives (from [van der Aalst et al., 1999, p. 2])

interfaces but disregards direct behaviour. It is seen as something that needs to be supported by the architecture: "*Conversely, to function effectively in a particular niche, an agent must exhibit the range of behavior required in that niche and, therefore, must have an architecture that supports the required behavior*" [Hayes-Roth, 1995, p. 4]. In this way this definition supports the motivating argument in this thesis that agent systems emphasise structure over behaviour.

– [Booch et al., 2004] generally defines "*the architecture of a complex system is a function of its components as well as the hierarchic relationships among these components.*" [Booch et al., 2004, p. 12]. In the more specific context of object-orientation the authors state that "*Combining the concept of the class and object structures together with the five attributes of a complex system (hierarchy, relative primitives [...], separation of concerns, patterns, and stable intermediate forms), we find that virtually all complex systems take on the same (canonical) form [...]. Collectively, we speak of the class and object structures of a system as its architecture.*" (adapted) [Booch et al., 2004, p. 16] This definition also disregards the behaviour and focuses solely on structures and relations.

There are a number of observations that can be made for these definitions. The specialised definitions at the end ([Dumas et al., 2013, Hayes-Roth, 1995, Booch et al., 2004]) are strongly influenced by their context and origin. [Dumas et al., 2013] and [van der Aalst et al., 1999] come from the fields of BPM and workflows. Consequently, they have a strong behavioural emphasis. [Hayes-Roth, 1995, Booch et al., 2004] originate from agent-orientation and object-orientation respectively. Hence, their focus lies on the structure of the system. These definitions are not very suitable for the purposes of this thesis as they exemplify the exact issue this thesis aims to address: The focus on one aspect due to the employment of one main modelling construct.

The general definitions at the beginning are different in that respect. The definitions in [Jansen and Bosch, 2005, Gacek et al., 1995, IEEE00, 2000, IEEE07, 2007, IEEE11, 2011] stand apart from the rest. They don't explicitly focus on either structure or behaviour but are defined in a way that includes these aspects.

The other definitions all incorporate both structure and behaviour to some degree, though with different emphases. [Bass et al., 2012, Perry and Wolf, 1992] are very structure-centric but both still consider the behaviour. The issue with these is that behaviour appears to be of lower importance (for the architecture) when compared to the structure. The remaining definitions though ([Taylor et al., 2009, Kruchten, 2004, Eeles and Cripps, 2010, Shaw and Garlan, 1996, Rozanski and Woods, 2005]) consider structure and behaviour as at least comparable w.r.t. their importance. There is still some preference to the structure (e.g. emphasis on components of system first and their interactions second) but these definitions are relatively suitable for this thesis. They give a viable perspective on the system which incorporates the aspects motivated and focussed on for the integration.

However, none of these definitions are entirely fitting for the purposes of this thesis. The common issue is that the scopes of these definitions are broader then structure and behaviour and yet also still too unclear about either one.

One of the reasons is that the definitions don't distinguish clearly enough between structure and behaviour. They often incorporate structures or structural elements and the interactions between them but fail to specifically delineate them. The issue here relates to processes (see Definition A.8 from Section 2.4). Processes can clearly be considered as architectural elements as well (e.g. as seen in the specialised definition in [Dumas et al., 2013]). They are, however, also part of the behaviour as they are used to describe interactions between different structural elements of a system. Distinguishing between processes as structures that describe such interactions and processes as structures that don't involve interactions obscures the lines between structure and behaviour.

Some definitions elaborate further by describing dynamic and static aspects of the system ([Taylor et al., 2009] and [Rozanski and Woods, 2005]). Dynamic elements are characterised as the behaviour at runtime, while static elements are characterised as the elements at design-time. In this way the definitions in [Taylor et al., 2009] and [Rozanski and Woods, 2005] determine the distinction between structure and behaviour as the distinction between static and dynamic aspects of the system. This distinction is also used, for example, in [Löwe et al., 2001]. It is however, still a bit too simplistic for the purposes of this thesis. It is again problematic w.r.t. processes. Processes may be explicitly defined as elements at design time and then executed at runtime. This would make them both dynamic elements as processes at runtime and static elements as processes as design artefacts.

There is also the problem of interdependency of static and dynamic aspects as pointed out in [Taylor et al., 2009]. The static aspects may change due to dynamic aspects and thus become dynamic aspects as well. This means that elements of the structure not only exhibit behaviour but can also become elements of the behaviour as well.

In addition to these issues the definitions often incorporate additional aspects of software systems. The problem here is that these other aspects are arguably part of the structure or behaviour. The definitions though make them explicit and distinct. For example the aspect of organisation of the system is named first in the definition in [Kruchten, 2004], even before structural and behavioural aspects. But the organisation within a system is, as supported by the definition in [Shaw and Garlan, 1996], part of the structure of a system. In general, the definitions are broader and go beyond structure and behaviour by extracting things from them. This makes it more difficult to determine structure and behaviour clearly.

Another open point is the determination of the significance or principality of architectural elements. Some of the definitions explicitly stipulate that the architectural elements need to be significant, fundamental or principal enough to be taken into account in an architecture [Taylor et al., 2009, Kruchten, 2004, IEEE11, 2011]. The reason for that is nicely put by

Perry and Wolf: "*It is very difficult to abstract design and architecture from all the details.*" [Perry and Wolf, 1992, p. 3] However, this leads to most of the definitions ignoring *internal* behaviour of the architectural elements. The processes happening within the architectural elements determine, to a large degree, their external behaviour as well. E.g., internal data processing determines when data is ready for external interactions with other components. For the context of this thesis internal behaviour is essential and needs to be regarded as an important, explicit part of the behaviour of a software system.

As this thesis explicitly deals with processes in all of these forms (e.g. processes as interactions between architectural elements (e.g. agents) and processes within architectural elements, processes at runtime, processes as artefacts (e.g. workflows)), the definitions presented so far are not sufficiently clear enough in their specification of structure and behaviour. (Parts of the) Structure can become (parts of the) behaviour due to dynamic (behavioural) aspects. Likewise (parts of the) behaviour may need to be considered as structural elements and artefacts. The definitions of structure and behaviour need to capture the interdependencies between the two aspects. The previous definitions of architecture provide excellent ideas and concepts that need to be incorporated into the notions of structure and behaviour but cannot serve as the basis for their definition.

4.2 Considering Petri Net Systems

A different approach to examining software systems as a whole is to examine a model of them. In this thesis these models of (software) systems are given as Petri nets. A (Petri) net system is given as a set of places, a set of transitions, the flow relation between them and an initial marking. The net system is executed with the initial marking serving as the initial state of the system. The following observations and discussions are based on views adapted from *unit theory* [Moldt, 2005]. For the purpose of this discussion, the net system is considered as a simple P/T net (see Definition 2.1 in Section 2.1). The nets-within-nets concept is added later in the discussion.

The static aspects of the system are the places, transitions and the flow relation. These static aspects remain and do not change[3] during the execution of the net system. Static aspects represent the possible states and determine possible firing sequences from the initial state.

These firing sequences (see Definition 2.2 in Section 2.1) are the dynamic aspects of the system. This includes first and foremost the actually executed firing sequence. During execution actions are performed by firing transitions which "move"[4] tokens from one state to another. While the possible firing sequences are defined by the static aspects through the flow transition between transitions and places they are part of the dynamic aspects. The difference to the actually executed firing sequences is that the latter are only determined at runtime due to concurrent actions and non-deterministic choices.

There is a final element of a Petri net system which doesn't quite fit into either static or dynamic aspects: the tokens. The tokens in a net system describe a marking of that net which in turn represents a state of the net system. The marking is part of the static aspects as it also exists outside of the execution of the net. During execution, though, each

[3] Adaptivity of nets is an advanced feature of certain Petri net formalisms and tools. However, it is not included in the basic net systems that are considered in this section. The topic of adaptivity is discussed further later on in this thesis.

[4] Tokens aren't actually "moved" in Petri nets. They are consumed by transitions on incoming arcs and created on outgoing arcs. When the figure of speech of "moving" tokens is used in this thesis it is merely to illustrate how a certain net works.

action causes the marking to change[5]. This means that the marking is also part of the dynamic aspects of the system. In other words, the marking connects static and dynamic aspects in a net system.

With these preliminary considerations, applying the terms structure and behaviour to a net system is possible. Figure 4.2 illustrates this. The structure of the system is given by the places, transitions, flow transition and the current marking (i.e. state) of the system. The behaviour of the system is the set of its possible and executed firing sequences, which are determined by the initial and subsequent markings. The current marking is located in Figure 4.2 in the grey area. This is to indicate markings as part of both structure and behaviour.

This view on net models covers all facets and properties of the net system. Even in more advanced higher-level Petri net formalisms nothing else is added on this conceptual level to the nature of Petri nets. In the end, everything else is mapped onto places, transitions, arcs and tokens. This addresses one issue with the definitions presented before. Structure and behaviour are the sole aspects under consideration and are clearly distinguished from one another. Markings are part of both, but under observation it is clear from context if they are viewed statically or dynamically. In fact, as indicated in Figure 4.2, the two aspects are orthogonal to each other with the markings providing the connection. This view is analogous to the view on software systems in the PAOSE approach ([Cabac, 2007], see Section 2.2.3). Orthogonality in this context means that the two aspects are distinct and their intersections clearly defined, but still span the entirety of the system.

Regarding the issue of principal/significant details of systems it is relatively simple to abstract or add details to a Petri net system (model). To remove details modellers can merge subnets into one transition or one place[6]. To add details a transition or place can be replaced by a complex subnet. This doesn't add any new conceptual elements to the net system or change the way it should be regarded. This is a well-researched area for Petri nets. [Murata, 1989], for example, proposes a number of transformation rules in nets. These rules can be used to either add or remove details from a net while maintaining interesting properties like boundedness or liveness. The problem with this approach is that there is always only one level of abstraction under consideration. In order to add or remove details a transformation is performed which creates a new model. Modellers would have to keep track of each of these models and switching between them would be cumbersome.

The most crucial issue of the previous definitions was the shortcoming w.r.t. the interdependency between structure and behaviour. This is also addressed in the view given by the net system. The possible firing sequences are determined by the places, transitions, arcs and current marking/state. In this case the behaviour/dynamic aspects are clearly dependent on the structure/static aspects of the system. On the other hand, the (statically considered) current marking, which defines the state of the system is a result of the executed firing sequence so far. The (again statically considered) possible next marking is determined (through concurrent, non-deterministic choice) by the possible firing sequences. In this case the structure/static aspects are dependent on the behaviour/dynamic aspects of the system. However, the interdependency is still limited in a way because of the limited scope of the tokens. Even considering tokens which are distinguishable and typed (e.g. in coloured Petri nets) it can be difficult to adequately capture a complex system state. Another issue is that tokens and the marking, while part of both structure and behaviour, don't actually contribute anything to them except for the representation of state.

[5] Unless the action consumes and creates the same amount of tokens in the same places. But this trivial case does not affect the argument as tokens are still consumed and created.

[6] If the subnet is transition- or place-bordered respectively.

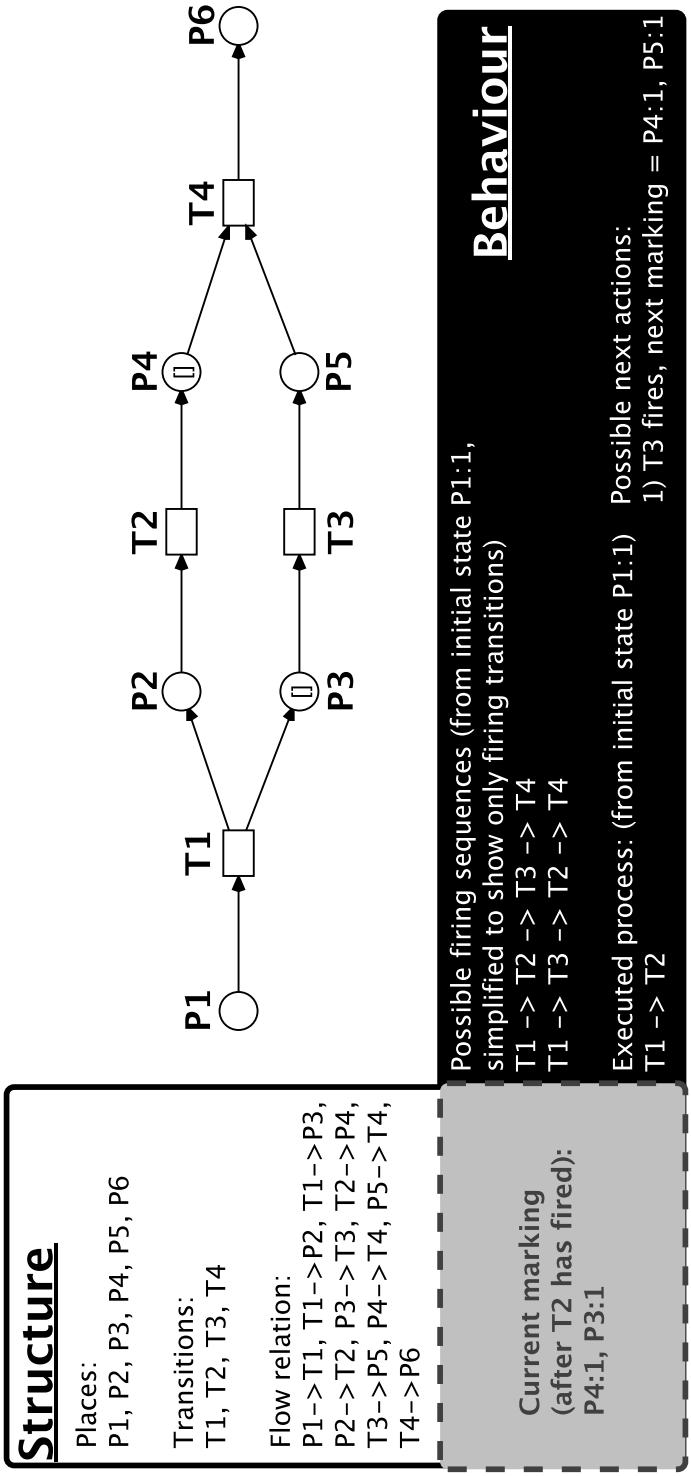

Figure 4.2: Structure and behaviour in a net system

Processes, in the sense of Definition A.8 from Section 2.4, also do not create as much issues in net systems. Processes usually can't be directly equated to firing sequences. Rather, a firing sequence represents one specific run of a process. If the process has no alternatives or concurrent behaviour, the firing sequence and process are equivalent. Each alternative and concurrent action within a process requires one or even more firing sequences to represent. This means that a process is represented by a set of firing sequences. Furthermore, in a net system a process can be defined on the entire net or just a subnet. Transitions in the net structure describe the tasks of the process, so the scope of the (sub-)net is directly linked to the scope of the process.

This means that processes are part of both the behaviour and the structure of a net system and can be regarded in both perspectives. The set of transitions and places can be regarded as the structural representation of a process. The firing sequences that represent the different possible runs of a process can be regarded as the behavioural representation of a process. This means that the line between structure and behaviour considering processes is not obscured.

Still, processes as structures in the behaviour, i.e. as artefacts used by the system, are difficult to realise in standard net systems. It is possible to solve this issue with complex net patterns simulating a process as an artefact but this requires a large amount of additional transitions and places that are quite difficult to keep track of.

This issue also applies to any specific components or other architectural elements in a system. They can be modelled and identified as subnets in the net system. However, having these components interact also requires a kind of simulation of their interactions with large and complicated net patterns.

In summary, the view of net systems addresses all the issues identified for the previous definitions. However, it does so only to a certain degree and in rather simple form. Structure and behaviour are distinguished clearly and cover the whole system, but it is difficult to separate specific components and have them interact. Processes are considered in different forms, but modelling a process as a structure inside the behaviour is not feasible. Interdependency is addressed, but is difficult to capture due to the relative simplicity of tokens. Different levels of abstraction can be achieved through net transformations but at any one time there is only one abstraction level available in a model.

All of these issues can be traced back to the fact that a simple net system only provides one level of abstraction. One could say it is too "flat" to capture multiple levels of abstraction. A hierarchy of abstraction levels is a basic requirement to support individual components, their interaction, complex tokens and abstract views. This requirement is achieved by using the nets-within-nets concepts (see Section 2.1.2). The nets-within-nets concepts can be applied for the context of this thesis as the reference nets are used as the main modelling formalism.

The nets-within-nets paradigm enables tokens in nets to be other nets. This allows for complex nesting of abstraction levels. The change in abstraction levels addresses the issues presented before.

Insignificant details can now be hidden inside of the hierarchy without having to create and keep track of new models. By pushing the details into a new net inside of the net system it remains in the system and can be accessed at any time.

Processes can be modelled as separate nets and executed within the net system. This enables them to be structure (the new net places, transitions, arcs and marking), behaviour (the set of firing sequences of the new net) and structure within the behaviour (the token in the net system). Components, likewise, can be modelled separately from the net system. The nets of these components can then interact (e.g. via synchronous channels) when

their tokens are involved in the firing of a transition. This encapsulates the structure and behaviour of these components and puts them into the context of the structure and behaviour of the net system. Neither processes nor components require any overly large or complex net patterns for this. The relocation into another level of the system naturally supports this.

The remaining issue is that of interdependency. When considering processes or components as separate nets these nets have their own structure and behaviour. This means that both structure and behaviour of the original net system have an influence on both structure and behaviour of these separate, subordinate nets. The structure of the overarching net system determines what can possibly happen to and in the subordinate nets, while the behaviour dictates what actually does happen. This represents a kind of control the overarching net system has over its subordinate nets. But this control is partially bidirectional as the net system can't force actions in the subordinate. It can't continue its own process if the subordinate doesn't or can't fire the necessary transition.

The degree of that influence and control is dependent on the specific model. The entire mechanism allows for everything from simply starting the subordinate net and letting it execute on its own to actually controlling every single transition in the subordinate net. This means that the interdependency between structure and behaviour is now no longer captured only in the "flat" net system through simple tokens. It cascades into the different levels of the net system hierarchy[7]. This provides the model with a much more finely nuanced representation of the interdependency between structure and behaviour. Taking reference semantics into account it is also possible for a token to refer to the net in which it exists. This self-reference enables applying this advanced representation of the interdependency to the net on the most abstract level of the net system itself.

Net systems following the nets-within-nets paradigm finally provide an appropriate view on systems for the purposes of this thesis. Structure and behaviour are clearly distinguished, different abstraction levels allow artefacts of both structure and behaviour to be used in any way possible and interdependency between structure and behaviour is completely captured.

This view justifies the choice of structure and behaviour as the aspects that should be integrated. They are the only two aspects of the system. Everything else is in some way mapped onto subsets of them. This means they can capture the entirety of the system.

4.3 Structure and Behaviour of a Software System

To summarise the previous section, examining (Petri) net systems as models of systems has provided a suitable and adequate perspective on structure and behaviour. The final step is to map and relate the specific concepts and ideas from software systems to the more abstract terms of places, transitions, arcs and markings.

This thesis uses Petri nets with nets-within-nets (reference nets, see Section 2.1.2) as the basis for all developed software systems. For example, agent systems follow the Mulan reference architecture (see Section 2.2.3) and workflows are implemented as (reference) workflow nets (see Section 2.3.2). Consequently, the step from net systems to software systems isn't a big one.

The general approach is to identify components of the system and realise them as separate, interacting nets in a hierarchic net system. This means that the structure of all nets in the net system defines the components and the infrastructure and organisation between them. Considering the net system in these specific terms, though, is nothing more

[7] The subordinate nets may have subordinate nets of their own.

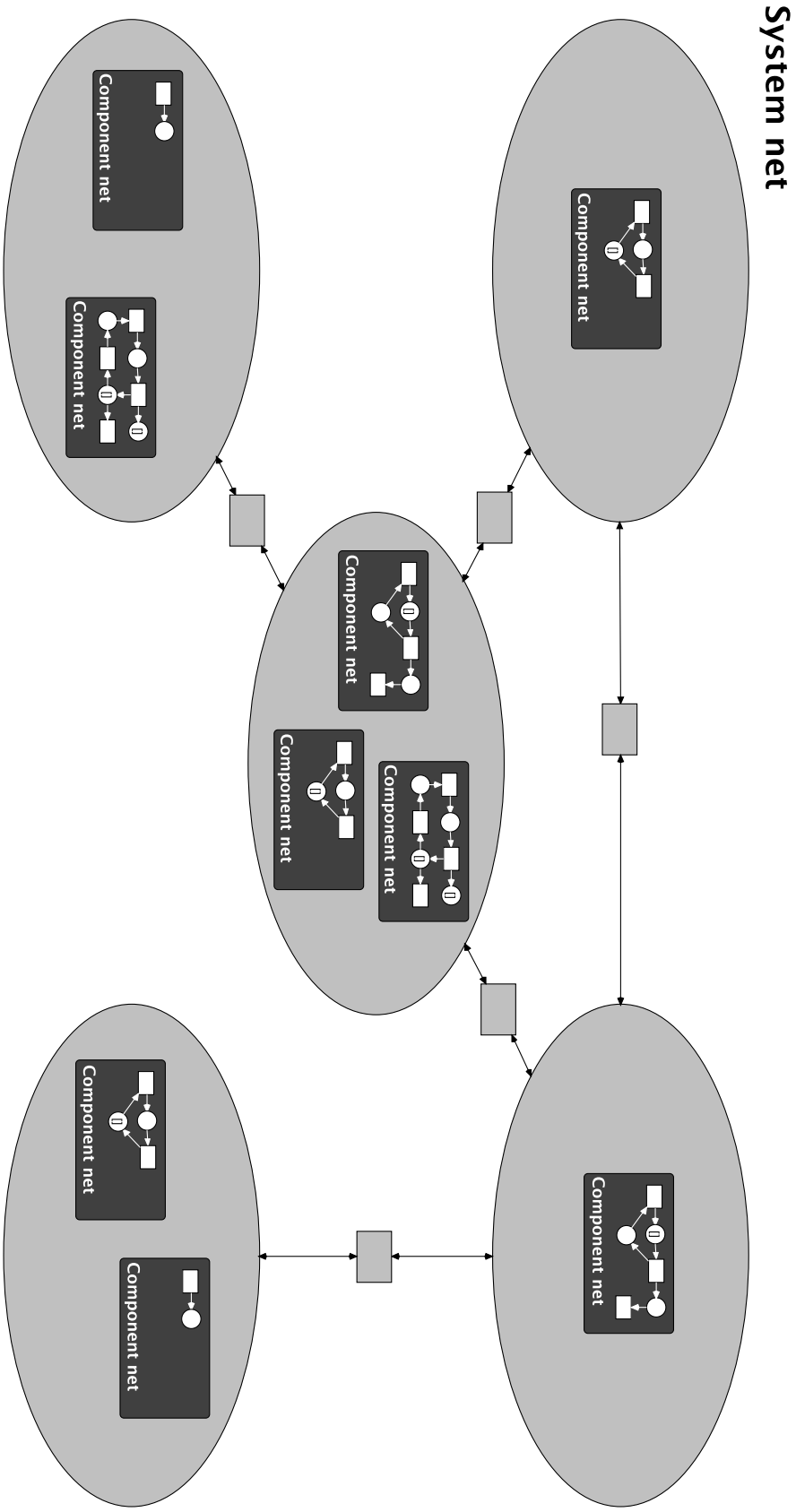

Figure 4.3: View on the structure of a software net system

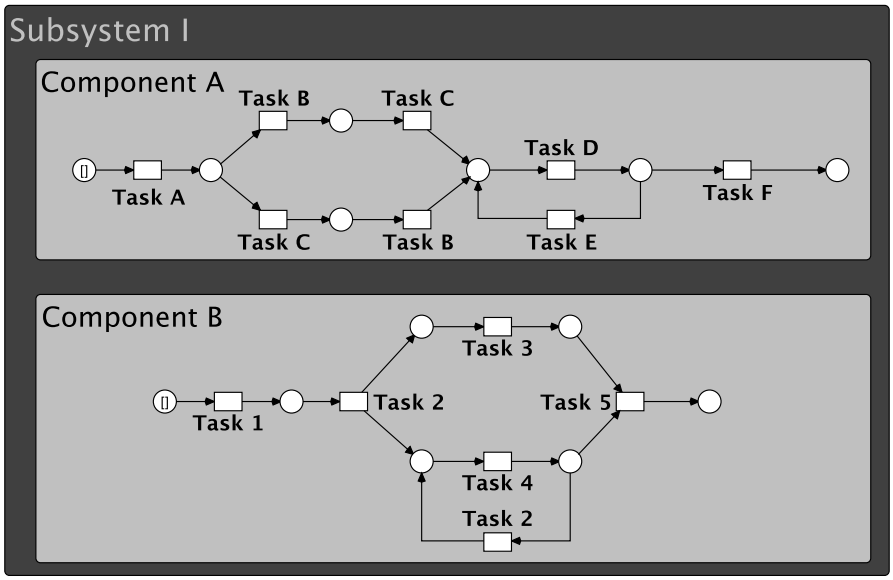

Figure 4.4: View on the behaviour of a software net system

than adding a level of abstraction to it. Components, infrastructure and organisation are simply abstractions of parts of the structure of the overall, flat net system with a specific application onto a software system. Consequently, these elements *are* the structure of the software system.

This view is illustrated in Figure 4.3[8]. The overall net (system net) represents the system as a whole. Components are nets (component nets) executed as tokens on the places of the system net. It should be noted that the system net may be just another component in an even larger system. The infrastructure of the system is defined by the places, transitions and arcs of the system net. Transitions between places determine if components on connected places can interact in some way (via the transition firing). Organisation is found in different ways in the system. Components are, for example, organised by their net structure or by their execution environments (i.e. places).

For the behaviour the same abstraction can be applied. In the net system the behaviour contains the possible firing sequences of that net system. By (logically) partitioning the net system's structure of nets into the software system's structure of components, infrastructure and organisation the firing sequences are now defined on these logical partitions of the system. This means that the behaviour of the software system is now the set of possible firing sequences (i.e. processes, see above) of all of its components and logical subsystems[9].

The view on behaviour is illustrated in Figure 4.4. The possible processes for two components A and B can be considered individually or combined as the processes of subsystem I, which contains the two components[10]. The processes contain the different tasks (i.e. units of work) the components execute, as well as their order. Extending this representation of processes to the entirety of a system yields the view its behaviour.

Figure 4.4 also calls attention to an important issue in this context. It illustrates the

[8]The figure represents a simplified and abstract view on a system. The figure only serves to illustrate the concept of structure of a software system in Petri net system terms.

[9]I.e. logical combination of different components.

[10]To keep the figure simple it is assumed that the processes of components A and B are independent from each other. If there were interactions between the two, the processes of subsystem I would have to represent these.

behaviour of components as a Petri net. That Petri net represents the process of the component. As a Petri net the process has its own structure. However, in this case the structure of the process is simply a representation and extract from the component's structure. The process' structure is irrelevant as its sole purpose is to *represent* the behaviour of the component. It is a logically partitioned view on the behaviour of the structural element.

The remaining part of net systems that has not been discussed yet in the context of software systems are the markings. In net systems the markings represent the state of the system and are part of both behaviour and structure. This does not generally change for the view on software systems. Markings still determine the state of the system as structure, while the changes in markings are part of the behaviour. This can be traced down to the components, in which markings represent the states of those nets/components.

One distinction needs to be made for tokens which are (or refer to) subnets. In the system net in which they are located these tokens represent the current state. This state, though, includes the component, its structure, its behaviour and its own state. In this way it is an abstraction of the component into just one part (token) of the state (marking) of the overall system net.

In conclusion, structure and behaviour of net systems can be mapped onto software system by following the general approach of Petri net based software engineering. How structure and behaviour are concretely characterised in systems is dependent on the specific kind of modelling technique used. This thesis focuses on agent and workflow-based modelling techniques. The specific modelling perspectives based on each technique are described in Chapters 5 and 6 for agents and workflows respectively. Concrete definitions for structure and behaviour are provided in those chapters.

5 Structure Based on Agents

The previous chapter established a general understanding of the term structure of a software system for this thesis. This current chapter consolidates this understanding with the modelling technique of agents and especially Mulan, Capa and Paose. It represents a refinement and specialisation of the basic understanding of structure and establishes a definition of structure, which takes agent-orientation in general and the Mulan agent model in particular into account. For the remainder of the thesis the resulting definition specifies structure and the structural perspective.

Four sections are included in this chapter. Section 5.1 examines the agent-oriented modelling perspective. Based on that perspective, Section 5.2 introduces the concept of fundamental agent actions. Section 5.3, then, uses the observations and conclusions from the previous sections to refine and specialise the general definition of structure introduced in Chapter 4. Finally, Section 5.4 concludes this chapter with a short discussion of agents as the chosen modelling construct for structure in this thesis.

5.1 The Agent-Oriented Modelling Perspective

Section 2.2 described what software agents are and how multi-agent systems work. The agent-oriented modelling perspective presented in this current section, is a creative and productive counterpart to the analytical view taken in that previous section. It doesn't describe how agents *work*, but rather uses the concepts related to agents to describe how to *approach* the creation and modelling of agent-oriented software systems. The perspective is abstract in nature and independent of the particular agent model or technology, since it builds on the fundamental parts of agent-orientation itself.

Note that the agent-oriented modelling perspective is not Paose (see Section 2.2.3) or any other agent-oriented modelling approach or methodology. Such approaches describe how to actually, practically model and engineer an agent system. The agent-oriented modelling perspective instead describes how to *approach* the actual modelling described in a methodology. It is a fundamental, abstract view on software systems and represents a motivation and inspiration for modelling software systems that facilitates the understanding of these systems during their creation. In other words, it is one possible way of thinking that can be applied to agent-oriented modelling.

All this does not mean that the perspective isn't part of some approaches. In fact, the perspective was devised in the context of and inspired by Paose research. The agent-oriented modelling perspective is implicitly inherent throughout the Paose approach, which is elaborated on in Section 5.1.4.

The agent-oriented modelling perspective serves an important purpose in this thesis. It is the adopted perspective for all concepts, models, aspects and prototypes that are either directly agents or based on agents. In combination with its counterpart of the workflow-based modelling perspective (see Chapter 6) it serves as the basis for the integrated modelling perspective (see Section 7.1.3).

The details of the agent-oriented modelling perspective are presented in the following four subsections. Section 5.1.1 presents the core idea of the perspective. Functionality

and its division onto agents are discussed in Section 5.1.2. Establishment of structure and behaviour are addressed in Section 5.1.3 Finally, Section 5.1.4 examines the perspective in the specific context of Mulan and Paose.

5.1.1 Everything is an Agent

In agent-orientation everything revolves around agents:

Agents are the main modelling construct: When modelling an agent system the first and foremost consideration is given to the agents. The agents provide the basis for and are at the centre of all modelling and design decisions.

Agents are the main partitioning construct: While observing the agent system (e.g. for monitoring or analysis and also during engineering) the agents are the main abstraction for partitioning the system for observers.

Agents are the main relational construct: When considering anything in an agent system it is always related to an agent. Knowledge, behaviour, communication, distribution etc. are always only considered in relation to the affected or observed agents.

Agents are the main functional construct: Functionality in agent systems is completely encapsulated by individual or sets of agents. Agents may use functionality provided by external elements (e.g. databases) but within the agent system these elements are only passive and do not act by themselves.

Agents are the main intelligence construct: Not just pure functionality is provided by agents in agent systems. Agents are also the main focus when system intelligence is added. This relates to both user and system support through proactive behaviour in agents.

These and more areas, such as scalability, research or communication, are all inherently focussed on and around the concept of software agents. The agent-oriented modelling perspective builds on this strong focus. At the abstract core of the perspective is one concise, clear and somewhat provocative statement:

Everything is an Agent

This statement reveals the general way of thinking about elements of the system. It assumes a purely logical and virtual system partition by abstracting from implementation details. In that view every element is considered as an agent. Interactions between the agents and the division of system functionality (see next subsection) become the main focus of attention.

Definition A.1 from Section 2.2 defines an agent as an autonomous system in some environment that is capable of autonomous action in that environment to meet its objectives. In other words, it is an autonomous entity in its environment which provides a certain functionality and interacts with its environment. That environment consists of other entities, which are all regarded as agents, since *Everything is an Agent* applies the abstract and conceptual view of autonomous entities to all elements of the system. Later on, during implementation, these elements can exhibit the concrete properties, characteristics and functions of specific agent models, yet those design decisions happen at a later stage.

In the view promoted by the agent-oriented modelling perspective, even the entire software system is considered as an agent. That agent is executed in some environment (on a computer) and is interacted with (used) by other agents, in this case human users or

other interacting systems. Since a (non-monolithic) system contains multiple elements, an agent representing a system also contains more agents. These agents, as subsystem, may then again contain other agents. This creates a hierarchy of agents containing agents and interacting with one another crossing multiple levels of the hierarchy.

Such a hierarchy has been considered before in [Jennings, 2000]. That article states that "*[...] at one level, entire subsystems can be viewed as singletons, alternatively, teams or collections of agents can be viewed as primitive components, and so on until the system eventually bottoms out.*" (adapted) [Jennings, 2000, p. 11] Here, the focus is on an abstraction onto primitive components which is in the current case provided by the concept of agent itself.

[Sterling and Taveter, 2009] also considers the notion of agents in a broader sense then just practically implemented software agents. It does not go as far as to regard every element of a system as an agent, but does consider the overall system, human users and people in general as agents. This is justified by focussing on an agent's awareness of its environment and its ability to react to changes in that environment.

[Shoham, 1989, Shoham, 1993] are early contributions in the context of agent-orientation and examine what kind of elements to consider as agents. It is stated that "*agenthood is in the mind of the programmer: What makes any hardware or software component an agent is precisely the fact that one has chosen to analyze and control it in these mental terms.*" [Shoham, 1993, p. 52]. This idea aligns well with *Everything is an Agent*, which represents the choice to analyse every element of a system in agent terms. To illustrate the point both [Shoham, 1989, Shoham, 1993] use the example of a light switch, which can be considered as an agent, albeit a "*very cooperative*" [Shoham, 1993, p. 53] one.

A similar approach to the agent-oriented modelling perspective is called *Agent-Based Modeling* in [Wilensky and Rand, 2015]. "*The core idea of Agent-Based Modeling is that many (if not most) phenomena in the world can be effectively modeled with agents, an environment, and a description agent-agent and agent-environment interactions.*" [Wilensky and Rand, 2015, p. 32] The core idea here matches well with the idea of *Everything is an Agent* in the agent-oriented modelling perspective. However, it has a more limited scope (only considering many or most elements) and is anchored on a much more concrete level as opposed to the abstract and conceptual level of *Everything is an Agent* and the overall agent-oriented modelling perspective.

The basic notion of considering everything as an agent can also be found in the original development of Mulan. [Rölke, 2004] includes some mentions of the general idea in the descriptions of the agent model and reference architecture. This is picked up in Section 5.1.4.

Again, it should be clarified that the statement *Everything is an Agent* is only an abstract way of consideration for the system and is not concerned with any practical modelling or implementation issues. In fact, it rather means that everything is a conceptually abstract agent, detached from any specific agent models, properties or other technical characteristics. Nonetheless, it also means that while everything *is* only an abstract agent, everything (in the end product of the system) *may be* a practically implemented agent. The core idea only influences modelling and design decisions, but does not enforce anything.

If everything is considered as an agent, it poses the question of how agent properties and capabilities are regarded in this view. The answer is that not all agents need to have the same capabilities and properties. Different needs of elements within the system require agents representing them to do or be capable of different things.

Take, For example, an execution environment (e.g. an agent platform in Mulan), which is an element of any system that needs to be explicitly acknowledged. It has to have

the capabilities of a container for other elements/agents and would need to be able to create, destroy, migrate and manage them as well. If considered as an agent, the execution environment would substantially differ from other agents of the system which represent elements on wholly other levels of execution.

Considering the agent execution environments or platforms as agents has been done before in the context of nested multi-agent systems. Recursive multi-agent systems [Giret and Botti, 2003] define a multi-agent system as consisting of so-called A-Agents which, in short, can be either atomic agents or other multi-agent systems. Holonic multi-agent systems [Fischer et al., 2003] consider holons of agents that appear and interact with one another as individual, atomic agents would. Ideas of holonic multi-agent systems are also found in some research in the particular context of Mulan and Capa, which is discussed in Section 5.1.4.

Agents representing execution environments are very complex with comprehensive functionality and extensive interactions. On the other end of the spectrum of *Everything is an Agent* lie the very simple and primitive elements of the system. For example, data documents or even simple data types can also be considered as agents. These agents would only store some amount of data in a specific form that can be read and modified by other agents. Consequently, they would be very simple agents with limited functionality.

Regardless of their complexity the consideration of elements as agents provides a number of benefits to the modellers. When all elements are considered as agents they share an abstract and uniform basis for the modellers to inspect their role in the system. Relationships between the different elements, even on wholly different levels of execution, can be examined more easily and consistently due to that shared basis. Overall, these general relationships and the specific interactions included in them also become more explicit. Modellers now need to actively think about and model them, which facilitates discovering errors and inefficiencies that could otherwise be lost in implicit assumptions. Finally, as described in Section 2.2.1, agents can exhibit a wide range of properties, including intelligence, mobility and adaptability. While not every element in the system needs these kinds of properties, considering the elements as agents with the *possibility* of these properties may lead to new ideas to solving specific modelling problems. Associations invoked by the consideration of elements as agents imbue these elements with implicit assumptions about their possible properties. For example, the system may have a data storage element that contains data which is needed in different execution environments at different times. This could classically be handled by a database, but, considered as an agent, mobility and the asynchronous communication may be utilised for this task. All of these effects can positively influence the design decisions of for agents systems.

On the other hand, *Everything is an Agent* also faces some issues. When considering every element within a system as an agent it becomes difficult to maintain the ideas of autonomy and interaction for every element in this abstraction. If, for example, simple elements, like passive data repositories, documents or simple data types, are indeed considered as agents it is difficult to realistically see much, if any, autonomy in them. Interactions with these simple elements are also extremely simplified to call/respond patterns. These are the cases which, according to [Shoham, 1993], can be considered as agents but may not be advantageous as such. However, as stated before, the agent-oriented modelling perspective represents a purely logical and conceptual view. Simple entities do not need to exhibit autonomy, complex interactions or other advanced properties of agents in any *practical* system. They may nonetheless for the sake of an *abstract* view be considered as autonomous agents that are simply "*very cooperative*" [Shoham, 1993, p. 53] and allow all changes to them. In the worst case, they are considered as more complex than they

actually need to be. In the best case, modellers realise that, for the specific purposes of a given system, certain usually simple elements may benefit from advanced agent features.

Another issue relates to the conceptual overkill and the practicability of the idea of *Everything is an Agent*. The benefits of the idea have been discussed above. To fully utilise these benefits a system needs to be sufficiently large. In small-scale, manageable systems it may simply not be necessary as modellers would probably be able to grasp and capture the intricacies of the system without them. For such cases explicitly applying the idea of *Everything is an Agent* may be impracticable overkill. However, in many cases it is enough to merely keep the idea in mind and apply it only to solve specific problems in the system. Once again, the idea is not about practical implementation but about abstract consideration of problems.

5.1.2 Functionality in Agent Systems

Functionality of an agent system is an important aspect of the agent-oriented modelling perspective. Every software system usually has a specific purpose or goal that should be achieved. This obviously also applies to a multi-agent system. The functionality of the system is the set of processes, mechanisms, capabilities and properties aimed at fulfilling the system's purpose. In short, the functionality of a system is its (correct) behaviour (see Section 4.3).

The particularity in multi-agent systems is that they consist of individual agents. This means that the purpose of the system is distributed among these agents during modelling. Each agent in the system has a specific purpose, a delegated objective, and needs to provide a specific functionality to fulfil that purpose.

At this point the agent-oriented modelling perspective begins to transition from the purely logical, abstract and complete view of *Everything is an Agent* to a more restricted yet more practically grounded view. Instead of considering all elements of the system, it now only considers elements that actively provide functionality[1]. Actively providing functionality means that the elements control and perform their own actions to fulfil their purpose within the overall system. In other words, in the context of functionality the agent-oriented modelling perspective emphasises *autonomy* in agents.

The counterpart to the actively provided functionality are the passive elements of the system. Passivity in this context denotes that these elements are utilised in the context of the behaviour of the other (active) elements. They can still provide important functionality but simply don't act on their own as the active (autonomous) agents can do. It is very important to note that the distinction between active and passive is not made based on the functionality itself, but on the way the element provides it. For example, a database might actively or passively provide its functionality. Actively, it could prepare common query results in advance or proactively administrate and clean/maintain/update its dataset. Passively, it would only perform these actions at the request of other agents, never acting alone on its own impulses. In both cases the basic functionality of the database remains the same. Yet, in one case it would be considered as an autonomous agent while in the other it would be more akin to a classical object (cf. "*Objects do it for free, agents do it because they want to.*" [Wooldridge, 2009, p. 29] and the corresponding discussion in Section 2.2.1).

If the purpose and functionality of the overall system are distributed among its different elements (agents), the interactions between these agents become even more important.

[1] *Everything is an Agent* is still in effect for the general view on the system. The restriction only applies to considering functionality. Modellers can also add and remove elements to the consideration of functionality at any time by determining their properties and role in the general, abstract view.

The functionality of the individual agents alone is not enough to achieve the desired overall purpose, except in trivial cases. Agents need to interact and cooperate with one another, creating an emergent behaviour that is greater than simply the sum of its parts. Emergence is "*the arising of novel and coherent structure, patterns, and properties through the interactions of multiple distributed elements*" [Wilensky and Rand, 2015, p. 6]. The combination of individual agent functionality/behaviour and the functionality emergent from the interactions of those agents fulfil the functionality requirements set by the intended overall purpose of a system.

The importance of interaction for the functionality of an agent system aligns with the agents' prescribed explicitness introduced by the idea of *Everything is an Agent*. As interactions are so important in agent systems the beneficial property of making them explicit through *Everything is an Agent* becomes even more distinct. Vital details lost in implicit assumptions about the interactions can critically impair the functionality of the overall system.

Interaction can be every kind of exchange of data between agents, from simple call/respond patterns to complex negotiations. At this point the agent-oriented modelling perspective takes another step closer to the practical aspects. It generally assumes inter-agent communication to be handled via asynchronous messages. If other communication types (e.g. synchronous communication) are required for specific purposes in the system these need to be handled separately as exemptions[2]. Other Technical details of the interaction are generally not important at this point (e.g. form of messages, FIPA compliance).

Of course, correctly specified interactions are useless without the internal functionality of the agent performing its purpose correctly. Internal functionality generally consists of performing actions on knowledge and data available to the agent. Sequence, order and conditions for those actions have to be defined correctly in order to achieve the desired purpose. The knowledge and data for those actions is provided to the agent either at modelling time or through the interactions with other agents.

Combining the correct specifications for the internal functionality and the interactions with other agents specifies the entire functionality for an individual agent. Agents need to process and work on their own data and knowledge and need to share the results with other agents that can utilise them further. Through the combination of internal functionality and interaction the agent can achieve its purpose in the system (cf. the distinction between internal and external events/operations in [Moldt, 1996]). If such a combination is provided to all agents in the system, the system's overall purpose can be achieved.

5.1.3 Structure and Behaviour

Structure and behaviour were identified in Chapter 4 as the two main aspects of any software system. Structure included the components, infrastructure and organisation of the system, while behaviour included all processes of and between the components in the system. Naturally, structure and behaviour also need to be covered in the agent-oriented modelling perspective. This subsection examines where and how structure and behaviour are included in that perspective.

The two main areas of the agent-oriented modelling perspective were discussed in the previous subsections. On the one hand, the perspective includes the consideration of every element as a (possible) agent, captured in the idea of *Everything is an Agent*. On the other

[2] If asynchronous communication does not appear suitable for the needs of the current system, modellers may need to reevaluate the choice of the agent concept.

hand, the perspective partitions the functionality onto the set of active elements (agents) which perform internal actions and interact to achieve their purpose.

Structure can be found in the first area. Considering everything as an agent represents a structural abstraction on the elements of the system. The components, i.e. agents, are treated uniformly and can thus be compared and set in relation to one another. This comparison/relation between the different agents also provides the organisation within the structure. Every agent has a distinct set of connections to other agents which determines its part in the organisation of the system. In fact, since components such as execution environments or other container-components are considered as agents as well, certain hierarchies begin to emerge in the system. Hierarchies also play an important part in the depiction of the infrastructure in the agent-oriented modelling perspective. Infrastructure is provided by specific agents and specific parts of the overall organisation of the system. Agents providing infrastructure are exactly those that factor into the hierarchical aspects, namely the containers of other agents. To complete the view on infrastructure the relations and connections between these container-agents and other agents need to be taken into consideration. This includes relations between the different container-agents, between the container-agents and the agents contained within them and between the container-agents and the (more abstract) container-agents they themselves are contained in.

Behaviour of the system can be found in the functionality and purpose. An agent system's purpose determines what functionality it has to provide. The functionality of an agent system in the agent-oriented modelling perspective consists of the internal functionality of each agent and the interactions between the agents. Internal functionality describes how, when and in what order an agent performs actions on its internal data and knowledge. Interactions describe how, when and in what form the agent exchanges data/knowledge with which other agents. Combined, both parts describe interconnected processes within an agent, which irregularly alternate between exchanges of data/knowledge and the execution of some internal actions. The set of all of these possible processes for one agent describes the behaviour of that one agent. Considering the individual behaviour of all agents combined yields a set of larger and more complex, composite processes. In these composite processes the agents are connected through the interacting parts of their own processes, assuming every message sent by an agent is received by another one. The set of composite processes describes the behaviour of the overall system.

In summary, the idea of *Everything is an Agent* is used in the agent-oriented modelling perspective to provide a structural abstraction that helps modellers capture the different facets of the structure within the system. Based on that abstraction, the purpose and functionality of the system are distributed in the system. Each agent has a certain purpose and provides a certain functionality which consists of internal and external parts. The resulting processes combined over all agents then describe the behaviour of the system.

As a consequence, the agent-oriented modelling perspective and the idea of *Everything is an Agent* can be regarded as organising their behaviour, i.e. processes and functionality, in a structural way. This leads back to the main motivation, since the structure clearly strongly influences and governs the behaviour, which is something the desired integration aims to address.

5.1.4 The Mulan and Paose Modelling Perspective

This section examines the agent-oriented modelling perspective in the specific context of Mulan and Paose (see Section 2.2.3). Using the agent-oriented modelling perspective with Mulan agents is quite natural. Historically, the idea of considering every element of a system as an agent has already influenced the original development of Mulan. At

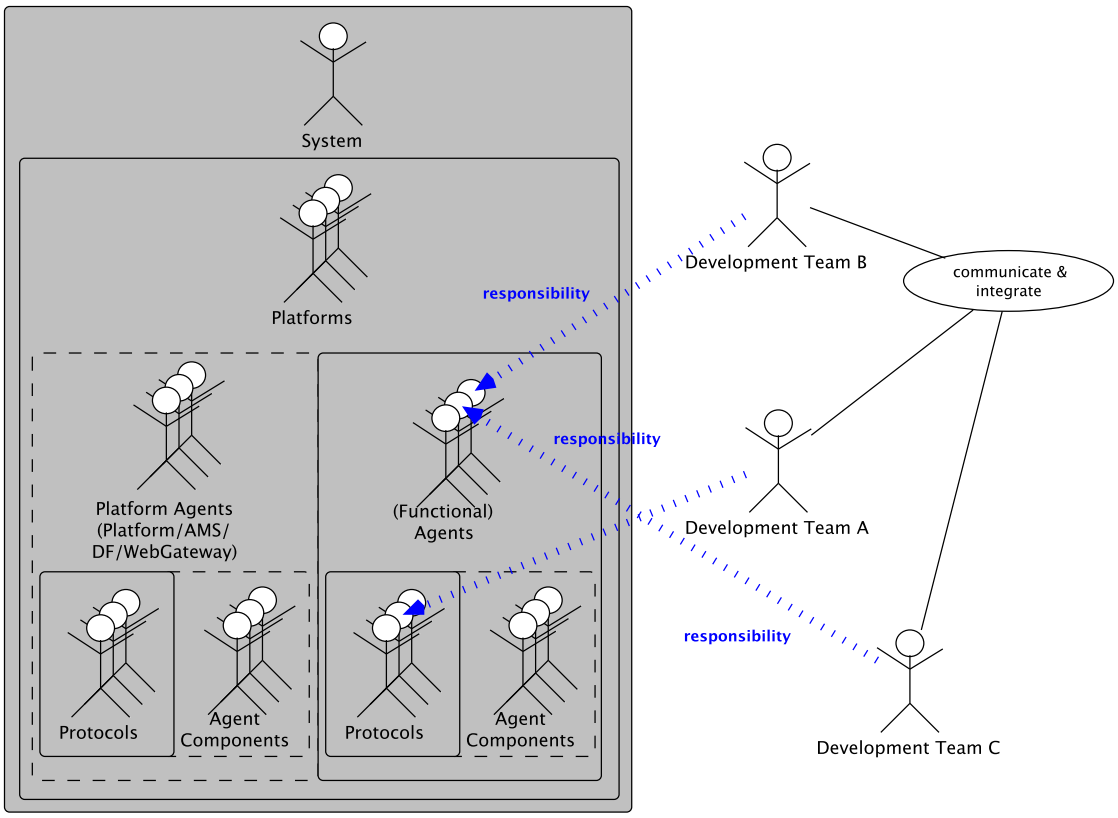

Figure 5.1: *Everything is an Agent* in Mulan and Paose

various points, [Rölke, 2004] deliberates on the idea of considering every element of the reference architecture as agents. In fact, [Rölke, 2004, pp. 180ff] examines an extension to Mulan in which the platform is implemented as an agent. The Capa implementation of Mulan actually implements a platform agent [Duvigneau, 2002, pp. 93ff] that represents the platform and provides the required functionality to other agents. It should be noted that the platform agent does not use the same agent net as standard agents, although the basic structure defined in the Mulan model is present. Further platform functionality is also implemented in and provided by a number of (standardised) agents, e.g. AMS, DF and WebGateway (see Section 2.2.3).

The research featured in [Schleinzer, 2007, Schleinzer et al., 2008] is strongly related to this and uses ideas from holonic multi-agent systems as seen in, e.g., [Fischer et al., 2003] by providing so-called plugin-agents. Plugin-agents act as plugins for other agents to expand their functionality. Consequently some agents serve as platforms and execution environments to other (plugin) agents, which the platforms use to expand their own functionality. The view adopted here is quite similar to *Everything is an Agent* since the functionality of an agent can be altered by the inclusion or exclusion of the plugin-agents it contains. This means that, conceptually, functionality and therefore behaviour is realised by agents.

Ideas in line with *Everything is an Agent* can also be found in the Paose approach. Paose uses multi-agent systems as a guiding metaphor for the development of systems [Cabac, 2010, p. 120ff.]. It considers not only the system in development as a multi-agent system, but also the team developing it. Each development sub-team, i.e. one or more developers, is regarded as an agent that has certain responsibilities (i.e. a purpose to fulfil,

see Section 5.1.2). Responsibilities define which component/part of the system (e.g. role, interaction, ontology, tool) a sub-team is in charge of. Sub-teams need to develop their component/part internally (i.e. the sub-team-agent's internal behaviour) and interact with other sub-teams to integrate their components into the whole of the system (i.e. the sub-team-agent's external interactions). This is illustrated on the right-hand side of Figure 5.1. Figure 5.1 is only a simplification of the process to show that the development teams need to interact themselves in order to integrate their responsibilities in the system. For more information on the use of multi-agent system as a guiding metaphor in PAOSE see [Cabac, 2007] and the short discussion in Section 2.2.3.

Moving away from the very abstract level of the agent-oriented modelling perspective and applying the concrete MULAN agent model provides additional insight into the perspective. The MULAN reference architecture features four levels of abstraction: system, platform, agent and protocol. Considering every element in these levels as an agent is possible. Overall, applying *Everything is an Agent* in the MULAN agent model yields a nested view illustrated in Figure 5.1

For agents this consideration is trivial. However, there are agent-internal components, like the knowledge base and the protocol factory, that can be considered as agents within the agent. They have a distinct interface through which they interact with the outside agent, fulfil a purpose in their (agent) environment and perform internal behaviour. Figure 5.1 sees these components on the same level as the protocols.

Likewise, the platform layer can also be regarded as an agent that contains other agents. As described above, this consideration has been intended historically and can also be found in the CAPA implementation. Agents may be either functional, i.e. developed for a specific, custom application, or default for the platform providing default functionality.

The remaining two levels, system and protocols, are more interesting in this context. Considering the system as an agent works mostly like considering the platforms. The system contains agents (platforms) that contain agents (agents) that contain agents (protocols). For the system the environment is more interesting, because it doesn't have an explicit one. Every other agent in the system is contained in at least one more level of agents. Only the system is not nested in an explicitly modelled environment. Of course, there is an environment around the system. It contains the deployment site, the users and other systems (e.g. operating system of its computer) it interacts with. In fact, the development teams are also part of this implicit environment as indicated in Figure 5.1. However, none of these elements are explicitly modelled and, for the purposes of the agent-oriented modelling perspective, do not exist. They do affect the modelling through requirements (e.g. technical, practical, functional), but they are nonetheless considered as non-existent. Being aware of and capturing these requirements coming from an otherwise implicit source is one of the challenges of modelling. A view of systems as agents is in line with the view taken in [Sterling and Taveter, 2009].

Regarding protocols as agents is also quite interesting. Unless a modeller chooses to (conceptually) regard data and other artefacts as agents, the protocols are the only agents in the *Everything is an Agent* MULAN that do not contain other agents[3]. This means they are a fixed end-point in the modelling. At this point the nesting stops, which, when considering an integration of agents and workflows, creates a fixed point at which to approach such an integration.

[3]Considering data or other artefacts in protocols as agents conforms with the view of *Everything is an Agent*. Still, the current discussion is associated with concrete MULAN systems and assumes a practical discussion level. The limitations regarding autonomy of simple elements as agents discussed in Section 5.1.1 strongly apply here. Consequently, for these practical reasons data and artefacts are not considered as agents on this level and at this point.

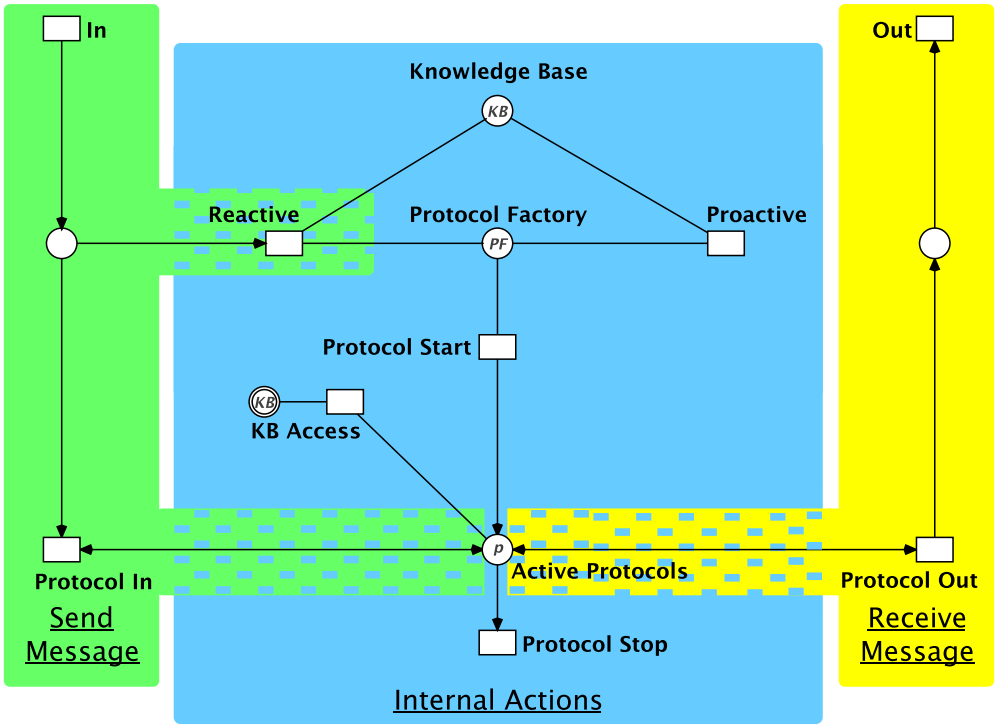

Figure 5.2: Basic agent actions in the standard MULAN agent net (based on [Rölke, 2004, p. 169])

Additionally, regarding protocols as agents results in relatively simple agents. A protocol in MULAN describes one part of the behaviour of one MULAN-level agent. Considering the protocol as an agent itself distils and encapsulates this individual behaviour, which creates a very easy-to-understand agent that has a very obvious and limited behaviour (i.e. the original protocol). A common basis for comparison and relation is created, which supports modellers in evaluating the role of the protocol within the agent and also the overall system.

5.2 Fundamental Mulan Agent Actions

As discussed in Section 5.1.2, the functionality of one agent consists of its internal functionality combined with the interactions it has with other agents. Under this consideration agent behaviour can be classified into three fundamental agent actions: *Internal actions*, *receiving messages*, *sending messages*. These actions are denoted as fundamental as they cover the *entirety* of agent behaviour. Even complex behaviour like decision making and inference can be captured in them. The topic of complex behaviour is addressed at the end of this section.

Regarding context, the descriptions here assume MULAN or CAPA agents, including the FIPA-compliant asynchronous communication. Figure 5.2 illustrates where in the standard MULAN agent net these actions can be found. Applicability and transferability to other agent models are discussed in Section 12.2.

Send Message Sending a message is one half of what an agent can do to interact with other agents. Messages may e.g. be sent to inform other agents of processed results, to

send requests for other agents to do certain actions or accepting or rejecting a request from another agent. In fact, the FIPA standards define 22 different communicative acts for agents (see [FIPA37H, 2002] and Section 2.2.2) which describe the most basic forms and domains for agents sending messages.

Looking at the MULAN agent model it is easy to identify sending a message as a fundamental agent action. In the standard agent net (see Figure 5.2 and Section 2.2.3) sending a message is realised in the right-hand branch of the net (set before a yellow background). This is the only part of the agent that can directly interact with the outside environment via outgoing communication. As such it represents one of the two interfaces the agent has.

The fundamental agent action *send message* describes the abstract activity[4] of an agent that is sending a message. This may conceptually involve the entire (miniature) process of (locally) looking up the recipient, creating the message, preparing it for sending and actually, technically sending it out. However, depending on the implementation some parts of that miniature process may be considered as internal actions or be captured by standardised mechanisms.

In the basic MULAN agent model seen in Figure 5.2 two transitions correspond to the *send message* action: the *out* and *prt_out* transitions. The latter receives the message from a protocol while the former actually sends it out by synchronising with the platform environment. As indicated in Figure 5.2, some parts of the *send message* action may also be found in the protocols. This depends on how the specific modeller chooses to recognise certain actions. Looking up the identifier of a recipient or creating a message are usually done in the protocol in MULAN. While these actions are agent internal they may arguably also be considered as part of the *send message* action. The transition[5] between internal actions and sending a message is fluent here and depends on the view of the particular modeller.

Receive Message Receiving a message is the other half of what an agent can do to interact with other agents. As every message sent by an agent should be received by another agent at some point the examples and classification into the FIPA communicative acts described for sending a message are applicable here as well. The only difference is that the reception of a message is purely reactive.

Considering the MULAN agent model the identification of the *receive message* fundamental agent action is almost mirror-inverted to the *send message* agent action. Reception of messages is handled in the left-hand side of the standard MULAN agent net in Figure 5.2 (set before a green background). This is the only part of the agent that directly interacts with the outside environment via incoming communication. As such it represents the second of the two interfaces the agent has.

Similar to sending a message, the *receive message* agent action also describes an abstract activity of an agent. The reception of a message involves different handling though. In general the reception involves the entire (miniature) process of technically receiving the message and passing it to the part of the agent that has to work with it. But again, some of the latter steps may also be considered as internal actions.

Looking at the MULAN agent in Figure 5.2 there are two transitions *in* and *prt_in* that are completely part of the *receive message* action. In contrast to sending a message though, there are now two additional areas where parts of the reception action can be considered

[4] The general notion of the word activity is meant here, not the workflow terminology given in Definition A.6 from Section 2.3.1.

[5] Meant as the crossover from one area to another, not *transition* in the Petri net terminology.

as internal actions. The first area concerns protocols. Passing the message to protocols and unpacking any data may be considered as part of the *receive message* action. However, in MULAN any processing of the message is usually considered as an internal action and part of the protocols. The second area concerns reactive behaviour. Some messages trigger reactive protocols to be started in the agent. Starting these protocols can arguably be considered as part of the reception of the message as it is a direct consequence. However, these parts of the abstract action are also usually considered as internal or administrative actions.

As with sending a message the particular modeller decides what is internal and what is part of receiving a message. In general the clear separation of these transitional areas is not crucial. However, later on in this thesis the fundamental agent actions are heavily utilised. At that point the separation is clearly made for the specific modelling purpose.

Internal Action The final type of fundamental agent action concerns the internal actions. Basically everything an agent does that is not explicitly concerned with interacting with other agents (i.e. sending and receiving messages) falls into this category. Data processing, decision making, accessing knowledge etc. are all internal actions.

In the MULAN agent model of Figure 5.2 the central transitions and places are involved with internal actions. These are set before a blue background in Figure 5.2. The transitional areas between internal actions and sending/receiving a message have been discussed in those respective paragraphs and won't be discussed here further. For the most part, internal actions happen within the protocols. Working on data, executing specific parts of the functionality (in MULAN executing Java and reference net code inscribed to the transitions of the protocol net) or communicating with other internal components of the agent (e.g. knowledge base, protocol factory) are just some of the examples of internal actions in protocols. Other internal actions concern the management of the agent, e.g. starting (proactive) protocols, startup or shutdown of the agent.

Looking at the CAPA implementation of MULAN another area for internal actions are the decision components (see Section 2.2.3). These completely internal behaviours receive data from protocols or the knowledge base and work on it without directly interacting with anything outside of the agent. In this way they can be considered as protocols featuring exclusively internal actions. If results or data generated within a decision component need to be sent to other agents the decision component communicates with or possibly even starts a new protocol to handle the communication. Figure 5.3 illustrates the view on decision components as purely internal actions.

Another way of considering decision components is to apply the idea of *Everything is an Agent* at this point and regard decision components as agents within the agent. This is illustrated in Figure 5.4 where the decision component area of the MULAN/CAPA agent is considered as a separate (though contained) agent. The internal decision component agent distinguishes between the three different actions like the overall agent. Internal actions happen in the central place with the blue background. Communication with protocols, other decision components, the knowledge base, etc. happen via the green and yellow areas (a transition in both green and yellow indicates bidirectional communication). Therefore, the general agent actions are still valid in that case. However, as the communication in CAPA is handled via synchronous channel the *send message* and *receive message* actions become more conceptual and describe which of the components/agents initiates the exchange of data. Analogous arguments can also be made for other agent-internal components like the knowledge base and protocol factory. These particular views, though, are not pursued further at this point in the thesis.

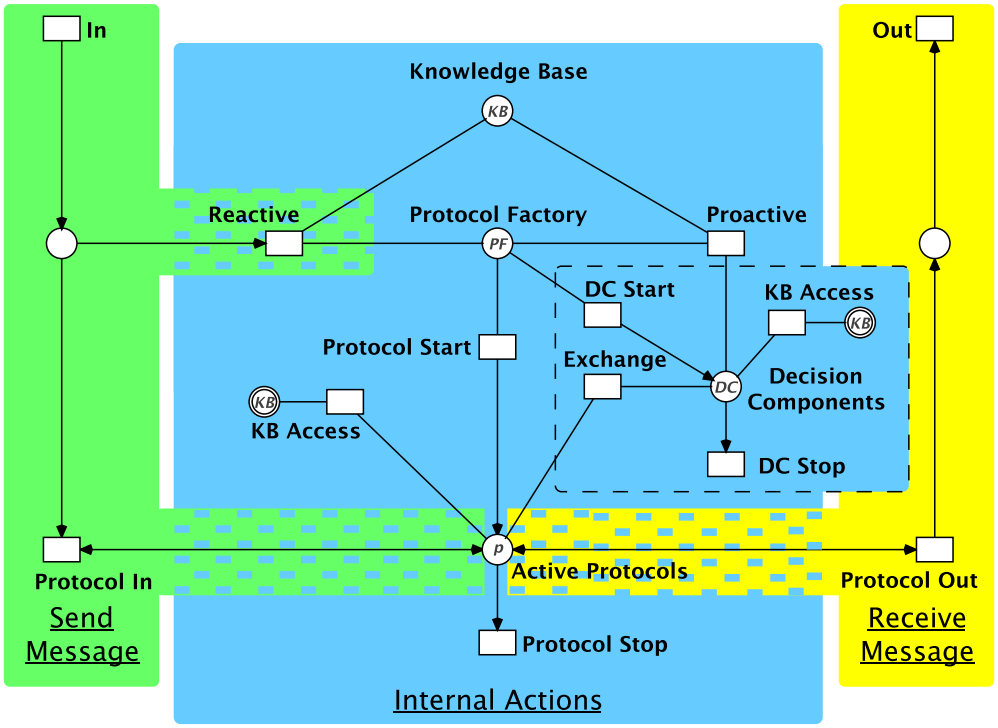

Figure 5.3: Basic agent actions in the extended MULAN agent net (based on [Cabac, 2010, p. 51])

Conclusion After having characterised and examined the extent of the fundamental agent actions in the previous paragraphs an important question remains: Do these actions actually cover all agent behaviour? It can be argued that complex, advanced behaviour is not included in the categorisation of fundamental agent actions. However, this is only true if the advanced behaviour is considered too concretely and atomically.

For instance, plan deliberation in BDI agents (see Section 3.1) is a special kind of internal decision making action. It may involve multiple steps, but these steps all happen within one agent, which make these steps internal agent actions. Even considering collaborative goal deliberation of groups of agents only adds further actions of sending and receiving messages and internally processing the results. The fundamental actions can also be applied to agent platforms considered as agents. The perspective changes slightly in this context, mirroring the view described above of decision components as internal agents.The agents of a platform can either be seen as fully internal components of the platform with only internal actions, or they can be regarded as individual agents. In the latter case, the internal interfaces of the platform become interfaces for sending and receiving messages enabling the distinction between all three agent actions. This means that agent management, from the platform side, can be regarded in the same terms as agent execution. From the individual agent side, agent management can be regarded in terms of fundamental agent actions as well. Instantiation, termination or migration of agents involve multiple steps in both the platform and agent in question. This includes all three types of actions. Internally, both the platform and agent in question need to make decisions and process data about their states. Regarding the exchange of messages the agent in question communicates with the platform (as an agent) to inform or request the specific management activities and receive confirmations and instructions about the process.

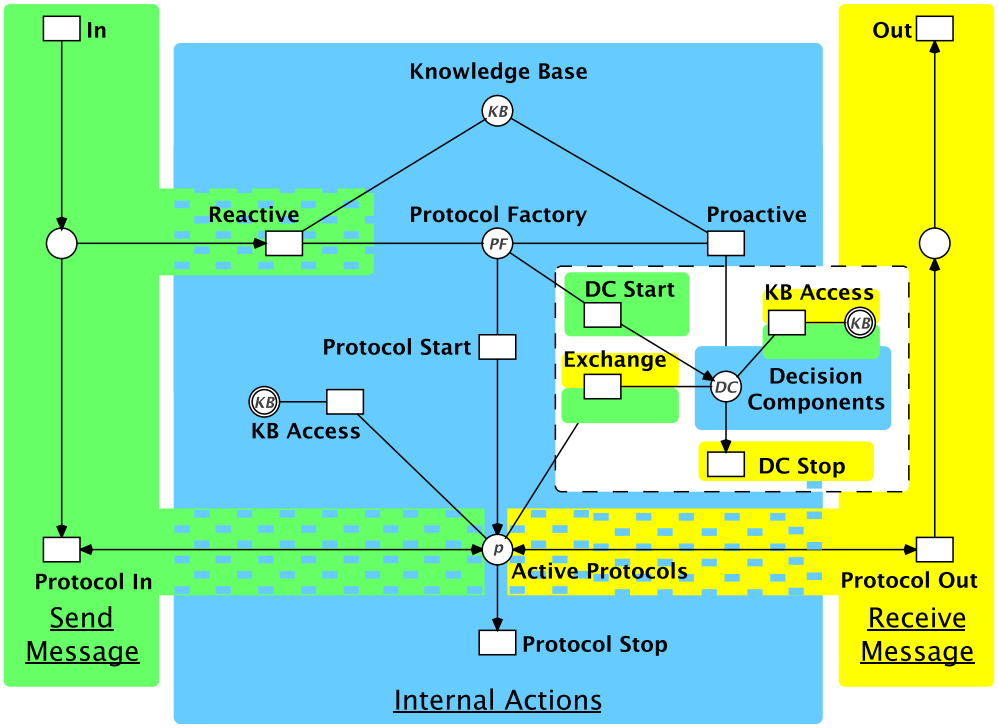

Figure 5.4: Decision components as agents in the extended MULAN agent net (based on [Cabac, 2010, p. 51])

Analogous arguments can be made for all complex types of agent behaviour. Fundamental agent actions describe the smallest units of works within the behaviour of agents. Complex agent behaviour may require multiple actions from multiple involved agents. However, such behaviour can always be broken down into individual steps, which can be mapped onto the three fundamental agent actions. This means that, in order to consider complex agent behaviour in terms of fundamental agent actions, only the scope of examination must be extended. In which dimension the scope is extended is fully dependent on the specific context. Abstraction level, involved agents and number of steps are just some of the possible dimensions of scope. But since the categorisation provided by the fundamental agent actions is logical and analytical in nature, the scope can be freely chosen.

Still, in some cases it may be cumbersome to use the set of fundamental agent actions as described here, leading to difficulties for modellers. This means that applying the fundamental agent actions to other agent architectures is possible, but it may be beneficial to examine the specifics of an agent architecture and determine if extensions to the fundamental agent actions are appropriate for efficiency or complexity reasons. Still, for the purposes of the specific context of this thesis, which takes Petri-net based agents as a basis for structural aspects of software systems, these actions are completely suitable.

Altogether, this section can be summarised in the following key term definition:

Key Term Definition B.1 (Fundamental Agent Actions)**.** The three fundamental agent actions are: receiving a message, sending a message and performing an internal action. These three agent actions can fully describe the behaviour of an agent.

5.3 Structure through (Mulan) Agents

This section combines the previous results of this chapter into a refined and concise definition of the structure of a software system based on the agent-oriented modelling perspective and MULAN agents in particular. The definition serves as the basis for any further examinations of structure and the structural parts of the integration examined and developed in the remainder of the thesis.

Chapter 4.3 discussed the structure of a software system as an abstraction of Petri net constructs onto components, infrastructure and organisation. Considering the MULAN agent model, such an abstraction is, due to the Petri net nature of MULAN, directly available and points directly at the different levels of the MULAN reference architecture.

Explicit components of the system are agent and platform nets. These components provide the standard (platform) and application-specific (agent) functionality of the system. Considering protocols as components is possible under *Everything is an Agent* (see Section 5.1.4), however at this point this particular viewpoint is not taken. Protocols, for now, describe the functionality the agents perform and are consequently (part of) the behaviour. The role of protocols and a complete view at *Everything is an Agent* are picked up again later in this section.

Infrastructure is provided by the platform and system nets. Platforms provide the "local" infrastructure for the agents. They manage life-cycles and communication. A system net on the other hand provides the global infrastructure as it defines which platforms can interact in which ways.

Lastly, organisation is provided by the relations between components. Therefore, it can be found in and between the agents and platforms. Since the relations between platforms are defined in the system, those parts of the organisation can also be found there. Organisation contains the assignment of (sets of) agents to platforms, the relations between agents given by their intended interactions, the connection of platforms in the system layer, etc. It should be noted that in an entity-relationship-model [Chen, 1976] there would be only 1:n relations downward through the MULAN architecture (e.g. one platform having n agents and one agent having m protocols), but many m:n relations within the architectural levels (e.g. agent interaction). Like the relations in multi-agent systems, organisation is dynamic, especially if agents are mobile and can change their execution environment.

Considering the discussions in Section 5.1.4, every element in the MULAN reference architecture can be regarded as an agent. This leads to the following abstract definition of the structure of a software system characterised by the agent-oriented modelling perspective and the concepts of the MULAN agent model and reference architecture:

Key Term Definition B.2 (Abstract Definition of Structure)**.** *The structure of a software system is constituted by (abstract) agents and the relations between them.*

As with the idea of *Everything is an Agent* this definition is, by itself, abstract and conceptual. While it focuses on the components and organisation it implicitly also captures infrastructure as described above. It considers the three levels of MULAN, agents, platforms and the system, all as the same kind of abstract agents. Between these abstract agents, diverse relations and connections are present[6]. In order to make it practically applicable it has to be brought closer to the implementation. A decision has to be made which abstract agents are actually implemented as practical agents. For the context of MULAN this is defined in CAPA where, besides the actual agents, the platform is also implemented by agents. Taking this under consideration results in the following refined, concrete definition:

[6] Relations between concrete agents are e.g. very different to the relations between agents and platforms.

Key Term Definition B.3 (Concrete Definition of Structure)**.** *The structure of a multi-agent system is constituted by agents and platforms and the internal and external relations between them.*

This definition takes concrete agents into account. While platforms are implemented as agents, it is beneficial to distinguish between the functional agents and the standard platform functionality agents (see Section 5.1.4). To emphasise this distinction the definition reverts back in terminology to the Mulan reference architecture. The distinction also makes the infrastructure and organisation aspects in the definition more explicit. Relations between agents and their platforms provide a basis for the infrastructure. Relations between (only) agents, (only) platforms and also those between agents and platforms describe the different kinds of organisational facets. In addition, the concrete definition stresses the fact that the system now includes concrete software agents and can be concretely referred to as a multi-agent system.

Up until now the idea of *Everything is an Agent* was used in a restricted way. To conclude this section the restriction is lifted for a short discussion. This reaffirms the agent-oriented modelling perspective but does not affect the previous definitions.

The only Mulan level missing from the abstract definition is the protocol level. Considering protocols as parts of the structure appears to be counter-intuitive. Since it makes the definition more difficult to comprehend, it was omitted from the definition above. However, considering protocols is possible, at least on the abstract and conceptual level, as was already discussed in Section 5.1.4. At this point, however, the question is not how to consider the protocol as an agent but rather how it affects the view on structure. Protocols describe behaviour and regarding them as agents turns them, on the abstract level, into parts of the structure. Nonetheless, this does not change the definition. Protocols are, in this consideration, agents that describe the behaviour. But, as agents, they also have a behaviour (purpose and functionality) that is equal to the behaviour they describe. In some conceptual ways the structure and behaviour may be considered merged at this point, but distinguishing between the agent that represents the behaviour and the actual behaviour executed by that agent provides the separation needed in the context of the definition. The abstract agents within the system still constitute the structure and are clearly separated from the behaviour.

For the concrete definition in addition to the protocols the system level is also omitted. The system is the sum of all components which means it also incorporates the entirety of the structure. The only problem is that, in Capa, it is usually only indirectly modelled through the agents and platforms and is therefore implicit. It doesn't add anything new to the structure. In classic Mulan the system layer is explicitly modelled to define the infrastructure between platforms. Capa, in a way, moves this infrastructure into the knowledge and data of the platforms. Both the Mulan and Capa views are justifiable. The concrete definition is closer to the practical implementation and consequently closer to Capa though, which is why the system layer is not considered. If the system was considered as an agent, it would also not add anything on a conceptual level as it could directly be seen in the same way as a platform. It would simply be a platform of platforms, another type of container (agent).

Protocols as agents in the concrete definition are a bit more difficult. They can still be considered as agents that represent the behaviour and also have a behaviour. The separation of structure and behaviour discussed above is still valid. However, these agents now actually need to practically execute that behaviour. At first glance, this would add another level to the agent hierarchy, a kind of protocol for the protocol-agents. These protocol-agents, though, would be markedly different from the regular agents. As discussed

in Section 5.1.4 protocols as agents would be extremely simple. They would have only one small behaviour, namely the one described in the protocol. This behaviour would be hard-coded into them, eliminating the need for another protocol level. Ultimately, this allows agents as protocols to be regarded in the same way as in the abstract definition. However, the concrete definition is more concerned with practical considerations. Realising protocols as agents would require a very efficient implementation and would be more difficult to administrate. It could also affect flexibility, as these protocol-agents would naturally be very rigid. Therefore, it is not feasibly to consider protocols as agents at this point. The idea is similar to that of the Plugin-Agent for Capa though (see [Schleinzer, 2007, Schleinzer et al., 2008]).

In summary, adding protocols to the abstract definition or the system and protocols to the concrete definition neither affects the definitions nor does it add anything to them. Protocols on the abstract level simply allow for a more comprehensive application of the idea of *Everything is an Agent*. The system only adds something to the concrete definition if classical Mulan is considered which is not emphasised in the concrete definition. Protocols on the concrete level act in the same way as on the abstract level, but are cumbersome to realise practically. All in all, agents and platforms are the more important and prominent features of the structure, which is why they are given priority in both definitions.

5.4 Structure and Agents in this Thesis

This chapter dealt with agents for modelling software systems. It described an abstract approach towards agent-oriented modelling. While the agent-oriented modelling perspective is a specialised model created for the context and purposes of this thesis, it still follows the general core of agent-orientation. That general core, the concept of agents as the main modelling construct, drove the discussions and results of this chapter. It is also reflected in the definitions of structure in Section 5.3.

Chapter 4 discussed how structure and behaviour are considered as the two main aspects of a software system. The choice of agents for the modelling and representation of structure was shortly addressed in the overall introduction and motivation. Now, the reasoning and justification of that choice can be provided in more detail.

From the discussions in this chapter it is clear that agents emphasise the structure of a software system. The idea of *Everything is an Agent* may represent the extreme end of the spectrum, but agents, even in general context, are, first and foremost, components of a software system. They are pieces of software that autonomously exist in some environment with a design objective they work to achieve. In the understanding of the structure of a software system from Chapter 4, structure consists of components, infrastructure and organisation. Agents actually and directly *are* these components of the structure. The infrastructure and organisation as well are given by agent execution environments, i.e. agent platforms, and, more indirectly, by the relations between the individual agents themselves. In other words and in reference to Definitions B.2 and B.3, the most tangible parts of the structure of a software system are directly defined through agents and agent platforms. This means that agents, as a modelling construct, directly model the structure of a software system. In reference to the fact that structure and behaviour together span the entirety of a software system, this in turn means that agents can be used to directly model one of the two dimensions of a complete software system. This ability is why they were chosen as one of the two modelling constructs for the integration desired for this thesis. They are the *structural modelling construct* for this thesis.

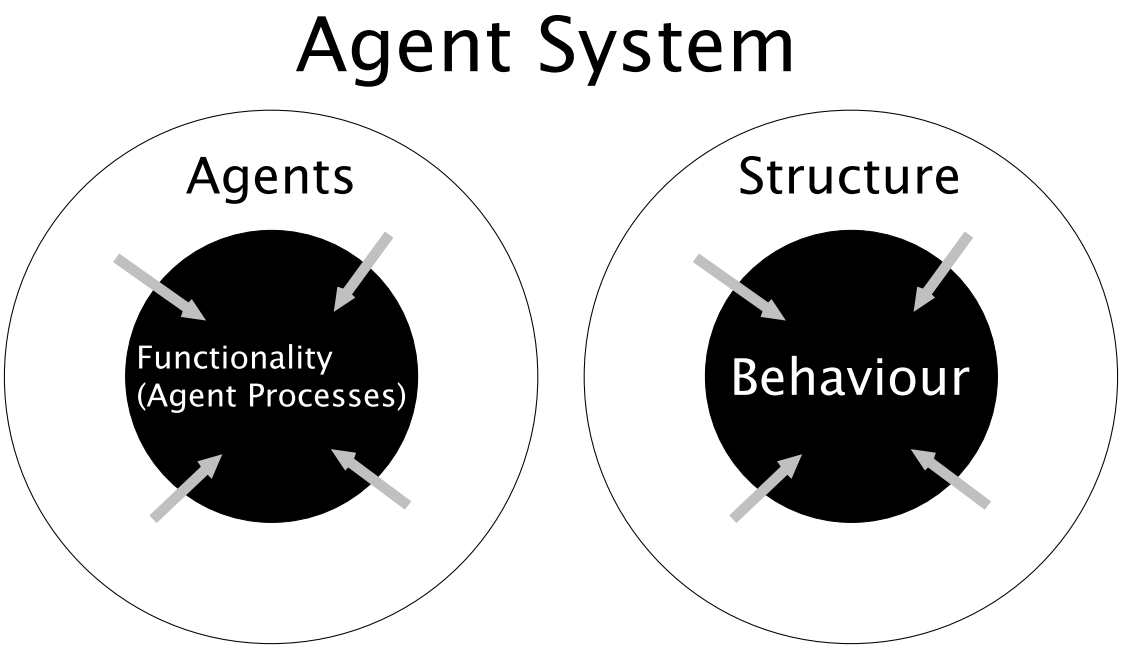

Figure 5.5: Structure and behaviour in an agent system

An open question when using the agent-oriented modelling perspective or agents in general is how the relationship between structure and behaviour is characterised. Structure is directly defined through agents and agent platforms. Behaviour in an agent system, as discussed in Section 5.1.3, is a set of composite processes consisting of processes for individual agents that contain internal and external actions describing the functionality of agents. This means that behaviour is not independently defined, but only modelled and represented in association with the executing agents. The result of this is that the structure is easier to grasp, i.e. easier to model and understand, in the agent-oriented modelling perspective as it benefits from the advantages of the direct structural abstraction of agents. Behaviour is negatively affected by the indirection and partial implicitness of the processes and the inevitable association to the agents. Modellers always have to take an extra step (mental or modelling) w.r.t. the behaviour that they don't need with the structure. This means that in large and complex scenarios it becomes ever more cumbersome and inefficient to represent, model and understand the behaviour.

This issue is illustrated in Figure 5.5. Agents determine the functionality and associated processes (left-hand side of Figure 5.5), which means that these processes are always considered in the context of individual or sets of agents. In other words, neither functionality nor processes occur without a superordinate agent or set of agents. On the more abstract level (right-hand side of Figure 5.5) this corresponds to the structure (the agents) determining and governing the behaviour (the functionality and processes). This relates to the overall motivation, which is why the right-hand side of Figure 5.5 can also be found in the Yin and Yang illustration from Figure 1.1 in the introduction. The Yin and Yang nature of an integration, as well as the changed relationship between structure and behaviour in such a system is picked up again in Chapter 7.

6 Behaviour Based on Workflows

After exploring structure based on agents in Chapter 5, the current chapter now examines behaviour based on workflows. The goal of this chapter is to consolidate and combine the general understanding of behaviour of a software system from Chapter 4 with workflows in general and workflow nets (especially Renew workflow nets) in particular. Mirroring what the previous Chapter 5 did for (Mulan) agents and structure, this chapter creates a refined and specialised definition. The result specifies behaviour and the behavioural perspective for the remainder of this thesis.

This chapter contains four sections. Section 6.1 describes the workflow-based modelling perspective. Building on that modelling perspective, Section 6.2 introduces the concept of the three basic workflow operations. Next, Section 6.3 compiles a definition of the behaviour of a software system through workflows and workflow nets. Finally, Section 6.4 concludes this chapter with a short discussion of workflows as the chosen modelling construct for behaviour in this thesis.

One important distinction that needs to be made for the following discussion is the one between workflow and process. Definition A.2 defines a workflow as an automation and facilitation of a process, while Definition A.8 defines a process as an ordered collection of tasks. In other words, a workflow is the realisation and implementation of a process consisting of tasks for use in a workflow system. Both refer to the same abstract entity. Process emphasises the concept of that entity, while workflow emphasises the technical implementation. Both terms are used to emphasise certain issues throughout the following discussions. If the separation between concept and implementation is unclear at any point, the issue is addressed.

6.1 The Workflow-Based Modelling Perspective

The workflow-based modelling perspective is the workflow-focussed counterpart to the agent-oriented modelling perspective (see Section 5.1). As the name implies, the workflow-based modelling perspective emphasises the concept of workflows instead of agents. Like the agent-oriented modelling perspective, the workflow-based modelling perspective describes a creative and productive perspective on *how* to approach the creation of a workflow system.

The term workflow-based modelling perspective was chosen in anticipation of an implementation focus later on. Process-based modelling perspective would also have been possible, but would have implied an emphasised conceptual level. The conceptual level of processes is still explicitly included in the workflow-based modelling perspective, though, as shown by the core statement of *Everything is a Process* (see Section 6.1.1).

The workflow-based modelling perspective mirrors the agent-oriented modelling perspective's purpose in this thesis. It is the adopted perspective for all concepts, models, aspects and prototypes that are either directly workflows or based on workflows. It also provides the basis for the behavioural/workflow aspects in the integrated modelling perspective in Chapter 7.

This section describes the workflow-based modelling perspective in four subsections. Section 6.1.1 discusses the core statement and idea of the perspective, while Section 6.1.2

discusses the role of cases and resources. Next, Section 6.1.3 examines how structure and behaviour are represented in the perspective. Finally, Section 6.1.4 shortly discusses the particularities of using workflow nets instead of general workflows.

6.1.1 Everything is a Process

When modelling a workflow system[1] the concept of a process is fundamental. The process is what the technical implementation of the workflow realises. As with agents in the agent-oriented modelling perspective, the process is at the core of any discussion:

Processes are the main modelling construct: Processes are the main focus in modelling workflow systems. They are considered with first priority in all design and modelling decisions in the system.

Processes are the main partitioning construct: Workflow systems are partitioned into processes. This partitioning is binding for all system observations, including monitoring and analysis.

Processes are the main relational construct: Every element of the system is always considered in relation to one or more processes. This includes cases, resources and other processes (e.g. subprocesses).

Processes are the main functional construct: The functionality of the workflow system is contained in the processes. While resources execute those processes, the modelling emphasises the functionality within the processes.

Processes are the main flexibility construct: Adaptability, or any other advanced process or workflow property, is always focussed on the process. Flexible and dynamic behaviour is governed by the processes.

This means that these, and more, areas of workflow system modelling all inherently emphasise the concept of a process. The workflow-based modelling perspective uses this emphasis and, just like the agent-oriented modelling perspective, sums it up in one abstract core statement:

Everything is a Process

This statement represents the main idea of the workflow-based modelling perspective: A logical partitioning of a system into abstract, related processes. There are two major questions to answer at this point: First, how is everything considered as a process? Second, how are these processes related?

To answer the first question one must look first at the different possible views of processes. Definition A.8 basically defines a process as a set of related and ordered tasks. The definition is deliberately vague as to the relation of the tasks.

The classical view on processes is to consider a process as a set of ordered tasks, semantically related by their context. This means that each process represents a specific application context and only contains tasks that are related to that context. This also means that the tasks in a process may not, and in fact usually are not, executed by one resource.

The workflow-based modelling perspective and *Everything is a Process* go beyond this classical view. The grouping of tasks into processes according to their context is one

[1] Please note the distinction between workflow system (Definition 2.18) and workflow *management* system (Definition 2.16).

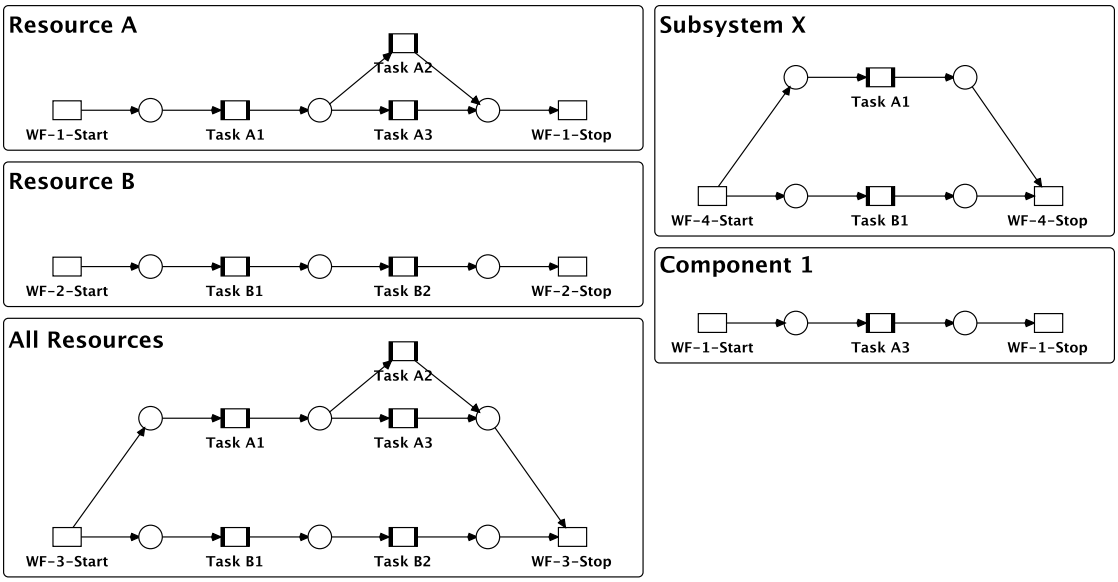

Figure 6.1: Different processes in a system

possibility. Another way of grouping tasks is by resource. By grouping all the tasks a specific resource executes in the overall system, a distinct process is generated. That process represents the entirety of the behaviour of that one resource. In fact, it is possible to fully describe the resource with that process. Resources, in general, are those elements of the system, which actually execute the work associated with the tasks of the processes. They are the structural elements of the workflow system. This means that by generating a process from a resource it becomes possible to describe a structural element of a system as a (behavioural) process. In other words, every structural element of a system can be described not only by its structural attributes and roles within the system, but also by the process it executes.

Any other, arbitrary grouping is also possible. Groupings according to subsystems, subcontexts, etc. are all valid. On the level of *Everything is a Process* the choice of processes is free. It represents one view of one element of the system, no matter if it is an application, an element, a subsystem or a certain aspect. Before proceeding with the implementation, though, the control flow between the tasks, which describes the dependencies between them, needs to be fixed. That control flow may not be visible in all processes. In an implementation, though, that control flow needs to be correctly implemented. This is one of the challenges modellers using the workflow-based modelling perspective need to keep in mind. Everything, meaning every element and aspect, of a system can be represented as a process by providing a suitable selection and grouping of tasks. However, the dependencies between tasks and processes need to be captured and implemented in some way at a later stage. This is picked up again in the discussion about process relations and properties of the idea later on.

Figure 6.1 illustrates the representation of different elements as processes. It shows a workflow system that consists of two resources (A and B) performing tasks. The upper left-hand side of Figure 6.1 shows the processes for both resources. These processes describe the complete (but simple) behaviour of resources A and B. Since A and B do nothing besides these processes they, as structural elements, are completely represented by their processes. The remaining boxes show different process representations of the system. The

lower left-hand side shows the complete process of all resources in the system. Combining the processes of all resources also yields a complete representation of the system as a process. The right-hand side picks out arbitrary subsystems and components that are involved in specific tasks. These are incomplete and arbitrary but showcase the idea of *Everything is a Process*. At any time a system modeller might need to consider a specific subsystem or component. If that is the case the modeller can use the process representation of that subsystem or component in order to make modelling and design decisions according to it.

If every element of the system can be considered as a process another question that needs to be answered is where the functionality of the system is situated. Shortly answered, the functionality is contained in the processes. The functionality is the aspect of the system, which fulfils the system's purpose. Previous paragraphs stated that every aspect of a system can be captured by a process. The aspect of functionality can be captured in a process or set of processes as well. These processes describe what exactly happens within the system. They consist of tasks that describe basic units of work. It might be argued that the tasks contain the functionality, but the individual tasks lack the information about the control and data flow that is required to fully describe the functionality. It might also be argued that the resources, which actually perform the work described in tasks and processes, contain the functionality. However, just like the tasks, the resources do not know how their work is connected. Consequently, the functionality is contained in the processes. Tasks and resources are essential to execute the functionality, but they do not contain it.

Grouping and selecting tasks according to different elements, contexts and aspects of the system answers the question of how everything is considered as a process. The processes also describe the functionality of the emerging system. The remaining question deals with the relations between the different processes.

In general, there are a number of ways of how processes can be related. Processes can trigger one another. If the execution of a process triggers another process they are related in the workflow-based modelling perspective. This relation includes processes triggered without any further interaction, as well as subprocesses and process hierarchies that need to be completed before the triggering process can continue beyond a certain point. Processes can also interact directly, which is another form of relation. This can happen when data from one process is required in another. Inter-organisational workflows can contain such interactions (see Section 2.3.3). It is also possible for processes to indirectly relate to one another. This may happen if a process needs to progress beyond a certain point before another one can continue itself. This is not a direct interaction, but an indirect dependence and another form of relation in *Everything is a Process*.

When considering arbitrary groupings of tasks in a system as processes the question arises if these arbitrary processes may relate to one another differently. The answer to this is that basic relation types do not change, but concrete relations may change due to different processes. Basically, whenever the relation is caused on the individual task level, the grouping of tasks does not affect it. Grouping tasks into different processes does not change the tasks itself. If a task requires interaction, that task requires that interaction in any process grouping. If a task has an indirect dependence to another task, the grouping is also irrelevant.

However, if the relation is caused by the grouping, it changes with different groupings. For example, if a task would start a new subprocess in one specific process grouping, there is another grouping that contains the subprocess within the original grouping. In that case the task would not trigger a new process, but its own process would continue on with the tasks of the original subprocess as part of the overall process. Still, this reasoning works

both ways as it is always possible to extract a number of tasks from a process and turn them into a separate subprocess. The basic kind of relationship of processes triggering processes and possibly creating hierarchies still exists in some form or another in any process grouping. How that relationship and hierarchy is structured, though, is completely dependent on the actual shape of (i.e. grouping of tasks into) the processes.

The previous paragraphs answered the basic questions raised by the statement of *Everything is a Process*. The core idea is, in general, not that much different than the idea of *Everything is an Agent*. Consequently, the properties, advantages and disadvantages of both ideas are very similar or even analogous. *Everything is a Process* provides a uniform perspective on all elements of the system. That uniform perspective allows system modellers to make modelling and design decisions from a common basis.

Something that is different from *Everything is an Agent* is that there is a larger dynamic and flexibility in the choice of process. As discussed above the choice of process is arbitrary. Grouping tasks by semantic application context or by resource are the most reasonable choices for processes. However, even in these cases a large degree of flexibility remains. By shifting the focus of the application slightly, different tasks may be involved in a process. This makes modelling more complex, but also gives the modellers more choices for optimising the system.

The choice of processes also determines the different relationships between them. Modellers may choose a complex hierarchy of subprocesses in order to abstract certain details from certain users. Or they may choose to create flat, but complex sets of processes that feature all details for all users. Modellers are free to choose both the optimal set of processes and the most suitable relationships and hierarchies.

By viewing all elements as processes, the properties of those processes also become available to the different elements. The scope of these properties is not as impactful as with agent properties, but process properties can also be useful in practical settings. On the one hand the division of work into tasks that make up the processes allows the usage of task principles. Atomicity of tasks allows modellers to make assumptions about the state of the system when tasks have been completed. Task rollback, too, might become a powerful tool for system modellers to deal with uncertainty or error situations. It may also become possible to validate or verify processes, in turn validating or verifying the different elements of the system like resources.

Regarding disadvantages, the freedom and complexity in the choice of processes is generally beneficial but needs to be properly handled. While everything may be a process, it may not be efficient to actually have everything be a process. Also, as mentioned before, the control flow between different tasks may not be captured in all process variations. This places the responsibility on the modellers to choose a suitable set of processes. One way to deal with this issue is to prescribe the use of a fixed set of processes. These processes contain tasks that are semantically related to one application and consequently contain the minimum necessary control flow. The remaining processes contain additional information, but are mostly different views on the system. Through these views, modellers can use the above mentioned advantages without having to worry about missing control flows or dependencies.

Conceptual overkill is another issue facing *Everything is an Agent*. Systems need to be sufficiently large to exploit any of the advantages. However, even in small systems the view of elements as processes can yield beneficial partial results. For more details on the properties of the idea please refer to the analogous discussion on *Everything is an Agent* in Section 5.1.1.

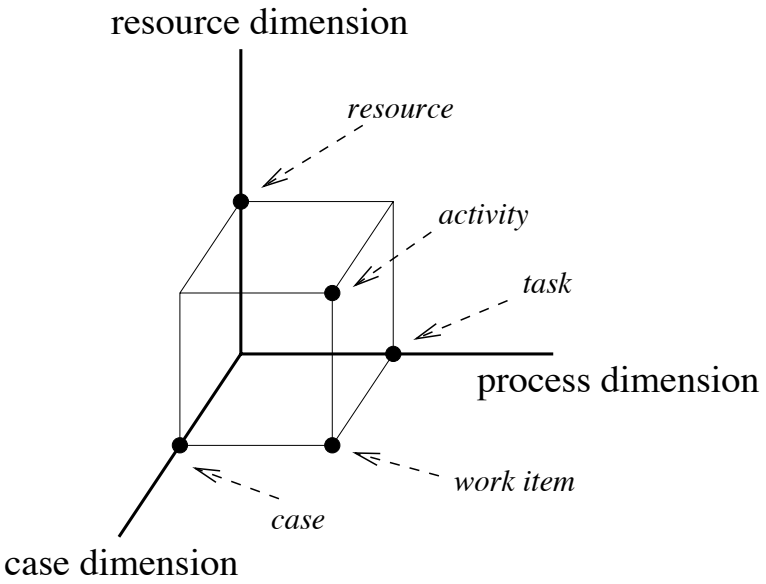

Figure 6.2: Dimensions of a workflow (repetition, from [van der Aalst, 1998b, p. 6])

Before moving on to a short discussion about the role of cases and resources there is another important issue regarding *Everything is a Process*. It concerns the distinction between process and workflow. As stated before both of these terms refer to the same entity, namely the collection of tasks according to a specific element, context or aspect of the system. Process emphasises the concept of that collection, while workflow emphasises the implementation. The term process was chosen for the statement *Everything is a Process*, because the idea behind the statement is firmly set on the conceptual level. *Everything is a Process* is **not** about actually describing everything as a process in the implementation. Rather, like *Everything is an Agent* in the agent-oriented modelling perspective, it is about conceptually considering every element as a process in order to gain a unified understanding of the elements of a system. The number of actually implemented workflows may drastically differ from the overall number of processes that may be considered.

6.1.2 Cases and Resources

The previous section described the ability to consider every element of the system as a process in the workflow-based modelling perspective. This current section now examines, how the process-related concepts of *case* and *resource* affect the workflow-based modelling perspective. As Figure 6.2[2] shows, resource and case are the two missing dimensions of a workflow. Together with processes, which were discussed in the previous subsection, they cover the entirety of a workflow system.

Definition 2.13 defined a case as a single enactment of a process. Cases are important in the workflow-based modelling perspective, because they represent the enactment of the processes described by *Everything is a Process*. At first glance, cases appear to be only relevant for the actual implementation with workflows. Since they relate to the enactment, their applicability to the conceptual idea of *Everything is a Process* appears counter-intuitive. However, as discussed before, *Everything is a Process* describes, in essence,

[2]This is a repetition of Figure 2.15. Since it is relevant for the current context it is reused to avoid having to browse back to the previous occurrence.

a virtual grouping of tasks. If the tasks exist in the workflow system, any particular virtual process can be used to represent them. This is independent of the workflow that is actually executed. For example, if the system consists of a set of workflows for a number of application contexts, it is also possible to consider the virtual processes of each resource involved in these contexts. When these tasks are executed in the context of a case, then some form of case also exists for the virtual processes. However, these cases are also virtual. As, for example, the virtual process of one resource may cross multiple cases of the application workflows, there is not one particular case for that resource. Rather, the virtual case is assembled by the different cases it crosses during its execution.

Moving on to resources, Definition A.3 defined a resource as an entity, which performs work in a process. Resources were already shortly discussed as processes. However, regardless of the consideration as a process, any workflow system still needs actual resources. They are the individual, active components that execute the functionality described in the processes. Without resources, no processes may be executed in any system. Resources are irrelevant for a system if they are not part of any processes. But, as mentioned before, a resource may be involved in many processes. This means that resources are orthogonal to processes and a cross-cutting concern in any workflow system. Still, within the workflow-based modelling perspective resources are not considered as stand-alone. Since the processes are at the centre of attention, resources are only regarded in the context of the processes they are involved in.

6.1.3 Structure and Behaviour

Chapter 4 identified structure and behaviour as the two main aspects of any software system. This section examines the two aspects for the workflow-based modelling perspective.

Behaviour is quickly identified. The workflow-based modelling perspective emphasises processes in the core idea of *Everything is a Process*. Every element of the system can be described by the tasks it is involved in. These tasks form a virtual process, which is why that particular view on the element is part of the system's behaviour. Whether or not that process is actually implemented as a workflow later on is irrelevant. It can be considered as a process, so it is part of the behaviour. Still, if everything is considered as a process, the question about what is actually executed is not answerable on the conceptual level of process. It is implicitly contained in the processes, but irrelevant on this level.

Structure in the workflow-based modelling perspective is provided by the resources executing the processes. As discussed above, while resources can be seen as processes (and are part of the behaviour in that view) they are still required as actually active elements in the system. The necessary consideration of them as active elements represents a view concurrent and orthogonal to the previously discussed behavioural view on them. In that view as active elements the resources are the structural components of the system. While considering conceptual processes the remaining parts of the structure, infrastructure and organisation, are implicitly included in the resources. Resources are related to one another in some way (organisation) and have an implicit access to the processes and their data (infrastructure).

Up until this point, the discussions in this section were mostly kept to the conceptual level related to the term process. However, when considering structure and behaviour of the actual workflow system, a shift in perspective towards the implementation is necessary. One factor that was ignored until now is that of workflow management. Workflow management functions make the implicit parts of structure and behaviour of workflow systems explicit.

For behaviour, the workflow management mechanisms handle the relationships between processes. Triggering other or subworkflows, providing the means for processes to access

shared data and interact that way, as well as unravelling dependencies is all handled by different management mechanisms in the workflow enactment service and workflow engines. Cases as enactments of specific behaviour are also part of the workflow management functionality. However, only the concrete cases of explicitly modelled processes are directly available. The virtual cases discussed in the previous subsection are usually not generated or observed at this point.

For structure the management functionality makes infrastructure and organisation explicit. The workflow management provides the infrastructure that resources need to access the processes, tasks and data required for their work. In that way, the workflow management connects resources to processes. Regarding organisation, the management functionality is responsible for managing not just the processes, but also the resources. Access rules, role hierarchies, rights management all provide essential functionality regarding the organisation of the structure provided by the resources.

The management elements themselves, like the workflow enactment service or the workflow engines, cannot be directly associated with either structure or behaviour of the workflow system. This is due to the fact that they are *not* part of the workflow system, but part of the management system framework supporting and executing the workflow system. Still, as described above, they strongly influence both structure and behaviour. They enrich both the structure and the behaviour by making relations between processes and the components (resources) explicit. But since they are outside of the scope of the workflow system under consideration this influence is not enough to directly consider them as either.

There is, however, a particularity regarding the management elements. According to *Everything is a Process* management elements can also be considered as processes. Management elements are responsible for a set of tasks in the workflow system. *Everything is a Process* states that an element can be considered as a process of the tasks it is involved in. If that perspective is taken, then a management element becomes the conceptual processes of the workflows it is responsible for executing. In that way, management elements *can* be regarded as part of the behaviour, even though they are actually outside of the scope of the workflow system.

In summary, behaviour is provided on a conceptual level by the processes generated by applying the idea of *Everything is a Process*. On a practical, implementation level the behaviour is provided by the implemented workflows and the relationships between them. Structure generally consists of resources as the structural components. Conceptually, resources implicitly contain all the information regarding structure. Practically, workflow management functionality is required to explicitly handle the infrastructure and organisation of the structure of the workflow system.

An aspect of note from these observations is that processes and workflows organise their structure in a behaviour-centric way. This means that in the workflow-based modelling perspective and the idea of *Everything is a Process* the structure is depending on the behaviour. Part of motivation of the integration is to reduce or eliminate this governance of structure through behaviour.

6.1.4 The Workflow Net Modelling Perspective

Since workflow nets are the chosen representation and implementation of processes and workflows in this thesis it is necessary to address them in the context of the workflow-based modelling perspective. In general, using workflow nets or any other workflow representation does not affect the modelling perspective in any principal way. The way a workflow is modelled only affects the processes if there are limitations to the modelling.

Workflow nets are based on Petri nets. Petri nets are a universal tool for describing concurrent systems. Consequently, they are not limited in their ability to describe complex situations in those systems. Considering higher level Petri net formalisms, like reference nets, the modelling abilities become even more accessible.

Another interesting aspect of using Petri nets as the basis for workflow modelling is the formal basis. By using Petri nets validation and verification of the workflow systems become possible. While these formal aspects are outside of the scope of this thesis, they are picked up in the discussion and outlook at the end.

6.2 The Basic Workflow Operations

The previous section described how processes and workflows are at the core of the workflow-based modelling perspective. Both processes (conceptually) and workflows (practically) consist of an ordered set of tasks. Tasks are logical, indivisible units of work. As such, they are the smallest building blocks available for processes and workflows. An examination of how tasks work and operate is an important result required for the understanding of behaviour of a workflow system. If that view on behaviour is to be applied to an integration approach of a system, the results become even more important.

To simplify the descriptions, the following always refers to workflows. This indicates a proximity to the practical implementation. However, analogous arguments and discussions can be made for processes on the conceptual level. To avoid repetition, these arguments are not made here.

The life-cycle of a workflow task is always the same. At some point of the enactment of a workflow, a task may become available to resources. The task becomes a workitem and is offered to the eligible resources. A resource can then request the workitem. If the request is successful, the workitem becomes an activity assigned to the resource. The resource then proceeds to execute the work associated with the activity. If at any point the work fails, the resource can cancel the execution of the activity. The local state of the workflow is reset (rollback) and the activity becomes an available workitem again[3]. However, if the resource finishes its work on the activity, it can successfully confirm it. In that case, any obtained results are passed to the workflow which can continue on with its execution.

From this life-cycle of a workflow task it is possible to identify three basic workflow operations: *Request workitem*, *cancel activity* and *confirm activity*. These three operations describe the entirety of the workflow *internal* behaviour. The internal behaviour of a workflow defines the overall behaviour (given by the tasks), these three operations are essential for the behaviour described by workflows.

Before continuing the discussions and descriptions on the three operations, two issues need to be addressed. The first one relates to the duality of the operations. As described above the operations describe the behaviour within the workflow. However, while the operations *happen* within the workflow, they are *performed* by two different components. These two components are the workflow engine and workflow resource. The workflow engine executes the workflow and the workflow resource executes the work on the task. Consequently both must perform any workflow operation (conceptually) synchronised, meaning in some way coordinated.

As stated before, the workflow engine is not actually part of the workflow system and consequently not part of structure or behaviour of the workflow system. However, the

[3] It is feasible to consider a situation in which a cancelled activity does not directly become an available workitem. If certain indirect, global conditions are not met anymore the task may not be active and consequently no workitem would exist.

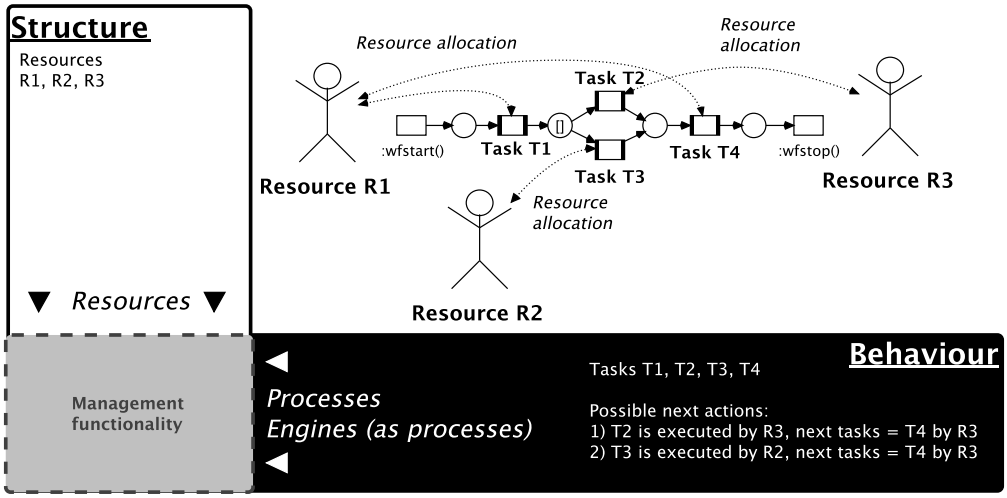

Figure 6.3: Connection between structure and behaviour in a workflow system

workflow engine can be considered as a process and, if that is done, actually becomes the set of workflows it is responsible for executing. In that view, the workflow engine acts as a part of the behaviour when performing the workflow operation. A less abstract, but equivalent, view on this situation is that the workflow engine acts on behalf of the workflow and conceptually becomes part of the behaviour that way. When the resource performs the same operation, it does so as part of the structure of the workflow system. This means that the duality of workflow operations is what determines and causes the connection between structure and behaviour. Figure 6.3 illustrates this. On the conceptual level, the duality is even more pronounced. As the conceptual level of processes does not involve management aspects, the duality happens directly between processes and resources. The question of duality needs to be addressed in any integration approach.

The second issue that needs to be addressed before the descriptions of the operations are addressed concerns implementation details. Workflow operations refer to tasks in general, as logical units of work in a workflow system. They are conceptual in nature and are applicable to any workflow description language. However, a specific technical basis facilitates the understanding of the concepts. Workflow (reference) nets are the chosen representation of workflows in this thesis overall. Consequently, they, and especially the Renew workflow task-transition (see Section 2.3.2), have been chosen as the technical basis for the following descriptions.

Request Workitem The first workflow operation within the life-cycle of any task is requesting it as a workitem. Requesting a workitem uses any input inscribed to the task, forwards it to the management system and either creates the activity directly or receives the activity from the management system.

Naming the operation *request workitem* appears to indicate that only the pull dynamic between workflow and resource is supported. Whenever a workitem becomes active, the information about it is provided to all eligible resources. The resources can then choose a workitem to request and send out the request. The workflow management system then decides whether or not to grant the request. If it is granted the operation is performed in both the engine and resource.

The other variant of the dynamic between workflow and resource is the push dynamic. Whenever a workitem becomes active, the workflow management system decides which

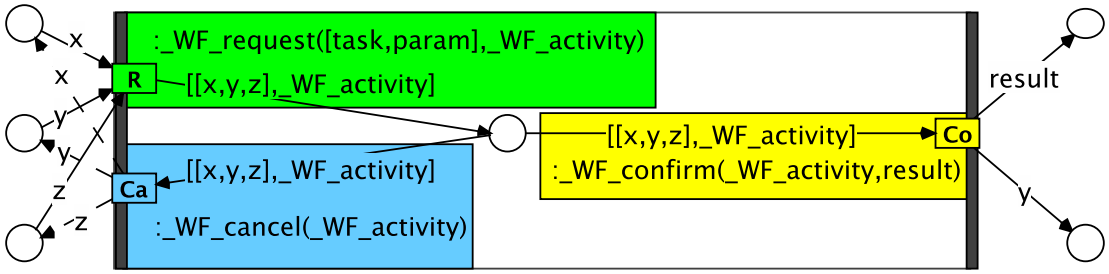

Figure 6.4: Basic workflow operations in the RENEW task-transition (based on [Jacob, 2002, p. 96]))

resource should perform it. It then assigns the workitem to the resource, in turn creating the activity analogously to the pull dynamic.

The difference between the two dynamics is whether the workitem is requested or assigned. Ultimately, for the current workflow operation, it does not matter. Basically, the operation describes the state change between workitem and activity. Regarding workflow dimensions (see Figure 6.2) it only adds the resource. Whether that resource chooses the workitem or whether it is chosen for it is irrelevant. Since the pull dynamic is the more common one in workflow management, the term *request workitem* was chosen for this context. Still, both dynamics are captured by this operation.

In the technical view of the RENEW task-transition, requesting is the initial internal transition *R*, which takes the input from the preconditions. The area is coloured in yellow. The activity object is created in the same firing step within the workflow management and passed via synchronous channel directly to the internal place of the task-transition. Figure 6.4 shows the task transition, with colour-coded areas representing each of the workflow operations. The request of a workitem is in the upper left corner, coloured green.

Confirm Activity The confirmation of an activity represents its successful completion. The resource completes the work associated with the task and transfers any result it obtained to the management system, which, in turn, transmits it to the workflow for further processing.

The confirmation of an activity is always initiated by the resource. The resource is usually the only component of the system that knows the progress of the work being done on the task. Monitoring tools might also have access to that information.

In the task-transition in Figure 6.4, the confirmation is implemented as the right-hand side internal transition *Co*. It takes the input and activity from the central place and receives, via synchronous channel from the management system, any result obtained from the resource. Output variables from the original input are passed through from the internal place unchanged.

Cancel Activity The cancellation of an activity represents its failure option. The resource cancels all work associated with the task and informs the management system. The management system then proceeds to cancel the transition for both the resource and the workflow (engine).

Usually, the cancellation is initiated by the resource. As with the confirmation the resource is, except for monitoring facilities, the only component that is able to make a decision to cancel. However, it is feasible for the engine to make that decision itself. In the simplest case, a timeout might be implemented for a task that determines if a resource

takes too long to finish the work associated with the task. Such behaviour of workflow engines, though, is rather specialised.

In Figure 6.4 the cancellation is implemented in the lower left-hand side internal transition *Ca*. As with the confirmation transition it takes the input and activity from the central place when it fires. However, it does not receive any additional data or objects from the management system and merely returns the objects mapped to the input variables back onto the precondition places.

A feasible alternative to the rollback associated with the cancel transition in the task-transition is to provide the failure option in some other way. Tasks are atomic, meaning that they need to happen fully or not at all. Other failure options include throwing exceptions, relinquishing control to another element (e.g. a human workflow owner) to fix the issue or even simply skipping the task (if optional tasks are supported in the workflow). As long as the activity has a failure option that can be called, this operation is still valid. For the context of this thesis, though, associating a local rollback with a cancellation is the standard option.

Conclusion After discussing the three basic workflow operations the remaining question is whether these three operations really do capture the entirety of the behaviour. There are aspects of a workflow, which do not correspond explicitly to tasks. Examples include the intermediate processing of results between tasks (e.g. changing data formats), routing decisions for the control flow (e.g. depending on results) and management actions like the instantiation and termination of the workflow. While these kinds of aspects are not explicitly related to the tasks of a workflow, they still represent units of work that need to be done. This means they are, in fact, tasks, albeit implicit ones. The resource for these tasks is the workflow itself. The workflow is also directly the management element for these tasks. It processes the intermediate results, routes the control flow and starts and terminates the process[4]. The workflow operations hold for these auxiliary tasks, though since the tasks are implicit they are implicit as well. At some point the workflow begins the work (request workitem) and completes it (confirm activity). Cancellations (cancel activity) are rather rare in these auxiliary tasks, but it is feasible for the workflow to throw and handle exceptions, which is similar.

In summary, the three basic workflow operations describe the entirety of a workflow. For the auxiliary mechanisms the task view is implicit, but still viable. Basic workflow operations represent another key term for this thesis:

Key Term Definition B.4 (Basic Workflow Operations)**.** The three basic workflow operations are: Requesting a workitem, confirming an activity and cancelling an activity. Through these basic workflow operations a workflow can be described in its entirety.

6.3 Behaviour through Workflow Nets

This section combines the previous results of this chapter into a refined and concise definition of the behaviour of a software system. The definition is used as the basis for any further examinations of behaviour and the behavioural parts of any integration examined and developed in the remainder of the thesis.

Chapter 4.3 examined the behaviour of a software system as an abstraction of Petri net firing sequences onto processes. Workflow Petri nets explicitly describe processes, which makes an application of them to the behaviour directly possible.

[4]The overall WFMS is also involved in starting and terminating the workflow, but, as stated before, the WFMS is outside of the scope of the workflow system.

For the purpose of this chapter, the distinction between processes and workflows has been made. Processes remain on an abstract and conceptual level. Consequently, it is possible to directly derive an abstract definition of behaviour that takes processes into account:

Key Term Definition B.5 (Abstract Definition of Behaviour)**.** *The behaviour of a software system is constituted by its processes and the relations between them.*

By emphasising the processes of the system, Definition B.5 directly captures the understanding of behaviour from Chapter 4.3. The definition also contains the relations between the processes. Relations are important, because they describe how the processes are interconnected among each other.

The abstract definition captures the core idea of *Everything is a Process* completely. As described in Section 6.1.1, the idea regards, on the conceptual level, every element, aspect and component of the system as a process. This definition does the same. On the abstract level this definition is targeted at, it is completely viable to consider processes as freely. What is actually implemented as a workflow is irrelevant for the sake of this definition.

In order to apply the definition to the practical level of an implementation with workflows it needs to be adapted. Here, not all elements are actually implemented as workflows. They can still be regarded as virtual processes (see previous discussions), but are not implemented as such.

Also, in order to be practically applicable for this thesis the concrete definition needs to take workflow nets into account. While this is a specialisation it is not a limitation. As discussed in Section 6.1.4, any other workflow description language can be taken. Workflow nets are the chosen description language in this thesis. Analogous definitions can be given for alternative workflow description, e.g. BPMN (see Section 3.2.1).

Key Term Definition B.6 (Concrete Definition of Behaviour)**.** *The behaviour of a workflow system is constituted by a set of workflow nets. These workflow nets are related to one another and internally constructed in such a way that they capture and cover the distinct tasks of the system in the correct order.*

Since this definition takes the implementation level of workflows into account, it specifies a workflow system as the software system under consideration. The first part of the definition is analogous to the first part of the abstract definition, replacing processes with workflow nets. The second part, however, specifies how the workflow nets need to be related.

The issue here is that of task coverage. The abstract definition considers every element as a process. In these processes there exists a set of tasks that actually need to be executed by the system. These tasks can be part of any number of virtual processes, e.g. the resource process, the subsystem process, the overall process. However, on the concrete implementation level these tasks need to be implemented in at least one workflow net. This is what the second part of the concrete definition describes.

The workflow nets of the system need to cover the different tasks that can happen in the system. They also need to capture the order of the tasks, including concurrency and parallelism. If the workflow nets fail to do so correctly, tasks may be out of order or missing entirely. The resulting system would not feature the intended behaviour.

In summary, the two definitions of behaviour capture the distinction between conceptual process and implemented workflow (net). The abstract definition falls in line with the workflow-based modelling perspective and the core idea of *Everything is a Process*. The concrete definition leaves the conceptual level and considers what is actually implemented as a workflow (net).

6.4 Behaviour and Workflows in this Thesis

This chapter examined workflow modelling. The core of this chapter is the workflow-based modelling perspective, an abstract approach towards modelling software system with workflows. As with the agent-oriented modelling perspective, the workflow-based modelling perspective represents a specialised model created for the context and purposes of this thesis. Still, the core idea of workflow management, namely putting the concept of workflow at the centre of all modelling considerations, is captured in it. That core idea is also what drove the different discussions throughout this chapter. It is also reflected in the main result of this chapter, the abstract and concrete definitions of behaviour given in Section 6.3.

Selecting workflows as the modelling and representation construct for the behaviour of a software system was shortly addressed in the overall introduction and motivation of this thesis. Now that the basics of workflow management, as well as the workflow-based modelling perspective have been discussed, the reasoning behind this selection can be explored in more detail.

This chapter has clearly shown that workflows emphasise the behaviour of a software system. Workflows are the technical implementation of processes, which, according to the understanding of behaviour from Chapter 4 and the abstract definition of behaviour in Definition B.5, constitute the behaviour. On the concrete level of Definition B.6, the behaviour is actually directly constituted by the workflows, or rather the specific format of workflow nets. This means that workflows define the elements of the behaviour of a software system. On the concrete level this even happens directly, while on the abstract level the indirection of processes and implementation exists. The latter, though, represents only a minor indirection as, as discussed before, all tasks of all (possibly virtual) processes need to be captured in at least one workflow implementation. This means that all (virtual) processes of the behaviour can also be virtually represented even on the concrete level of workflows.

Overall, workflows, as a modelling construct, directly model the behaviour of a software system. As behaviour, together with structure, spans the entirety of a software system, this means that workflows can be used to directly model one of the two dimensions of a complete software system. This is why they were selected, next to agents, as a modelling construct for the integration desired for this thesis. They are the *behavioural modelling construct* for this thesis.

Due to the emphasis on behaviour, using workflows in general or the workflow-based modelling perspective influences the relationship between structure and behaviour. As discussed above, behaviour is directly defined through workflows, which makes workflows a direct abstraction of the behaviour. This means behaviour is easy to model, represent and handle even in large and complex scenarios. Structure in workflow systems, as discussed in Section 6.1.3, is given by the resources related to the behaviour. Resources are only considered in and defined for the contexts of workflows. From the view of the workflow system, resources that do not participate in its workflows do not exist. This means that there is a fixed dependence between the structural elements (resources) and the behavioural elements (workflows). Structure, in workflow systems, is not independently defined, but only modelled and represented in association with behavioural elements, the workflows. This indirection negatively affects the modelling, representation and handling of structure in large workflow systems. Modellers have to take an extra step when considering any structural issues that they don't have to take with the behaviour. This makes modelling, representing and handling the structure more difficult, cumbersome and inefficient.

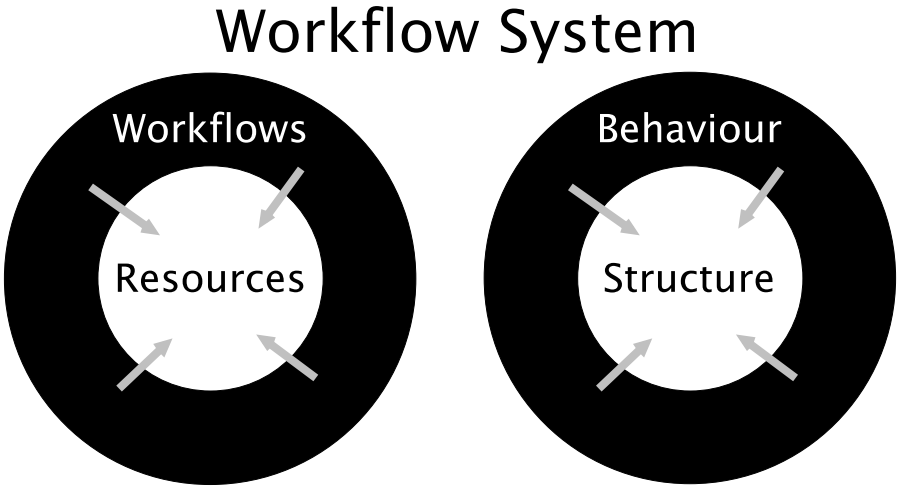

Figure 6.5: Structure and behaviour in a workflow system

Figure 6.5 illustrates this situation. Workflows of the system determine the resources participating in them (left-hand side of Figure 6.5). These resources are always and only considered in the contexts of the workflows they can be active in. In other words, they don't exist without a workflow containing tasks they can execute. More abstractly (right-hand side of Figure 6.5) this means that the behaviour (workflows) determines and governs the structure (resources). As with the conclusion of the previous chapter about agents and structure, the issue illustrated in Figure 6.5 relates to the overall motivation and especially the Yin and Yang illustration from Figure 1.1 in the introduction. This is picked up again in the following chapter.

7 Specifying an Integration

The previous chapters described the agent-oriented and workflow-based modelling perspectives. Based on these perspectives, definitions of structure and behaviour have been presented. These definitions represent the foundation for the integration of agents and workflows.

This current chapter uses the results of the previous chapters in order to approach such an integration. Note that throughout this thesis the term integration can refer to both the process of integrating structure (through agents) and behaviour (through workflows), as well as the end result of that process, namely an approach/model/system that features an integration. The context of where the term is used clarifies its meaning.

The chapter is structured as follows. Section 7.1 discusses the vision of the integration of agents and workflows. It describes and specifies, informally, what an integrated agent and workflow system looks like. This includes further details about the motivation and the roles of agent and workflow concepts, as well as a focus on the idea of an integrated entity. Next, Section 7.2 discusses application scenarios in which the integration vision can be deployed to gain advantages over the classical agent/workflow approaches. Section 7.3 then defines a set of specific criteria for an integration of agents and workflows. These criteria represent properties and characteristics an integration approach should exhibit. They are based on the vision and specification of an integration and are used in the next chapters to evaluate the possible integration approaches. Finally, Section 7.4 consolidates the results of the thesis so far into a set of requirements the overall results have to achieve.

7.1 Vision of an Integration

Up until now, this thesis established an understanding of the foundation for an integration. As discussed in the previous chapters, agents and workflows as modelling constructs emphasise structure and behaviour respectively. Consequently, an integration of agents and workflows is an integration of structure and behaviour of a software system.

Building upon the established understanding of structure (through agents) and behaviour (through workflows), this section now provides a specification and vision of an integration of agents and workflows with more concrete and confirmed details than were available in the introduction of this thesis. It deals with the relationship between structure and behaviour in an integration, with the perspective on modelling and with the nature of the modelling constructs. This current section generally characterises features and elements of an integration, independent of any concrete integration approach. Its aim is to promote a perception of what an integration conceptually is before the following chapters deal with specific integration approaches.

This section consists of four subsection. Section 7.1.1 examines how structure and behaviour are related in an integration system. Following that, Section 7.1.2 discusses how modelling constructs are characterised in an integration. Finally, Section 7.1.3 describes how the modelling perspective of an integration is characterised.

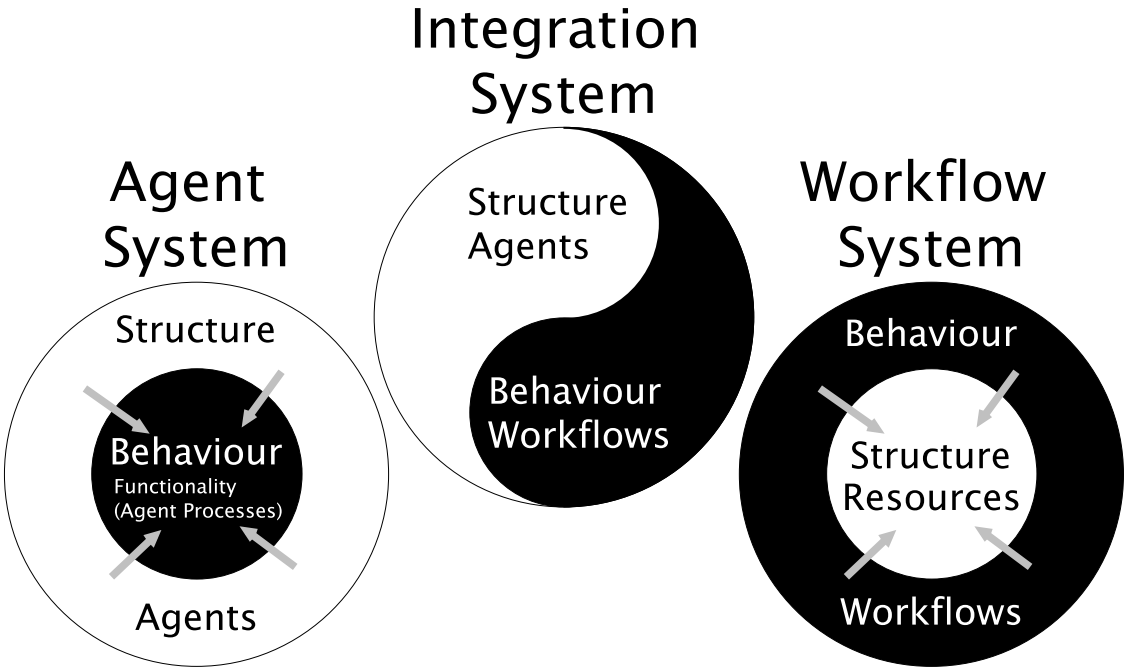

Figure 7.1: Structure and behaviour in agent, workflow and integration systems

7.1.1 Structure and Behaviour in an Integration

This section considers the relationship between structure and behaviour in an integration system. As discussed in Section 5.4 for agents and Section 6.4 for workflows, when using a modelling construct such as agents and workflows, the aspect emphasised by the modelling construct becomes the principal aspect of the developing system.

The left- and right-hand parts of Figure 7.1 reiterate the illustration of this issue. In agent systems the structure (agents) governs the behaviour (functionality and agent processes). In workflow systems the behaviour (workflows) governs the structure (resources). In both variants the emphasised aspect is easier to model, represent and handle, while the subordinate aspect suffers from the indirection introduced by the emphasis on the superordinate aspect.

One of the biggest motivating factors of pursuing an integration of agents and workflows is to eliminate this indirection. Together, structure and behaviour describe the entirety of a software system and are equally important. Without behaviour the structure simply exists and without structure to execute it the behaviour cannot achieve anything. Consequently these two aspects *should* be treated on the same level of abstraction and importance. Neither one should govern the other one and both should be modelled, represented and handled with the same relevance.

The core issue here concerns the levels of abstraction in modelling. When modelling with one particular modelling construct such as agents or workflows, the subordinate aspects and constructs are not on the same level of abstraction. The main modelling construct is at the top of a hierarchy, which describes the relations between aspects, concepts, constructs, mechanisms and so forth. For example, an agent message is being sent as a fundamental agent action, which is part of an agent process, which is part of the agent functionality, which is associated with an agent. Everything can be related and traced to the central concept of agents. For every element in the system there is a clear chain upwards the

hierarchy towards the agent[1]. The steps in these chains are the different abstraction levels for modelling. Clearly, the functionality of agents, which consists of its processes, is on a less direct level of abstraction than the agent itself. This leads to the issues discussed before about indirections in modelling the structure and behaviour. As behavioural constructs are not treated on the same abstraction levels as structural modelling constructs the modelling and representation of behaviour is negatively affected by this inequality and indirection. In other words, to model behaviour in agent systems modellers always have to model *around* the structural construct of agents. In small-scale systems this effect is negligible, but modelling behaviour in large-scale systems can become cumbersome and inefficient. At worst, errors due to inadequate tools and representations can be introduced, which can affect crucial parts of a system.

For workflows analogous arguments can be made when considering a hierarchy with workflows at the top. For example, a resource performs a basic workflow operation of requesting a workitem, which is the instantiation of a task, which is contained within a workflow.

A system which integrates agents and workflows achieves an equality of abstraction levels and importance between the two constructs. The middle part of Figure 7.1 illustrates this. Instead of one aspect governing the other one, both aspects exist on the same level of abstraction and have the same importance. They are connected, strongly related and depend on one another, yet neither one has any privileged or controlling stake.

In fact, the orthogonality of structure and behaviour in regards to spanning the entirety of a system is exploited to realise their equality. For each individual concept/construct the subordinate aspect is replaced with the respective opposite superordinate aspect. The behaviour is described by workflows while the structure is described by agents. This means that, from an agent perspective, the agent functionality and processes have been replaced by workflows. From a workflow perspective, the resources have been replaced by agents. However, this replacement doesn't mean that anything is lost. Rather, it is a question of representation.

In an agent system the behaviour is defined through agent functionality which is provided by agent processes. In an integration system the *behaviour of agents* is still defined as agent functionality. Nothing changes in this regard, but the agent functionality is actually provided by the workflows of the system.

On the other hand, the structure in a workflow system is defined through the resources executing the tasks of the workflows. In an integration system the *structure for the workflows* of the system are still the resources executing the individual tasks. Here, also nothing changes, except that the resources are actually agents.

Considering the system from an agent-centric viewpoint the only difference between agent system and integration system is that the behaviour is ultimately defined in workflows. For the agents the workflows still describe agent processes which make up the agent functionality. Considering the system from a workflow-centric viewpoint there is also only one difference between workflow and integration system, namely that the structure is ultimately defined by agents. For the workflows these agents are still the resources which execute tasks in the workflows.

Conceptually, neither the agent nor workflow sides need to adapt. Workflows still rely on resources as structure and agents still consider their functionality as behaviour. From

[1] It may be argued that, in the previous example, the agent is not the top of the hierarchy. It can be considered that the agent is being executed on an agent platform which is part of the overall multi-agent system. However, the direction of the hierarchy reverses in this step. In an agent-oriented and agent-centric view the multi-agent system consists of platforms which execute particular agents. This means that this is another branch of the hierarchy in which the agent is at the top.

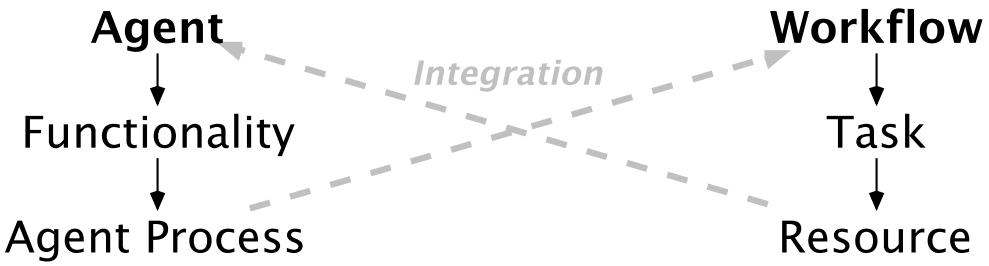

Figure 7.2: Nested integration hierarchies

the structure-centric viewpoint of agents and the behaviour-centric viewpoint of workflows there is only a difference in representation when it comes to behaviour and structure respectively. Yet, through the mutual replacement of the representation of that secondary aspect, the two concepts are incorporated into one another. This mutual incorporation is part of the foundation of the integration. In fact, it realises the desired equality of the abstraction levels and importance of agents and workflows.

The mutual incorporation solves the issues regarding abstraction levels by creating a nested hierarchy. On the agent-side the hierarchy is still topped by the agent construct. On the workflow-side the hierarchy is still topped by the workflow construct. Both of these hierarchies still exist. Agents and workflows are still used as modelling constructs. Each construct considers itself in a hierarchy with itself at the top. However, at some step down the relationships within the hierarchy are transformed. Instead of continuing down the hierarchy, the top of the respective other hierarchy is reached.

When considering the agent hierarchy (left-hand side) in Figure 7.2 the agent processes are transformed into workflows[2]. These workflows consist of tasks which are being executed by resources. This is the workflow hierarchy that is, through the integration, being added to the agent hierarchy. However, when continuing down that workflow hierarchy, the resources again are transformed into agents, which returns to the top of the agent hierarchy. This nesting creates a loop. The resources of the workflows are the agents which execute the workflows which are executed by the resources which are the agents and so on. This means that, while no new element is actually added to either hierarchy, they both contain each other in such a way that the abstraction level becomes meaningless. In the agent view the workflows are still the behaviour and subordinate to the agents. However, simply switch the view to the workflow view and the agents constitute the resources which makes them subordinate to the workflows. Since, as stated before, this is only a matter of representation and viewpoint the subordination and indirection simply do not exist in an integration. Agents are both subordinate and superordinate to workflows and vice versa. Essentially, they have been equalised.

While the mutual incorporation represents part of the foundation of the integration, the further details of the specification of the integration are largely dependent on the concrete technical integration approach. Issues that need to be addressed are more practical in nature and include, for example, the concrete management of agents and workflows in an integration, the technical interface between agents and workflows and the runtime

[2]The exact point where workflows show up in an integration is dependent on the concrete integration approach. It is possible that the whole functionality is modelled as workflows, but in this example the individual agent processes are regarded as the workflows. An analogous argument can be made for agents as resources.

focus. It is feasible for each of these issues to be handled in various ways. Management of agents and workflows may happen in parallel, merged or independent of one another. The interface between agents and workflows may be kept simple to exchange runtime and administration data or it might be a complex permanently updating mechanism. These kinds of practical issues build upon the conceptual integration and mutual incorporation and describe how the conceptual equality in modelling and abstraction levels is practically realised, supported and facilitated.

7.1.2 Modelling Constructs in an Integration

The second part of the foundation of the integration concerns the modelling constructs. Within an integration, agents and workflows should be equally important, should be handled on equal levels of abstraction and neither one should govern the other. The intention behind this equality is to enable utilising the strengths of agents for structural issues of system modelling and the strengths of workflows for behavioural issues of system modelling. Basically, agents and workflows should be equally treated and considered in all issues regarding modelling. Figure 7.1 illustrates this as a metaphorical Yin and Yang imagery. Agents and workflows should be in balance with one another. The mutual incorporation into each other, illustrated by the black speck in the white area and white speck in the black area, already realise much of this. However, the question that arises is the following: If agents and workflows are balanced and equal in all modelling issues, why are they still considered as a separate modelling construct?

This question can't be answered directly. In many cases it is beneficial to view agents and workflows separately. Much of the strengths of agents and workflows come from their ability to directly model elements of structure and behaviour. This ability allows the use of agents to represent components of large and distributed system with complex relations between them. It allows workflows to be used for representing and relating processes in inter-organisational contexts featuring intricate interdependencies. When it comes to representing and modelling issues directly associated with structure and behaviour, then agents and workflows should be considered as separate modelling constructs even in an integration. If agents and workflows are integrated so far as to be indistinguishable from one another, their strengths for this kind of modelling are diminished.

In other cases, though, it is beneficial to view agents and workflows in a less separated way. Strengths of agents and workflows also lie in the mechanisms associated with them and in their properties. Agents can, e.g., exhibit autonomy, mobility and intelligence. Workflows feature, e.g., task atomicity and loose coupling of resources and tasks (easier division of work). Making these strengths available at the same time requires agents and workflows to merge into a hybrid modelling construct. This is in fact what happens in agent-based WFMS and workflow-based agent management systems (see Sections 3.3.1 and 3.3.2), albeit in a limited and one-directional way (see discussion in Section 13.4). The ability to use properties and mechanisms of agents and workflows in the respective other construct is one of the major strengths of an integration. If agents and workflows are integrated too little, the conceptual gap between them as separate modelling constructs becomes too large to exploit this strength.

In yet other cases it is beneficial to have a modelling construct change from being an agent to being a workflow or the other way around. This is strongly related to the previous issue, as it deals with agent and workflow properties. However, it goes beyond this. A workflow may represent any process of the system. If that process gets stuck and can't continue its execution it could change to being an agent and autonomously attempt to solve the problem before turning back into a workflow and continuing as the process. This

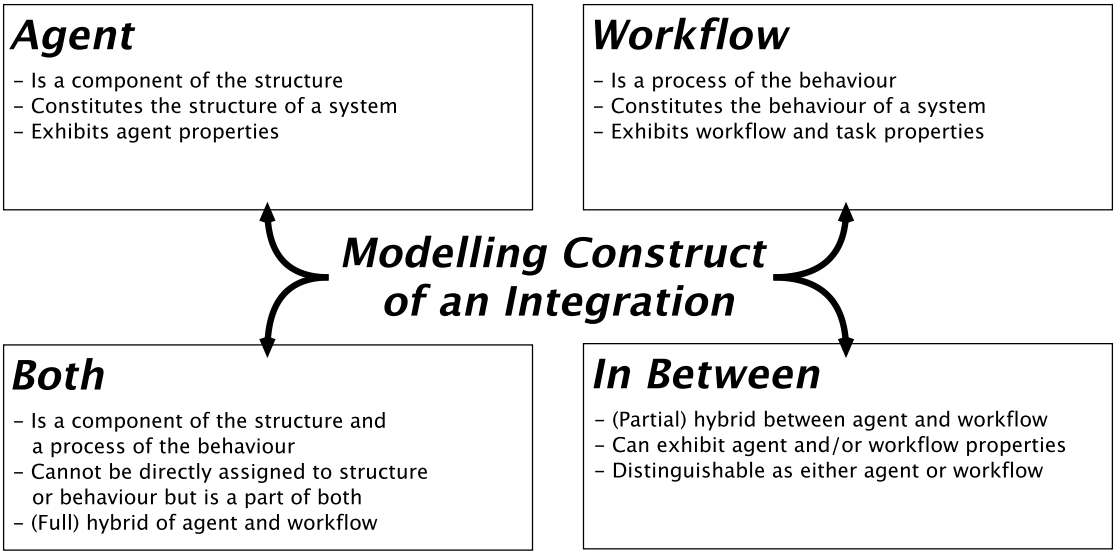

Figure 7.3: Concepts represented by the modelling construct of an integration

is different from just using agent or workflow properties as it completely changes the role and function of the modelling construct. The ability to dynamically change between structural and behavioural construct is another major strength of an integration of agents and workflows.

In summary, modelling constructs in an integration should be able to be seen as agents and workflows separately, should be able to be seen as a merged hybrid between agents and workflows and should also be able to dynamically switch between agent and workflow. This is why the question of why agents and workflows are considered separately even though they are equal in all modelling issues is so difficult to answer. Yes, they are equal and there are cases in which they should be used in a merged, hybrid fashion. But there are also cases in which they should be considered separately. Even more so, there are cases in which the modelling construct should dynamically change between agent and workflow. So the answer to the question is that agents and workflows should and shouldn't be considered as separate modelling constructs and also do so dynamically.

What this leads to is that, as direct modelling constructs, agents and workflows are not suitable for an integration between them. In general a modelling construct needs to represent a concept. In its role as a representative of that concept the construct needs to be able to act and be interacted with as that concept. For an integration the modelling construct needs to be able to represent four concepts. This is illustrated in Figure 7.3.

Modelling construct represents agents: The modelling construct needs to be able to represent agents, which means that it needs to be able to act as and be interact with as an agent. It needs to exhibit agent properties and also fill the same role concerning the structure of a software system as prescribed in Definition B.3. Since agents are components of the structure and also constitute the structure in agent systems, the modelling construct of an integration needs to be able to constitute the structure in an integration system.

Modelling construct represents workflows: The modelling construct needs to be able to represent workflows, meaning processes of the behaviour. Again, this means that it needs to be able to act as and be interacted with as an agent, as well as exhibit

workflow and task properties. It also hast to be able to constitute the behaviour in an integration system as prescribed in Definition B.6, just as workflows do for workflow systems.

Modelling construct represents both agents and workflows: The modelling construct needs to be able to represent merged hybrids of agents and workflows. A full hybrid of agent and workflow is concurrently active as both at the same time in the system. Agent and workflow aspects can't be completely distinguished in such a full hybrid and it exhibits the full potential, properties and mechanisms of both individual concepts. As such, a full hybrid is a component of the structure and a process of the behaviour at the same time and thus can't be directly assigned to either, even though it is a part of both.

Modelling construct represents something between agent and workflow: Something in between agent and workflow is a partial hybrid of the two. Agents that exhibit and use workflow properties and vice versa fall into this concept. Partial hybrids may still be clearly distinguished as agents or workflows, but are still hybrids of both, though to a lesser degree as the full hybrids.

The previous four ways a modelling construct can represent a concept are called the *states* of modelling constructs in an integration. Each state determines a different type, shape or form of a modelling construct. The term state was chosen here because it strongly implies a transient or even fleeting dynamic. In fact, an important point in all of these states is the dynamic.

The modelling construct needs to be able to dynamically switch between agent, workflow and (full and partial) hybrid states. At one point of the execution of a system it may act as and be interacted with as an agent, at another point it as a workflow. At yet another point it can also act as a hybrid described in the previous point. These switches can occur any number of times and in any number of ways that have been designed by the system modellers. Depending on the current state of the modelling construct, meaning whether it is agent, workflow or a hybrid, its role in the structure and behaviour and the properties it exhibits change accordingly.

Given the four concepts that need to be represented by the modelling construct of an integration it is not feasible to refer to the modelling construct in either agent or workflow terms. The modelling construct within an integration represents agents, workflows, both at the same time or something in between. It is, tough, in general, neither one of the four. It may take the shape of one of the four, but due to the dynamics no such shape is guaranteed to be persistent. The modelling construct of an integration is called an entity[3] or integrated entity in this thesis:

Key Term Definition B.7 (Integrated Entity)**.** An integrated entity (or entity for short) is the modelling construct within an integration of agents and workflows. It is capable of representing the different concepts that are required in such an integration. This means it can dynamically act as and be interacted with as an agent, a workflow, a hybrid of both or something in between.

The integrated entity is the second part of the foundation of the integration. The first part, the mutual incorporation of agents and workflows into one another, relates the concepts regarding equality of importance and abstraction levels. However, if the integration

[3]The term entity does **not** refer to entities from the context of entity-relationship models [Chen, 1976], general databases or any other field of computer science. Rather, it refers to the colloquial meaning of a generic thing that exists. As the entity as a modelling construct represents four different concepts, such a generic terminology is fitting.

stopped there, agents and workflows would still be separate modelling constructs. As discussed, this separation is desired in some cases, but without the ability to have dynamic and merged hybrid constructs the integration efforts would be insufficient. Without the entity construct the integration would simply be an unstructured combination of agents and workflows. The entity construct is a modelling abstraction that begins to provide a distinct form and function to the integration that enables exploiting the different strengths of the four concepts the entity is designed to represent.

As with the mutual incorporation in the previous section, more concrete details about entities and how they are realised can only be given in the context of a specific integration approach. Entities may fully implement the four concepts they represent or they may be a virtual representation of these concepts. Entity management is also an important issue, as it has to incorporate both agent and workflow management, as well as deal with the dynamic and hybrid aspects. These issues are practical in nature, but build upon the conceptual foundation provided by the entity construct.

7.1.3 The Integrated Modelling Perspective

The previous chapters discussed a structure-centric modelling perspective based on agents and a behaviour-centric modelling perspective based on workflows. In each of these perspectives the basic concept (i.e. agents or workflows) is ubiquitous, universal and pervasive. This is captured in the core ideas of *Everything is an Agent* and *Everything is a Process*. As discussed in the previous section, an integration uses a generic entity as its modelling construct. This current section now discusses how the integration in general and the use of an entity that can be agent, workflow, both or something in between affects the general modelling perspective. The result is an integrated modelling perspective, which abstractly and conceptually describes the enhanced modelling perspective referred to in the hypothesis of this thesis.

As with the agent-oriented and workflow-based modelling perspectives, the core idea can be summed up in the following way:

Everything is an Integrated Entity

In other words, this core idea states that every element of the system can be dynamically considered agent, workflow or a (full or partial) hybrid of both. Elements can dynamically change between its states but may also remain in a state.

The core idea represents a change in focus from the individual agents and workflows to entities. Yet, that change is more of an extension than it is a shift to something else. If one chooses to consider an entity as an agent, the modelling perspective is the same as the agent-oriented modelling perspective from Section 5.1. If, on the other, one chooses to consider an entity as a workflow, the resulting modelling perspective is the workflow-based modelling perspective from Section 6.1. Lastly, if entities are considered as hybrids the resulting modelling perspective is an amalgamation of the agent-oriented and workflow-based modelling perspectives.

Consequently, the integrated modelling perspective has three characterisations based on how entities are considered. These characterisations can change dynamically in accordance to the dynamic changes of the states within the entities. For entities considered as agents the agent-oriented modelling perspective is in effect. For entities considered as workflows the workflow-based modelling perspective is in effect. Further details of these two states are not discussed here as these perspectives were already discussed before.

It is, however, noteworthy that the consideration of entities in these perspectives requires them to be fully available in an integration. In other words, any integration approach

must fully support both the agent-oriented and workflow-based modelling perspectives. Otherwise, it would not be possible to support integrated entities as regular agents or regular workflows. While the ultimate goal of the integration is to utilise features from both, the restriction of the regular variants would severely impede modelling efforts. In some cases, entities as hybrids are simply not needed.

The amalgamation of agent-oriented and workflow-based modelling perspectives valid for hybrid entities is the focus of the remaining discussions in this section. The question that needs to be answered is how the modelling perspective is affected if the modelling construct is both an agent and a workflow at the same time. For partial hybrids this question is easier to answer. If an entity is considered as an agent, but utilises some workflow properties and mechanisms, the agent-oriented modelling perspective is still in effect. The analogous is true for a workflow that exhibits some agent properties. Here the workflow-based modelling perspective is still in effect. Generally, if an entity representing a hybrid concept can still be distinguished in a clearly structural or behavioural view the agent-oriented and workflow-based modelling perspectives are still valid.

For example, an entity representing a workflow that exhibits some agent intelligence in task to resource allocation is still a workflow. The same is true for an entity representing an agent that handles its human user interaction via a task interface. Still, the agent-oriented and workflow-based modelling perspectives are slightly enhanced. Depending on the degree of the hybrid integration, agent properties and mechanisms need to be taken into consideration in the structure of a workflow and workflow properties and mechanisms need to be taken into consideration for the behaviour of an agent. This extends the options and capabilities available in the modelling perspectives but does not change them in a conceptual way.

If, on the other hand, the hybrid concept is so fully integrated that the entity can't be clearly distinguished between agent and workflow, the agent-oriented or workflow-based modelling perspectives can't be used as easily. Issues regarding structure and behaviour from agent and workflow perspectives are merged in this case. One of the core foundations of the integration is the mutual incorporation of agents and workflows into one another. This means that behaviour of a system is modelled through workflows while structure is modelled through agents. If the entity used as a modelling construct is both agent and workflow it is part of both structure and behaviour. Still, it can't be clearly associated with either one exclusively without limiting the consideration.

The entity is an agent (structure) that has its functionality (behaviour) represented by a workflow. The entity is also a workflow (behaviour) that has its resources (structure) represented by a set of agents. This means that the entity constitutes both structure and behaviour in a dual way. It is not possible to separate behavioural and structural aspects when the entity is considered as a whole.

This means it is only possible to consider an individual part of the entity as either structural or behavioural. The entity can be represented partially as an agent or a workflow, and can thus be used in the traditional individual ways. However, this partial representation does not affect the entity as a full hybrid of agent and workflow. If it is considered as an agent it can be related to other agents or workflows but it is *still* also the workflow which may have different relations to possibly other agents and workflows, and vice versa.

What this ultimately means for the modelling perspective is the following. If an entity represents a full hybrid of agent and workflow, it can't be fully captured in traditional terms of structure and behaviour. Only partial considerations of that entity can yield useful views. Partial considerations means considering the entity as either an agent or a workflow and suppressing the respective other aspect in so far that the entity is only

considered as a partial integration (in which agent and workflow aspects can be clearly distinguished). For these partial considerations the agent-oriented and workflow-based modelling perspectives can be applied. A complete view of the full hybrid can only be achieved with (at least) two separate, partial views that individually capture the entity with agent and workflow focus.

A final issue regarding the modelling perspective concerns sets of entities. In the individual modelling perspectives there was no distinction in the modelling construct. Consequently, the modelling perspective didn't change between the individual consideration of modelling constructs and sets of modelling constructs. In an integration sets of entities may represent different concepts. For example, a modeller may consider two entities in the system, one representing an agent and one a workflow. At first glance, neither the agent-oriented nor workflow-based modelling perspective can be applied. The first wouldn't work for the workflow, the second wouldn't work for the agent.

At this point, however, the dynamic of the entity comes into play. If a set of entities is examined in which individual entities are considered as different concepts, the view on all entities can be changed dynamically to consolidate the modelling perspectives. An entity may be individually considered as an agent, but that doesn't mean that it can't be considered as a workflow in another situation. In fact, that is part of the essence of the integration. An entity **can** be anything of agent, workflow or both. Just because it is considered as one concept at one time, it doesn't mean it can't be considered as another concept the next time. When that change in perspective happens, though, the agent becomes a workflow with agent properties or vice versa. That means the considerations for partial hybrids of agents and workflows are in effect, but the modelling perspectives for individual agents or workflows can be applied.

Since everything is considered as an integrated entity, this can be applied to every element of the system in any configuration of entities representing different concepts. If the modelling perspectives of a set of entities do not match and are incompatible, modellers can simply change their consideration. Whether changing the consideration to all agents or all workflows is dependent on the specific needs of the situation, but in general both are continually available.

7.2 Application Areas

The current section discusses some of the application areas in which an integration can be applied well. Of course, the integration is not limited to these particular application areas. It can be used in any context for any kind of scenario. The following areas, though, possess special and particular requirements that can directly benefit from the nature of the integration.

Note that the integration is a complex endeavour, regardless of the specific approach that is ultimately chosen. The mutual incorporation of agents and workflows, as well as the utilisation of integrated entities that can dynamically exhibit four different states, increase the complexity of the required management facilities. Furthermore, from a modelling perspective, the capabilities of agents and workflows are combined, as well as additional hybrid and integration capabilities added. This increase in complexity can be considered as the cost of the integration. As a consequence, it is reasonable to assume that large-scale and complex systems will more directly benefit from an integration. Here, the envisioned benefits provided by the combination and integration of the strengths of agents and workflows can yield a more easily recognisable positive effect. This is reflected by the fact that, as indicated in the introduction chapter, the integration is intended more for

large-scale system than small-scale ones. These points are picked up again in the later discussion chapters.

This section examines application areas of the integration in three subsections. Section 7.2.1 discusses inter-organisational contexts. Section 7.2.2 then examines distributed software development. Finally, Section 7.2.3 discusses the general areas in which an integration can be applied advantageously.

7.2.1 Inter-organisational Contexts

Section 2.3.3 discussed inter-organisational workflows and defined inter-organisational BP in Definition 2.23. In summary, inter-organisational BP are BP that are executed by multiple organisations in collaboration. Real-world scenarios in which inter-organisational BP occur are called inter-organisational context in this thesis. This terminology is used in anticipation of the change from workflows as a modelling construct to integrated entities.

In inter-organisational contexts different organisations collaborate in some way to achieve a common goal. Often, that goal is too cost-intensive, difficult or even impossible to achieve by one organisation alone. One of the particularities of inter-organisational contexts is that the participating organisations are both actors *and* processes at the same time.

Organisations act autonomously. They, or rather their executives, decide on actions and reactions to stimuli from the outside. While the decisions may be done by actors within the organisation, the organisation, as a black box, performs the actions and reactions itself. Organisations also interact. In an inter-organisational context organisations exchange goods or services to further the collaboration. All in all, organisations are elements in inter-organisational contexts that act and interact in different ways. This is what makes then actors in these contexts.

Organisations also contain a process. If the overall inter-organisational BP is considered as the orchestration of all participating organisations, there is a subprocess for each organisation that contains all the tasks that are associated with a particular organisation. While these tasks are performed by elements within the organisation, they are clearly part of the operations of that organisation. When considering the organisation as a black box, these tasks are actually executed by the organisation directly, which leads back to the previous actor viewpoint. As such, for the inter-organisational context the subprocess of the organisation is represented by the organisation itself. This is what makes the organisation a process within an inter-organisational context.

By being both an actor and a process, organisations exhibit a duality in scope that is difficult to capture with just agents or workflows. If the inter-organisational context is modelled with agents, the process nature of organisations is neglected. If it is modelled with workflows, an organisation's nature as an actor is overlooked. What is needed for modelling the duality of organisations is a kind of hybrid between actor and process.

In an integration organisations can be represented by integrated entities. If these entities are considered as agents, which represent actors, that facet of the organisations modelled by the entities is highlighted. If these entities are considered as workflows, which represent processes, that facet of the organisations is highlighted. Additionally, entities enable the modelling and representation of hybrid facets of an organisation.

Hybrid facets in inter-organisational contexts involve situations in which the nature of a participating organisation can't or shouldn't be clearly distinguished between actor and process. The entity modelling an organisation in that case would represent the hybrid concept between agent and workflow. Such hybrids can be used, for example, when an organisation receives goods or services from another organisation. In that scenario, the organisation is clearly an actor which interacts with other actors, but in terms of process

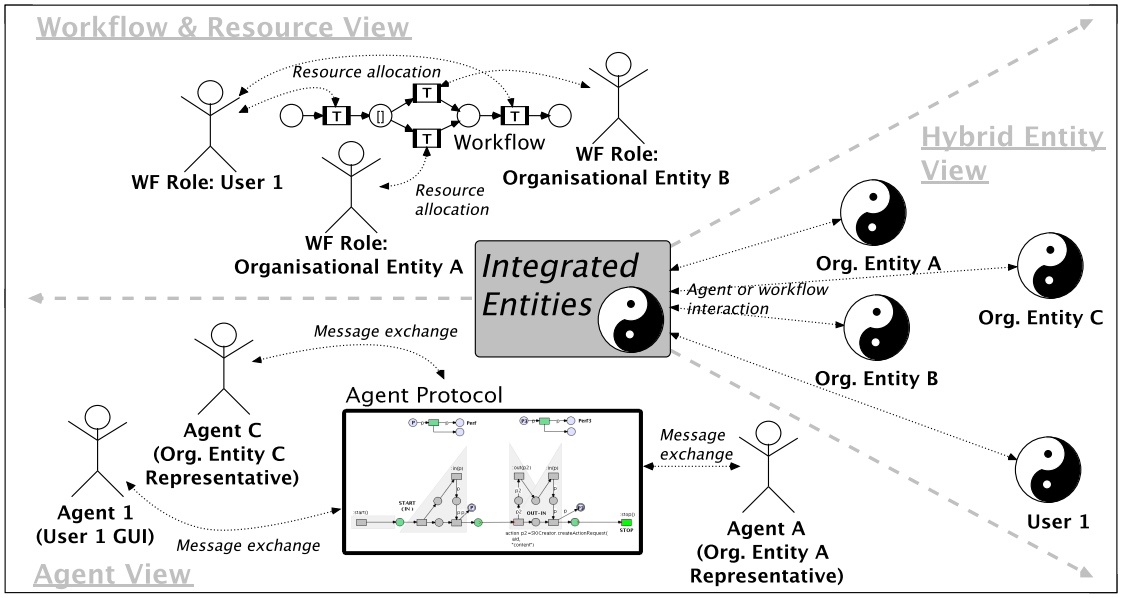

Figure 7.4: Entity views for inter-organisational contexts

the reception is also a task in the overall inter-organisational BP. In these kinds of scenarios the duality of actor and process or, more generally, structure and behaviour is not just alternating between the two facets but exhibiting the two facets simultaneously. An integration of agents and workflows allows for the modelling of inter-organisational contexts in which the duality, both simultaneously and alternating between actor and process, can be modelled and represented directly.

All in all, this kind modelling with integrated entities yields a threefold view on the inter-organisational context. This is illustrated in Figure 7.4. Organisations and all elements of organisations are practically modelled and implemented as integrated entities (centre element of Figure 7.4). Depending on the needs of modelling, modellers can consider these integrated entities in three (virtual) views. Note that the integrated entities at the core always remain integrated entities. The views only represent a logical perspective on these entities that highlights beneficial properties and mechanisms and conceals others that are not as useful or beneficial in specific situations.

If a workflow/process view is required, the entities are seen as workflows and resources (upper view in Figure 7.4). Here, all issues regarding behaviour, e.g. processes, tasks, resources, allocation, etc., can be considered and handled. If an agent/actor view is required, the entities are seen as interacting agents (lower view in Figure 7.4). In this view, issues regarding structure, e.g. actors, components, organisation[4], etc., are considered and handled. Finally, if neither individual view is sufficient for a problem/situation, the hybrid entity view can be used by modellers. In such situations, mechanisms and properties from both agents and workflows are required. The hybrid view combines the required mechanisms and properties and presents them in an entity view to the modellers. Modellers can then work with the entities to solve the issues that required more than just individual agents or workflows.

In summary, organisations in inter-organisational contexts exhibit a duality in being both actors and processes. An integration of agents and workflows as discussed in this chapter is capable of directly modelling and representing this duality. The integration allows

[4]Meant in this instance in regards to the structure, namely relations between components and administration.

modelling the complex interdependencies in the duality of organisations more realistically then it would be possible with just agents or workflows.

7.2.2 Distributed Software Development

In distributed software development, the development of a software system is split among multiple teams. The core of distributed software development is that the different development teams work completely independently and don't need to be in the same location geographically or work at the same time. They still need to interact with one another, though, and manage the overall collaboration, which are the main challenges of distributed software development. In other words, they are distributed and work autonomously but need to interact with one another in a systematic way.

Distribution and autonomy are predominant agent properties, while systematically organising interaction is a strength of the workflow domain. Consequently, an integration of agents and workflows is quite suitable for realising a support and facilitation for distributed software development. In fact, the requirements and nature of distributed software development is not too different from inter-organisational contexts. When considering each development team as an organisation, the basic premise is quite similar.

Development teams are actors that interact with one another. They also represent a process that they execute that is embedded in the context of an overall process to develop a software system. With these considerations the discussions, observations and results for the application of an integration to inter-organisational contexts can be directly applied here as well. Especially the duality of actor and process is in effect.

For concrete development approaches the duality can be even more pronounced. In Paose (see Section 2.2.3) a multi-agent system is considered in the two orthogonal perspectives of structure defined through agent roles and behaviour defined through agent interactions. This, in itself fits well with the integration approach that supports a similar view on systems. Even more so, Paose considers its view on multi-agent system as a guiding metaphor for any aspect of the development, including the development teams. If the development teams are considered with the same view on orthogonality of structure and behaviour, the integration can be directly applied there. Development teams can act and be interacted with in their agent role, which would have an entity modelling them exhibit the agent concept. Development teams also represent interactions and processes, which would have an entity modelling them exhibit the workflow concept. Entities can support these two facets easily and dynamically. But more so, the intersections between roles and interactions, meaning where agent roles would participate in an interaction, also need to be represented when supporting Paose. Entities support these facets of the distributed development in Paose by representing the hybrid concept.

7.2.3 Other Application Scenarios

The applications to Inter-organisational contexts and distributed software development already discussed the duality of actors and processes as a major issue that can be addressed by an integration. While this duality is especially pronounced in the previously discussed application areas, it can be found to some degree in any area. Every actor in a system, be it a software component, a human user or some automated mechanisms, has some form of behaviour. That behaviour may be extremely simple or highly complex, but it is always possible to identify some sort of actions that form a kind of process.

This means that any actor has a process. The actor can be considered as such, or it can be considered as a process. In fact, if it performs an action from within the process it can be considered as both the actor and the process at the same time.

Applying agents and workflows to actors and processes respectively enables, in turn, the application of an integration of agents and workflows. If the actor/process is modelled as an entity, it becomes possible to model and represent its inherent duality.

The open question is whether or not the actor/process can benefit from the integration. In inter-organisational contexts and distributed software engineering the interdependencies between structural and behavioural roles are not only important but can also be highly complex. Especially when different organisations/development teams interact issues can arise if process-related behavioural and actor-related structural interdependencies are not adequately supported. By enabling the direct modelling of these interdependencies through entities that can dynamically switch between agents, workflows and hybrids, the support of these scenarios is vastly improved. Direct modelling makes previously implicit issues explicit, which allows modellers to recognise and address them.

Of course, another asset of the integration is the ability to enable the use of concrete agent and workflow properties simultaneously. Here, any classical agent and workflow application domain can benefit from an integration. Workflow management can utilise agent intelligence, mobility or autonomy to better capture real-world situations or solve problems during runtime. Agent management can utilise the task concept to better allocate and distribute work or interact with human users in a standardised way. This is what is being done in agent-based workflow management and workflow-based agent management (see Sections 3.3.1 and 3.3.2). However, the integration goes beyond the capabilities of these efforts as it does not feature a focus on either exclusive agents or workflows. This is why the application areas in which such "partial" integration are useful can conceptually benefit even more from a "full" integration that features entities. The concrete benefits, however, are dependent not only on the concrete integration approach but also on the technical implementation. Consequently, this discussion is picked up again later, when both a conceptual approach and a technical implementation of an integration have been presented.

7.3 Integration Criteria

Before the next chapter deals with concrete integration approaches, the vision and specification of the integration needs to be refined in a way that makes an evaluation of the different integration approaches possible. For this purpose, the current section defines thirteen criteria for an integration. Each criterion deals with a facet or feature of the integration. Each of these facets or features corresponds to one or more issues discussed in this part of the thesis so far (Chapters 4 through 7). By means of these criteria, integration approaches are evaluated as to how well they conform to the vision and specification.

Overall, none of the criteria can be assumed with a simple checklist nature for evaluation. Each criterion defines a condition, which optimally supports the integration efforts. Thus, if a condition is fully realised, the integration is optimally supported in the current, particular facet or feature of the integration. The key word here is *fully*. For each criterion there is a range to which the criterion can be fulfilled, which is discussed for each criterion.

In general, the integration criteria can be classified into four categories. There is some overlap between categories, but all criteria can be reasonably associated with one main category. The first category relates to the *modelling construct*. It contains the criteria that describe how integrated entities should function and which properties they should possess. Criteria of this category are marked with the identifier **MC**. The second category relates to *structure and behaviour*. Criteria in this category deal with how structure and behaviour are seen in an integration, as well as the interface between them. Criteria of this category

are marked with the identifier **SB**. The third category deals with the *transfer of properties*, including criteria that describe how agent and workflow properties and mechanisms are shared between agents and workflows in an integration. Criteria of this category are marked with the identifier **ToP**. The fourth and final category deals with *management*. It contains criteria that, on a conceptual level, describe how integrated entities are managed. Criteria of this category are marked with the identifier **Mgt**.

These four categories and the criteria within them cover the entirety of the conceptual integration. The first two categories wholly cover the conceptual vision of the integration, as well as how the two main aspects of a software system (structure and behaviour) are handled. The third and fourth categories bridge the gap to the concrete approaches. They describe the conceptual basis for the transfer of properties, a desired core feature of an integration, and the management of integrated entities within an integration. These categories cover the issues as far as possible without assuming or associating any concrete, technical and implementation details. Such details would incorporate, for example, issues of entity organisation, monitoring/administration, Petri net aspects, concurrency, ontology or specific tools and techniques. Any such details would be dependent on concrete integration approaches and consequently can't be taken into consideration in these criteria as their goal is to objectively and independently evaluate different approaches.

The following descriptions are structured in the same way for all thirteen criteria. Each criterion description starts with a short introduction to the criterion providing the general context. Then, the criterion itself is stated and generally explained. This is followed by a discussion about the range of the criterion. Afterwards, the integration characteristics are discussed. This describes in how far agents and workflows are integrated, combined or just provided next to one another for the current criterion. Finally, the importance of the criterion as to its role in the overall integration is discussed.

Note that these integration criteria are repeated in the key terms glossary at the end of the thesis for easier reference in later chapters.

Integrated Entities

As described before, modelling in an integration can no longer be considered with just agents or workflows. Instead, the integration uses integrated entities (see Definition B.7) as a modelling construct. The current criterion describes the core idea behind integrated entities:

Criterion MC1: Integrated Entities

The main modelling constructs are integrated entities,
which can be agent, workflow, both or something in between.

What this means was discussed, in detail, in Section 7.1.2. In short, an entity can have different states. In each state it represents a different concept. It can represent an agent, in which case it emphasises the structure of a software system. It can represent a workflow, in which case it emphasises the behaviour of a software system. It can also represent a full or partial hybrid between agent and workflow.

Range: In a purely agent-oriented variant this criterion would state that the main modelling construct is an agent. For workflow management it would state the main modelling construct is a workflow. The former corresponds directly to the agent-oriented modelling perspective, while the latter corresponds to the workflow-based modelling perspective. These variants both describe the decomposition of a system into structural and

behavioural main modelling constructs respectively. They realise the weakest form of integration for this criterion.

The strongest form of integration is a realisation in which the main modelling construct completely complies with Definition B.7. The modelling construct can be agent, workflow or an arbitrary (i.e. full or partial) hybrid of the two.

In between the weakest and strongest forms realisations feature limitations to the modelling construct. These limitations can extend to the state of the modelling construct or the available features. For example, a relatively weak form of realisation would allow for a modelling construct to be an agent or a workflow, but not both at the same time. An example of a relatively strong, but not optimal, realisation would allow for the modelling construct to limit itself to partial hybrids instead of full hybrids.

Integration Characteristics: For this criterion the integration characteristics are straightforward. This criterion describes one of the core ideas of the integration of agents and workflows. In fact, it literally directly integrates agents and workflows into the novel modelling construct of integrated entities.

Importance of Criterion: As stated before, the concept of integrated entities is essential for any integration. An integration approach requires agents and workflows to not only be available at the same time, but also be available in the same element. Without integrated entities any approach would only feature agents and workflows next to one another. This means it is the basis for modelling in an integration in both the practical and mental (w.r.t. the way of thinking about a system) ways.

Entity Dynamic

The previous criterion described the basic states of an integrated entity, i.e. what concept an integrated entity represents. This current criterion builds upon those four states and concerns itself with dynamic between them. It describes how an integrated entity should be able to freely and dynamically switch between the states defined in Criterion MC1.

Criterion MC2: Entity Dynamic

Entities are dynamic in their states and can freely switch between them in order to represent different concepts.

The dynamic of integrated entities was also discussed, in detail in Section 7.1.2. In general, the dynamic refers to the ability that an entity can, at both modelling- and runtime, switch between being an agent, being a workflow and being a (full or partial) hybrid. If a sequence of situations requires an entity to be agent, then workflow, then agent, then hybrid and so on, a concrete integration approach should be able to accommodate this.

Range: The weakest realisation of this criterion is not directly defined by an agent or workflow variant. Instead, the weakest variant causes an entity once defined as agent, workflow or hybrid to always remain in that state.

The strongest realisation allows an entity to be in the state as it is needed at any point. It enables using the best tools and mechanisms from agents and workflows to be used in situations where they are most suitable. The strongest realisation would also allow having the entity itself decide, via some form of (agent) intelligence, to switch its state during runtime.

Other, less strong, realisations could limit the states of the entity. For example, an approach might allow an entity to be defined as an agent or a workflow. That entity might not be able to change its state fully to the other concept but it might be able to utilise mechanisms from the other concept. This would mean that it could not change from agent to workflow, and vice versa, but from agent to partial hybrid with an agent form.

One particularity of this criterion is that it is feasible to be irrelevant in some cases. If an approach does not feature any dynamic for entities, but always considers entities as full hybrids than this criterion is automatically fulfilled. Full hybrids are both agent and workflow at the same time anyway and can only be considered in agent and workflow views. This trivially realises the criterion as the consideration can be changed dynamically anyway.

Integration Characteristics: This criterion is responsible for integrated entities to be able to switch their state. It is no longer just possible to have an entity be agent, workflow or hybrid, but an entity can also dynamically switch between the three states. While previous Criterion MC1 described the basis of the integration, this current criterion presents an integral extension to that basis. This is also why it is being considered separately, even though it is strongly related to Criterion MC1.

Importance of Criterion: Dynamically changing between agent, workflow and hybrid states enables a large amount of flexibility in modelling a system. This flexibility can be used by system modellers to fully exploit the capabilities of both agents and workflows. An entity that can dynamically switch between its states can be used in any situation and can rely on any mechanism that is best suited for that situation, regardless of whether or not the mechanism is agent-oriented or workflow-based in nature.

If the criterion isn't fulfilled modellers are limited to static modelling constructs. The issues of static modelling constructs can be bypassed by starting a new instance of an entity in the required state and providing the necessary data, but only at the cost of modelling clarity, convenience and efficiency.

Logical Entities

This current criterion concerns itself with the logical view of elements of a system. It is consequently related to the core idea of the integrated modelling perspective of *Everything is an Integrated Entity* from Section 7.1.3.

Criterion MC3: Logical Entities

All elements of the system can be considered as logical entities.

What this criterion describes is the view on the elements of a system that an integration approach should take. While not every element of the system is practically implemented as an integrated entity, it should always be possible to take the logical perspective that an element could be one. This ensures that the system can be regarded in a way that is compatible with all modelling perspectives discussed in this part of the thesis.

Range: In its weakest realisation this criterion cannot be applied at all, since no element can be considered as an integrated entity. This would indicate a complete lack of integration efforts. Even with weak realisations of Criterion MC1 integrated entities would still be available as partially integrated agents or workflows. If no integrated

entity can be identified in even a purely logical perspective, the system does not integrate agents or workflows at all.

For this criterion a minimum baseline for the fulfilment can be set. At the very least, every practically implemented integrated entity can be seen as an integrated entity. This directly replaces the previously discussed weakest variant where no element can be regarded as an integrated entity. The consequence of this minimum baseline is that it can be clearly stated that the bigger the set of elements that can be regarded as integrated entities becomes, the better this criterion is fulfilled. The strongest realisation of this criterion then enables all elements of the system to be considered as entities, which would fully implement the idea of *Everything is an Integrated Entity*.

Integration Characteristics: This criterion integrates the different modelling perspectives. It describes that the integrated modelling perspective, discussed, in detail, in Section 7.1.3, can be used in the integration.

Importance of Criterion: The importance of this criterion lies in the way a system modeller considers the elements of a system, If every element can be considered as an integrated entity, it is possible to actually implement it as one. Of course, as discussed for the ideas of *Everything is an Agent* and *Everything is a Workflow*, not every element should be implemented as an integrated entity. However, it can be helpful to consider an element as one to solve particular issues and problems. In this case, these consideration include that the element can take the form of an agent, a workflow or a hybrid. This supports the modellers in their decision making.

Overall, this criterion introduces a mental flexibility into the modelling within an integration. It enables the utilisation of the different modelling perspectives discussed throughout this part of the thesis. In turn, this allows modellers to choose the best suited perspective on any element of the system they are creating.

Structure and Behaviour of the System

Definition B.3 defined the (concrete) structure of a software system. Definition B.6 defined the (concrete) behaviour of a software system. These definition are valid for the purposes and scope of this thesis. Consequently, any integration approach must comply with these definitions. The following criteria describe this compliance for the context of integrated entities. Since the discussions are analogous for both they are being handled in one description.

Criterion SB1: Structure of the System

Structure is constituted by entities considered as agents and platforms.

Criterion SB2: Behaviour of the System

Behaviour is constituted by entities considered as workflows.

Note that this is not a redefinition or further refinement of Definition B.3 or Definition B.6. Those definitions are still valid. The current criteria merely state how integrated entities are mapped and associated to these definitions.

Also note that, until now, entities representing agents have only been considered as regular agents. Criterion SB1 also considers them as platforms. In the view proposed by Paose and Capa, agent platforms are merely special agent implementations that possess

functionality to contain and manage other agents. This view is also adopted in this thesis. Consequently, the distinction between agents and platforms is important only for the sake of the structure definition, but irrelevant for the nature of the entities representing them.

Range: The range of these criteria is contingent on how the structure and behaviour of a system are formed. There is very little range here as the structure is either constituted by agents and platforms or entities representing agents and platforms and the behaviour is constituted by workflows or entities representing workflows. In the workflow dimension the structure is constituted by resources, which would not fulfil Criterion SB1. In the agent dimension the behaviour is constituted by functionality and agent processes, which would not fulfil Criterion SB2. The only way to fulfil these criteria is to have structure through agents and behaviour through workflows. This can only happen in combination in an integration.

Integration Characteristics: In these criteria there is very little integration in of itself. However, in unison they define how structure and behaviour are represented by integrated entities. In that way these criteria define the role of the different states of integrated entities in the overall modelling of the system.

Importance of Criterion: These criteria associate the concrete definitions of structure and behaviour to the particular context of an integration and integrated entities. As structure and behaviour were identified as the two main aspects of a software system these criteria are highly important. They define for an integration that the structure of the system is provided by entities representing agents and platforms and that behaviour of the system is provided by entities representing workflows. This excludes any other sort of structure or behaviour in an integration.

Mutual Incorporation

As described in Section 7.1.1 agents and workflows are mutually incorporated into each other in an integration. This circumstance is captured in the following two criteria. Just like with criteria Criterion SB1 and Criterion SB2 the descriptions are analogous for agents and workflows, which is why they are combined in this section.

Criterion SB3: Mutual Incorporation I

Entities as agents interact via workflows.

Criterion SB4: Mutual Incorporation II

Entities as workflows are executed by agents.

The first of these criteria captures that workflows replace agent processes as agent functionality. The second one captures that agents replace resources as the elements responsible for executing tasks in workflows. The second one also captures that agents act as workflow engines providing the execution environment for the workflows. As agents constitute the structure and workflows constitute the behaviour the two criteria also describe the main interface between structure and behaviour.

Note that these criteria highlight a significant difference between entities considered as agents and considered as workflows in the integration. When entities need to interact directly, they are considered as agents. When they, or rather the tasks within them, are executed, they are considered as workflows. This is a conceptual assignment of

responsibilities. Consequently, there is no need for criteria that state that agents are executed by something or that workflows interact via something. These kinds of mechanisms are the results of the overall integration.

Range: The weakest realisations of these criteria correspond to the using regular agent interactions for Criterion SB3 and workflow resources and engines for Criterion SB4. This again would correspond to there being no integration at all. The closer the realisation of agent interactions is to workflows, and workflow resources to agents, the better these criteria are fulfilled. In the strongest realisation the agent interactions are fully-fledged (entities as) workflows, while all workflows are executed by (entities as) software agents (as resources and engines).

Integration Characteristics: As described in Section 7.1.1 the mutual incorporation of agents in workflows and vice versa is one of the core mechanisms of an integration. Consequently, these criteria contain a large degree of integration. Agents are incorporated and merged into workflows and workflows are incorporated and merged into agents.

Importance of Criterion: As the mutual incorporation of agents and workflows into one another is a core mechanism of an integration its importance for an integration is correspondingly high. If agents and workflows are not incorporated into one another the integration cannot happen as the two concepts would only exist besides one another. In this way, this criterion enables a coupling between integrated entities beyond what agents and workflows can accomplish themselves These two criteria correspond to the realisation of the equality of the abstraction levels and importance of agents and workflows They also define the major interface between agents and workflows in integrated entities.

Properties and Mechanisms

One of the goals of an integration is to provide the properties and mechanisms of both agents and workflows for modellers to work with. This is why an integration should capture this circumstance.

Criterion ToP1: Properties and Mechanisms

Entities can exhibit agent and workflow properties and mechanisms.

What this criterion describes is that integrated entities can make free use of agent and workflow properties and mechanisms. It could be argued that this is just another facet of Criterion MC1, as entities representing agents, workflows or hybrids should automatically be able to exhibit the corresponding mechanisms and properties. However, the current Criterion ToP1 should be treated separately. It is feasible that entities might not be able to be seen as both agents and workflows, but might still exhibit agent *and* workflow properties. This would correspond to a partial integration like agent-based workflow management or workflow-based agent management. Consequently, the transfer of properties can happen independently from other integration criteria and should therefore be handled separately.

Range: In the weakest realisation, entities as workflows can only exhibit workflow properties and mechanisms and entities as agents can only exhibit agent properties and mechanisms. Hybrids of any kind are not possible in the weakest form. The weakest realisation, again, corresponds to no integration between agents and workflows

whatsoever. Stronger realisations are the partial integrations discussed before. Partial integrations describe entities acting as agents that can exhibit workflow properties and mechanisms but still retain their agent nature, as well as entities acting as workflows that can exhibit agent properties and mechanisms while remaining workflows. The strongest realisation of this criterion corresponds to a full hybrid that exhibits all properties and mechanisms of agents and workflow at the same time. Generally, this criterion is better fulfilled the more agent properties and mechanisms are available for workflows and vice versa.

Integration Characteristics: As far as integration characteristics, this criterion represents another kind of incorporation of agents into workflows and vice versa. Instead of the mutual incorporation describe in Criterion SB3 and Criterion SB4, it describes the transfer of mechanisms and properties.

Importance of Criterion: This criterion is very important for an integration as it relates directly to the strengths of an integration. Without the transfer of properties and mechanisms, the strengths of agents remain with agents and the strengths of workflows remain with workflows. This criterion enables the use of any mechanism and property in each state of an integrated entity. This means that without fulfilling this criterion an integration would be useless as agents couldn't gain any practical benefits from also being workflows and vice versa.

Agent Actions and Workflow Operations

Previous chapters identified the fundamental agent actions and basic workflow operations as the atomic steps in the execution of agents and workflows respectively. In that capacity, they represent information about the execution of agents and workflows on the highest level of detail.

Criterion ToP2: Agent Actions and Workflow Operations

Entities perform fundamental agent actions and basic workflow operations.

What this criterion means is that it should be possible to regard the execution of integrated entities in terms of the fundamental agent actions (see Definition B.1) and basic workflow operations (see Definition B.4). If it isn't possible to do so, the entities do not conform to the basic understanding of agents and workflows as described in this part of the thesis. Since that understanding was created and defined as general as possible, entities that do not conform to this view have to exhibit their agent- or workflowhood in some way that is incompatible with the traditional ways.

Note that this criterion can be seen as a specialisation of Criterion ToP1. Agent actions and workflow operations are mechanisms from agents and workflows respectively, which means they are captured in that capacity in Criterion ToP1. However, Criterion ToP1 dealt with the transfer of properties to exploit strengths, while this current criterion deals with the fundamental view on the execution of entities. Fundamental agent actions and basic workflow operations play a twofold role in an integration. They are both mechanisms of agents and workflows, but they also describe the execution in atomic steps. The former is captured in Criterion ToP1 and pertains to the flexibility of modellers to choose the best tools for different situations, while the latter, captured in the current criterion, creates a baseline for understanding and considering the execution of entities. Consequently, this criterion is handled separately from Criterion ToP1.

Range: The range of this criterion is relatively vague. As stated in the discussions in previous chapters, the fundamental agent actions and basic workflow operations are defined in a very general way. Other variants of actions and operations are possible, though they can always be mapped back to the ones discussed here. However, this possible mapping may make the evaluation more difficult.

Integration Characteristics: This criterion does not assume a direct integration. Rather, it only requires agent actions and workflow operations to be available in some way. Figuratively speaking, this criterion only requires agent actions and workflow operations to be available next to one another.

Importance of Criterion: The importance of this criterion lies in the ability of an integration approach to be able to apply the most basic understanding of agents and workflows. It concerns the core of how agents and workflows are regarded that underlies the agent-oriented and workflow-based modelling perspectives. Since the support of these modelling perspectives is essential to the integration, this criterion is also essential.

Regular Agent and Workflow Systems

As discussed in Section 7.1.3 an integration must also be able to implement traditional agent and workflow systems. The following two criteria describe this circumstance. Since they follow analogous explanations, they are combined into this subsection.

Criterion ToP3: Regular Multi-Agent Systems

Entities as agents can build regular multi-agent systems.

Criterion ToP4: Regular Workflow Systems

Entities as workflows can build regular workflow systems.

What these criteria describe is that if a modeller chooses to use entities only as agents, the resulting system is indistinguishable from a multi-agent system designed with a traditional agent approach with the same agent feature set. The analogous is true for a system using only entities as workflows. The indistinguishability is restricted to the features and external execution and performance of the system. Internally, the integration mechanisms can be arbitrarily complex. The direct consequence of these criteria is that the integration can't restrict the features of individual, traditional agents and workflows and also can't enforce integration features upon them.

Range: The range of these criteria is dependent on how close to a traditional agent or workflow system the system using integrated entities can come. If the systems are indistinguishable, the criteria are completely fulfilled, which constitutes the strongest realisation. The more differences there are to traditional agent or workflow systems, the weaker the realisation and the lesser the criteria are fulfilled. It is difficult to define a weakest realisation, as the differences can continue to amass. An example of a very weak realisation would be an agent system that can only interact in strict terms of workflows.

Integration Characteristics: These criteria put minor restrictions on the integration. They prescribe that it should always remain possible in an integration to build traditional agent and workflow systems. While it does not restrict the integration capabilities directly, it requires the integration features not to be imposed onto agents and workflows. This does not prohibit any integration features, but keeps them from being mandatory.

Importance of Criterion: These criteria are important because they enable the full support of agent-orientation and workflow-management in an integration. If both criteria are fulfilled by a concrete integration approach or framework, then regular agent and workflow systems can be built. This creates a baseline that encompasses the entire functionality provided by agents or workflows. Anything the integration adds to that functionality through hybrid entities is considered additional value.

These criteria also ensure that no functionality of traditional agents or workflows is lost in the integration. If that were possible, the integration would, in those areas, be less powerful than the individual concepts. This would be paradoxical to the goal of the integration which is to provide more than agents and workflows are capable of doing individually.

Functionality

In agent systems the system functionality is divided amongst the different agents and is the behaviour of the system. In workflow systems the system functionality is divided, in its description, amongst the different workflows. In an integration it must be possible to distribute the functionality among entities. This is trivially fulfilled if all entities are exclusively agents or workflows. In those cases, the statements from traditional agent-orientation and workflow management are valid. However, in an integration it also has to be possible for the functionality to be distributed amongst entities, even if these entities have different states.

Criterion Mgt1: Functionality

Functionality is distributed among entities that can have different states.

This means that it has to be possible to realise, define and describe what the system does not just between agents or workflows exclusively, but crossing over between entities as both agents and workflows. The fulfilment of this criterion ensures that the integration is not just superficial. Take, for example, partial integrations like agent-based workflow management systems. These systems feature agent-enhanced workflows that can exhibit properties and mechanisms of workflows. Nonetheless they emphasise the workflows as the main modelling construct. This means that, concerning management and modelling, the functionality of the system is only distributed amongst the workflows. These kinds of partial integrations are valid and useful in general contexts, but do not go beyond the scope of traditional workflow management (or agent management in the analogous case) and are thusly not suitably full integrations for the purposes of this thesis.

Range: The range of this criterion hypothetically ranges from functionality only distributed amongst agents or workflows exclusively in its weakest realisation to being freely distributed amongst entities in all possible different states in its strongest realisation. An integration approach that falls between these two extremes would separate specific parts of its functionality and assign agents or workflows to them. Variations in this separation are too extensive to generally state what characteristics would constitute a better fulfilment of this criterion. Specific separations may be advantageous in some settings, but useless in others. Consequently, the exact effects have to be examined and evaluated for each specific, concrete integration approach independently.

Integration Characteristics: The integration characteristics of this criterion concern what the system does, its functionality. It integrates agents and workflows in such a way

that all parts of the purpose of the system can be freely assigned to the most suitable entity state.

Importance of Criterion: As stated before this criterion ensures an integration beyond the superficial use of some properties and mechanisms. It states that the functionality of the system, its purpose and what it does, need to be able to be described by entities that represent both agents and workflows. This ability is what causes the integration to actually happen on the level of the individual concepts and not just on the level of properties. Therefore, this criterion is very important for any integration approach.

Hierarchies

Agents can exhibit logical hierarchies. Considering agents as platforms for other agents, and considering *Everything is an Agent* are all supported in the agent-oriented modelling perspective. Workflows, too, can exhibit such hierarchies. Here, the hierarchy idea is even more pronounced, as subworkflows being represented as tasks are an established mechanism. Consequently, entities need to be able to form hierarchies as well. But, similarly to Criterion Mgt1, the trivial case of hierarchies of entities exclusively representing agents or workflows, is uninteresting. These hierarchies are already possible with the traditional concepts and need to be supported by an integration trivially and according to Criterion ToP3 and Criterion ToP4. What an integration needs to support in addition to the traditional hierarchies are hierarchies consisting of both agents and workflows.

Criterion Mgt2: Hierarchies

Entities in different states can form logical hierarchies.

This criterion enables hierarchic relations between integrated entities on the logical level. It is, in essence, a more general variant of Criterion SB3 and Criterion SB4, but on a virtual level. It allows integrated entities of one state to have logical precedence and control over subordinate entities of other states. With this criterion, entities can logically contain or control entities of any state, which again can contain entities of any state and so on.

Range: This thesis assumes, in the context of the agent-oriented and workflow-based modelling perspectives, that agents and workflows can form logical hierarchies. Consequently, the weakest realisation of this criterion is that entities can only form hierarchies of agents or workflow exclusively. The strongest realisation does not impose any limits on the logical hierarchies of entities. Less strong realisations would impose some limitations, e.g. that the top of the hierarchy is always given by an entity as a workflow. The characterisations of the realisations differ similarly as those of Criterion Mgt1. Consequently, no general statement can be made about what constitutes a better or worse realisation without knowing the details of an integration approach. The ability to form logical hierarchies needs to be evaluated for each approach independently.

Integration Characteristics: This criterion describes how entities can logically form hierarchies. Logical hierarchies are important for modellers to determine and define prime elements in their systems that initiate, control or contain other elements. The integration characteristics of this criterion apply an integration way of thinking to those hierarchies and allow free association of entity states throughout hierarchies.

Importance of Criterion: This criterion combines agents and workflows on the level of logical entity management. This means that, through this criterion, entities of different

states can be set into hierarchic relation with one another. If this weren't possible, only entities representing the same state could be related hierarchically. Consequently, this criterion is important to ensure modelling flexible by not limiting the hierarchies of entities.

Additionally, logical hierarchies describe how elements are associated and which elements are initial or prime to the system. The top node of a hierarchy may describe the initial, or an initial, element of the system that initiates other parts. If these parts were enforced to be of the same state as the initial element, modellers would lose a vast amount of flexibility and the capabilities of the integration would be severely limited.

Criteria Conclusion

In the next chapter, the integration criteria are applied to a number of different integration approaches. There, they are used to evaluate the approaches in order to select the best option for implementation and further research.

To conclude the general description of the criteria, this current passage examines how traditional agents and workflows would be evaluated in the integration criteria. The results of this examination can be seen in Table 7.1.

Overall, it can be seen that, as would be suspected, agents and workflows both do not evaluate very well for an integration. They only completely fulfil the few criteria which take on a decidedly agent or workflow perspective (i.e. Criterion SB1 and Criterion ToP3 for agents and Criterion SB2 and Criterion ToP4 for workflows). These criteria describe features of an integration that are directly adopted from agent-orientation or workflow management. In these cases, the respective other traditional concept does not fulfil the criteria at all.

Most of the criteria are partially fulfilled by agents and workflows. These are the criteria that combine and merge features from agent-orientation and workflow management. Consequently, both agents and workflows fulfil their part of the criterion while failing at the respective other parts. The criteria in question are Criterion MC1, Criterion MC3, Criterion ToP1, Criterion ToP2, Criterion Mgt1 and Criterion Mgt2. Even though these criteria are partially fulfilled, they are only trivially partially fulfilled, as nothing is added to agents or workflows beyond what is traditionally available. Consequently, the evaluation in these criteria would also be poor.

The final criteria Criterion MC2, Criterion SB3 and Criterion SB4 are not fulfilled at all by either agents or workflows. These are the criteria that actually assume not just a combination of agents and workflows, but an actual integration. These criteria require agents and workflows to be integrated in at least some way to be fulfilled. Traditional agent and workflow concepts cannot fulfil these criteria.

7.4 Thesis Results and Requirements

This chapter specified an integration of structure and behaviour based on agents and workflows. It described the vision of an integration, some application areas for it and finally the concrete criteria which an approach needs to fulfil in order to realise this vision. In this way this chapter and part established the concrete context for the goal of this thesis.

To reiterate from the introduction: "In this thesis the two concepts agents and workflows are combined and integrated in order to create the hybrid system. The research, development and provision of this combination and integration are the overarching *goal* of the thesis. On a conceptual level this goal is constituted by a conceptual model and approach for the

Criterion	Traditional agents	Traditional workflows
Criterion MC1: Integrated Entities	Partial (only agents)	Partial (only workflows)
Criterion MC2: Entity Dynamic	N/A (only agent state)	N/A (only workflow state)
Criterion MC3: Logical Entities	Partial (only agents)	Partial (only workflows)
Criterion SB1: Structure of the System	Fulfilled	N/A (Structure through resources)
Criterion SB2: Behaviour of the System	N/A (Behaviour through agent interactions)	Fulfilled
Criterion SB3: Mutual Incorporation I	N/A	N/A
Criterion SB4: Mutual Incorporation II	N/A	N/A
Criterion ToP1: Properties and Mechanisms	Partial (only agent properties and mechanisms)	Partial (only workflow properties and mechanisms)
Criterion ToP2: Agent Actions and Workflow Operations	Partial (only agent actions)	Partial (only workflow operations)
Criterion ToP3: Regular Multi-Agent Systems	Fulfilled	N/A
Criterion ToP4: Regular Workflow Systems	N/A	Fulfilled
Criterion Mgt1: Functionality	Partial (functionality in agents)	Partial (functionality in workflows)
Criterion Mgt2: Hierarchies	Partial (agent hierarchies)	Partial (workflow hierarchies)

Table 7.1: Integration criteria applied to traditional agents and workflows

motivated combination and integration. This model and approach are supplemented by a set of prototypes constituting a working proof-of-concept realisation and implementation. This proof-of-concept is the goal of this thesis on the practical level."

The combination and integration of agents and workflows is exactly what has been specified in the vision in Section 7.1. With this, one result has already been established. It serves as the foundation and guiding ambition for further results, which build upon it to create new concepts and practical realisations. Therefore, it is now possible to further refine the goal of the thesis into concrete requirements.

These requirements also need to take the research questions into account. To reiterate from the introduction:

1. "How can the concepts "agent" and "workflow" be combined and integrated in a reasonable, conducive and beneficial way?"
2. "Which beneficial effects can be achieved through a combination and integration of agents and workflows, potentially and substantiated through a technical proof-of-concept, and in which scenarios are they best applicable?"

Note that, for readability reasons, the following only refers to an integration of agents and workflows instead of a combination and integration. As the discussions in Section 7.1 have shown, the combination is included in and subsumed by the integration. The integration provides an additional abstraction onto the combination, as well as a form and function to it. Consequently, it is easier and more concise to consider an integration of agents and workflows as the main topic of the results.

One of the first issues that needs to be clarified is which specific results need to be obtained in order to achieve the goal and answer the research questions. The following describes the partial results that need to be obtained. Together, these partial results constitute the integration of agents and workflows as specified in the goal and research questions of the thesis. They also describe the basic research process followed in this thesis.

A specification of the integration: The detailed nature of an integration of agents and workflows was unspecified at the beginning of this thesis. Therefore, the concrete specification of what the goal and research questions refer to is the first partial result required for this thesis. In fact, this current part of the thesis, especially this current chapter, have already provided that result in the form of the vision of an integration.

A conceptual integration approach: After establishing the specification of the integration, the next partial result requires the definition of a conceptual integration approach. That integration approach describes how, in concept, the integration specification can be realised. In other words, the approach prescribes how concretely agents and workflows are (combined and) integrated in a system. The integration approach constitutes the "conceptual model and approach" referred to in the goal of the thesis.

To achieve this partial result different integration approaches are developed on an abstract level and evaluated against the specification of the integration. The criteria for this evaluation have already been established in Section 7.3. From that evaluation the best fitting approach is selected and then developed and refined into the required partial result. This way of developing the integration approach ensures that the main research, development and realisation efforts for this thesis take the best direction and also contributes to answering the first research question.

A technical proof-of-concept for the conceptual approach: Having established the conceptual model for the integration in the previous partial result, the next step is to

implement the model in a practical and technical system. This system represents the working proof-of-concept for the conceptual integration approach and refers to the practical part of the goal of this thesis. By implementing the conceptual approach its technical feasibility is ensured and it is shown that agents and workflows can in fact be integrated as prescribed.

The proof-of-concept is to take the form of a development framework with which it is possible to build applications utilising the now implemented integration concepts. Using an agile-like development approach for the proof-of-concept, which takes the form of a set of iterative and incremental prototypes, assures that unforeseen issues and challenges can be adequately addressed. The prototypes have achieved the proof-of-concept status when the developed system constitutes the desired framework for the creation of integrated agent and workflow applications and when that framework fully aligns with the conceptual model.

A set of application prototypes using the integration: The final partial result is a set of application prototypes built with the proof-of-concept for the conceptual approach. This set of prototypes serves a twofold purpose. First, it confirms that the proof-of-concept integration framework, and by extension the conceptual integration approach, can indeed be used to build practical applications. Second, it establishes a foundation on which to discuss the properties and mechanisms realised through the integration, by applying the integration specification, model and proof-of-concept to a practical scenario. It is only possible to determine the effects the integration has on system modelling by providing such applications.

Overall, the nature and direction of this thesis is *conceptual*. This means that the focus clearly lies on the first two partial results, the specification of the integration and the development of the conceptual model. The practical aspects, including the proof-of-concept and the application prototypes, are still important but their main value is to supplement and support the conceptual results.

In order to achieve the goal and answer the research questions posed in the introduction, the developed integration needs to be examined and discussed thoroughly. To facilitate the discussions, the following presents a set of requirements to the results. These requirements are largely based on the goal and research questions, yet also include more general concerns.

Alignment with the integration vision: The goal of the thesis described in the introduction refers to a desired combination and integration of agents and workflows without specifying its details. That specification was developed in this current part of the thesis and presented as the integration vision in Section 7.1. The integration vision establishes the basic frame of reference for any result concerning the integration. Therefore, all other partial results of the thesis, especially the conceptual integration model and its proof-of-concept must align with the integration vision.

Technical feasibility of the developed concepts: This requirement refers to the creation of the proof-of-concept. More specifically, it requires establishing that the developed practical system is, in fact, a working proof-of-concept realisation and implementation for the developed integration approach. In other words, to fulfil this requirement it must confirmed that the practically developed system aligns with the concepts and mechanisms prescribed in the conceptual integration model. Note that technical feasibility here means that the conceptual approach can, in fact, be technically implemented as intended and planned in a particular modelling/implementation language. For this thesis that language is provided by reference Petri nets using Java as an inscription

language. Other implementation options, e.g. using other programming languages or implementations for (mobile) operating systems, are outside of the scope of this thesis. This requirement must be fulfilled, because otherwise conceptual and practical results of this thesis do not match.

Capability to build applications utilising the integration: The proof-of-concept implementation of the integration approach is, as prescribed in the partial result description, intended as a framework for the creation of applications that utilise both agents and workflows. This current requirement ensures that it is actually capable of doing so and is used to ensure part of the usefulness of the results. If it were not fulfilled, it would not be possible to model and implement anything with the results of this thesis.

Transferability: The context of this thesis clearly emphasises Petri net-based agent and workflow models. The reasoning behind this, as exposed in the introduction, is to provide a common and well-understood basis for the models on which to build the integration. However, the results of this thesis should provide answers applicable beyond the current Petri net context. That way researchers from other fields can examine applying the concepts developed here in their own work. This requirement does not prescribe general transferability, but simply that the results are not *only* valid for Petri net-based models.

Note that this requirement does not include the actual transfer to other models beyond Petri nets. In other words a transfer to and realisation of the results in other contexts should be possible and feasible, but the transfer is not actually performed.

Improvement and conceptual maturity: This requirement ensures that the results of this thesis improve in some way upon the state-of-the-art related to agents, workflows and their integration, meaning the results must have some additional value over other, related work. These additional benefits emphasise the conceptual results, i.e. the developed integration approach, over the technical results, i.e. the proof-of-concept implementation, due to the conceptual nature of this thesis. This means that the benefits and improvements must refer to the conceptual approach but must only be substantiated as feasible in the technical realisation. In other words, the benefits need to be distinguished as reasonably realisable if not for technical limitations and open points in the prototypical implementations. Of course, the technical results should strive to actually implement as many benefits as possible, yet technical issues and limitations should not interfere with this requirement in a conceptually focussed thesis.

As a consequence of this requirement and the conceptual nature of this thesis, it should be noted that developing a practical system to rival commercially available systems and performing large-scale application tests are outside the scope of the thesis.

Synergies: A purpose of the desired integration is to utilise or exploit it to gain additional benefits through the synergies between agents and workflows. These synergies need to be highlighted and exposed. While this can be seen as a specialisation of the previous requirement, it should be treated explicitly and separately, because it emphasises the basic interplay between agents and workflows and not (only) the improvement related to the state-of-the-art. The previous requirement related to the concrete results (models and prototypes) of this thesis, while this requirement examines the results and takes beneficial lessons from it.

Inter-organisational contexts and other application areas: Section 7.2 examined a number of application scenarios in which the integration is envisioned to be useful, first and foremost inter-organisational contexts. Those discussions only take the integration

vision into account. The actually developed integration model and proof-of-concept need to be examined w.r.t. the application areas. Inter-organisational contexts were motivated as a major research focus and the results need to exhibit beneficial properties at least in this area. This requirement therefore insists on an examination and evaluation of the results for the application in inter-organisational aspects. Other application areas should also be examined and evaluated, but are considered as optional by this requirement.

The partial results, requirements, research questions and goal are all picked up again near the end of the thesis in Section 12.7. That section evaluates the individual aspects described here and determines if the goal of this thesis has been achieved.

Part C

Conceptual and Practical Integration

Part C presents the conceptual and practical contributions and main results of this thesis. Chapter 8 proposes a number of abstract integration approaches developed for this thesis. These approaches are based on the integration vision established in the previous parts and are evaluated based on the defined integration criteria. Chapter 9 defines the Agent-Activity integration approach. It concretises the idea of an integration using agent actions and workflow operations and provides the blueprint for further implementations. At its core, the Agent-Activity approach contains the Agent-Activity modelling construct. Around that construct a reference architecture is created that describes how a system supports and manages Agent-Activities. The implementation of the Agent-Activity integration approach is presented in Chapter 10. The Paffin-System fully realises the integration approach as a framework with which to build applications. These applications can utilise the properties and mechanisms of agents, workflows and the integration of both.

The purpose of this part is to describe the main results of this thesis. The Agent-Activity integration approach represents the main conceptual result. It constitutes the conceptual model and approach of the integration of agents and workflows desired by the goal of this thesis. It describes how to integrate agents and workflows in a methodical way. The Paffin-System, on the other hand, represents the main practical result. Concerning the goal of this thesis, the Paffin-System establishes the proof-of-concept implementation of the conceptual model and approach of the integration.

8 Possible Integration Approaches

This current chapter presents and evaluates different abstract integration models and approaches that have been developed for this thesis. These models and approaches are abstract in so far, that they represent only detailed sketches of their integration efforts. They are not fully refined. The result of this chapter is the evaluation and selection of one abstract approach that is developed further into a more concrete and fully realised approach and finally implemented in the next chapters of the thesis.

The chapter contains two sections. Section 8.1 describes the different approaches. The concepts and ideas behind each one are described, followed by an individual evaluation according to the integration criteria and further practical aspects. Afterwards, Section 8.2 performs the final, overall evaluation of the previously described approaches. It compares the individual evaluations to one another and finally selects the approach that is further developed in the remainder of this thesis.

8.1 Developed Integration Approaches

This section presents six abstract models of possible integration approaches that have been developed for the context of this thesis. Early work and discussions on these models has been published in [Wagner and Moldt, 2011].

The six abstract models approach the challenge of an integration of agents and workflows in different ways. The first four approach the challenge via facets of the vision of an integration described in Section 7.1. These facets include the management facilities, the entities/agents/workflows and the actions and operations. The final two models are focussed on utilising reference Petri net mechanisms to achieve the integration.

Overall, neither the set of approaches nor their respective models are intended to be exhaustive or complete. As the vision of an integration was set out to be general in nature, there is a multitude of possible integration approaches and even more models to represent them. The approaches presented here are designed to be general enough so that most other approaches can be considered as specialisations of them. These approaches are also, from an intuitive perspective, the most promising ones.

Additionally, the selection and development of the models of these approaches was intentionally restricted. This thesis assumes a Petri net-based context, which is why only models following the Mulan agent model and workflow nets were considered. This does *not* mean that the integration approaches or the concepts behind the models are not viable for or applicable to other agent or workflow models. It only provides a fixed basis for the selection, description and evaluation and also lays the groundwork for the implementation in the intended Petri net-based practical context. The general applicability of the integration approaches and results is discussed at the end of the thesis.

Evaluation Process The evaluation process is handled identically for each integration model. First, the model is described. Focus here is given to the core concepts and ideas, i.e. the integration approach behind the model. Afterwards, the evaluation according to the integration criteria defined in Chapter 7 is applied.

Next, the model is practically evaluated and its overall practical/technical aspects are discussed. This is handled in an additional, special criterion, the practicability. That criterion is not concerned with the integration aspects or based on the integration vision and therefore defined here separately.

Criterion Pra: Practicability

The integration approach is practically and technically feasible.

The issues discussed for practicability regard the technical realisations and the later usability in modelling. For example, feasibility of realisation, predicted required effort of realisation, any issues for the implementation, difficulties/inconveniences for modellers and issues in management are points that are discussed. These kinds of issues are entirely dependent on the approach and model and not issues regarding the integration directly, which is why they are not captured by the regular integration criteria. The range, integration characteristics and importance w.r.t. the integration are also not applicable. Practicability is, of course, highly important in general and it can range from poor practicability (i.e. an approach very difficult to realise) to very high practicability (i.e. an approach that is easily realised).

Following the discussions of the integration criteria and practicability, a short discussion of further aspects of the approach and/or model is provided. This captures all aspects that don't relate directly to the integration or practical issues. Finally, the evaluation is concluded by a summary of features, advantages and disadvantages.

8.1.1 Integration via Management

Integration via management approaches the integration at the level of the management of the main modelling constructs. The approach is based on the dissertation thesis of Christine Reese [Reese, 2010]. The model is adjusted to the context, terminology and practice of this current thesis.

Core Idea

The core idea of this model and approach is to perform the integration effort on the level of agent and workflow management. Both management systems are provided and integrated into what [Reese, 2010, p. 141] calls a novel "unit" management system. On top of that novel management system agent/workflow applications are built.

Internally, all agent behaviour (agent-internal and interactions between agents) is implemented as workflows. This means that the system can be viewed from an agent or a workflow perspective. In the agent perspective the individual components and subsystems are focussed on. Analogously, in the workflow perspective the processes of the system are focussed on.

Figure 8.1 illustrates the core idea. Agent and workflow management are integrated into an integrated management. That management facility can then be used to create applications using agents and workflows that can freely and without restrictions interact with one another using the management facilities. Note that the unrestricted interaction between agents and workflows is handled via the integrated management, which is why that connection passes is part of both management and application as indicated in Figure 8.1.

For more details of this approach refer to Section 3.3.3 where the dissertation thesis of Christine Reese is discussed.

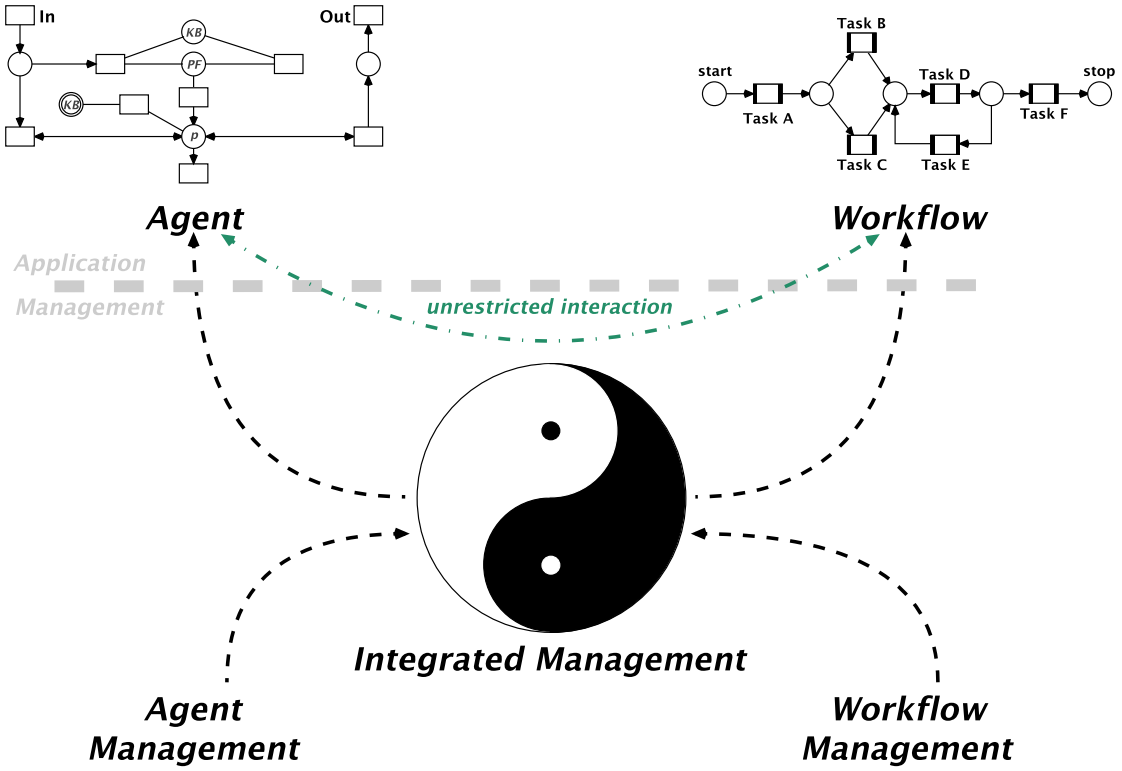

Figure 8.1: Integration via management

Criteria Evaluation

Criterion MC1: Integrated Entities: This approach and model has to be considered separately for the management and application modelling levels. For application modelling a clear separation for agents and workflows persists. It is possible to switch between structure and behaviour perspectives during runtime, but the constructs under consideration are always either agents or workflows. These agents and workflows may exhibit properties of the respective other concept, but beyond this partial hybridisation there is no full hybrid as defined for integrated entities in Definition B.7.

On the management level the novel units are similar to integrated entities. [Reese, 2010] does not elaborate on the nature of the units, only that they are the result of an integration of agent and workflow management. Consequently, it is feasible that they can be realised as integrated entities as defined in this current thesis. This current model assumes this realisation. However, the units are restricted for the application modelling level in that they do not feature a full hybrid state between agent and workflow. Each unit is either agent or workflow at any one time. This is also reinforced by the approach, which has the management system using separate agent and workflow management systems making the unit or entity management only available virtually.

Ultimately, both perspectives do not fully feature integrated entities as defined in Definition B.7. Application modelling separates agents and workflows too much and the internal management does not feature a fully hybrid state. Overall this means that this criterion is only fulfilled in a limited and partial way.

Criterion MC2: Entity Dynamic: With the restrictions discussed for the previous criterion in place, the units in this approach can dynamically switch between agent and workflow state. Elements of the system can be considered in both agent and workflow terms, although only exclusively. This means that this criterion is fulfilled as far as the restrictions from Criterion MC1 can allow it.

Criterion MC3: Logical Entities: Due to the internal unit management system it is possible to consider all agent/workflow elements of the system as integrated entities, keeping in mind the restrictions to integrated entities stemming from Criterion MC1. There is no limitation to any other element of the system being considered as a logical entity.

Criterion SB1: Structure of the System: The model and approach are compatible with the concrete definition of structure. All structural elements are either agents or platforms or can be represented by agents or platforms (e.g. human users or databases).

Criterion SB2: Behaviour of the System: The model and approach are compatible with the concrete definition of behaviour. All agent behaviour is realised through workflows. Since all structural elements are agents or platforms, this means that all behaviour in the system is realised with workflows.

Criterion SB3: Mutual Incorporation I: Agent internal and external behaviour is realised through workflows. This means that this part of the mutual incorporation is completely fulfilled.

Criterion SB4: Mutual Incorporation II: While agent behaviour is realised as workflows, these workflows are only executed by the agents as resources. The agents are not the engines of these workflows. The workflows are executed in the workflow management system behind the unit management system. This means that this criterion is only partially fulfilled for agents as workflow resources.

Criterion ToP1: Properties and Mechanisms: Both agents and workflows in the application can utilise properties and features from the respective other concept. However, this transfer of properties is limited in so far that an agent can't become a workflow at the same time (see the limitations for Criterion MC1). This means that transfer of properties is only fully possible for partial integrations of agents and workflows which still emphasise either agents or workflows in the foreground.

Criterion ToP2: Agent Actions and Workflow Operations: Agent (internal and external) behaviour realised as workflows satisfies this criterion. Agents still exhibit the fundamental agent actions, although they are coupled here with workflow tasks in the workflows that describe the behaviour. That means that both actions and operations are supported in both agents and workflows.

Criterion ToP3: Regular Multi-Agent Systems: It is possible to exclusively use agent functionality to create a regular agent system.

Criterion ToP4: Regular Workflow Systems: It is possible to exclusively use workflow functionality to create a regular workflow system.

Criterion Mgt1: Functionality: The functionality in this model is distributed amongst agents and workflows. However, all aspects of the functionality concerned with behaviour are distributed amongst only workflows, while all aspects of the functionality concerned with structure are distributed amongst only agents. This means that this criterion is only partially fulfilled, as the functionality aspects cannot cross the boundary between agents and workflows.

Criterion Mgt2: Hierarchies: Hierarchies are fully supported in this approach. Units are explicitly stated to be able to contain other units (i.e. agents and workflows).

Criterion Pra: Practicability: The development plan of the approach is provided in detail in [Reese, 2010]. In summary, it can be realised by starting off with regular agent and workflow management systems and advancing the integration through five tiers. After the regular management systems on the first tier, the second on features agent-based workflow management and workflow-based agent management. The third tier combines the possibilities of the two variants of the previous tier and provides them in an unrestricted and unstructured way. Tier four then restricts and structures the integration for both agents and workflows separately again. The fifth and final tier again combines the two variants of the previous tier to provide a structured and reasonably restricted integration of agents and workflows. That fifth tier corresponds to the model described in this section.

While feasible, the realisation of this approach as described in [Reese, 2010] is cumbersome. Providing an unstructured integration (third tier), then restricting it (fourth tier) just to combine it again (fifth tier) is conceptually sound and can provide a good understanding of the integration. However, the consequence is that the fifth tier entails all of the technical encumbrances from the previous tiers in addition to its own.

Approaching the fifth tier, i.e. the current model, directly without following the described approach is possible, though also relatively difficult. Agent and workflow management systems feature different abstraction levels. A Mulan agent system consists of platforms, on which agents are executed which execute protocols. A workflow system consists of workflow management systems that execute workflows. Workflows contain one less level when viewed this way. There is also the duality of tasks concerning engines and resources to consider in workflows. All of this means that a direct integration of the management systems would have to contend with the conceptual and abstraction differences between agents and workflows. Overcoming these differences and providing a technically and conceptually sound realisation of an integrated entity management system would be difficult.

Additional Aspects

There are no additional aspects to discuss for this approach.

Conclusion

In conclusion, the integration via management systems is a conceptually well founded though ultimately practically ineffective model of the integration. It does not feature integrated entities on the direct modelling level and the entities used in the background do not fully realise the potential defined for integrated entities in the vision of the integration from Section 7.1. Structure and behaviour of a software system are well covered here, although the mutual incorporation does not feature agents as workflow engines. The transfer of properties criteria are all fulfilled, although the restrictions stemming from the limited scope of the integrated entities prohibit the use of full hybrids of agents and workflows. From the management criteria, the boundary crossing of functionality is limited, yet logical hierarchies are fully supported. The practicality of the approach is rather poor, since the approach described in [Reese, 2010] entails a lot of technical encumbrances and a new, direct approach is difficult due to the difference in abstraction in agent and workflow management. The individual results are summarised in Table 8.1.

Criterion	Evaluation	Comment
Criterion MC1	limited	Restricted units instead of integrated entities
Criterion MC2	full*	Dynamic changes in units
Criterion MC3	full*	All elements can be units
Criterion SB1	full	
Criterion SB2	full	
Criterion SB3	full	
Criterion SB4	limited	Agents are not workflow engines
Criterion ToP1	slightly limited	Transfer of properties only for partial hybrids
Criterion ToP2	full	
Criterion ToP3	full	
Criterion ToP4	full	
Criterion Mgt1	limited	No boundary crossing for functionality
Criterion Mgt2	full	
Criterion Pra	poor	

Table 8.1: Evaluation overview for integration via management

Overall, this approach to integrate agents and workflows via the management systems should not be chosen for the practical realisation in this thesis. The limitations with regards to the integrated entities cascade throughout the facets of the integration and would prevent a realisation of this approach to achieve the full envisioned potential. Combined with the poor practicality of the approach, this leads to an overall weak evaluation. The approach is, however, conceptually well founded and has, after all, inspired the research for this thesis. [Reese, 2010] distinctively elaborates a vision of an integration of agents and workflows that, regrettably, falls just short of the vision set for this thesis.

8.1.2 Intermediate Model

Integration via an intermediate model uses integrated entities in the background of the system. For system modelling distinct agent and workflow facets of the internal integrated entities are used.

Core Idea

The core idea of this model and approach is to use integrated entities only in the background. The integrated entities used here have distinct facets that represent the entity as an agent and the entity as a workflow to the outside of the system. Figure 8.2 illustrates the core idea. For system modellers, only the agent and workflow facets are available. However, the facets are only virtual representations of parts of the integrated entity. This means that changes in one facet that cause changes in the other one are automatically adopted. For example, adding a task to the workflow facet automatically adds the task and the corresponding agent behaviour to the agent facet. All active elements of the system are realised as integrated entities and system modellers can freely choose the facet they need at any time.

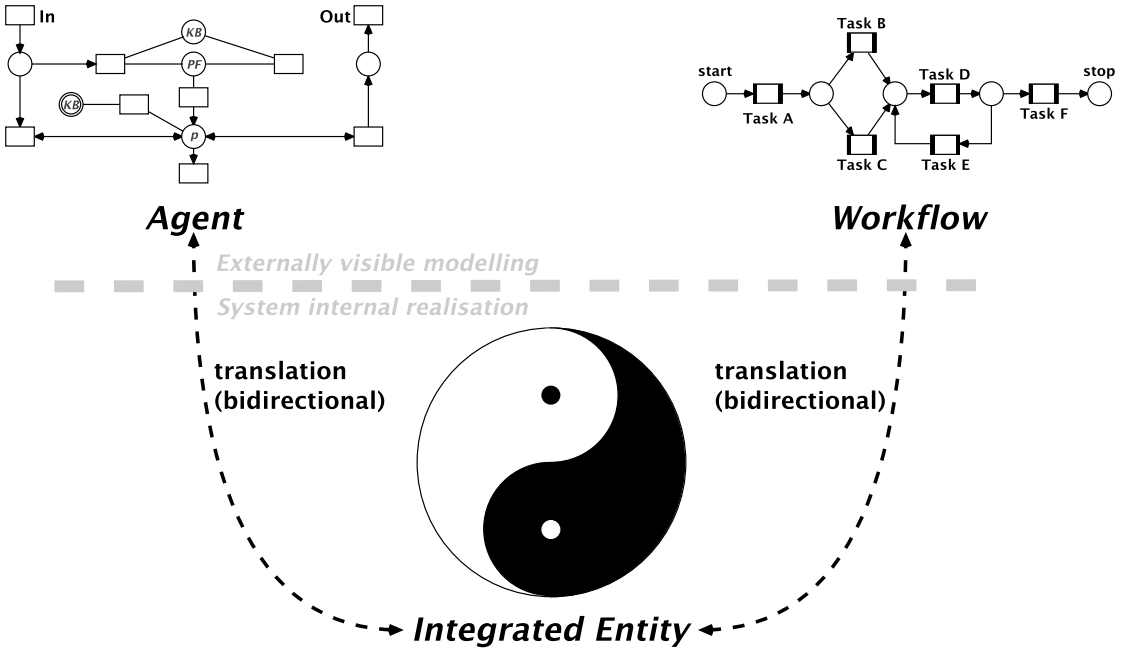

Figure 8.2: Intermediate model

Criteria Evaluation

Criterion MC1: Integrated Entities: As with the previously discussed approach integration via management, the main issue with the intermediate model in regards to the use of integrated entities is that there is a clear distinction between agents and workflows for the system modeller. The limitations are not quite as severe as with integration via management though. Modellers see only the agent and workflow facets at any one time. The facets are only representations of those parts of the integrated entities that correspond to agent or workflow functionality and mechanisms. This means, however, that, unlike with integration via management, an integrated entity can be a full hybrid of both agents and workflows. Modellers and monitors are only able to see the agent or workflow facets at any one time. The other facet can be concurrently active, modellers and monitors only need to switch their perspective. Still, even though the switching of facets is possible at any time, the restriction to modelling only one facet at any time is still a limitation w.r.t. this criterion. At best, it is an inconvenience for modellers and monitors, at worst it impedes or even prohibits modellers from utilising advanced and complex integration mechanisms by obscuring them between the facets.

Criterion MC2: Entity Dynamic: On the management level integrated entities are completely dynamic in their state. Still, they can only be viewed as agents or workflows on the application level at any given time. This represents only a minor limitation for this criterion, because the dynamic is still fully available for the execution.

Criterion MC3: Logical Entities: On the management level this criterion is completely fulfilled. However, on the application level, elements can only be considered as agents or workflows separately. While it is possible for all elements to switch between agent and workflow facets, this criterion is concerned with how modellers may consider the elements of the system. In contrast to Criterion MC2 it is not enough for this criterion

to have the integrated entities available in the background and be able to switch between the facets here. Consequently, this criterion is severely limited for this model and approach.

Criterion SB1: Structure of the System: All structural elements are realised as integrated entities. Considering these entities as agents fulfils this criterion. Platforms, as special agents, are also covered here.

Criterion SB2: Behaviour of the System: All behavioural elements are also realised as integrated entities. When these entities are considered as workflows, this criterion is fulfilled.

Criterion SB3: Mutual Incorporation I: Behavioural elements of the system are realised through integrated entities using their workflow facets. This includes the interactions between integrated entities as agents, consequently fulfilling this criterion.

Criterion SB4: Mutual Incorporation II: Structural elements of the system are realised through integrated entities using their agent facets. This includes all resources participating in any tasks in integrated entities as workflows. Regarding engines, for the workflow facet of each entity the agent facet of that entity is the workflow engine. Therefore, this criterion is completely fulfilled.

Criterion ToP1: Properties and Mechanisms: As all agents and workflows in an application are just facets of the integrated entities in the background, the transfer of properties is fully realised. Both the agent and workflow facets can access, utilise or depend on the concurrently running mechanisms from the respective other facet.

Criterion ToP2: Agent Actions and Workflow Operations: As all integrated entities can be considered in their agent and workflow facets, the entities can perform all agent actions and workflow operations.

Criterion ToP3: Regular Multi-Agent Systems: It is only possible to simulate regular multi-agent systems in this approach and model. All integrated entities are workflows and agents at the same time, even if only one facet is visible or used at any time. To simulate a regular multi-agent system, one can ignore the workflow facets, but they can't be switched off.

Additionally, workflow functionality replaces the behavioural functionality (e.g. protocols, interactions and decision components) of agents. It is not possible to exclude or turn off the workflow functionality from the behaviour, but it is possible to limit and restrict it in such a way that regular agent behaviour can be simulated. This can be done, for example, by simplifying resource and role descriptions and disabling task atomicity etc.

Criterion ToP4: Regular Workflow Systems: As with regular multi-agent system it is only possible to simulate regular workflow systems. Agent facets can only be ignored, not switched off. Also, the structure of the system is realised through integrated entities using their agent facets. Consequently, any workflow system built in this model needs to realise their resources as agents. The agents can be kept extremely simple, but they can't be removed from the system.

Criterion Mgt1: Functionality: As all integrated entities can exhibit their agent or workflow facets, the functionality is freely and completely (including boundary crossing) distributed among the entities in any state.

Criterion Mgt2: Hierarchies: Logical hierarchies of entities are fully supported.

Criterion Pra: Practicability: From a practical side, this approach and model are difficult to realise. Two major challenges need to be addressed. First, the intermediate model needs to be defined. This model needs to incorporate structure through agents and behaviour through workflows fully, meaning that every relevant mechanism from agents and workflows needs to be made available. Not all mechanisms are compatible and can simply be made available concurrently. For example, agent autonomy clashes with controlling and management mechanisms from workflows. Solutions for these kinds of issues need to be found and incorporated into one unified intermediate model.

The second challenge is directly related to this. After having defined the intermediate model a translation towards the agent and workflow facets needs to be defined. Irrelevant parts of the intermediate model need to be masked as to not infringe on the accuracy of each facet. Also, the translation also needs to ensure that changes made in a facet can be translated to the intermediate model. This translation of changes needs to ensure the integrity of both facets and the intermediate model.

Overall, these two challenges represent major obstacles for the realisation and implementation of this model and approach.

A practical aspect besides the realisation regards the structure and behaviour. While Criterion SB1 and Criterion SB2 are completely fulfilled, they are, in a way, overly fulfilled. Structure consists of integrated entities using their agent facets. Behaviour consists of integrated entities using their workflow facets. However, each integrated entity also concurrently has its respective other facet active, even though it is not visible. This means that the structure also consists of workflows and the behaviour also consists of agents. Being only facets of the integrated entities behind this means that this does not violate the definitions of structure and behaviour or the corresponding criteria in itself. However, these circumstances muddle up the clarity and directness of how the structure and behaviour are seen. It becomes more difficult to distinguish them clearly.

Additional Aspects

The approach and model are, in nature, similar to the integration via management discussed before. For application modelling both integration via management and the intermediate model provide only agents or workflows. The main difference between the two lies in how and where the integration is provided. Integration via management takes two management systems and integrates them into a new unit management system that provides a combined and integrated interface. The intermediate model integrates agents and workflows directly. In integration via management there are actual agents and workflows active in the system, in the intermediate model all elements are integrated entities. This results in the differences in evaluation described above.

Conclusion

In conclusion the integration via an intermediate model is a promising approach. It suffers from some limitation regarding the use of the agent and workflow facets for application modelling. Integrated entities are only available in the background, while modelling uses only agents and workflows separately. While this does not directly prohibit the possibilities of the integration achieved here, it makes accessing and utilising these possibilities more difficult. Modellers need to work around the limitations in order to fully utilise everything that is generally possible.

Criterion	Evaluation	Comment
Criterion MC1	limited	Only agent and workflow facets are modelled, no direct integrated entities
Criterion MC2	slightly limited	Dynamic only complete in background
Criterion MC3	limited	Limited considerations due to agent and workflow facets
Criterion SB1	full*	Integrated entities for structure are also workflows
Criterion SB2	full*	Integrated entities for behaviour are also agents
Criterion SB3	full	
Criterion SB4	full	
Criterion ToP1	full	
Criterion ToP2	full	
Criterion ToP3	limited	Only simulation. Workflow facets can't be switched off.
Criterion ToP4	limited	Only simulation. Agent facets can't be switched off.
Criterion Mgt1	full	
Criterion Mgt2	full	
Criterion Pra	poor	

Table 8.2: Evaluation overview for the intermediate model

Structure and behaviour in this approach and model correspond well to the criteria and definitions. However, as discussed, neither the workflow nor the agent facets of integrated entities can be turned off. This leads to difficulties with the distinctions and handling between structure and behaviour.

Mutual incorporation is ensured, as well as the transfer of properties and the use of agent actions and workflow operations. The realisation of regular multi-agent system and workflow systems is limited, as they can only be simulated. Functionality and hierarchies are fully supported.

From a practical point of view, the model and approach evaluate poorly. The realisation faces two major challenges in the intermediate model and the correct translations between the different facets. Also, the above mentioned ambiguity of distinction between structure and behaviour is a negative factor. Table 8.2 summarises the evaluation of the intermediate model.

Overall, this model and approach are not suitable for further development in the context of this thesis. The limitations regarding the application modelling (only agent and workflow facets, instead of direct integrated entities) are too significant. The remainder of criteria evaluate adequately to completely, but the issues regarding modelling with entities are not balanced out. The poor practicality also counts towards the disadvantages of this model and approach.

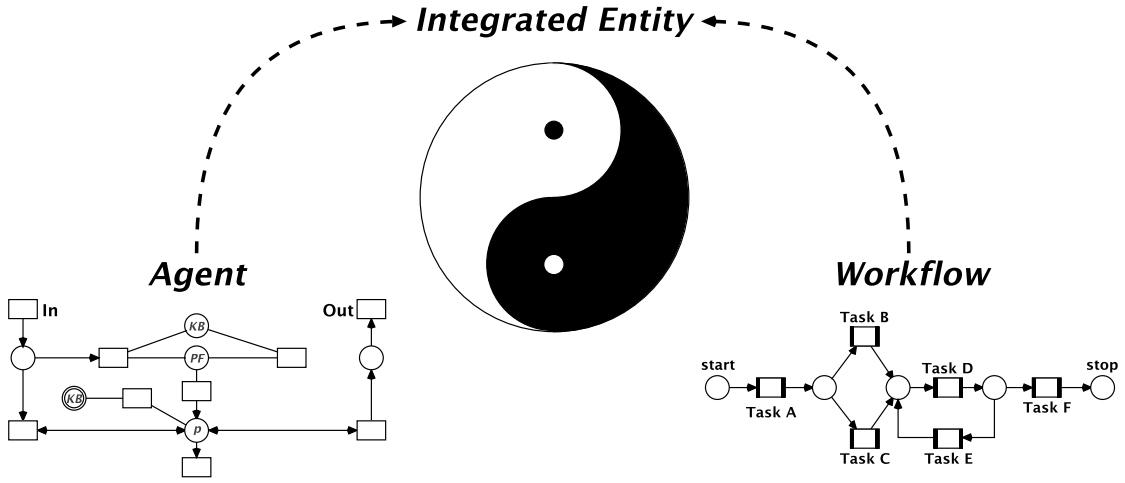

Figure 8.3: Combined model

8.1.3 Combined Model

Integration via a combined model utilises integrated entities as the main modelling construct. Opposed to the previously discussed integration via an intermediate model, the combined model is not translated into agent or workflow facets, but rather is directly available for system modelling.

Core Idea

The core idea of the combined model is to integrate agents and workflows in such a way that modellers have direct access to the integrated entities for modelling. It is illustrated in Figure 8.3. The integrated entities in this approach completely comply with Definition B.7. It is very similar to the approach and model of the intermediate model, but the difference is that there is no translation into facets for agents and workflows. Rather, the modellers directly use any mechanism and functionality of both agents and workflows as they see fit. As with the intermediate model, all active elements of the system are realised as integrated entities.

Criteria Evaluation

Criterion MC1: Integrated Entities: Integrated entities are directly available to the modellers. These entities comply with Definition B.7, which means they can be agent, workflow, partial hybrid or full hybrid. Consequently, this criterion is completely fulfilled.

Criterion MC2: Entity Dynamic: The integrated entities used in the combined model are fully dynamic, which means that they can freely switch between their possible states.

Criterion MC3: Logical Entities: Since there are no restrictions to the integrated entities used in the combined model, it is possible to consider all elements as integrated entities.

Criterion SB1: Structure of the System: Integrated entities in the agent state constitute the structure of the system, fulfilling this criterion.

Criterion SB2: Behaviour of the System: Integrated entities in the workflow state constitute the behaviour of the system. Consequently, this criterion is fulfilled.

Criterion SB3: Mutual Incorporation I: Behavioural elements of the system are realised through integrated entities in a workflow state. This includes the interactions between integrated entities as agents, fulfilling this criterion.

Criterion SB4: Mutual Incorporation II: Structural elements of the system are realised through integrated entities as agents. This includes all resources participating in any tasks in integrated entities as workflows. Regarding engines, entities as workflows are executed on their own. It is possible to see the entity as an agent as the engine for its own workflow, however, unlike with the intermediate model, that agent side may be undeveloped beyond the basic and default capacities of an agent. Consequently, it is not as directly possible to assume that each workflow is executed by an engine provided by itself as an agent. This weakens the fulfilment of this criterion slightly.

Criterion ToP1: Properties and Mechanisms: As integrated entities feature a dynamic state of agent, workflow and hybrid they can utilise any property and mechanism from any state. This fulfils this criterion completely.

Criterion ToP2: Agent Actions and Workflow Operations: Integrated entities can be both agents and workflows dynamically. This means that they can also perform agent actions and workflow operations without restrictions.

Criterion ToP3: Regular Multi-Agent Systems: Regular agent systems can be built using the combined model. However, similarly to the intermediate model, the workflow aspects can only be ignored and not completely shut off. This means that the combined model also can only simulate a regular multi-agent system. Still, the limitation to this criterion is not as strong as with the intermediate model. In the intermediate model the change to workflow facets ultimately would enforce the use of workflow functionality. In the combined model integrated entities can actually function as pure agents without accessing workflow functionality. From a conceptual point of view the multi-agent system is only simulated, yet from a practical point of view it is a regular multi-agent system.

Criterion ToP4: Regular Workflow Systems: Regular workflow systems can be built using the combined model. Conceptually, the realisation of a regular workflow system would only be a simulation. Practically however, there is no distinction between the simulation and a regular realisation. The arguments are analogous to the ones made for Criterion ToP3.

Criterion Mgt1: Functionality: As all integrated entities can dynamically exhibit agent or workflow states, the functionality is freely and completely (including boundary crossing) distributed among the entities in any state.

Criterion Mgt2: Hierarchies: Logical hierarchies of entities are fully supported.

Criterion Pra: Practicability: The main challenge for the realisation of this approach and model is the development of the combined model itself. Similarly to the intermediate model, the combined model needs to capture all elements of agents and workflows at the same time. While the intermediate model would restrict or remove certain mechanisms and features that clashed between agents and workflows, the combined model needs to provide the entire feature set of both agents and workflows. This is due to the fact that integrated entities in the combined model need to be able to exhibit a classic state without the background help of another facet. The need for

the incorporation of the entire feature set significantly complicates the development of the model. On the other hand, the second challenge of the intermediate model, i.e. the translation into facets, is eliminated. Overall though, the realisation is similarly difficult to the creation of the intermediate model.

Note that, in contrast to the intermediate model, modellers have full control over the state of each integrated entity. Consequently, the issues regarding structure and behaviour becoming less distinguishable and do not apply here. Each entity can exhibit the different dynamic states, but the structure is always constituted by agents while the behaviour is always constituted by workflows.

Regarding the practicability of this approach another issue has to be discussed. The integration of the combined model is highly unstructured. Agents and workflows are set next to one another and their entire feature set made available. This is similar to the third tier of the integration approach described in [Reese, 2010]. Like the combined model, the third tier is immensely powerful and expressive. However, without a structured approach towards the integration this power and expressiveness is difficult to harness by system modellers. The possibilities are overwhelming and may impede one another making an efficient utilisation hard to achieve. The combined model is also, from a technical point of view, bloated, as certain mechanisms and features from agents and workflows are, in their purpose, redundant.

Overall, the practicality of the combined model is poor. It is difficult to realise and suffers from the same issues as the third tier of the integration from [Reese, 2010] suffers from.

Additional Aspects

There are no additional aspects to discuss for this approach.

Conclusion

The combined model is very expressive and powerful. However, it is difficult to both realise and, when realised, properly utilise to gain advantages from the integration. Regarding the integration criteria, the model and approach evaluate very well. Integrated entities are directly available to the modellers, their state is dynamic and every element can be regarded in terms of integrated entities. Structure and behaviour are also fulfilled according to the criteria and the mutual incorporation is present, although there is no prescription of agents being the engines for workflows. The transfer of properties, as well as the use of agent actions and workflow operations, is fully supported. The realisation of regular agent and workflow systems is possible and, even though this is conceptually still only a simulation. Functionality is freely distributed amongst the integrated entities and hierarchies are supported. Table 8.3 summarises the individual points of the evaluation.

Overall, the combined model is very promising from its evaluation of the integration criteria. However, the practicality is very poor. Not only is the approach and model difficult to realise, it is also difficult to utilise once it has been realised. The possibilities for system modellers are too overwhelming and complicated to be used efficiently. Consequently, this model and approach should not be implemented further.

8.1.4 Integration via Actions and Operations

Integration via actions and operations utilises the fundamental agent actions (see Definition B.1) and basic workflow operations (see Definition B.4) as the basis of an integration.

Criterion	Evaluation	Comment
Criterion MC1	full	
Criterion MC2	full	
Criterion MC3	full	
Criterion SB1	full	
Criterion SB2	full	
Criterion SB3	full	
Criterion SB4	limited	Agents as engines is not necessarily prescribed
Criterion ToP1	full	
Criterion ToP2	full	
Criterion ToP3	slightly limited	Conceptually a simulation with no practical side-effects
Criterion ToP4	slightly limited	Conceptually a simulation with no practical side-effects
Criterion Mgt1	full	
Criterion Mgt2	full	
Criterion Pra	poor	

Table 8.3: Evaluation overview for the combined model

It makes actions and operations available to integrated entities whose state is defined by which actions or operations they currently execute.

Core Idea

The core idea of this model and approach is similar to that of the combined model. System modellers have access to an integrated entity as defined by Definition B.7. Unlike the combined model though, the integration is not realised directly through the integrated entities. Rather, it is realised through the actions and operations the integrated entities perform. In other words, the integrated entities are blank containers that have all the potential functionality of both agents and workflows. Only through the actions and operations that system modellers define for them is that potential accessed and the different states of the integrated entities constructed.

An agent is an integrated entity that executes only agent actions. A workflow is an integrated entity that executes only workflow operations. A (partial) hybrid is an integrated entity that executes both agent actions and workflow operations, but not at the same time. A (full) hybrid is an integrated entity that executes both agent actions and workflow operations simultaneously. Figure 8.4 illustrates the integration via actions and operations.

Criteria Evaluation

Criterion MC1: Integrated Entities: The model and approach use integrated entities as the main modelling construct. As described in the core idea, the deciding factor if an integrated entity is an agent, workflow or both are the actions and operations it executes. Consequently, this criterion is completely fulfilled.

Criterion MC2: Entity Dynamic: The state of an integrated entity is dependent on the agent actions and workflow operations it executes. Actions and operations are fleeting,

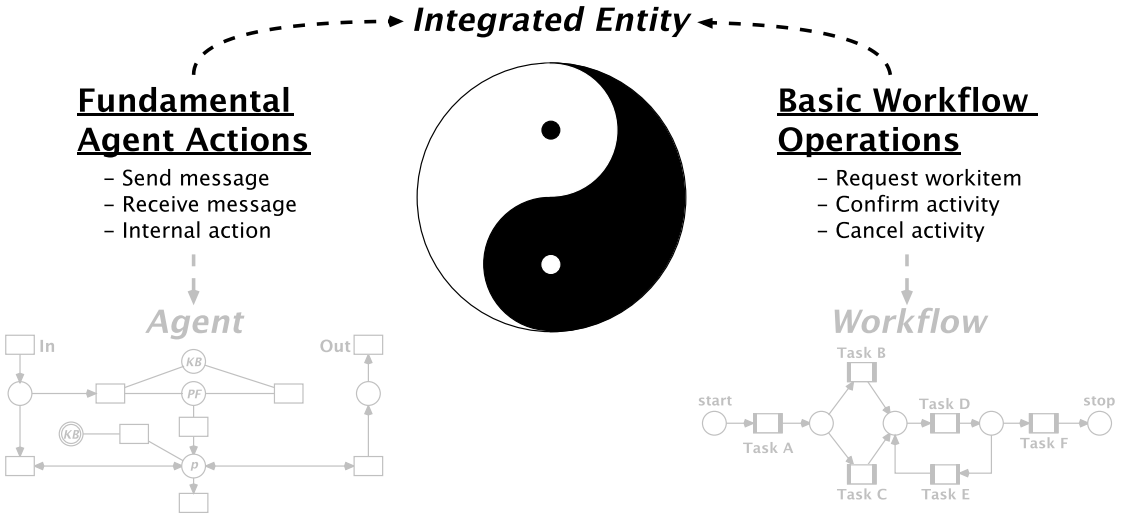

Figure 8.4: Integration via actions and operations

meaning that they are limited in duration. That means that an integrated entity in one state can change into another state by performing other actions or operations. For example, an entity can communicate as an agent and then turn into a workflow and provide a task. Afterwards it can provide another task that internally uses agent intelligence to support a human user. In that scenario the entity is first an agent, then a workflow and then a hybrid. These changes are completely dynamic, thus completely fulfilling this criterion. Still, the fleeting nature of actions and operations may also be detrimental. An entity is only in a specific state as long as the action or operation is executed. To fully utilise an integration, certain situations may require longer periods of being in a specific state. Still, this represents only a slight limitation.

Criterion MC3: Logical Entities: Integrated entities in this model and approach are blank containers with the potential to be agent, workflow or both. This is a very general view of the entity, which enables the consideration of every element of the system as such an integrated entity.

Criterion SB1: Structure of the System: The structure of the system is constituted by integrated entities that perform agent actions and are thus agents. Platforms are also covered as specialised agents that still utilise agent actions.

Criterion SB2: Behaviour of the System: The behaviour of the system is constituted by integrated entities that perform workflow operations that describe tasks and are thus workflows. The workflow tasks of these workflows are abstractly associated with specific work. That work can include agent actions and other workflow operations. However, being subordinate to the original task the behaviour of the integrated entity is still only described by a workflow task, thus not violating this criterion.

Criterion SB3: Mutual Incorporation I: The basis of agent interactions are the agent actions of sending and receiving messages. These are available and usable in this model meaning that agents can use regular agent interactions. These can also not be prohibited, since this would restrict the agent side severely in this approach and model. However, agent interactions can also be encapsulated into workflow tasks. That way workflows would control and manage the interactions between agents in the system. This satisfies this criterion, yet due to the lack of an enforcement it is slightly limited.

Criterion SB4: Mutual Incorporation II: Integrated entities as agents serve as the resources for the integrated entities as workflows being executed in a system. The agents can represent human users but may also participate in agent interactions as workflows (see Criterion SB3). When an integrated entity performs workflow operations it becomes that workflow. Similarly to the intermediate model that entity is also the engine for that workflow. Unlike with the combined model, the additional assumptions about the nature of integrated entities, i.e. that they have access to the full potential of agents, make this view possible.

Criterion ToP1: Properties and Mechanisms: Outside of the context of the concrete actions and operations being executed each integrated entity is a blank container with full access to agent and workflow functionality. This means that the entity in any state can access the full feature set and mechanisms through the use of the appropriate agent actions and workflow operations. This criterion is consequently completely fulfilled.

Criterion ToP2: Agent Actions and Workflow Operations: The free use of agent actions and workflow operations is the core idea of this model and approach. Therefore this criterion is directly completely fulfilled.

Criterion ToP3: Regular Multi-Agent Systems: To build a regular agent system in this model and approach, system modellers simply use exclusively agent actions during modelling. The potential for workflow functionality remains but is never accessed and doesn't impede or hinder the execution of the agents.

Criterion ToP4: Regular Workflow Systems: In order to build a regular workflow system, system modellers need to only use workflow operations for modelling. That way the resulting system only contains integrated entities that are workflows, without agents or agent functionality interfering with workflow execution.

Criterion Mgt1: Functionality: The functionality of the system can be freely distributed amongst the integrated entities. The state of these entities is only reliant on the actions and operations they execute. This means that the functionality is freely distributed amongst agents, workflows and hybrids.

Criterion Mgt2: Hierarchies: It is possible to create logical hierarchies of integrated entities in this approach and model. Since the state of the integrated entities is dynamic and freely assignable depending on the actions and operations they perform, the form and shape of the hierarchies created through them is equally as dynamic and free. Boundary crossing is also supported.

Criterion Pra: Practicability: The realisation of this approach appears similarly challenging as the combined or intermediate model. However, the focus on agent actions and workflow operations simplifies the challenge of realisation. Since the state of each integrated entity is defined through its actions and operations, the incompatibilities between agent and workflow functionality can be avoided. There is also no need to provide some kind of translation into specific facets or other models.

The usability of this approach and model is higher than with the comparable intermediate and combined models. System modellers can rely on and easily access the regular, classical agent or workflow functionality. Integration is clearly defined through the combination of agent actions and workflow operations. This provides a clear point of access for modellers to create complex integration mechanisms.

One issue of note is that of the behaviour of the software system. While the behaviour is constituted by integrated entities as workflows, the nature of entities causes similar,

though less explicit, issues with the clarity of distinction between structure and behaviour. The potential for agent functionality in integrated entities requires it to be a blank agent without functionality even if no agent actions are executed. In other words, even if the entity is a blank container for agent functionality it is still an empty and idle agent. This means that every element of the behaviour is also an agent. This, in turn, leads to similar possible issues with the distinction of behaviour and structure as in the intermediate model. For structure, no comparable issues persist, since the workflow potential does not affect the agent potential in a similar way. Overall, the issues are not as explicit as with either the combined or the intermediate model. The fact that it only affects the behaviour also mitigates the effect.

Overall, the practical evaluation of this model and approach is moderate to good. It is moderately difficult to realise and can suffer from distinguishability issues for behaviour and structure, but is easy to handle and utilise.

Additional Aspects

The integration via actions and operations at first glance appears to be a specialised version of the combined model with more concrete assumptions about the nature of the integrated entities. However, while similarities in the approaches exist, it is founded on basic principles of agents and workflows. Section 5.1.4 discussed agent protocols as a fixed end-point within the modelling. Protocols consist of agent actions, which means that these are the building blocks for that fixed end-point. It is the point where the nesting of platforms, agents and protocols stops. Tasks and the workflow operations performed on them are a similar fixed end-point for workflow modelling. The integration via actions and operations is based on these fixed end-point. Agent actions and workflow operations are the most elemental building blocks available in agents and workflows that persist on a tangible yet most abstract level of consideration and modelling. Consequently, an integration model and approach building upon actions and operations is justified in of itself and distinguishes itself from the similar combined model by virtue of the aforementioned principles.

Conclusion

The integration via actions and operations is similarly powerful and expressive as the combined model. But, through its additional assumptions and the use of fixed modelling end-points, it is more practically usable and achieves solutions for the issues faced by the combined model. Table 8.4 summarises the results for integration via actions and operations. Integrated entities are fully supported as defined in Definition B.7. Their state is completely dynamic and all elements of the system can be considered in these terms. The only issue here concerns the fleeting nature of actions and operations and how this might affect certain specific situations in the integration. Structure is constituted by entities as agents and behaviour by entities as workflows, although entities as workflows always exhibit some part of their potential agent state. This effect is only minor as it is only present for the behaviour and not as strongly developed as in the intermediate model. Agents can interact via workflows, though this is not enforced and regular agent interactions are still possible. Workflows are executed by entities as agents in both resource and engine capacities. The transfer of properties is fully possible and supported, agent actions and workflow operations as well. Regular agent and workflow systems can be naturally built by ignoring the potential for the respective other concept. Functionality can be freely distributed amongst the integrated entities in different states and logical hierarchies are

Criterion	Evaluation	Comment
Criterion MC1	full	
Criterion MC2	slightly limited	Fleeting nature of actions and operations
Criterion MC3	full	
Criterion SB1	full	
Criterion SB2	full*	Prescribed structural nature of integrated entities
Criterion SB3	slightly limited	Possible, yet not enforced
Criterion SB4	full	
Criterion ToP1	full	
Criterion ToP2	full	
Criterion ToP3	full	
Criterion ToP4	full	
Criterion Mgt1	full	
Criterion Mgt2	full	
Criterion Pra	moderate to good	

Table 8.4: Evaluation overview for actions and operations

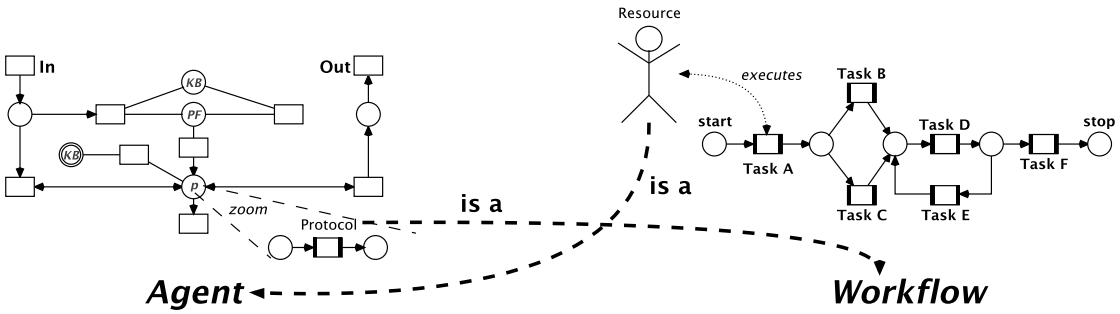

Figure 8.5: Nesting

also arbitrarily possible. The practical evaluation of the approach and model is moderate to good, as the estimated effort for realisation is moderate.

Overall, the integration via actions and operations is very promising. It evaluates fully in most integration criteria. It uses fully integrated entities that are dynamic and ubiquitous. The transfer of properties is available without restrictions. It is also relatively easy to realise and handle/utilise. There are some minor issues regarding the prescribed structural nature of all entities, yet these can be circumvented with later tool and methodology support.

8.1.5 Nesting

The nesting approach and model rely on the idea of nets-within-nets (see Section 2.1.2). Agents execute workflows that contain agents as resources that again contain workflows. Additionally utilising the reference semantics of reference nets allows the creation of a complex, yet conceptually sound integration approach.

Core Idea

The core idea of the nesting model and approach is to take the mutual incorporation of agents and workflows from the vision of the integration from Section 7.1 and build the integration around that concept. Agents contain workflows that describe and execute their behaviour. These workflows contain, as tokens, their own resources, which are again agents. Those agents in turn contain workflows that describe and execute their behaviour. This hierarchy continues until, at some point, a reference to a previous level is reached, in which case the hierarchy becomes a sort of loop. This is illustrated in Figure 8.5. System modellers can enter the hierarchy at any point, creating the opportunity to examine the same system on different levels of abstraction with different (agent or workflow) perspectives.

Criteria Evaluation

Criterion MC1: Integrated Entities: This integration model and approach does not feature explicit integrated entities. It may be argued that a subset of the hierarchy of agents and workflows may be seen as an integrated entity though. However, this is only a consideration and would not affect the modelling constructs available and relevant to this criterion. Consequently, this criterion is not fulfilled at all.

Criterion MC2: Entity Dynamic: Since there are no explicit integrated entities available in this model and approach, the dynamic can, in of itself, not be fulfilled either. However, it is possible to switch between agent and workflow views dynamically by going up or down one level in the hierarchy. This partially fulfils the dynamic criterion, albeit in a very limited way.

Criterion MC3: Logical Entities: As there are no explicit integrated entities available in this model and approach, it is not possible to consider elements as such.

Criterion SB1: Structure of the System: Structural elements of the system are all realised as agents, fulfilling this criterion.

Criterion SB2: Behaviour of the System: Behavioural elements of the system are all realised as workflow, fulfilling this criterion.

Criterion SB3: Mutual Incorporation I: The mutual incorporation of agents and workflows is the core of this model and approach. Consequently, all agents execute workflows as their exclusive behaviour, which includes the interactions between them.

Criterion SB4: Mutual Incorporation II: Workflows contain agents which act as their resources. Agents contain workflows as their behaviour. Consequently, this criterion is fulfilled because the former captures agents as resources, while the latter captures agents as engines.

Criterion ToP1: Properties and Mechanisms: Transfer of properties is not possible in this model and approach. Agents only execute workflows and workflows are only executed by agents.

Criterion ToP2: Agent Actions and Workflow Operations: Only agents can perform agent actions and only workflows can contain workflow operations. Consequently, this criterion is only trivially fulfilled.

Criterion ToP3: Regular Multi-Agent Systems: It is possible to build regular agent systems with this model and approach. Agents are *only* agents, although the behaviour is prescribed to be realised as workflows. The hierarchy needs to be restricted to agents at the top, which execute (as engines) workflows (their behaviour), which are executed

by other agents (as resources). By also restricting workflows to only exhibit the kind of behaviour that is available to regular multi-agent system (e.g. refraining from using workflow management for task and resource control) the workflow functionality can be kept from interfering with the agent concepts. That way the workflows simulate regular agent behaviour, which, together with the structure provided through pure agents, realises a regular multi-agent system.

Criterion ToP4: Regular Workflow Systems: It is not possible to build regular workflow systems with this model and approach. A restriction of the hierarchy and agents, as was done for Criterion ToP3, is not possible the other way around. The reason for this lies in the duality of engines and resources. Workflows are executed by agents as the resources. However, these agents contain a behaviour that is required for them to execute the workflows as resources. That behaviour again consists of workflows, for which these agents act as the engines. Even if the workflows for which the agents are engines only contain tasks for which the same agents are resources again, this prohibits building any regular workflow system as there would always be an additional level of workflows necessary.

Criterion Mgt1: Functionality: Functionality is distributed among agents and workflows. However, the functionality is restricted to the firm hierarchy the nesting creates. It can't be freely distributed. Consequently, this criterion is fulfilled only in a limited way.

Criterion Mgt2: Hierarchies: While hierarchies are a core concept of this model and approach, the hierarchies are restricted in their form and function. Hierarchies here always alternate between agent and workflow levels. Beyond that, it is only possible to consider logical hierarchies of exclusive agents and workflows. Consequently, this criterion is only trivially fulfilled.

Criterion Pra: Practicability: The realisation of the nesting model and approach is straightforward. It is mainly based on nets-within-nets and reference semantics, which are naturally available in the reference net formalism and Renew. Agent and workflow models are also available for reference nets, which restricts the challenges of the implementation to the realisation of the mutual incorporation. The main areas where work is required in this regard are the interfaces between agents and workflows. These need to be properly defined to accomplish the desired hierarchy.

The usability for system modellers is also comparatively high. Modellers only need to know the established agent and workflow models because they are used with little changes. The only difficulties may arise through complex hierarchies and loops. It is feasible to have hierarchies of agents and workflows with a multitude of levels before the loop is reached. Furthermore some branches of the hierarchy may loop back earlier than others. Jumps between levels of hierarchies are also possible. All of this creates a complex net of relations between agents and workflows that has to be carefully maintained and might be difficult to grasp.

Overall, the practicality of this approach remains high though.

Additional Aspects

It has to be noted that this approach and model has a very low degree of actual integration of agents and workflows. Instead of integrating them on some abstract level, it only combines them through one of the core principles of the integration, the mutual incorporation. The result can be observed in the fact that this model and approach evaluates poorly in many

Criterion	Evaluation	Comment
Criterion MC1	none	No explicit integrated entities
Criterion MC2	severely limited	No explicit integrated entities, only switching between levels in hierarchy
Criterion MC3	none	No explicit integrated entities
Criterion SB1	full	
Criterion SB2	full	
Criterion SB3	full	
Criterion SB4	full	
Criterion ToP1	none	Agents and workflows are highly separated
Criterion ToP2	trivial	Agent actions are only available to agents, while workflow operations are only available to workflows
Criterion ToP3	full	
Criterion ToP4	none	Hierarchy/loop prohibits creation of adequate restriction
Criterion Mgt1	limited	Hierarchy/loop restricts distribution of functionality
Criterion Mgt2	trivial	Hierarchies only with agents or workflows or with the prescribed hierarchy/loop
Criterion Pra	high	

Table 8.5: Evaluation overview for nesting

of the integration criteria. This highlights that the integration requires more than just a combination and part of the vision to function as desired and intended.

Conclusion

The integration via nesting is an interesting approach coming from a distinct reference Petri net direction. However, as discussed in the additional aspects, it suffers from the fact that there is a very low degree of actual integration between agents and workflows. The result is that it evaluates quite poorly with the integration criteria. Table 8.5 summarises the results for integration via nesting. There are no explicit integrated entities available in nesting. Dynamic is also not available for entities, although the switching between levels of the hierarchy enables a very limited form of dynamic between agents and workflows. Structure and behaviour are supported according to the criteria and definitions. The mutual incorporation is fully captured, because the core idea of the approach builds on that part of the integration vision. Due to the strong separation of agents and workflows there is no transfer of properties. Agent actions and workflow operations can also only be trivially found in agents and workflows respectively. The approach can be used to create regular agent systems, however there is no restriction available that allows the creation of regular workflow systems. Finally, the distribution of functionality is limited by the hierarchy and the hierarchies themselves can only be trivially formed. The approach and model excel at the practical aspects, as it is easy to realise and, except for complex hierarchies, relatively easy to handle.

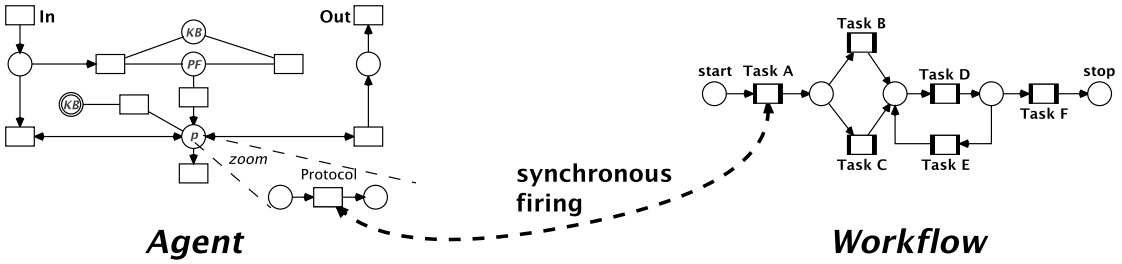

Figure 8.6: Integration via synchronous channels

Overall, the nesting approach should not be further pursued for development. The restrictions regarding entities and the consequences regarding important aspects of an integration, e.g. the transfer of properties, are too severe to qualify this model and approach for further research. The only mitigating factors, the high practical feasibility and the complete mutual incorporation, are not enough to balance out the shortcomings of the integration via nesting.

8.1.6 Integration via Synchronous Channels

The integration via synchronous channels uses the synchronous channels available to certain higher level Petri nets formalisms, including reference nets, to achieve an integration of agents and workflows. This model and approach connects agent and workflow models, rather than merge them in some way.

Core Idea

The core idea of this model and approach revolves around the concept of synchronous channels. Synchronous channels are a Petri net mechanism for firing synchronisation and bidirectional data exchange between transitions during firing. Here, they are used to connect agents and workflows. Instead of utilising some kind of merged model, the integration via synchronous channel maintains two separate models: one for agents and one for workflows.

Each of these models is a fully-fledged system according to its own concept. Together, these systems act as subsystems to the integration system and provide the full functionality for that integration system. These systems both describe the same real-world application, which means that there is an increased overlap between the individual functionalities of the subsystems. Most areas of the application are present in both the agent and workflow models. Exceptions include only those areas that can only be captured by either agents or workflows. Those parts of the two models that describe the same area/part/element of the application are the ones that are connected via synchronous channel. For system modelling, this also means that in these cases there is an automated translation for changes made in either the agent or workflow model into the respective other one.

For example, when an agent action is added to the agent model, a workflow task corresponding to that action is added in the workflow model. Another example is if a resource is added to the workflow model, then a new agent is added to the agent model. That way both the agent and workflow models describe the same application from different perspectives which aims to realise the desired and envisioned integration. The model and approach are illustrated in Figure 8.6.

In principle, this integration approach shares similarities with the third tier of the overall integration architecture proposed by [Reese, 2010] (see Section 3.3.3). In contrast to the higher tiers, which are similar to integration via management (see Section 8.1.1), agents and workflows are provided side-by-side. The key difference between the integration via synchronous channels and the third tier of the integration architecture is that there is an explicit, predefined and structured connection between agent and workflow parts provided by the synchronous channels.

Criteria Evaluation

Criterion MC1: Integrated Entities: The approach and model do not feature explicit integrated entities. However, each element of the system is present in both the agent and workflow models. This means that, in a way, each element is both agent and workflow at the same time, albeit in a virtual way. Overall, due to the implicitness of the entities this criterion is only fulfilled in a limited way. There are no integrated entities available as explicit modelling constructs, even though the agents and workflows being modelled possess counterparts in the respective other model.

Criterion MC2: Entity Dynamic: There is no real dynamic in this model and approach. Each element is, at any time, both agent and workflow. Dynamic can only be available in those parts that are not present in both models. Even when elements are active in only one model, the other one is only idle but still present. However, the intention behind this criterion is fulfilled through the constantly active virtual hybrid state of agents and workflows. If an element is always an agent and a workflow the modeller can choose which model to observe and change. This creates a virtual kind of dynamic which fulfils this criterion in a limited way.

Criterion MC3: Logical Entities: As there is no explicit integrated entity concept available in this model and approach, it is impossible to consider elements as such. The virtual entities discussed for Criterion MC1 are not suitable for this conceptual consideration. Without the explicit concept, modellers can only consider elements as agents, workflows or both. While this is similar in effect to the entities it misses the abstraction provided by the concept of integrated entities.

Criterion SB1: Structure of the System: The structure of a software system consists of elements in the agent model. This criterion is consequently fulfilled. Note that, similar to the intermediate model, the workflow model is always active at any given time. This means that all elements of the structure are also workflows. This is discussed further in the practical evaluation.

Criterion SB2: Behaviour of the System: The behaviour of a software system consists of elements in the workflow model. This criterion is consequently fulfilled. Due to the duality of this model and approach, each of the elements in the workflow model has a corresponding element in the agent model. This means that all elements of the behaviour are also agents. This is discussed further in the practical evaluation.

Criterion SB3: Mutual Incorporation I: In the agent model, agents interact via regular agent interactions. However, these agent interactions correspond to and are also present as workflows in the workflow model. Consequently, this criterion is fulfilled, albeit in an indirect way.

Criterion SB4: Mutual Incorporation II: In the workflow model, workflows are executed by regular workflow resources. These resources, however, correspond to elements,

i.e. agents, in the agent model. Analogous arguments can be made for the engine side. There are some engines available in the workflow model and these engines correspond to agents in the agent model. Consequently, this criterion fulfilled.

Criterion ToP1: Properties and Mechanisms: Transfer of properties is available only in an indirect way. It is possible to enrich one of the two models with model-specific functionality and mechanisms. These can't be automatically translated into the respective other model. That way the properties can be transferred. However, this represents a limitation to the transfer of properties. Agents only have access to agent functionality and workflows only have access to workflow functionality. The only thing that is possible is that the respective counterpart in the alternate model can concurrently enrich the functionality indirectly.

Criterion ToP2: Agent Actions and Workflow Operations: Agent actions are only available in the agent model, while workflow operations are only available in the workflow model. This means that this criterion is only fulfilled trivially.

Criterion ToP3: Regular Multi-Agent Systems: It is possible to build regular agent systems by modelling exclusively in the agent model. While the workflow model is active at any given time, it only contains automatically translated functionality from the agent model.

Criterion ToP4: Regular Workflow Systems: It is possible to build regular workflow systems by modelling exclusively in the workflow model. Agents are still active in the system at any given time, but they only contain the functionality that was defined in the workflow model and automatically translated to the agent model.

Criterion Mgt1: Functionality: Every element of the system corresponds to respective elements in the agent and workflow models. In that way the functionality can be considered as distributed freely amongst agents and workflows. However, this is only a virtual distribution. The entire functionality is distributed amongst the agents in the agent model and also amongst the workflows in the workflow model. This is a strong restriction to this criterion.

Criterion Mgt2: Hierarchies: Logical hierarchies can be formed throughout the different models. There is no restriction here as to which variant (in regards to agent or workflow model) is used in a logical hierarchy elaborated for system modellers. Consequently, this criterion is completely fulfilled.

Criterion Pra: Practicability: Both the realisation and later on usage of this model and approach are difficult. In order to realise this model and approach the main challenge of automatically connecting corresponding elements in both models needs to be overcome. Additionally, after the connection is established an automated translation of changes between the models needs to be implemented. This is even more difficult in this case, as it is for the intermediate model. The intermediate model requires translations from the agent and workflow facets to the intermediate model and back. The integration via synchronous channels needs to translate directly from agent to workflow model. This direct translation is more difficult because it is impossible to create auxiliary constructs that could transform and store data in the intermediate model.

Using the integration via synchronous channels is also quite difficult. System modellers need to keep track of both models concurrently and understand where and how changes in one can affect changes in the other. For example, deleting a resource in a workflow might lead to that agent being deleted. However, if another agent is designed to

interact with that original agent that interaction partner is lost. Making this model and approach usable would require a large effort regarding tool and methodology support.

As with the intermediate model, the constant duality of agents and workflows in this model can lead to some practical issues. While Criterion SB1 and Criterion SB2 are fulfilled, they are, in a way, overly fulfilled. Both structure and behaviour consist of both agents and workflows. In the intermediate model, the workflows in the structure and agents in the behaviour are only facets of the integrated entities in the background. In the integration via synchronous channels there are actual, separate workflows that correspond to agents of the structure and vice versa for the behaviour. This aggravates the issues regarding the clarity of structure and behaviour. In the intermediate model agents of the structure may have a workflow facet, but it is still just one integrated entity (and vice versa for workflows). In the integration via synchronous channels there are two elements that correspond to one (virtual) element. This means that distinguishing between structure and behaviour becomes more difficult, as does modelling and handling them.

Overall, the practicality of this approach evaluates very poorly.

Additional Aspects

As with the integration via nesting, the degree of integration between agents and workflows has to be noted. Due to the correspondence and connection between the different models the degree is slightly higher though. The individual criteria consequently evaluate better than they did for nesting. Still, this approach highlights that a connection and correspondence between agent and workflow models in itself is not enough to realise the desired and envisioned integration.

Conclusion

Integration via synchronous channels differs vastly from the other integration models and approaches presented here. It completely separates agents and workflows into two models, even more so than integration via nesting which at least incorporates the models into one another. In order to realise an integration of the two models, integration via synchronous channels connects corresponding elements of the system through synchronous channels that transmit data and control the firing through synchronisation.

The model and approach therefore do not feature explicit integrated entities. However, corresponding elements of the agent and workflow models create a kind of virtual, indirect integrated entity. There is consequently no real dynamic in these virtual entities and since they are not explicit, elements can't be considered in their terms. Structure and behaviour criteria are fulfilled, even though the issues of distinction and indirection introduced through the constantly active duality can lead to issues. Transfer of properties, likewise, is only available indirectly and agent actions and workflow operations are only trivially available in their respective models. Regular agent and workflow systems can be built. Distribution of functionality is limited to the separated models, but logical hierarchies are completely supported. The practicality of the approach is very poor as it is very difficult to realise and handle. Table 8.6 summarises the results for integration via synchronous channels.

Overall, the indirections in this model and approach make it unsuitable for the purposes of this thesis. For system modellers the absence of explicit integrated entities impedes the utilisation of integration properties and advantages. Furthermore, the very poor practical evaluation also speaks against further refinement.

Criterion	Evaluation	Comment
Criterion MC1	limited	No explicit integrated entities
Criterion MC2	limited	Every element is always an agent and a workflow
Criterion MC3	none	No explicit integrated entities
Criterion SB1	full*	Elements of the structure are always also a workflow
Criterion SB2	full*	Elements of the behaviour are always also an agent
Criterion SB3	full*	Indirectly realised
Criterion SB4	full*	Indirectly realised
Criterion ToP1	Indirect	No actual transfer, only utilisation of corresponding element in alternate model
Criterion ToP2	trivial	Agent actions are only available in the agent model, while workflow operations are only available in the workflow model
Criterion ToP3	full	
Criterion ToP4	full	
Criterion Mgt1	limited	Functionality separation due to the models
Criterion Mgt2	full	
Criterion Pra	very poor	

Table 8.6: Evaluation overview for synchronous channels

8.2 Overall Evaluation

This chapter presented the different abstract models and approaches towards an integration of agents and workflows that have been developed for this thesis. Each of the abstract approaches described in this chapter has interesting features and particularities. Each attempts to solve the challenge of an integration in a different way coming from a different perspective, angle or concept. Further development of all of these approaches, though, is not desired. The approaches have all been developed far enough to enable a comparative evaluation. In order to focus the research for the rest of the thesis and provide more concrete results, one of the approaches will now be selected for further development. The individual evaluations performed in this chapter are the basis for that choice.

Table 8.7 summarises the individual results of each approach and model. The table is an amalgamation of the previous Tables 8.1 through 8.6. It was stripped of details, please refer to the original tables for those.

Integration via management and integration via an intermediate model both mainly suffer from a lack of integrated entities available for direct system modelling. This restricts the possibilities for modellers too much to qualify them for further development. Additionally, they are also poorly evaluated in practical aspects. Integration via nesting and integration via synchronous channels also suffer the lack of directly accessible or, in the case of nesting, non-existing integrated entities. Furthermore, the core ideas for these approaches restrict the integration too much, which is reflected in their poor evaluation for many of the

Criterion	Management	Intermediate	Combined	Actions/Operations	Nesting	Synchronous
Criterion MC1	limited	limited	full	full	none	limited
Criterion MC2	full*	slightly limited	full	slightly limited	severely limited	limited
Criterion MC3	full*	limited	full	full	none	none
Criterion SB1	full	full*	full	full	full	full*
Criterion SB2	full	full*	full	full*	full	full*
Criterion SB3	full	full	full	slightly limited	full	full*
Criterion SB4	limited	full	limited	full	full	full*
Criterion ToP1	slightly limited	full	full	full	none	indirect
Criterion ToP2	full	full	full	full	trivial	trivial
Criterion ToP3	full	limited	slightly limited	full	full	full
Criterion ToP4	full	limited	slightly limited	full	none	full
Criterion Mgt1	limited	full	full	full	limited	limited
Criterion Mgt2	full	full	full	full	trivial	full
Criterion Pra	poor	poor	poor	moderate to good	high	very poor

Table 8.7: Evaluation overview for all models and approaches

integration criteria. Integration via synchronous channels is also very difficult to realise and handle. The relatively high evaluation of practical aspects for integration via nesting is not enough to qualify it for further development.

Integration via a combined model and the similar integration via actions and operations are the most promising approaches. Both evaluate very well w.r.t. the integration criteria. They both exhibit some issues, though these are minor and can be compensated for with later tool and methodology support. From a purely integration-centric point of view, these two are consequently comparable and both equally suitable for further development. The deciding fact, then, are the practical aspects. Here, the integration via actions and operations is clearly favourable. The integration via a combined model suffers from the same problems regarding usability the third tier of an integration in [Reese, 2010] suffers from. The integration via actions and operations provides a more concrete and structured approach based on a fixed end-point in modelling. This results in a vastly increased usability compared to the other approach. Consequently, the selection of an abstract approach for further refinement is the integration via actions and operations.

9 Integration via Agent-Activities

The abstract integration model and approach selected for further development is the integration via fundamental agent actions and basic workflow operations (see Section 8.1.4). This chapter now takes that abstract approach and further specifies the details to realise a concrete and explicit variant of the approach. Regarding the goal of this thesis, this chapter establishes the conceptual model and approach for the motivated combination and integration of agents and workflows. That model is called the Agent-Activity integration approach and is the main result of this chapter. Together with the results from the previous chapters, the Agent-Activity integration approach answers the first research question of this thesis.

The chapter consists of three sections. Section 9.1 presents the Agent-Activity concept. The Agent-Activity concept describes the direct integration model, called the Agent-Activity construct, used in an integrated entity. A practical system that utilises the Agent-Activity construct and integrated entities requires a specific form and specialised functionality. That functionality is described in a reference architecture, presented in Section 9.2. Together, Agent-Activity concept and reference architecture represent the concrete Agent-Activity integration approach. Finally, this chapter is concluded with a discussion and final evaluation of the completed Agent-Activity approach.

Note that the Agent-Activity approach was originally described in [Wagner and Moldt, 2015a]. This chapter depicts an extended and more detailed representation of the approach.

9.1 The Agent-Activity

To reiterate, the integration via agent actions and workflow operations utilises the fundamental agent actions (see Definition B.1) and basic workflow operations (see Definition B.4). Integrated entities serve as the main modelling construct and are considered to be blank containers with access to full agent and workflow functionality. Each integrated entity has the potential to be an agent, a workflow or a (partial or full) hybrid. Depending on whether agent actions or workflow operations are performed by the integrated entity that potential is accessed.

That basic approach of integration via actions and operations now needs to be made more concrete and explicit. Among the main open questions are how, where and in what form the actions and operations are provided to and executed by the integrated entities. This section answers these questions to provide the required concrete model.

Section 9.1.1 describes the Agent-Activity concept. The Agent-Activity concept combines a set of agent actions and workflow operations into an abstract task that describes the behaviour of an integrated entity. Then, Section 9.1.2 presents a basic Petri net model of the Agent-Activity concept to consolidate the descriptions. That section also focuses on the core components within an Agent-Activity.

Before continuing, the terminology of the Agent-Activity context should be clarified. The Agent-Activity concept describes the basic idea of combining agent actions and workflow operations into an abstract task. The Agent-Activity construct is the concrete

model and realisation of that idea and is a key tool for concrete system modelling. The Agent-Activity construct is continuously refined in this chapter and the prototypes of the next chapter. The Agent-Activity integration approach is the overall, concrete integration approach, including the Agent-Activity concept and the reference architecture described later in this chapter. Agent-Activity, as a general, standalone term, refers to an instance of an Agent-Activity. Whether it is an instance of the concept or the construct is clarified by the direct context.

The terminology of the Agent-Activity is essential to this thesis and is recorded in the following key term definitions:

Key Term Definition C.1 (Agent-Activity Integration Approach)**.** The Agent-Activity integration approach is a concrete approach to integrate agents and workflows. It consists of two main components, the Agent-Activity concept (see Definition C.2)and the Agent-Activity reference architecture (see Definition C.7).

Key Term Definition C.2 (Agent-Activity Concept)**.** The Agent-Activity concept combines agent actions and workflow operations into an abstract task that is executed by an integrated entity. Depending on the actions and operations of the Agent-Activities it executes, an integrated entity can be agent, workflow, both or something in between.

Key Term Definition C.3 (Agent-Activity Construct)**.** The Agent-Activity construct (or Agent-Activity modelling construct) is a concrete model and realisation of the Agent-Activity concept.

Key Term Definition C.4 (Agent-Activity)**.** The term Agent-Activity refers to an instance of the Agent-Activity concept or construct.

9.1.1 The Agent-Activity Concept

First, the prerequisites of an integrated entity for an integration via actions and operations need to be defined. The basic approach just states that each integrated entity has access to the full functionality of both agents and workflows. When not executing any agent actions or workflow operations this means that each integrated entity is just a blank container with the full potential of agents and workflows. Figure 9.1 and the following paragraphs further specify the functionality, potential and blank state of integrated entities.

Agent: Integrated entities need to be able to execute the three fundamental agent actions. Apart from that they need to provide the base functionality and facilities of regular agents. The chosen agent model for this thesis is Mulan (see Section 2.2.3). Consequently, the integrated entities need to provide the components defined in that agent model, namely a knowledge base and a protocol factory.

Decision components, as a further part of the extended Mulan model, can be seen as internal agents of an agent or as entirely internal behaviour (see Section 5.2 and especially Figure 5.4). Both variants are captured through agent actions, meaning they don't affect an approach based on agent actions in any conceptual way. A decision on how exactly to incorporate them only needs to be made for a practical implementation. Until then, decision components are excluded from further discussions to simplify the descriptions.

Further prerequisites about integrated entities for an integration via actions and operations concern the connection and relation to the entity environment. The

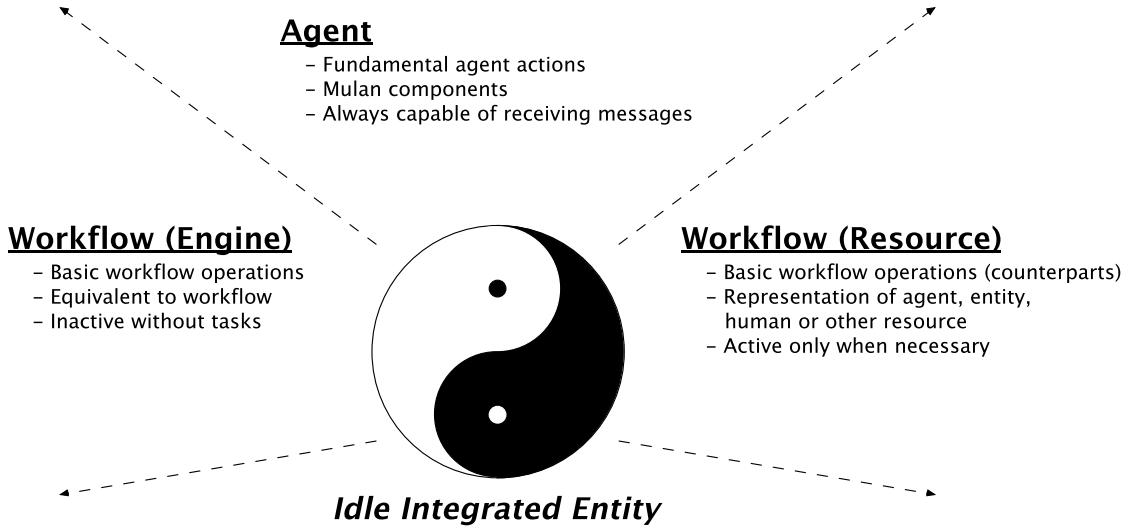

Figure 9.1: Idle integrated entities

integration model presented in this section is only concerned with the individual integrated entities and their execution of agent actions and workflow operations. Therefore, for the AGENT-ACTIVITY *concept*, in itself, it is only important that messages sent also arrive at some other integrated entity in the system. Further prerequisites, which are required to achieve the full AGENT-ACTIVITY *approach*, are addressed as part of the reference architecture in Section 9.2.

One particular item of note regarding integrated entities as blank agent containers is that, even if no agent actions are being executed, the entity still needs to be able to receive agent messages. Because of this an integrated entity, even in its blank state, is always still an idle agent. This issue was already discussed for the basic approach in Section 8.1.4.

Workflow (engine): On the workflow side integrated entities need to be able to execute the three basic workflow operations. Due to the duality of workflow tasks (see Section 6.2) the operations need to be considered from an engine and a resource perspective.

The engine side executes the workflow operations directly within the workflow. An execution of a workflow operation here directly signifies the beginning or end of work on one task. Due to the different modelling abstraction levels of agents and workflows (see Section 7.1.1) an entity that can be an agent generally can't directly be a workflow. To overcome this issue, integrated entities are seen as the workflow engines for workflows. This does not represent a conceptual limitation or restriction. In fact, Definition A.7 only states that a workflow engine is the runtime environment for a workflow. Therefore, it is possible to consider a workflow engine as equivalent to the workflow that is an amalgamation of all the workflows that engine executes. If the workflow engine only executes one workflow (at a time) that engine is equivalent to that workflow. This perspective on workflow engines is adopted for the AGENT-ACTIVITY approach.

As stated above, for a workflow (engine) the main requirement of an integrated entity is its ability to execute the basic workflow operations. This assumes that there are workflow management facilities available in the environment and that the integrated entity is capable of interacting with these facilities in order to successfully execute the operations. This includes internal facilities for the integrated entity to report available workitems, receive and answer workitem requests[1], activity confirmations and activity cancellations. The remainder of the required functionality is described in the reference architecture in Section 9.2.

Unlike the idle agent state, a workflow engine is not active when no workflows are being executed. It is feasible to deactivate this functionality and only activate it whenever necessary. Consequently, an integrated entity is not always[2] a workflow engine.

Workflow (resource): To complete the workflow side of an integration, integrated entities also needs to be able to represent workflow resources. From the conceptual perspective taken in this chapter, it is irrelevant what kind of resource is represented by the integrated engine. The integrated engine can represent itself as a resource (e.g. when agent interactions are modelled as workflows), a human user (e.g. as a proxy or user agent) or any other kind of resource (e.g. a printer or other device). The conceptual principles remain identical for each kind of resource.

The main requirement of an integrated entity as a resource is the resource functionality, especially the execution of workflow operations from the resource side. The description of the required resource functionality is kept on an abstract level for this conceptual approach and is enriched with technical details only in the practically implemented prototypes.

Integrated entities need to support the execution of the workflow operations by providing the counterparts to the operations on the engine side. This includes the following in typical execution sequence:

- Awareness of available workitems
- Initiating requests of workitems on behalf of the represented resource
- Handling results of workitem requests or workitem assigns
- Initiating the start of the actual work associated with an activity in or by the represented resource
- Awareness of the state of its own activities
- Initiating a confirmation or cancellation on behalf of the represented resource
- Providing means to forward results of actual work associated with a task
- Handling the results of a confirmation or cancellation

Note that the previous list does not contain the functionality to actually perform the work associated with the tasks. It is possible that that functionality is provided by the entity or it may be external functionality only controlled by the entity. Regardless, it is not part of the functionality required by the approach for the integrated entity. As with the workflow engine side, the workflow resource side assumes that there are workflow management facilities available in the system which are interfaced with the

[1] Whether a workitem request is a *pull* request or a *push* assign is irrelevant here. From the perspective of a workflow engine both are requests from an external source irregardless if that source is the resource (pull request) or a management facility (push assign).

[2] And possibly never if it doesn't execute any workflows.

functionality above. More details about those management facilities are provided in the reference architecture in Section 9.2.

Also note that it is possible that an entity never needs to act as a workflow resource. Consequently, this functionality is only active when explicitly modelled and necessary.

Having specified the functionality and form of the integrated entities, the next step is to specify the form, provision and execution of the individual agent actions and workflow operations. The most straightforward way of executing agent actions and workflow operations is to provide them directly as building blocks for the behaviour of integrated entities. The behaviour of entities is the same area/level of modelling as are protocols in Mulan and workflows[3]. If modelled that way, an integrated entity would be an agent if its behaviour executed an agent action and a workflow if its behaviour executed a workflow operation.

However, this is not enough to achieve the vision of an integration as desired for this thesis. Simply enabling both agent actions and workflow operations yields only a limited and unstructured combination of agents and workflows. Such integrated entities could only alternate between being an agent and being a workflow, with the exception of instants where concurrency (e.g. of Petri nets in the background) executes an agent action and a workflow operation simultaneously. In fact, the state of being agent or workflow would generally be only instantaneous, since actions and operations exhibit a fleeting nature (see the discussion of the basic approach in Section 8.1.4). What is missing is a modelling abstraction that actually *integrates* agent actions and workflow operations instead of just combining them. Such a modelling abstraction is needed provide form and function to the combination of actions and operations.

To modelling abstraction is provided by taking a step back from the individual agent actions and workflow operations and considering entity behaviour on a more abstract level. Individual agent actions are often related. For example, when a message is received, it is processed, an answer generated and that answer finally sent. Workflow operations are also, albeit trivially, related since a workitem that is requested has its activity, at some point, confirmed or cancelled. This is the solution to providing a modelling abstraction.

Instead of considering individual actions and operations one needs to consider related, ordered sets of actions and operations. That way it is possible to describe abstract tasks of an integrated entity. In this context the term task does not specifically refer to a workflow task, but still retains the meaning of logical, indivisible unit of work prescribed by Definition A.4. These abstract tasks are the titular Agent-Activities. Since these Agent-Activities consist of both agent actions and workflow operations they can be used to not only represent abstract tasks of agents or workflows, but also to represent abstract tasks of hybrids between agents and workflows as well. More concrete examples of Agent-Activities are provided at the end of this section.

Considering Agent-Activities as abstract tasks also means that combinations of Agent-Activities can be considered as abstract workflows. These abstract workflows are not connected with the workflow operations performed *within* the Agent-Activity, but rather represent a conceptual view of the overall relations between Agent-Activities. This view is important w.r.t. the integration and is discussed later on in this chapter.

Note that instances of Agent-Activities are always executed by only **one** integrated entity. Interactions involving multiple integrated entities always require separate Agent-Activities in each of the involved entities. Also note that Agent-Activities always prescribe a specific ordering relation between the actions and operations. That relation

[3] Again, note the different levels of modelling abstraction in agents and workflows and the equivalence of agents and workflow engines.

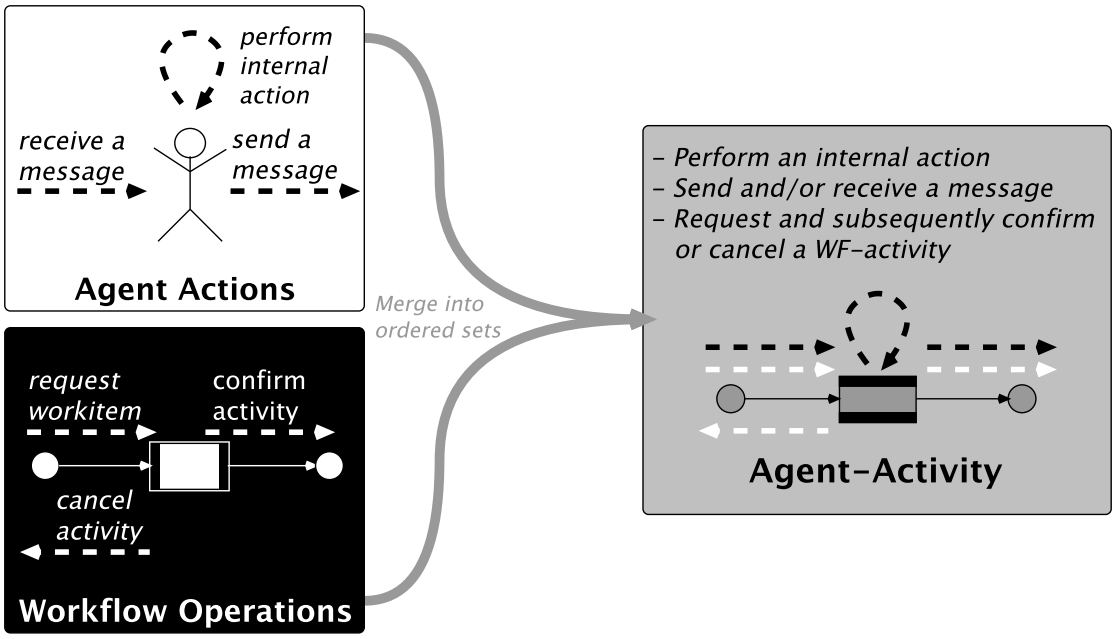

Figure 9.2: The Agent-Activity concept (modified from [Wagner and Moldt, 2015a])

can describe any kind of order, including concurrent actions/operations and loops, but it must specify when, in the context of an Agent-Activity, an action or operation is executed. The ordered set of actions and operations of an Agent-Activity is called the *internal process* of the Agent-Activity. The term process here is used according to Definition A.8.

Key Term Definition C.5 (Internal Process of an Agent-Activity)**.** The internal process of an Agent-Activity is the ordered set of actions and operations the Agent-Activity combines and is able to execute. It represents a process in the sense of Definition A.8.

The term Agent-Activity refers to the integration itself. It incorporates terms from both agents (i.e. Agent) and workflows (i.e. Activity) and symbolises the amalgamation happening within an Agent-Activity by merging the individual terms.

The concept of combining and integrating agent actions and workflow operations into Agent-Activities is illustrated in Figure 9.2. On the left-hand side the actions and operations are shown. Actions and operations are merged into the ordered sets (internal processes) that constitute the Agent-Activities on the right-hand side.

Since multiple connected actions and operations are merged into an Agent-Activity, the state of an entity executing an Agent-Activity is no longer instantaneous. During the period of execution of an Agent-Activity the integrated entity remains in the state that the Agent-Activity determines for it. To specify the duration: An Agent-Activity starts when it is *triggered*. Agent-Activities describe abstract tasks of integrated entities. Tasks are defined as logical, indivisible units of work. Therefore, Agent-Activities adopt workflow task-like atomicity, meaning that they happen either fully or not at all. An Agent-Activity successfully *finishes* or *aborts* as a failure. This terminology was purposefully chosen to distinguish starting and stopping a workflow task, i.e. request/confirm/cancel, from starting and stopping an Agent-Activity, i.e. trigger/finish/abort.

Note that triggering, finishing and aborting Agent-Activities represent additional, administrative management actions. They are not considered as a semantic part of any Agent-Activity, but rather as necessary auxiliary constructs, which are not further discussed for now.

An Agent-Activity is represented as a Petri net transition with thick horizontal bars as seen on the right-hand side of Figure 9.2. This representation is based on the Renew workflow task transition which uses thick vertical bars (see Section 2.3.2). In later prototypes the Agent-Activity is actually implemented as a specialised, complex transition in Renew.

As stated before an entity's state is determined by the Agent-Activities it is executing at any time. If an Agent-Activity consists only of agent actions (black, dashed arrows in Figure 9.2) the executing entity acts as and, for all intents and purposes, is, during the execution of that Agent-Activity, an agent. If an Agent-Activity consists only of workflow operations (white, dashed arrows in Figure 9.2) the executing entity acts as and, for all intents and purposes, is, during the execution of that Agent-Activity, a workflow or rather a workflow engine equivalent to the workflows it executes. If an Agent-Activity consists of both agent actions and workflow operations the executing entity also acts as both agent and workflow and is a hybrid of both. The extent of hybridisation in that case depends entirely on how the actions and operations are combined. Finally, if an integrated entity executes multiple Agent-Activities, each constituting different states, these states aggregate. If, for example an entity concurrently executes a pure agent Agent-Activity and a pure workflow Agent-Activity, it is both agent and workflow at the same time for the duration of the execution of both Agent-Activities. With no limitations[4] to modellers, it is possible to freely nest workflow tasks and agent actions within one another providing the capabilities of full hybrids.

The combination and integration of agent actions and workflow operations into the concept of an Agent-Activity is the core of the overall Agent-Activity integration approach. In fact, it is the core concept that actually enables the integration to be realised according to the vision of an integration laid out in Section 7.1. The following sections continue to provide more concrete details about the specification of the Agent-Activity. To conclude the description of the Agent-Activity concept the following presents a list of different Agent-Activities and their internal processes as small-scale application examples:

Agent Agent-Activity*:* The entity accesses its knowledge base and retrieves some data (internal action), creates a message from the retrieved data (internal action) and sends that message to another entity (send message).

Agent Agent-Activity*:* As a proxy, the entity receives a message (receive message) and automatically forwards it to a previously specified other entity (send message). Then, it awaits confirmation that the original message was received (receive message).

Agent Agent-Activity*:* The entity periodically checks a value representing its workload in its knowledge base (internal message on a timed loop). If the value is in the normal range, the entity doesn't react. If the value is too high or too low it broadcasts an appropriate message to all relevant other entities in the system informing them that it is available and can take on a higher workload or has to deny further work requests for the time being.

[4] Except, of course, the operative limitation that an activity has to exist, i.e. its workitem successfully requested, before it can be confirmed or cancelled.

Workflow Agent-Activity*:* A task to fill a form for a new customer. It is requested by a sales representative human user that is represented by the entity (request workitem) and then confirmed or cancelled by him/her (confirm or cancel activity).

Workflow Agent-Activity*:* Entity A provides a workitem, which is requested by entity B (request workitem). Entity B represents itself as a resource. The task associated with the workitem and activity is automatically performed by the entity B itself. Entity B acts as an agent or even another workflow or hybrid if the behaviour initiated by the task contains Agent-Activities. When entity B is finished it confirms or cancels the activity (confirm or cancel activity) depending on whether the activity was successful. While this Agent-Activity involves entity B with an unknown and possibly agent state, this is still an Agent-Activity that determines entity A to be only a workflow. From the perspective of the executing entity A the kind of resource is irrelevant, even though the utilisation of workflow tasks for the interaction between entities (as agents) is a distinct feature of the integration.

Workflow Agent-Activity*:* A set of tasks guiding students through a complex exercise in a teaching environment. Each task is handled analogously to the previous workflow examples with the human users being the students. Note, that this example purposefully includes multiple tasks in one Agent-Activity. An Agent-Activity can contain any number of workflow tasks and even nest them within one another. As long as only workflow operations are utilised the Agent-Activity still determines the integrated entity executing it as a workflow.

Hybrid Agent-Activity*:* Entity A provides a workitem, which is requested by another entity B representing a human user (request workitem). The activity can't be confirmed by the human user (or entity B representing him/her). Rather, entity B sends the results to entity A (receive message). The results are then checked by entity A, e.g. for format compliance (internal action). Depending on whether the results are satisfactory regarding the checks, entity A itself confirms or cancels the activity (confirm or cancel activity). During the execution of this Agent-Activity the integrated entity A is both agent and workflow. In fact, the final step is truly hybrid behaviour of the entity. While the confirmation or cancellation are workflow operations they are performed by and under the authority of entity A as an agent.

Hybrid Agent-Activity*:* Entity A provides a workitem, which is requested by another entity B representing a human user (request workitem). If the human user doesn't finish the activity in time, monitored by a periodic check of the elapsed time (internal action), a supervisor represented by another entity C is informed that there may be a problem (send message). Regardless of the possible intervention of the supervisor, the human user represented by entity B can still confirm or cancel the activity (confirm or cancel activity).

Hybrid Agent-Activity*:* Entity A provides a workitem, which is requested by another entity B representing a human user (request workitem). When the workitem is successfully requested, the entity A gathers all data related to the task and user and creates a command for an external time tracking and project management tool from it (internal action). The command is then executed by entity A and the data stored temporarily (internal action). When the human user has confirmed or cancelled the activity (confirm or cancel activity), the stored data is used to create another command for the external tool for further time tracking.

Hybrid Agent-Activity*:* An entity A stands by to receive a message from another entity B (receive message). When entity A receives the message it provides a task for itself. That task is assigned, by the workflow management facilities in the system, directly to entity A itself (it is still a request workitem operation from the engine perspective). The behaviour that is initiated for that task can again be anything. Once that is finished entity A confirms the activity (confirm activity) with itself. Here, again, there is truly integrated behaviour. The entity in question acts, on autonomy of itself as an agent, as both a workflow engine and a workflow resource.

Hybrid Agent-Activity*:* An entity performs some agent functionality in this Agent-Activity (any combination of internal action, send message and receive message). At each step it checks the intermediary results for certain conditions (internal action). If these checks return as critical values the data is gathered (internal action) and provided as a workflow task to a human administrator. Requesting the corresponding workitem (request workitem) means that the administrator is checking the problem. A confirmation (confirm activity) signals the entity that there is no problem and that it can continue with the remaining agent functionality. A cancellation (cancel activity) signals the entity that there is a problem, in which case the current data is stored in the knowledge base (internal action) and the Agent-Activity is aborted.

Hybrid Agent-Activity*:* An entity stands by to receive any message from a specific group of other entities (receive message). When a message is received the data from it is extracted (internal action) and provided as a parameter for an otherwise standardised task for a human system monitoring user. That task only contains the request workitem and confirm activity operations. A cancellation in this case is not necessary from an application perspective, since the monitor is only made aware of potential issues in the system. The confirmation only models that the monitor has acknowledged the message. Any further behaviour is outside the scope of the functionality of this Agent-Activity.

9.1.2 Basic Petri Net Model

After having described the Agent-Activity concept in the previous section, this current section provides a basic Petri net model. That model can be seen in Figure 9.3. The Petri net structure in that figure is the basic version of the Agent-Activity construct (see Definition C.3). The current model serves as the basis for any implementation.

The model in Figure 9.3 contains a number of uplinks of synchronous channels. These Uplinks represent the interface to management facilities and functionality within the integrated entity executing the Agent-Activity. Details of that functionality, as well as functional models and implementations, are discussed in the context of the reference architecture in Section 9.2 and the prototypes in later chapters. For this model it is assumed that the management facilities are able to directly and instantaneously provide the necessary data objects to the Agent-Activity.

Note that, without the management facilities serving as an execution environment, the model in Figure 9.3 is not functional, i.e. the transitions are unable to fire. However, for the current purposes, i.e. the further specification of the Agent-Activity concept, a more complex functional model including the management would not provide additional insights into the Agent-Activity concept.

The previous section described the Agent-Activity concept in its basic form. There are two major, practically-oriented enhancements to that concept included in the model presented in this section. While not affecting the fundamental concept and function of the

Agent-Activity, these enhancements improve the usability and versatility. Since these enhancements have, historically, been part of the Agent-Activity since its inception they are presented here as part of the Petri net model. The enhancements are the introduction of a triggering mode and the flexible update of the Agent-Activity at runtime.

Triggering of Agent-Activities mustn't necessarily happen directly. It is feasible to postpone the triggering of an Agent-Activity, even when all direct preconditions are satisfied. For postponing the triggering of an Agent-Activity there is a distinction rooted in agent-orientation. Apart from a direct triggering mode without delay, a reactive and proactive triggering mode for an Agent-Activity are possible. Reactive triggering means that triggering is postponed until the integrated entity receives some kind of external stimulus. That stimulus can be a message from another entity or some kind of direct instruction from the system or a human user. Finally, proactive triggering means that triggering is postponed until the integrated entity itself decides to trigger the Agent-Activity. This can happen when a certain value is achieved in the knowledge base or through more complex decision making mechanisms, which would equate to another Agent-Activity performing internal agent actions. Reactive and proactive triggering can be very useful from an application modelling point of view and are included in the Agent-Activity concept because of this.

Likewise, the flexible update of an Agent-Activity is a desirable feature. As Agent-Activities are not instantaneous and may be executed for a prolonged period of time, the circumstances of the execution may change. In such cases an Agent-Activity that was started may not be required anymore, at least not in its original form. Of course, it would simply be possible to abort the Agent-Activity and retrigger it with the changed parameters. However, this is a rather severe reaction if, for example, only one agent action should be omitted in the Agent-Activity or if an additional branch of actions or operations should be added. Therefore, the Agent-Activity model features the possibility to update the Agent-Activity at runtime. What exactly can be updated and how the update, or rather the migration of data between the states, is realised is entirely dependent on the implementation. For the model presented in this section it is only important that there is an update mechanism that is capable of replacing all runtime data of an Agent-Activity including its internal process at any time during the execution.

In general, the Petri net structure of the model in Figure 9.3 defines only the life-cycle of an Agent-Activity instance. It defines how and where it is triggered, updated, aborted, finished and the actions and operations of the internal process executed. All of the data of an Agent-Activity instance needs to be stored and made available in another component. That component also needs to keep track of the state of the Agent-Activity as far as that state is not concerned with the direct life-cycle, i.e. everything that is handled in the Agent-Activity net structure of Figure 9.3. Important elements that the component needs to track are the actual internal process, its state/progress, the different parameters, (intermediate) results etc. The component is called the Agent-Activity-Object (AAO).

Key Term Definition C.6 (Agent-Activity-Object (AAO))**.** The Agent-Activity-Object (AAO) completely encapsulates the substance of an Agent-Activity instance. It keeps the internal process, runtime state, parameters and any other data pertaining to an Agent-Activity instance during that instance's life-cycle.

Object, in this case, does not refer to an object in the sense of object-oriented programming. It refers to a general, abstract object that serves as the container for the data and non-life-cycle-related state. What kind of object the AAO actually is entirely dependent on the implementation. It might, in fact, be a Java object, an option that was

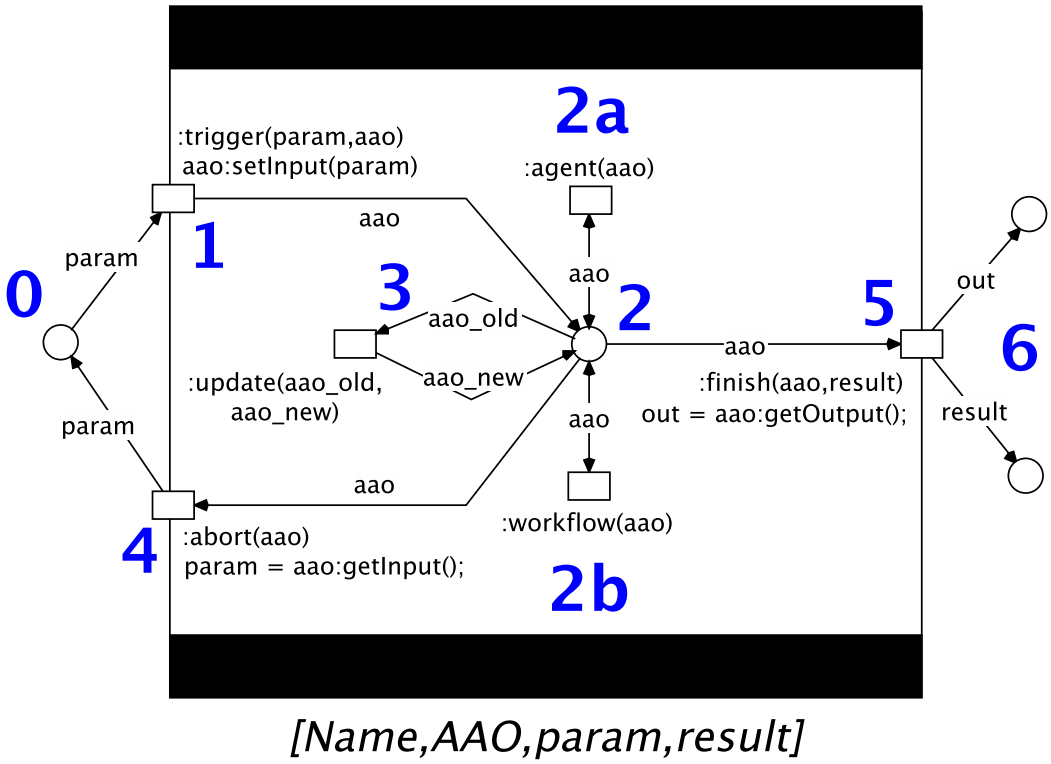

Figure 9.3: Basic Agent-Activity Petri net model
(modified from [Wagner and Moldt, 2015a])

considered during the development of the concept and later prototypes. In the model in Figure 9.3, however, the AAO is considered to be another Petri net. This choice was made in anticipation of later prototypes that implement the AAO as a Petri net as well. The exact functionality contained within the AAO is not relevant for the purposes of this discussion. Details and functional models and implementations are discussed in the context of later prototypes. For now, the AAO is just a container for all the data of an Agent-Activity, including the description and representation of the internal process. Apart from that the most important function of the AAO is to keep track of the state of the internal process, i.e. which actions and operations have already been executed and which are still to be executed.

Note that the use of the AAO emphasises modularity and reusability in Agent-Activities. An Agent-Activity instance encapsulated by an AAO can be reused elsewhere easily. It is also very easy to replace or modify an AAO without affecting the composition and relations of Agent-Activities in the overall behaviour of an integrated entity. By providing this additional level of abstraction, it increases the usability of Agent-Activities in general.

The following description is based on an excerpt from [Wagner and Moldt, 2015a, pp. 10–11], where this basic Agent-Activity construct was originally presented. Note that inscriptions of the model have been changed since the original presentation in order to have a uniform terminology throughout all models and prototypes of this thesis.

0 - Preconditions: The starting points of the model are the preconditions of the Agent-Activity. There can be an arbitrary number of preconditions for any Agent-

ACTIVITY, although, to keep the model simple, Figure 9.3 just features one. Preconditions are all the places connected by arcs to the transition **1**. Input parameters for the AGENT-ACTIVITY are inscribed in (variable *param*) and provided by these arcs.

1 - Triggering the AGENT-ACTIVITY*:* When the preconditions are satisfied the AGENT-ACTIVITY can be triggered. This is when the AGENT-ACTIVITY actually begins. As described in the general concept, there are three different variants to trigger an AGENT-ACTIVITY: direct, reactive and proactive. Triggering mode is not visible in the model, as it is specified directly in each AGENT-ACTIVITY and is managed by the facilities within the entity. Those management facilities ensure that triggering only occurs when both the preconditions of the AGENT-ACTIVITY and the preconditions of the triggering mode are fulfilled. When the triggering begins the management facilities of the entity read the data for the AGENT-ACTIVITY from the knowledge base. From that data and the parameter data the initial AAO is instantly created, transferred to the AGENT-ACTIVITY and put on the central place **2**. The parameter data is set as one of the parameters of the AAO via a synchronous channel downlink.

2 - Executing an action or operation: The central place holds the AAO during the main part of the execution of the AGENT-ACTIVITY. From this place the AAO can execute the individual actions and operations. The AAO can execute all actions and operations concurrently that are enabled in the current state of the AGENT-ACTIVITY and internal process. Agent actions fire the transition **2a**, workflow operations fire the transition **2b**.

3 - Updating the AAO: Updating an AAO is handled by the transition connected to the *:update(aao_old,aao_new)* channel. It removes the old AAO (aao_old) from the central place **2** and puts an updated AAO (aao_new) back. Updates are initiated by the management facilities within the executing integrated entity.

4 - Aborting the AGENT-ACTIVITY*:* If at any point of the execution of the AGENT-ACTIVITY a problem occurs or the entity itself decides to abort the execution, this part of the net is fired and a local rollback initiated. The transition connected to the *:abort(aao)* channel is fired, controlled by the management facilities in the executing integrated entity. It removes the AAO from the central place **2** and initiates a cleanup in the management and administrative facilities of the entity. The original input parameters are read from the AAO and the different tokens put back onto their respective precondition places. The AGENT-ACTIVITY can then trigger again.

5 - Finishing the AGENT-ACTIVITY*:* Once the internal process of the AGENT-ACTIVITY has successfully been completed the management facilities in the executing entity can initiate the finalisation of the AGENT-ACTIVITY. The transition connected to the channel *:finish(aao,result)*, removes the AAO from the net-structure and generates the result token[5] from it. Any output tokens carried through the AGENT-ACTIVITY from the input parameters unchanged are also read from the AAO at this point. The management facilities then finish the AGENT-ACTIVITY administratively.

Parts **4** and **5** realise the only ways to end the execution of an AGENT-ACTIVITY. This realises the described task-like atomicity of an AGENT-ACTIVITY, as it can only be successfully finished in part **5** or aborted with a local rollback in part **4**.

6 - Postconditions: The postconditions are the places connected to the AGENT-ACTIVITY finish transition **5** by outgoing arcs. As with the preconditions there can be an arbitrary number of them.

[5]The result is one object, although it is feasible to be a composite like a tuple or a list.

This concludes the descriptions of the general concept of the Agent-Activity. The following section describes the execution environment necessary for supporting Agent-Activity in integrated entities. More concrete, technical and practical details on the Agent-Activity are provided in the context of the descriptions of the prototypical implementations in the following chapters.

9.2 The Agent-Activity Reference Architecture

The previous section described the Agent-Activity concept. The Agent-Activity concept in of itself only describes how an Agent-Activity it functions inside an integrated entities. In order to actually execute Agent-Activities, an integrated entity and the system surrounding it require a specific form and specialised functionality. This form and functionality is defined in the reference architecture[6] described in this current section. The reference architecture supplements the Agent-Activity concept to create the full Agent-Activity integration approach.

Key Term Definition C.7 (Agent-Activity Reference Architecture)**.** The Agent-Activity reference architecture describes how a system needs to be constructed that features Agent-Activities to realise the integration of agents and workflows. It consists of four nested levels, from the bottom: Process-protocols, integrated entities, platform and management, system.

The reference architecture for the Agent-Activity integration approach is illustrated in Figure 9.4. The development of the reference architecture was strongly influenced by the Mulan reference architecture (see Section 2.2.3). In fact, the basic form of four nested layers is retained. Even more so, when ignoring the workflow aspects the two reference architecture are nearly identical in their composition.

Indeed, rooting the reference architecture in Mulan was a deliberate design decision. The Mulan reference architecture represents an established and well-understood description of multi-agent systems through (reference) Petri nets. Since the integration models of this thesis are also based on (reference) Petri nets, exactly this kind of description is required for the agent-side of the Agent-Activity approach. That agent-side only needs to be supplemented with workflow and hybrid management functionality in order to also support the workflow and integration sides of the Agent-Activity.

Consequently, the approach chosen to develop the reference architecture was to take the Mulan reference architecture as the concrete basis and enrich each level of it with the required workflow functionality. The result is the reference architecture seen in Figure 9.4.

Agent-Activities are the core of the reference architecture. They are combined into larger sets of behaviours called process-protocols (*process-protocol* level of the reference architecture). Process-protocols are executed by integrated entities (*integrated entity* level of the reference architecture). These integrated entities are executed in management systems in the *platform and management* level. The elements of the platform and management level make up the *system* level. That level is implicit, meaning that it is not modelled directly, but is rather generated automatically by the elements contained within it.

[6] Note that the term reference architecture describes an architecture that modellers can refer to when developing systems that features the Agent-Activity integration approach. The term does *not* imply a connection to the reference Petri net formalism. Reference Petri nets can, and are in later chapters, used to implement the Agent-Activity integration approach including the reference architecture. However, this is just one of the possible implementations.

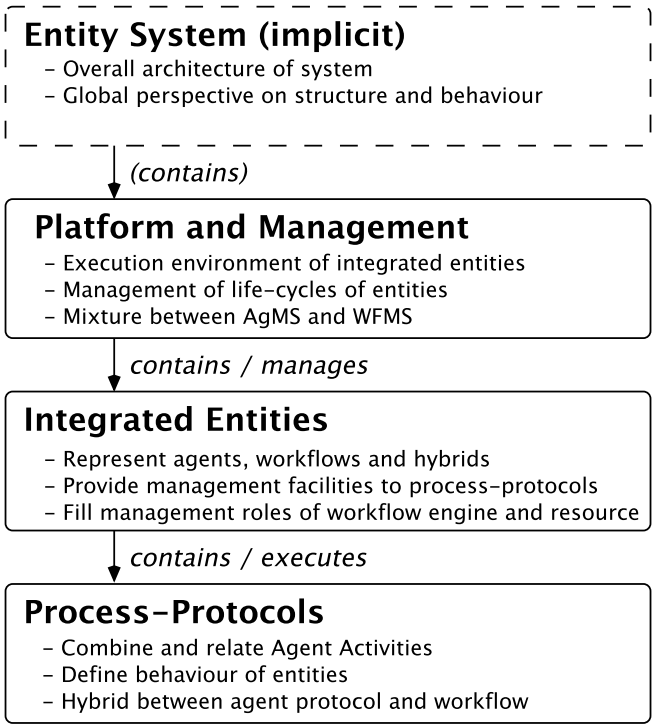

Figure 9.4: Reference architecture for the Agent-Activity integration approach (modified from [Wagner and Moldt, 2015a])

Key Term Definition C.8 (Process-protocol)**.** Process-protocols are larger behaviours of integrated entities that combine Agent-Activities into ordered processes.

Due to the different abstraction levels of modelling in agents and workflows (see Section 7.1.1) their management levels also differ. A workflow system consists of workflows being managed by a workflow management system. The core of a workflow management system, according to the WfMC (see Section 2.3.1), is the workflow enactment service (Definition 2.17) which contains workflow engines (Definition A.7) that execute the workflows. An agent system, according to the Mulan reference architecture, consists of agent platforms that execute agents that execute protocols. This represents a mismatch between the two management views, which had to be addressed in the reference architecture for the Agent-Activity integration approach.

The relation between the Agent-Activity reference architecture and the levels of workflow and agent systems can be seen in Figure 9.5. More details are provided when the individual levels are discussed in the following.

Level 1: Process-protocols The first level of the reference architecture describes the behaviour of integrated entities. The level and elements of this level are called process-protocols. This references the fact that it is a mixture of a workflow process and an agent protocol. Here, the individual Agent-Activities are combined into more complex behaviours. Note that, like the Agent-Activity itself, each process-protocol instance is executed by exactly one integrated entity. However, an integrated entity can have an arbitrary amount of process-protocols defined for it.

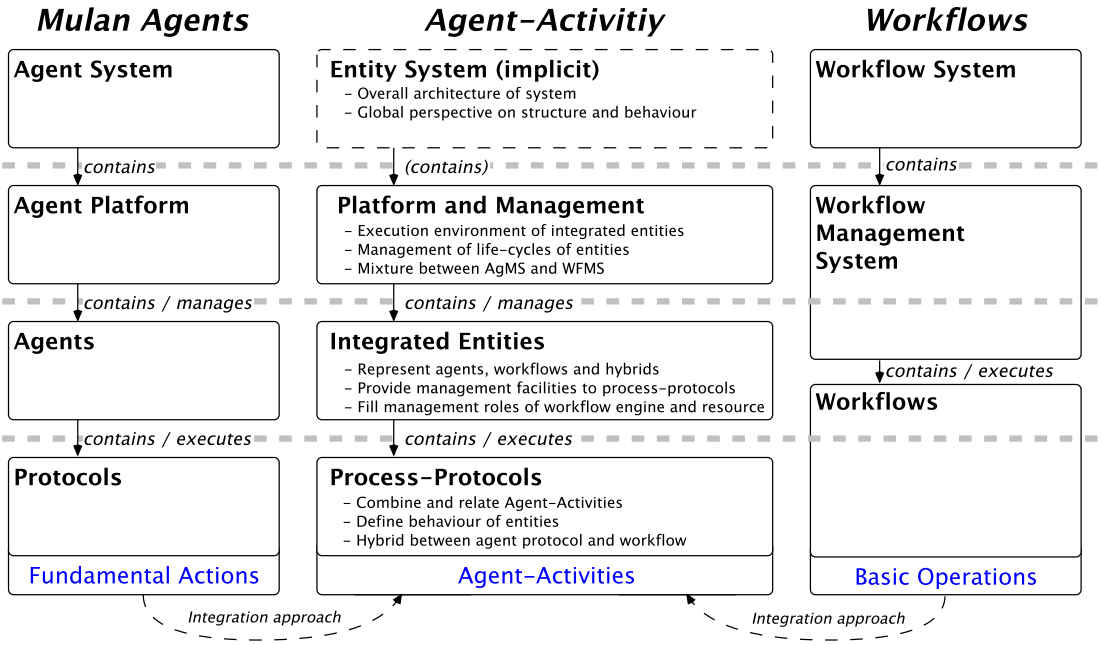

Figure 9.5: Origins of the AGENT-ACTIVITY reference architecture

Regarding the corresponding levels of agents and workflows there is a direct correspondence to the protocol level of the MULAN architecture. In fact, process-protocols including only AGENT-ACTIVITIES that contain exclusively agent actions and regular agent protocols are equivalent. As indicated by Figure 9.5, no such direct correspondence can be made for workflows. The issue here is that integrated entities need to be considered as the workflows they execute as workflow engines. This issue was discussed before in the beginning of Section 9.1.1 and is shortly readdressed in the discussion of the following level. For the process-protocol level this means that it corresponds to part of the workflow level and contains the description and execution of all tasks (and related workflow operations).

Level 2: Integrated Entities The second level of the reference architecture contains the integrated entities and is named likewise. The integrated entities are the main modelling construct of the AGENT-ACTIVITY integration approach. They execute process-protocols and the AGENT-ACTIVITIES contained within them. Depending on the AGENT-ACTIVITIES they execute they can be agents, workflows, both or hybrids.

Consequently, integrated entities correspond directly to agents in the MULAN reference architecture. Integrated entities that execute only agent actions in their process-protocols are equivalent to agents of MULAN. As discussed previously in the beginning of Section 9.1.1, a similar observation can't be made for workflows. An integrated entity is, conceptually, closer to a workflow engine than to a workflow. However, in order to equalise the abstraction levels between agents and workflows they are equated with the workflows they execute. While the integrated entity is a workflow engine it is also the workflow itself, indicated as such in Figure 9.5. This also explains why the conceptual workflow level spans beyond the process-protocol level and into the entity level.

In addition to being agent, workflow and workflow engine the integrated entity also needs to represent resources in the system. Consequently, a large amount of management

functionality is required and provided by facilities internal to the integrated entities. These facilities are called the management backend of the integrated entity. That management functionality includes:

- Management of currently active AGENT-ACTIVITIES of an entity
- Management and execution of triggers for AGENT-ACTIVITIES
- Workflow engine management of workitems and activities in currently executed process-protocols and AGENT-ACTIVITIES
- Workflow resource management of workitems available to the represented resource
- Workflow resource management of currently active activities of the represented resource
- Agent management of agent interactions and internal agent mechanisms
- Provision of agent components (e.g. knowledge base, process-protocol factory)
- Support and facilitation of both agent actions and workflow operations
- Provision and management of interface to further management functionality in the platform and management level

Key Term Definition C.9 (Backend)**.** The (technical) backend of an integrated entity provides the internal management facilities for each integrated entity in the AGENT-ACTIVITY reference architecture. It contains management functionality to support the execution and triggering of AGENT-ACTIVITIES, as well as of all agent actions and workflow operations (in both engine and resource variants).

Level 3: Platform and Management The third level of the reference architecture describes the direct execution environment for integrated entities. The level is called platform and management, as are the elements of it. Each platform and management can execute an arbitrary number of integrated entities. A platform and management is responsible managing the life-cycle and interactions between the integrated entities executed on it. It also provides support for communication between integrated entities on different platform and management elements in a system.

This level corresponds directly to the agent platform level of MULAN. However, as the additional workflow management functionality is continuously active, it is never completely equivalent to a MULAN agent platform. On the workflow side, the discrepancy introduced by the entities being workflow engines is solved. The platform and management level contains all the workflow management functionality besides the workflow engine functionality.

Consequently, the functionality required on this level includes the entirety of agent and workflow management (except workflow engines). As a reference for the required management functionality the FIPA (see Section 2.2.2) standards and WfMC reference model (see Section 2.3.1) are used.

- Management of integrated entity life-cycles
- Management of communication between integrated entities
- Agent management according to FIPA standards including an AMS and a DF
- Workflow management WFES and interfaces according to the WfMC

Level 4: System The fourth and final level is the system level. The system level is the implicit representation of the entire system. It provides a global perspective on both the structure and behaviour, provided through the integrated entities in the appropriate states.

It being implicit means that it is not explicitly modelled. This is a departure from the classical Mulan towards the more practically oriented Capa implementation of that architecture. The system level there is automatically and implicitly generated by the agent platforms active in the system. The same principle is applied here. There is no explicit connection between the individual elements of the platform and management level necessary because it is assumed that they are capable of communicating and interacting with one another in a technical way.

On this level, the correspondence to agent and workflow systems is balanced. The system level directly corresponds to both agent systems in Mulan and workflow systems.

Functionality on this level is not mandatory. However, to support modelling, administrating and monitoring a system a number of options are available here. The generation of a perspective on the global structure and behaviour, for example, is a versatile and desirable tool for system modellers. This is picked up again later in this thesis.

9.3 Final Conceptual Evaluation

This chapter presented the Agent-Activity integration approach. It is a refinement and specification of the basic approach of integration via agent actions and workflow operations. It consists of two main parts, the Agent-Activity concept and the reference architecture. To conclude this chapter, the approach is discussed as a whole and finally evaluated using the integration criteria from Section 7.3. The latter is done to examine if the desired integration characteristics have changed compared to the basic approach of integration via actions and operations.

An integrated entity that is capable of executing agent actions and workflow operations exhibits an agent state whenever it executes agent actions and a workflow state whenever it executes workflow operations. This is the basic approach, as discussed in Section 8.1.4, of an integration via agent actions and workflow operations. That basic approach evaluated well in regards to the individual integration criteria.

The concrete Agent-Activity integration approach fully realises the basic approach and enriches it with a number of things. First and foremost, the Agent-Activity provides an additional abstraction that allows the integration to happen not just instantaneously, but for an extended period of time. This allows for entities to actually utilise integration mechanisms beyond the instant that takes up the execution of an individual action or operation.

Related to that, Agent-Activities also vastly simplify the determination of state of an integrated entity. Agent-Activities are executed for a period of time and during that time the entity that executes the Agent-Activity is what the internal process of the Agent-Activity determines it to be. If the Agent-Activity contains only agent actions it is an agent, if it contains workflow operations it is a workflow and if it contains both it is a hybrid. A modeller only needs to consider which Agent-Activities are executed at any one time to recognise the state and function of the integrated entities. This also improves the ability to distinguish between structure and behaviour, which was an issue in the basic integration via actions and operations because all entities are always idle agents. While this is still true for integrated entities using Agent-Activities it is alleviated by the clear distinction provided by the currently executing Agent-Activities. Agent-Activities also happen either fully or not at all, adopting task-like atomicity aligning with

the consideration of Agent-Activities as abstract tasks of integrated entities. This can be utilised by system modellers to make assumptions about the behaviour of the system.

The basic Petri net model adds some additional features. Triggering modes enable the postponement of Agent-Activities until additional external parameters are fulfilled. This increases the versatility of the concept. Additionally, the flexible update mechanism introduces the desirable trait of adaptivity. Through it, changing runtime parameters can be handled without losing already made process. However, the extent of the effect the flexible update mechanism has is entirely dependent on how it is implemented. Another feature added by the basic Petri net model is the separation of life-cycle and substance of the Agent-Activity. The Agent-Activity construct shown in Figure 9.3 defines and controls the life-cycle of an Agent-Activity instance while the AAO encapsulates its substance. Parameters, input/output data, (intermediate) results, state and the internal process are all encapsulated in the Agent-Activity-Object (AAO). The AAO signifies the single object that needs to be changed if an Agent-Activity is to be adapted. It can also be easily injected into any other Agent-Activity construct in the behaviour of an integrated entity where the same actions and operations are needed again. Consequently, the AAO emphasises modularity and reusability of the Agent-Activity.

Lastly, the reference architecture provides a system and management perspective for the Agent-Activity. It describes what a system executing Agent-Activities needs to provide in terms of form and functionality. It also provides an understanding of which parts of an agent and workflow system can be found where in an integration system.

In conclusion, the Agent-Activity approach adds details to the basic approach of integration via actions and operations that are not only essential to actually realising such an integration, but also directly improves it in a number of areas. To complete this part of the discussion, the following reexamines the different integration criteria and discuss them in the context of the results obtained in this chapter.

Criterion MC1: Integrated Entities: Integrated entities are fully used in the Agent-Activity integration approach. Whether an entity is an agent, a workflow or a hybrid depends entirely on the Agent-Activities it executes at any time.

On a related side note, an interesting idea is whether or not just the integrated entities are considered as agents, workflows or hybrids, but also the Agent-Activities themselves can be considered in these terms. Conceptually, that is possible. By equating the prescribed behaviour of an agent/workflow/hybrid defined in the Agent-Activity with the agent/workflow/hybrid itself the Agent-Activity becomes that agent/workflow/hybrid. This is an interesting approach because it emphasises an even stronger hierarchy not just of integrated entities (see Criterion Mgt2) but also *within* integrated entities. However, this idea is not pursued further for this approach or the following prototypes.

Criterion MC2: Entity Dynamic: The state of the integrated entities is determined by the Agent-Activities it executes. In contrast to the basic approach via actions and operations, the Agent-Activity approach addresses the issues stemming from the fleeting nature of actions and operations. Instead of changing the state dynamically according to the individual actions and operations, the longer lasting Agent-Activities determine the state. This provides the same, full amount of dynamic for the state of the integrated entity without suffering from issues related to instantaneous actions and operations.

Criterion MC3: Logical Entities: Every integrated entity is a blank container that is only "filled" with substance through the Agent-Activities. Consequently, every element can be regarded in these terms without restrictions.

Criterion SB1: Structure of the System: The structure of the system is constituted by integrated entities that perform agent actions in Agent-Activities and are thus agents. Platforms are also covered as specialised agents that still utilise agent actions. There is no change to the basic approach.

Criterion SB2: Behaviour of the System: The behaviour of the system is constituted by Agent-Activities being executed by integrated entities. An Agent-Activity describes an abstract task or, in other words, a logical unit of work. Tasks form workflows, meaning that Agent-Activities as abstract tasks form abstract workflows. These workflows are not connected with the workflow operations happening within the Agent-Activities, but rather describe the overall behaviour of the integrated entities. Integrated entities execute the Agent-Activities as abstract workflow engines. Here, an argument analogous to the concrete level can be made, where integrated entities as workflow engines for the workflow tasks/operations *within* the Agent-Activities are equated to those workflows. Consequently, the behaviour of the system is constituted by the Agent-Activities executed in it, which in turn constitute abstract workflows thus fulfilling this criterion. The difference between the abstract workflow and the concrete behaviour contained within the Agent-Activities is equivalent to the subordination relation described for the general approach.

Criterion SB3: Mutual Incorporation I: The behaviour of integrated entities is defined in its process-protocols that contain Agent-Activities. Modelling interactions between entities as agents with Agent-Activities containing workflow operations is fully supported. However, regular agent interactions are also fully supported and are not restricted. Consequently, there is no change to the basic approach.

Criterion SB4: Mutual Incorporation II: Integrated entities can represent any kind of resource. They also serve as the workflow engines for the workflows they execute. Due to the agent functionality being constantly (though idly) active, both of these roles also coincide with the entities being agents, thus fulfilling the criterion.

Criterion ToP1: Properties and Mechanisms: Integrated entities have full access to agent and workflow functionality. They can consequently freely combine any mechanism of either functionality by employing and combining agent actions and workflow operations accordingly. This completely fulfils this criterion.

Criterion ToP2: Agent Actions and Workflow Operations: Agent-Activities consist of an arbitrary combination of agent actions and workflow operations, thus fulfilling this criterion directly.

Criterion ToP3: Regular Multi-Agent Systems: Regular multi-agent system can easily be built by using only Agent-Activities that contain agent actions exclusively.

Criterion ToP4: Regular Workflow Systems: Regular workflow systems can be built by using only Agent-Activities that contain workflow operations exclusively.

Criterion Mgt1: Functionality: As the state of each integrated entity is completely dynamic and dependent entirely on which Agent-Activities it executes, the functionality, which is distributed amongst the entities, is also distributed amongst a dynamically changing set of agents, workflows and hybrids. Therefore, this criterion is completely fulfilled.

Criterion Mgt2: Hierarchies: Integrated entities may initiate and even control one another. Consequently, logical hierarchies are possible to examine. Since the state of the integrated entities is dynamic and dependent on the Agent-Activities being executed,

Criterion	Evaluation	Comment
Criterion MC1	full	
Criterion MC2	full	*Improved from basic approach*
Criterion MC3	full	
Criterion SB1	full	
Criterion SB2	full	Agent-Activities determine state *Improved from basic approach*
Criterion SB3	slightly limited	possible, yet not enforced
Criterion SB4	full	
Criterion ToP1	full	
Criterion ToP2	full	
Criterion ToP3	full	
Criterion ToP4	full	
Criterion Mgt1	full	
Criterion Mgt2	full	
Criterion Pra	good	*Improved from basic approach*

Table 9.1: Evaluation overview for Agent-Activities

that hierarchy also consists of a dynamic set of agents, workflows and hybrids. That dynamic also means that boundary crossing between concepts is fully supported.

Criterion Pra: Practicability: Realisability and usability of the Agent-Activity approach are higher than of the basic approach. This is natural, since the additional details provide clear instructions on what exactly the system has to look like.

Regarding the distinction between structure and behaviour the abstraction provided by the Agent-Activity concept also alleviates the issues in the basic approach of integration via actions and operations. Even though each entity is always an (idle) agent, the Agent-Activities an entity executes are the primary, deciding factor if an entity is an agent, a workflow or a hybrid. Consequently, structure and behaviour are clearly distinguishable through the Agent-Activities.

Table 9.1 summarises the evaluation. The Agent-Activity approach addresses the limitation of the basic approach regarding the dynamic and solves the issue regarding the inherent (idle) structural nature of entities by declaring Agent-Activities as the primary factor in determining the state of an integrated entity. The practicality of the approach is also better. Equally as important, the Agent-Activity approach doesn't evaluate worse or more restricted in any other criterion. Consequently, the more concrete Agent-Activity approach can be seen as a distinct improvement to the basic approach of integration via actions and operations. Not only does it provide the details necessary for a realisation and implementation, the more concrete nature also allowed for some issues to be resolved.

In conclusion, the Agent-Activity integration approach represents a fully elaborated approach that describes an integration consistent with the vision and specification of an integration as described in Section 7.1. The evaluations made in this section support the decision to choose the basic approach of integration via actions and operations and elaborate on it. Having exhausted the conceptual means of examining and discussing an integration,

the next step is to implement prototypes for the Agent-Activity approach that serve as a proof-of-concept for further examinations, analysis, application and discussion. The following chapter presents these prototypes.

Before moving on to the prototypes, the following presents a list of requirements a practical and technical system needs to implement in order to constitute a proof-of-concept for the Agent-Activity approach. That list is used at the end of the prototypes chapter to attest that the implemented system is, in fact, a proof-of-concept. These requirements describe the distinct mechanisms, constructs and components defined by the Agent-Activity approach. They are independent of the integration criteria. To qualify as a proof-of-concept of the Agent-Activity integration approach a system only needs to fulfil the requirements. Afterwards, in a separate discussion step, that system can be evaluated against the integration criteria to also validate that the integration is achieved as desired. For this chapter, the list also represents a concise summary of the Agent-Activity approach.

The Agent-Activity *is the main modelling focus:* Agent-Activities are the main focus of modelling. An Agent-Activity is an abstract task that encapsulates an internal process consisting of an ordered set of fundamental agent actions and basic workflow operations.

Agent-Activities *are executed by integrated entities:* Agent-Activities describe the behaviour of integrated entities as defined in Definition B.7. Therefore, Agent-Activities are executed by integrated entities (as opposed to being stand-alone constructs or being executed by entities not complying with Definition B.7).

Dynamic state determination of the executing integrated entity: The state of an integrated entity, meaning if it is an agent, a workflow, both or something in between, is defined dynamically at any point in time by the Agent-Activities it is executing. If an entity executes an Agent-Activity with only agent actions it is an agent. If it executes an Agent-Activity with only workflow operations it is a workflow. If it executes an Agent-Activity with both agent actions and workflow operations the entity is a hybrid of both agent and workflow. Concurrent Agent-Activities aggregate their state, meaning that if, for example, an entity executes an Agent-Activity with only agent actions and another one with only workflow operations concurrently, that entity's state is both agent and workflow. Of the highest importance is that the state of the integrated entity is fleeting and dynamic, meaning that an entity can, for example, be a workflow at one point, an agent at the next and a hybrid after that.

Agent-Activities *are atomic management units:* Agent-Activities not only combine agent actions and workflow operations, but also provide form and function to that combination. This constitutes the integration of agents and workflows through the added abstraction. As a direct consequence, the Agent-Activities need to be atomic as to ensure that the form and function provided by them is achieved. Agent-Activities either happen fully or not at all.

The Agent-Activity *life cycle is observed:* The basic life cycle of an Agent-Activity, illustrated by the basic Petri net model from Section 9.1.2, is observed. This means an Agent-Activity can be triggered to be active and then executes its actions and operations. While it is active it can be updated or aborted. When the internal process is finished, the Agent-Activity itself can also finish.

Integrated entities are agents: This concerns the capabilities and idle state of the integrated entities executing the Agent-Activities. In general, the integrated entities must

represent agents according to the MULAN agent model within the prototype. This means they must be able to send and receive messages and also be able to execute internal actions.

Integrated entities are workflow engines: An integrated entity must have capabilities of providing local workflow management for workflow tasks defined through workflow operations in its own AGENT-ACTIVITIES. In this way, it must act as a workflow engine representing itself as the workflows it is executing.

Integrated entities are workflow resources: On the other hand, an integrated entity must be able to represent a workflow resource. It must have the capabilities of processing available workitems, requesting them and supporting the resource it represents in activity execution including the confirmation and cancellation of its activities.

Local and global management: All local management, especially workflow engine management, is provided by the entities themselves. In the runtime environment of those entities, global management facilities for both agents and workflows must be provided.

The reference architecture is distinctly realised: While the previous point already implicitly dealt with the levels of the reference architecture, that reference architecture must also be complied with explicitly in a practical system. This relates to the relationships between the different levels, especially the encapsulation and nesting of lower levels in the directly higher level. Process-protocols combine AGENT-ACTIVITIES into behaviour for and are executed in integrated entities. Integrated entities are managed by and executed in elements of the platform and management level. The entirety of the platform and management level is situated in the system level, which, opposed to the other three levels, is only implicitly modelled.

10 Prototypes

This current chapter presents the different prototypes that were implemented during the creation of this thesis. Iteratively building on the results and lessons learned of each prototype concluded in the creation of the **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System). The Paffin-System represents the technical proof-of-concept desired for the Agent-Activity integration approach. It was first presented in [Wagner et al., 2016b]. More information can be found at the Paffin-System website[1]. The Paffin-System as a technical proof-of-concept constitutes the technical part of the goal of this thesis as discussed in the introduction.

This chapter is structured as follows. Section 10.1 provides some general notes on the prototypes discussed in the rest of the chapter. Next, Section 10.2 presents the main proof-of-concept for the Agent-Activity approach, the Paffin-System. Other prototypes not directly related to the Paffin-System are described in Section 10.3. Finally, Section 10.4 concludes this chapter.

10.1 General Notes on the Prototypes

This section presents some general notes on the prototypes. The purpose of this section is to provide general and overarching information about the different prototypes and their descriptions.

Relation of the Prototypes This chapter only presents the major prototypes created for this thesis. Minor prototypes for testing ideas or mechanisms concerning the integration of agents and workflows are not presented. Results from minor prototypes were ultimately subsumed and incorporated into the major, more technically elaborate prototypes. The chronological order of the major prototypes is listed below:

1. Paffin Entity: Section 10.2.5
2. Paffin Platform: Section 10.2.13
3. The Agent-Activity Net Structure and AAO: Section 10.2.2
4. Paffin Backend Control: Section 10.2.6
5. Paffin Backend Agents: Section 10.2.8
6. Paffin Backend Engine: Section 10.2.10
7. Paffin Backend Resource: Section 10.2.11
8. Paffin Platform WFMS: Section 10.2.14
9. Internal Process and Paffin Net Components: Section 10.2.3
10. Paffin Web GUI: Section 10.2.12
11. Paffin Modelling: Section 10.2.4

[1] `https://www.paffin.de`, maintained by the author of this thesis (last accessed May 28th, 2017)

12. Reactive and Proactive Triggering of Agent-Activities: Section 10.2.7
13. Paffin Decision Components: Section 10.2.9
14. Application Prototypes: Chapter 11 (Concurrently to other (framework) prototypes)

As indicated by the section references, the chronological order differs from the descriptive one in the thesis. The descriptive order provides a clearer understanding of prototypes as it is based on the factual relations of the prototypes, not the chronological. Prototypes are described from the bottom up w.r.t. the reference architecture. After a general overview of the system in Section 10.2.1, the descriptions start off on the process-protocol level. These prototypes, described in Section 10.2.2 and Section 10.2.4, implement the net structures, net components and subnets found in and directly related to process-protocols. Moving on to the Paffin level of the reference architecture, the prototypes in Sections 10.2.5 through 10.2.12 describe the Paffin net and its internal backend, which is the key component of the Paffin. Also included is the graphical web interface for human users, which is important for the description of the applications later on. The third level of the reference architecture consists of the Paffin platforms. The corresponding prototypes are described in Section 10.2.13 and Section 10.2.14. These prototypes describe the platform implementation, including the important workflow management aspects and database connection. The final level of the reference architecture, the system level, is only implicit. As a consequence, there are not implementation prototypes. The system level is addressed again in the discussion and evaluation chapters later on.

Development of the prototypes followed ideas from agile software development approaches [Beck, 1999, Coleman and Renaat, 1998, Gloger, 2010, Hazzan and Dubinsky, 2008, Meyer, 2014]. Within each prototype incremental and evolutionary iteration cycles were utilised in variation with reviews, tests and evaluative discussions with peers.

Even more so, the prototypes of the Paffin-System in themselves represent agile, incremental extensions of functionality. All prototypes described in Section 10.2 extend the functionality and expressiveness of the Paffin-System with additional, significant mechanisms. Each prototype does so by building on top of the already implemented functionality (cf. the chronological order of prototypes above). Naturally, these extensions can necessitate changes in already implemented components. The result is an evolution spanning the different prototypes, caused by internal (i.e. within each prototype) and external (i.e. caused by the development of other prototypes) cycles.

Ultimately, only the final version of each prototype is described in this chapter. The evolution a prototype underwent, though, is, wherever relevant, included in the descriptions.

Description of the Prototypes In the following section each individual prototype of the Paffin-System is described in an individual subsection. Each subsection, except the overview of architecture and ontology in Section 10.2.1, follows the same basic descriptive structure. That structure contains the following components, which are roughly oriented around the Paose development approach.

1. ***Purpose and Functionality:*** Shortly describes the purpose and importance of the current prototype and also outlines the functionality associated with that purpose. This provides the general context within the overall development of the Paffin-System.
2. ***Design:*** Provides a basic overview of how the functionality is implemented, including components associated with the prototype and the relations between them. Also discussed are the major design decisions of each prototype. Note that all design decisions were made by the author of this thesis, though they were often discussed

with colleagues and peers. An exception to that is the Paffin Web GUI prototype described in Section 10.2.12. This prototype was a collaboration with Dennis Schmitz, therefore the design decisions of that prototype were also made in collaboration.

3. ***Implementation:*** Provides details of the implementation of the current prototype. Specific parts of both the reference net and Java implementations are highlighted for that purpose.
4. ***Conclusion:*** Concludes the description of the prototype by examining how the purpose/functionality was achieved and discussing overarching aspects, issues and challenges, including, if applicable, future work visions.

The state of the described prototypes is also important to note. While the descriptions are categorised along the iterative, main prototypes, the actual content relates to the *final* state of the software.For the following descriptions this means that any intermediary results and states that were made obsolete at some point during development are not discussed here.

Figures of Petri Nets from the Prototypes The Paffin-System prototype spans six Renew plugins that consist of more than 140 reference nets and numerous Java classes. The ontology of the overall system alone consists of more than 100 concepts. Consequently, it is not feasible to present all nets and classes of the prototypes within the scope of this thesis. Rather, the descriptions of the prototypes describe the realised functionality and highlight certain features and mechanisms through the most important nets and classes.

In general, the figures presented in this section directly correspond to nets used in the prototypes. However, many have been rearranged to fit better into the page layout of this thesis. Due to the large size of some nets, some figures also only show partial excerpts of the nets. These instances are clearly marked.

10.2 Paffin-System

The **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-system (Paffin-System) represents the technical proof-of-concept for the Agent-Activity integration approach. It implements a framework for modelling and executing systems featuring Agent-Activities by realising the reference architecture and providing suitable modelling tools. The core of the Paffin-System consists of 13 iterative prototypes.

Before beginning with the main descriptions the following key terms need to be defined:

Key Term Definition C.10 (Paffin-System)**.** The **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System) is a modelling framework and technical implementation of the Agent-Activity integration approach.

Key Term Definition C.11 (Paffin Entity)**.** A Paffin entity (or integrated Paffin entity or Paffin for short) is the implementation of an integrated entity in the Paffin-System.

The Paffin-System website[2] contains relevant links to additional information about the Paffin-System, including instructions on how to access the current source code repository and Paffin Redmine project. Note that the following descriptions refer to the state of the prototype in April/May 2017. The prototype is still being actively developed in the context of successor theses.

[2] `https://www.paffin.de`, maintained by the author of this thesis (last accessed May 28th, 2017)

10.2.1 Paffin-System Overview

The Paffin-System, developed in incremental prototypes, was iteratively extended to realise the full Agent-Activity integration approach. Each prototype implemented only part of the approach, so that all individual prototypes in combination represent the entire Paffin-System.

Technologically, the Paffin-System was not built from scratch. As the reference architecture of the Agent-Activity approach was developed from the basis of the Mulan architecture, it was only appropriate to utilise the Capa framework, the implementation of Mulan (see Section 2.2.3), for the implementation of the reference architecture. In doing so, the full agent side of the Agent-Activity approach was already implemented from the beginning. Additionally, modelling and monitoring tools of the Capa framework, like the MulanViewer (see Section 2.2.3), were directly available for the Paffin-System.

Using the foundation of Capa, the main tasks for the creation of the Paffin-System revolved around implementing and incorporating the workflow management and integration concepts. The basic approach for each prototype was kept deliberately simple. First, identify the existing (Capa) nets and classes that would be associated with the intended functionality. Second, determine if the existing nets could be modified for the new functionality or if new nets or classes needed to be implemented. Complete replacement of nets and classes was never necessary during the development of the Paffin-System. The nets and classes of Capa all fulfil a certain purpose for the agent model. Since that agent model should be fully supported by the Paffin-System all of its functionality was required. Third, implement the new desired functionality in the nets and classes designated for it. Fourth and last, test the new functionality for that prototype. Prototype tests were kept simple and separated from the applications, which can be regarded as final, larger scale tests of the entire Paffin-System. They are not discussed further in these descriptions, but are examined in Chapter 11.

This current section presents an overview of the entire Paffin-System. It assumes, in advance, the full realisation of the Paffin-System that is achieved through the 13 individual prototypes. Its goal is to facilitate the following descriptions by providing context in the form of a complete system overview. Some global features, like the ontology, also can't be assigned to one specific prototype and need to be discussed here.

The Paffin-System is implemented as a number of Renew plugins. These plugins are provided in a separate Renew project called `Paffin`. Through the use of Apache Ant[3] as a build tool and xml configuration files, the Paffin plugins are incorporated into Renew for development and execution. At the time of writing the Paffin framework consists of six Renew plugins:

- `Paffin`: is the main plugin. It provides the ontology for the entire project.
- `PaffinEntity`: realises all functionality associated with an individual Paffin entity.
- `PaffinPlatform`: realises all functionality associated with a Paffin platform and management.
- `PaffinComponents`: provides standardised modelling (net) components for the Paffin context.
- `PaffinModelling`: provides a Renew formalism realising the Agent-Activity as a single transition for modelling.

[3] `http://ant.apache.org/` (last accessed May 28th, 2017)

– `PaffinWebGui`: realises a default graphical user interface (GUI) based on web services and accessible via internet browser.

In these plugins the entire functionality of the Paffin-System is implemented. Additional plugins from the context of Renew, Mulan, Capa and Paose are used by the Paffin-System. These are considered as part of the runtime environment and only discussed when the features or mechanisms in the individual prototypes require them for explanations.

Being implemented in Renew with the Capa basis means that most components of the Paffin-System are implemented as reference Petri nets. Additional functionality is provided in auxiliary Java[4] classes. Individual components are connected in several ways. Nested nets are directly connected and communicate via synchronous channels. Net components, i.e. net structures within nets, may also be modularly combined to create composite nets. Components in different Paffin entities utilise asynchronous agent messages. Finally, certain components, especially those regarding the database and web GUI, communicate via Java calls or JSON[5] messages.

Figure 10.1 shows the overall Paffin-System architecture. Each component of that figure corresponds to one net or class realising a specific bit of functionality in the Paffin-System. The Agent-Activity reference architecture can be clearly distinguished in the architecture of the Paffin-System.

On the process-protocol level individual agent actions and workflow operations are combined into an internal process, which is nested in the Agent-Activity-Object (AAO), which is being executed within the Agent-Activity-Transition[6], multiple of which together form process-protocols.

Process-protocols themselves are executed in Paffin entities, which contain internal components responsible for handling and managing Agent-Activities, agent actions and workflow operations (as engine and resource). These internal management components are generally referred to as the *technical backend* of a Paffin entity (see Definition C.9), implemented in multiple reference nets. Capa agent base functionality, including application-specific decisions components (DC), knowledge base and factory, are adopted from Capa with only minor changes and extensions to facilitate the incorporation of integration aspects. Due to their nature as integrated entities, Paffins require both an agent and a workflow interface. Technically, both interfaces are implemented through the adopted Capa agent interface using agent messages exclusively. The agent interface does so directly and explicitly, while the workflow interface uses transparent (for users and modellers) administrative (i.e. automatic and non-application-specific) messages to and from the management facilities on the platform and management level. These administrative messages are controlled and processed by two standardised decisions components implementing the engine and resource interfaces respectively. The technical unification of agent and workflow interfaces into one interface between Paffin entities and the platform and management level was a technical design decision, elaborated on later in the individual prototypes, favouring uniformity and maintainability.

In addition to the regular Paffin entities, which implement application functionality, the GUI Paffin entity is a special kind of Paffin providing (only) the GUI for human users. Each Paffin entity that requires a GUI initiates and controls, via administrative agent messages, one GUI Paffin. That control is implemented in another standardised DC

[4] `https://java.com/` (last accessed May 28th, 2017)

[5] JavaScript Object Notation, `http://www.json.org/` (last accessed May 28th, 2017)

[6] The Agent-Activity-Transition implements the Agent-Activity construct (see Section 9.1.2) as a single transition at modelling time which is translated by Renew during compilation into a complex Petri net structure for execution.

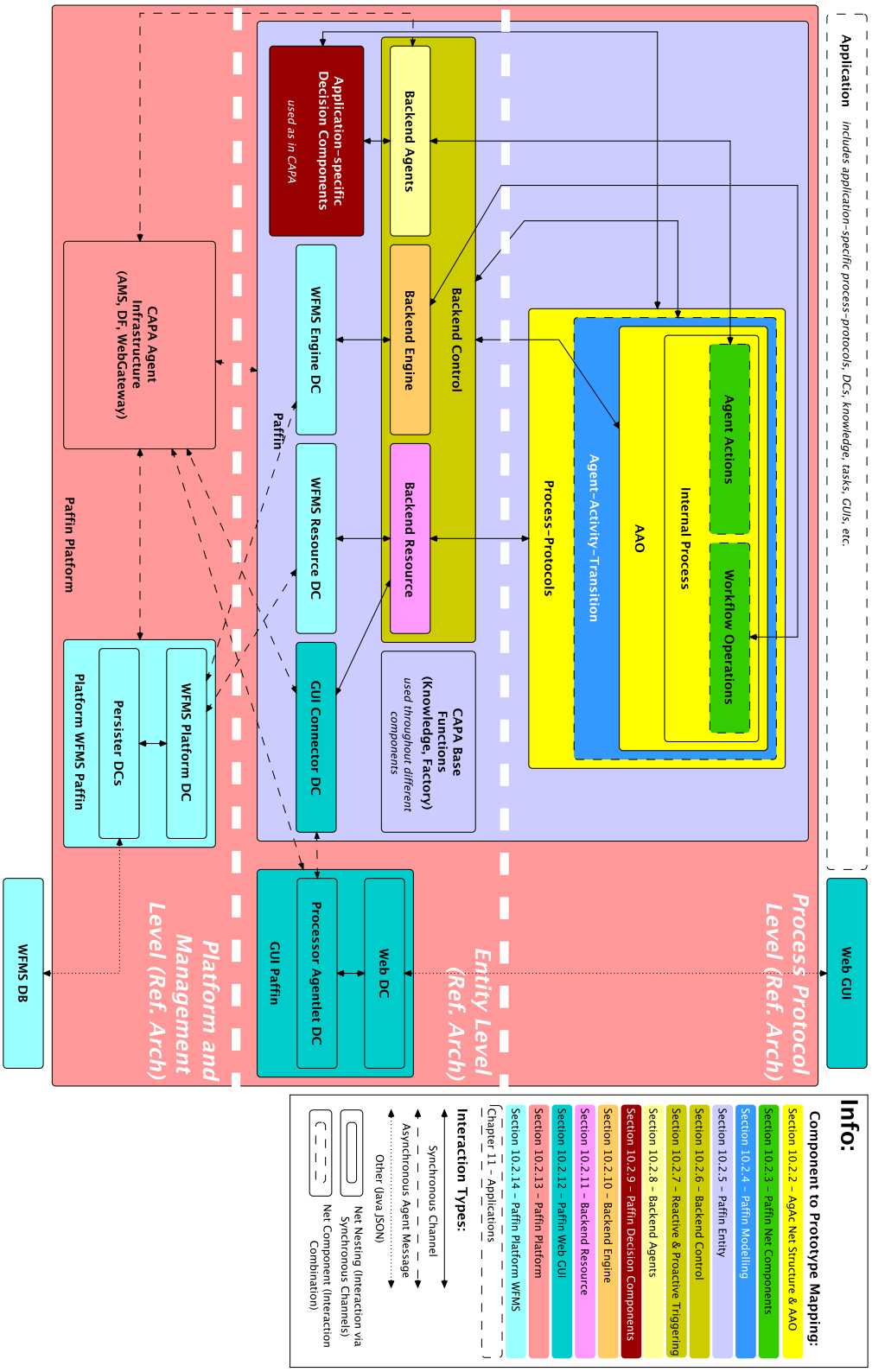

Figure 10.1: Paffin-System architecture overview(modified from [Wagner et al., 2016b])

in the regular Paffin, which communicates with a standardised DC in the GUI Paffin, which in turn uses the Capa WebGateway to provision and display the GUI.

On the platform and management level, the basic Capa infrastructure is maintained and extended with a platform-specific WFMS implemented as a standardised Paffin entity. That standardised platform WFMS Paffin entity has the same significance as and is started automatically alongside the standard AMS and DF Paffins adopted from the FIPA standards (see Section 2.2.2) and the Capa platforms. The platform WFMS utilises standardised decisions components to control interaction with engines and resources, as well as with an external database used for accessing persistently stored data.

The legend of Figure 10.1 indicates where in the following sections each component is discussed in higher detail.

The purpose of the Paffin-System is to provide a framework for the creation of applications utilising an integration of agents and workflows. Consequently, Figure 10.1 indicates the applications as the topmost level of the architecture building on all components beneath it. Most of the components of the Paffin-System architecture are standardised. Only a few of them need to be actively modelled for an application. These include the process-protocols and all components contained within them, the decision components, the knowledge, the database task data and the GUI. Application modelling in the Paffin-System is discussed in Section 11.1 before the specific applications are presented. In that section the overall architecture is also revisited in Figure 11.1, in which the artefacts required for application modelling are highlighted.

In order to get the different components of the architecture to understand one another, a common vocabulary is required. This vocabulary is provided by an ontology in Capa, describing Java classes that correspond to concepts that are used within a system. Each concept has a defined number of fields of arbitrary (Java) types. The ontology mechanism from Capa was directly adopted into the Paffin-System. The main ontology diagram for the Paffin-System can be seen in Figures 10.2 and 10.3[7]. Figures 10.2 and 10.3 do not contain all concepts of the Paffin-System. Additional concepts were introduced with the GUI plugin. Those additions are discussed in the context of the corresponding prototype.

The ontology diagram in Figures 10.2 and 10.3 describes all the concepts that are used in the framework provided by the Paffin-System. Applications built in the framework require access to only a few of the concepts. In general these concepts are mostly used for framework-internal management purposes. Due to the size of the ontology model it is unreasonable to present each concept individually, many of which only differ in slight, semantic ways. The following instead presents the basic groups of ontology concepts to provide an overview.

As an illustrative example to provide a better understanding of the role of the ontology, take the concept of `paffin-task` for example. It can be found as the top right-most element of the *Workflow Concepts* group of concepts in Figure 10.3. A `paffin-task` describes a workflow task within the Paffin-System. It has seven fields: A name, a title, a description, a set of required permissions, an assignable flag (indicating if this task can be automatically assigned) and two objects designating what a Paffin requesting this task must do to start the associated work either with a human user with a GUI or as an automatic resource itself. A `paffin-task` object contains all the general data of a task and defines, for the components of the Paffin-System, how that data can be accessed. Inheriting from the `paffin-task` concept are the `paffin-workitem` and `paffin-activity` concepts which enrich the task data with runtime and execution data.

[7] Due to the size of the ontology diagram it is split into two figures in this document. It still is only one modelling artefact.

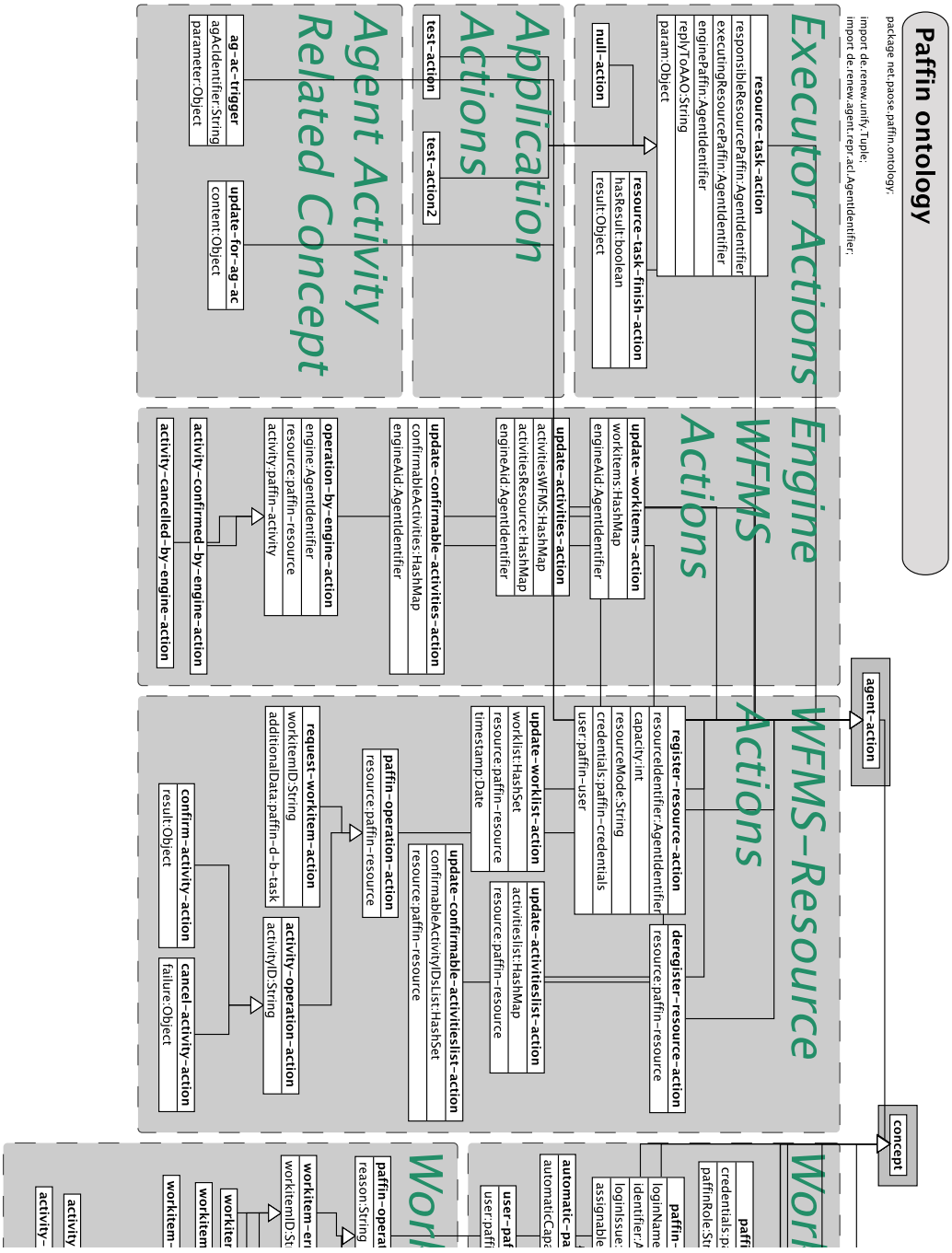

Figure 10.2: Paffin-System ontology (without GUI concepts), part 1

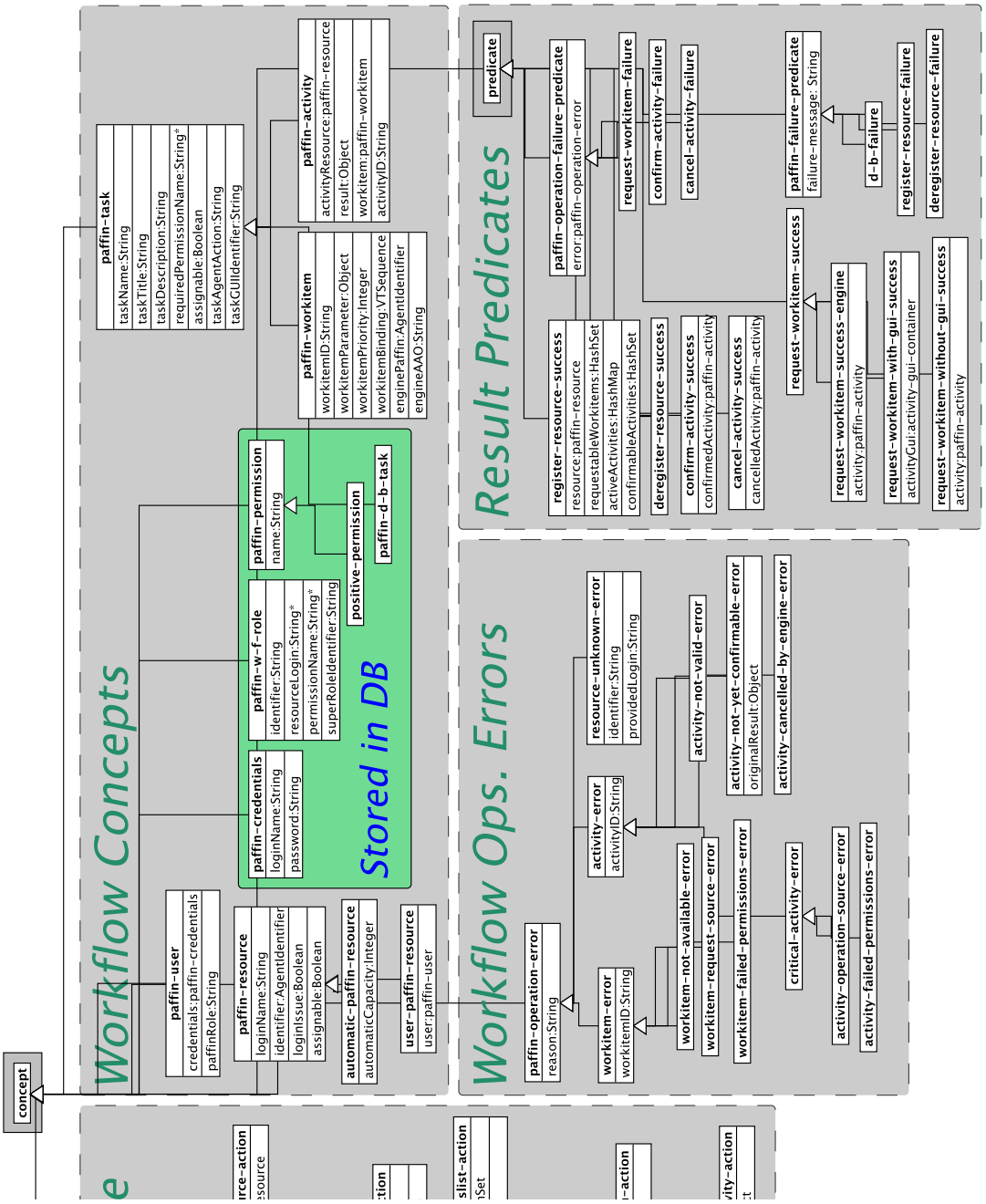

Figure 10.3: Paffin-System ontology (without GUI concepts), part 2

As an explanatory note for Figures 10.2 and 10.3, there are three standardised concepts in each ontology diagram: `concept`, `agent-action` and `predicate`. These are used by the RENEW ontology plugin to create specific forms of ontology concepts that provide slightly different functionality. Ontology entries inheriting from the `concept` superclass are generic concepts/objects containing getters and setters for each field. Entries inheriting from `agent-action` describe specialised messages being sent from one agent (here PAFFIN) to another. Note that `agent-action` does not directly refer to fundamental agent actions in the sense that they are used in this thesis. Incidentally though, `agent-action` concepts are compiled into Java objects, which are sent as agent messages, which does indeed relate them to that particular fundamental agent action. Finally, `predicate` concepts describe special ontology objects that are intended as answers and notifications or results for `agent-action` concepts.

Executor Actions and Application Actions: Executor actions are used in workflow tasks that are executed by PAFFIN entities as resources. These PAFFIN entities need to know which process-protocol to start for a task and ontology concepts inheriting from the ResourceTaskAction are used to determine this. The Application Actions are test actions that illustrate how to use executor actions.

AGENT-ACTIVITY *Related Actions:* This group of actions are used to trigger AGENT-ACTIVITIES reactively or inform their PAFFIN of updates to them at runtime.

Engine WFMS Actions: Actions that are exchanged between PAFFIN entities acting as workflow engines and the platform WFMS are contained in this group. These include actions to inform the WFMS of available workitems, (confirmable) activities and of operations performed directly by the engines on tasks.

WFMS Resource Actions: This group of actions contains those actions that are exchanged between the platform WFMS and PAFFIN entities acting as resources. Actions for registering and deregistering a resource, updating different lists of workitems and activities and initiating workflow operations are contained in this group.

Workflow Concepts: This group contains the concepts directly related to workflow management. Resources, users, tasks, workitems, activities, credentials, roles and permissions are described here. These concepts allow the system to manage elements described by these concepts. This group also contains the concepts that are stored in the database (indicated in Figure 10.3). These concepts contain additional data and information about workflow management that needs to be persistently stored.

Workflow Ops. Errors: Concepts from this group are used to indicate that workflow operations have experienced some kind of error. The platform WFMS can observe these errors and determine how to solve the existing issues according to these concepts. Possible errors include, for example, invalid activities, invalid credentials of a resource, missing permissions etc.

Result Predicates: The result predicates group contains predicate concepts that are used to inform all participants of the WFMS (engines, resources and the platform WFMS) of the results of workflow operations. These are used to confirm or acknowledge operations or inform participants of errors that have occurred. The concepts of the workflow operations errors group are used as content for the concepts of this current group.

Clearly, the ontology diagram for the PAFFIN-System is very workflow-centric. Except for the executor actions and the AGENT-ACTIVITY related actions groups, all other groups

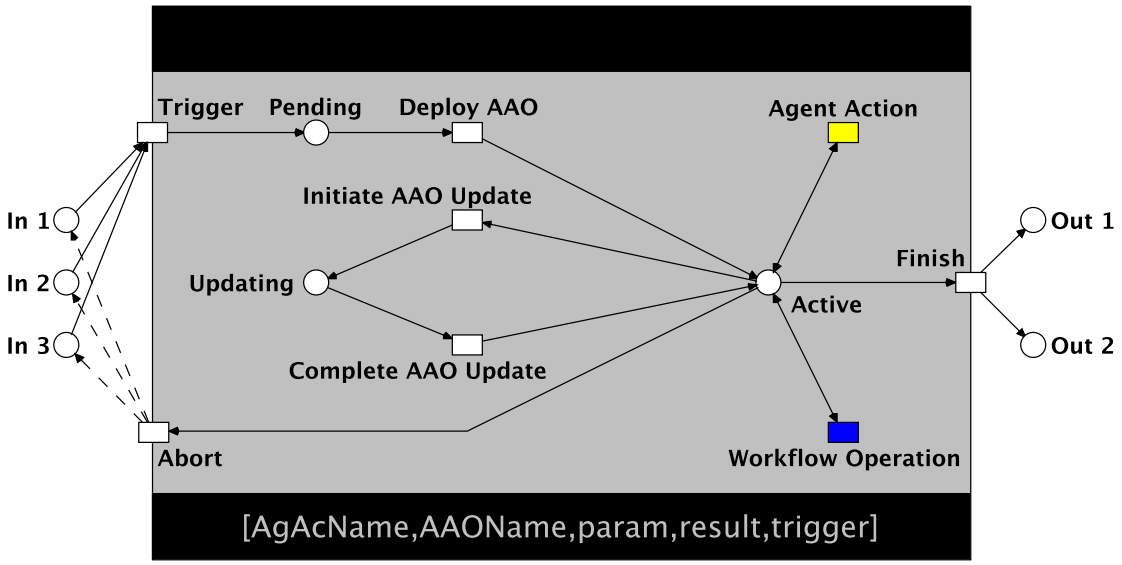

Figure 10.4: Conceptual net structure of the Agent-Activity

are concerned with workflow management. This can be traced back to the Capa origin of the Paffin-System. To reiterate, the Paffin-System builds upon Capa and incorporates workflow and integration management into that framework. Consequently, all ontology concepts regarding exclusively agents are provided by the Capa basis and do not need to be redefined.

10.2.2 The Agent-Activity Net Structure and AAO

Purpose and Functionality The Agent-Activity net structure defines the complete life-cycle of an Agent-Activity, from triggering to finishing or abortion. Within that net structure, an instance of an Agent-Activity, including all data, is represented and encapsulated by an Agent-Activity-Object (AAO, see Definition C.6). The Agent-Activity net structure represents the realisation of the Agent-Activity concept (see Definition C.2) and the main modelling building block for the process-protocols (see Definition C.8) of the reference architecture of the Agent-Activity approach (see Definition C.7). The required functionality of both the Agent-Activity net structure and the AAO are equivalent to the basic Petri net model for an Agent-Activity construct from Section 9.1.2.

Design The coarse design of the Agent-Activity net structure can be seen in Figure 10.4. The only directly discernible differences between the model of Figure 10.4 and the basic Petri net model from Section 9.1.2 and Figure 9.3 is that the triggering and updating an Agent-Activity are no longer instantaneously handled by one transition. This change was made, since instantaneous triggering and updating would limit the management options. Instantaneous triggering would require the AAO to be instantly ready for execution, which would only be possible to implement for very simple variants that could not handle complex states, parameters and internal processes. An instantaneous update mechanism faces analogous challenges.

In the Paffin-System, triggering happens in two steps. The first step retains the name *trigger*, since it is the first administrative action being performed for an Agent-Activity

instance. Triggering the Agent-Activity in the Agent-Activity net structure initiates the creation of the AAO in the technical backend of the Paffin entity and puts the Agent-Activity into a *pending* status. After the AAO is created and all management preparations[8] are complete, the AAO is passed back to the Agent-Activity net structure in the second step named *deploy AAO*. Updating also happens in two steps. The first step, *Initiate AAO Update*, initiates the update with the technical backend by handing over the existing AAO. The technical backend can then update the AAO as desired, while the Agent-Activity net structure is in the *Updating* state. Once the AAO has been updated by the technical backend, the updated or new AAO is handed back via the *Complete AAO Update* transition to the *Active* place. Here it can resume or restart its work according to the update.

From a coarse design perspective, the remainder of the Agent-Activity net structure directly corresponds to the basic Petri net model from Section 9.1.2

Two major design decision need to be highlighted for this prototype:

Functionality range of the AAO: The capabilities of the AAO are another important design decision made during the development of this prototype. The AAO encapsulates all data and state pertaining to the execution of one Agent-Activity instance. As such, it not only needs to be able to store parameters and other runtime data, as well as the internal process, it also needs to provide an interface between the description of agent actions and workflow operations in the internal process and the technical backend of the Paffin entity. Furthermore, access to runtime data also needs to be provided to the internal process, which might use it in its actions and operations. Consequently the design of the AAO requires the following functionality:

- Initialising the Agent-Activity
- Storing and providing access to the runtime parameters
- Storing, instantiating and providing access to the internal process
- Providing the interface for the execution of agent actions and workflow operations
- Providing management access for workitem and activity lists
- Terminating the Agent-Activity

As the runtime data storage, interface and management functionality is required for *all* Agent-Activities the design also specifies that that part of the AAO is standardised. This leaves the internal process of the Agent-Activity as the only part of the AAO that is specific to a particular Agent-Activity. Consequently, from a modelling perspective the AAO can and commonly is equated to the internal process it contains.

Agent-Activity *parameters:* The Agent-Activity parameters can be seen in the lower bar in Figure 10.4. They represent the data objects necessary for the execution of an Agent-Activity. The parameters, in order, are:

- The name of the Agent-Activity
- The name of the AAO
- The input parameters as variables from the preconditions
- The result as a variable for the postconditions
- The triggering mode

[8] These are discussed in the prototype for the technical backend control in Section 10.2.6.

The name of the Agent-Activity identifies the Agent-Activity for management purposes. The name of the AAO identifies the specific AAO used in this Agent-Activity. As discussed above, the AAO is standardised except for the internal process it contains, which means the name of the AAO is, in practice, the name of the (non-standardised, application-specific) internal process. By distinguishing between the name of the Agent-Activity and the name of the AAO/internal process it is possible to realise the same Agent-Activity in different variants by simply exchanging the AAO. The remainder of the Agent-Activity parameters are the variables for input variables to identify (unify) data objects from the preconditions to be used during the execution of the internal process and output and result variables to generate the output and results as tokens on the postconditions. The triggering mode uses either "direct", "reactive" or "proactive" String objects to indicate the triggering mode. Ultimately, these parameters are used as inscriptions for the Agent-Activity as a transition (see prototype in Section 10.2.4).

Implementation Overall, the implementation of the Agent-Activity net structure and AAO is incorporated into the `PaffinEntity` plugin. The implementation of the Agent-Activity net structure is a reference Petri net structure including all inscriptions and details. It can be seen in Figure 10.5. Note that the variant in Figure 10.5 shows the net component variant of the Agent-Activity net structure, a reusable modelling component discussed in Section 10.2.3. The AAO is also implemented as a reference net and can be seen in Figure 10.6. The net in Figure 10.6 hides the implementation details in the initialisation and state/data access for readability purposes. As a reference net, the AAO is naturally embedded into the Agent-Activity net structure as a token, so that the interplay between the two can realise and manage the life-cycle and execution of an Agent-Activity as specified in the basic Petri net model from Section 9.1.2.

Most details of the Agent-Activity net structure are directly discernible from Figure 10.5, especially due to the similarity to previous models of the Agent-Activity construct. However, some points need to be addressed explicitly. The parameters are provided as a Tuple token in a place connected to the triggering transition. It was implemented that way to be more compatible with the vision (and later implementation, see Section 10.2.4) of the Agent-Activity as a single transition. By providing the parameters as an input token object, it was also possible to utilise variables in the trigger transition inscription. This, in turn, enabled the syntax check provided by Renew to be better utilised, improving usability for system modellers. Another interesting detail pertains to the central place, where the AAO is placed during the execution of the internal process. Here, a Tuple token containing both the AAO and the internal process (as a net) are used instead of just the AAO. This implies a separation between the AAO and the internal process, which is, however, not the case. Rather, the internal process is still completely encapsulated and controlled by the AAO. The *reference* token to the internal process was only added to the Agent-Activity net structure to provide direct access for inspection. Without this reference inspection would only be possible by first accessing the AAO and then accessing the internal process. Eliminating this, admittedly rather minor, inconvenience did improve modelling.

The AAO net shown in Figure 10.6 realises the complete functionality required of an AAO. Most of the net implements the required functionality in a straightforward way. The upper parts, hidden in Figure 10.6 for readability purposes, realise the initialisation, storage of and access to runtime data and the instantiation of the internal process, provided during instantiation as a net name identifier. The only prerequisite of that identifier is uniqueness

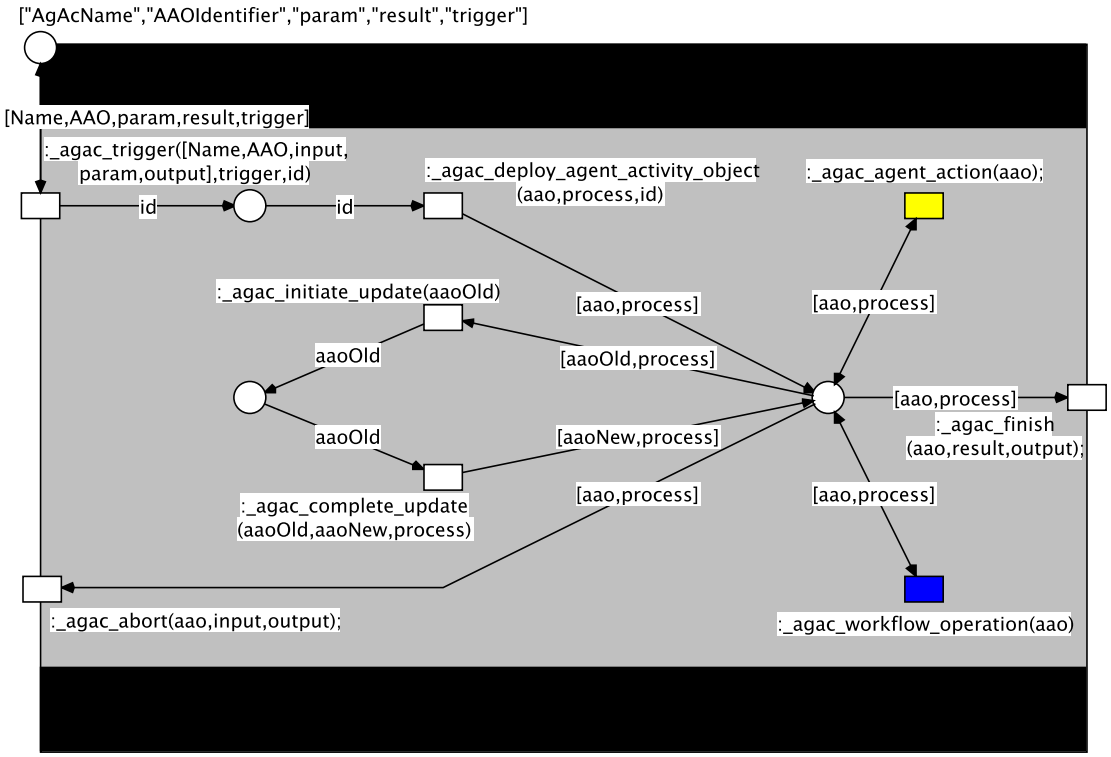

Figure 10.5: Actual structure of the Agent-Activity (net component version)

and that the net is available somewhere in a folder in the known Renew net path. The middle part implements the workflow operation functionality. Providing the interface for workflow operations and the creation of workitem and activity lists are handled here. In the lower left part the interface for agent actions is provided, being markedly simpler than the workflow interface due to the Capa basis of the prototype. Finally, the lower right part implements the termination of the AAO in various ways. Correctly finishing the Agent-Activity, aborting it, resetting it (as an update mechanism) and "killing" it (instant removal from the management system in case of critical failures) are supported.

Conclusion The purpose of this prototype was to realise the core modelling elements of the Agent-Activity integration approach. Building directly on the specification provided by the Agent-Activity concept, the prototype completely fulfils that purpose. The Agent-Activity net structure defines the basic life-cycle, while the AAO encapsulates all associated instance data and states. Some design decisions have been made causing minor modifications from the concept that ultimately improved the modelling and management effort. Overall, there is also very little that can or needs to be improved in future extensions. Changes and extensions elsewhere in the system might affect the AAO or Agent-Activity net structure, but no major functionality can be added directly to them.

10.2.3 Internal Process and Paffin Net Components

Purpose and Functionality Each Agent-Activity contains an internal process of agent actions and workflow operations. Modelling of this internal process (see Definition C.5) is the subject of this prototype. The required functionality for this prototype is the provision of modelling elements for the internal process of an Agent-Activity.

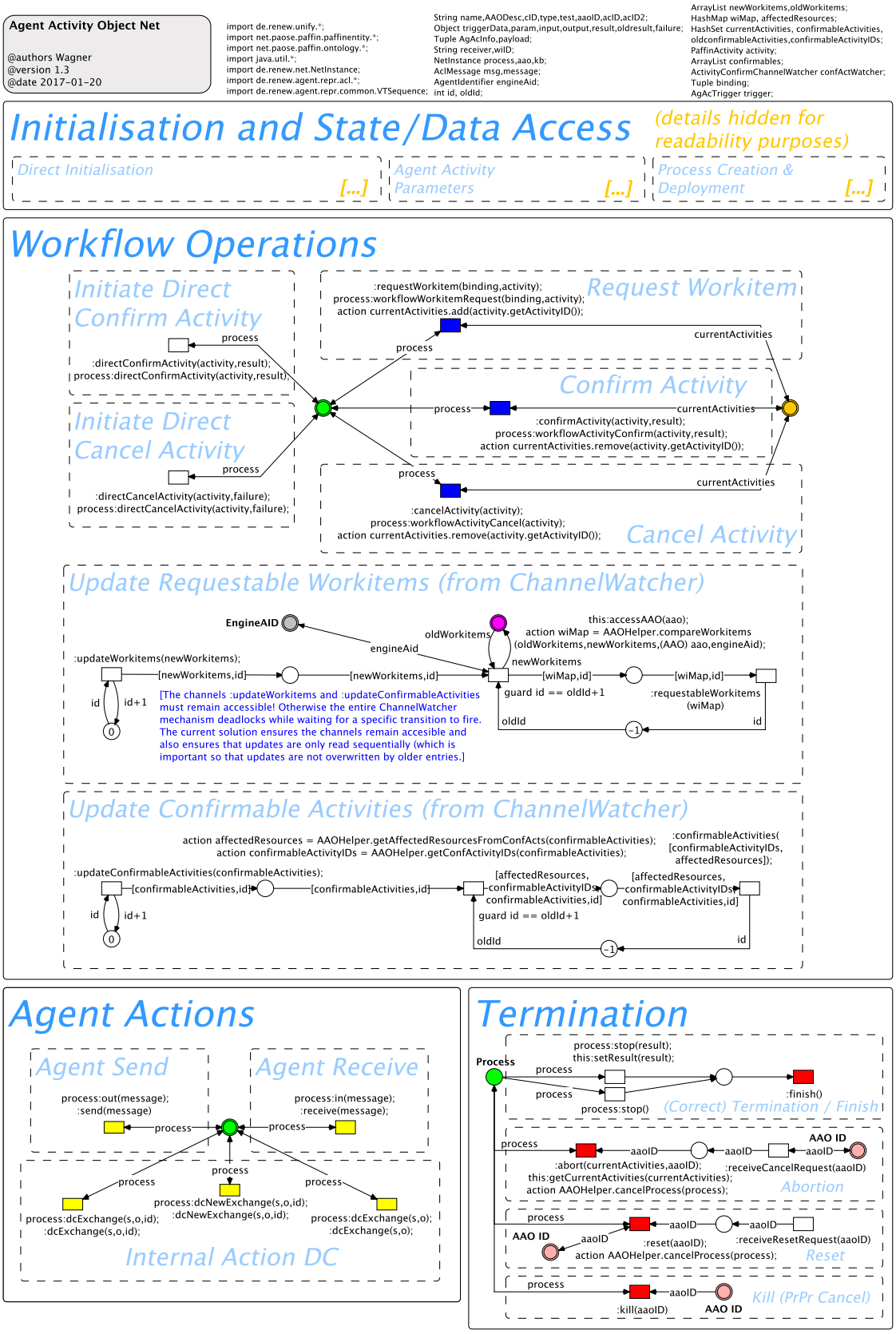

Figure 10.6: The AAO net (compact version)

Design Overall, the internal process is implemented as a reference net. Reference, and Petri nets in general, excel at representing processes, which is why this choice is only natural. It also maintains the focus on the same modelling artefacts (i.e. reference nets) throughout the prototypes.

In order to support modelling, net components, as introduced by [Cabac, 2002], have been chosen as modelling elements. Net components are an established and proven tool in Paose (see Section 2.2.3), which improves modelling effort through reusable, easily recognisable and standardised modelling elements.

The coarse design of this prototype defines, which actions and operations should, by default, be supported in an internal process of an Agent-Activity. Of course, the six fundamental agent actions and workflow operations, as the main elements of the internal processes, are included. In addition to that, administrative actions, i.e. start and stop of the internal process, are also included. For each action, operation and administrative action net components have been implemented.

As a core element of the overall Agent-Activity approach there are a number of important design decisions related to this prototype:

Connectivity of actions and operations: This design decision relates to the interface of the internal process to the AAO and technical backend of the Paffin entity. Basically, since all of the involved components are implemented as nets the technical shape of the interface is prescribed as synchronous channels. To more easily distinguish the different actions and operations, each of them uses a different, unique and standardised channel. The agent action internal action is different, however. Since it happens internally to the Paffin entity, it does not need a channel to the technical backend or the AAO. However, internal actions might require access to the knowledge base of the Paffin entity or to another internal component (e.g. another subnet). Consequently, the inscription of the internal process is not restricted in any way regarding synchronous channels. In fact, some additional synchronous "shortcuts" that allow access from the internal process, which is referenced only in the AAO and Agent-Activity net structure, to other internal components, like the knowledge base or decision components, needed to be provided. Furthermore, the internal process needed to have standardised start and stop administrative connections to the AAO that also utilise specific, unique synchronous channels.

Engine confirm/cancel: Traditionally, resources confirm or cancel the execution of a workflow activity. Since in the Agent-Activity approach Paffins representing workflows can also act autonomously and intelligently as agents, the capability of confirming or cancelling no longer needs to be limited to the resources. A Paffin entity representing a workflow and equivalently being the workflow engine, should be able to directly and autonomously confirm or cancel an activity.

An example of how this could be used is if a resource takes too long to complete a time critical task. In that case the workflow (engine) Paffin could directly cancel the activity and reenable it as a workitem for another resource. Another example transfers the quality control to the workflow (engine) Paffin. The resource sends (as an agent action) the result to the workflow (engine). The workflow (engine) then evaluates and checks the result and confirms the activity only when the result is adequate, otherwise it cancels the activity and tries again.

Direct engine operations have been implemented for the Paffin-System and are also available as net components. Note that confirm/cancel and direct confirm/cancel do

not represent different workflow operations. They are, however, modelled separately to explicate their usage and simplify the interface to the technical backend.

Task relation: Workflow operations always relate to one specific task. In fact, a triplet of request workitem, confirm activity and cancel activity wholly describes one workflow task in an internal process of an Agent-Activity. However, the operations are modelled separately to allow for freely structuring and combining them with agent actions or for nesting them in other tasks. Consequently, a modelling issue arises of how to relate different workflow operations to one particular task. To solve that issue, the design decision was made to store the activity data (containing the task data) in a place once the request workitem is successfully completed. That place could then be connected to a confirm or cancel activity operation that uses that activity data to identify the correct task and confirm or cancel it. This represents a flexible net-based mechanism that avoids any specific "hard-coding" of activity data into the net which would be error prone and more difficult to handle by modellers.

Directly related to the task relation is another design decision. Triplets of workflow operations wholly describe a task, as described above. However, should it be allowed to start a task in one Agent-Activity and end it in another? Use cases for this kind of behaviour are feasible, as longer-running tasks may require complex internal behaviour. That behaviour might be too complex to justify incorporating in only one Agent-Activity. Consequently, the decision was made for the Paffin-System to support splitting workflow tasks between Agent-Activities. This required minor adjustments to the technical backend, to realise a handover of management associations between Agent-Activities and workflow tasks.

Scope of sending/receiving messages: A final important design decision regarded the scope of sending and receiving a message, which in the conceptual approach is (deliberately) ambiguously defined. The first question is, if technically sending or receiving the message by an entity/agent is part of the action or if the action is confined to the internal process. Here, the clear separation between the standardised parts and the application-specific parts has been used to answer the question. Sending or receiving a message is always handled in the same way by the Paffin entity. This principle mechanism was adopted with little changes directly from Capa. Only the initiation of sending or waiting for the reception of a message needs to be modelled for each application. Consequently, from a modelling perspective only the latter should be considered as the corresponding action. Conceptually, the technical sending or receiving of the message is also part of the action, although this is irrelevant to modelling and can therefore be ignored by it.

The second question is, if the creation of the message object (for sending) and the processing of the message object (for receiving) are part of the send/receive actions. While messages are always created before being sent out and (usually) processed after being received, these should not be considered as part of the sending or receiving. The creation or processing of a data object outside of the context of messages are clearly internal agent actions. For conceptual clarity these actions should also be considered as separate internal agent actions even if they are directly followed by or follow sending or receiving a message.

Implementation For each of the elements discussed in the coarse design, net components were created using the existing net components plugin for Renew. That plugin only requires the net components as individual reference net files and icons for each net component.

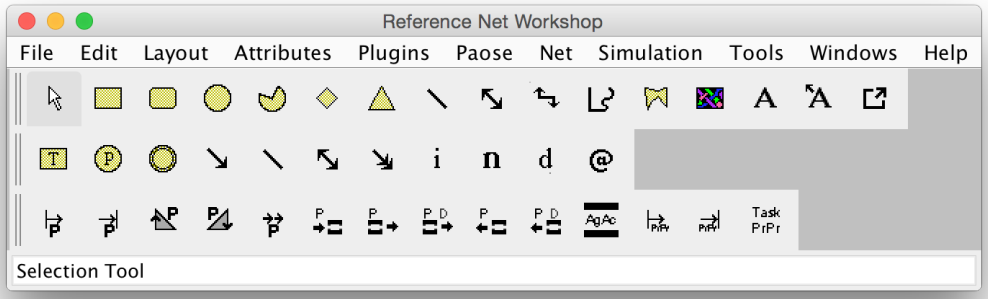

Figure 10.7: PAFFIN net components incorporation into RENEW main window

Those files are provided as a separate plugin, `PaffinComponents`. The net components can be loaded into a running RENEW instance via the menu commands provided by the original RENEW net components plugin. Figure 10.7 shows the loaded net components in the RENEW main menu (lowest row of buttons). Selecting one of the components and then clicking on a net drawing creates the full net component for the selected action/operation in that net drawing.

Figure 10.8 shows an example of an internal process created only with net components. Individual actions and operations are indicated by boxes with dashed lines. The process starts with the administrative start net component which also adds the declaration node and the comment box in the top left corner. Next, the process awaits an agent message. Here, the separation between receiving the message and processing (at the top) is clearly visible. These actions are modelled together in one net component, yet they are clearly distinguished. After the message reception a workitem request action is performed. When the final transition of that block fires all data tokens relevant to the activity are put onto the slightly enlarged place. That place can be connected to a confirm or cancel net component. Those net components are designed with only a placeholder place to remind modellers to connect a task to them. The confirm block in Figure 10.8 has been connected to the task (via virtual place) while the cancel block below it retains the placeholder for illustrative purposes in this example. Note that when the activity is confirmed the process can continue directly, but when the activity is cancelled, it happens in two steps. The first is to inform the technical backend that the cancellation can happen and the second, controlled by the technical backend, fires the final transition of the cancel block and the bottom of the associated request workitem block synchronously. That firing restores the preconditions of the workitem request block, thus reenabling the workitem. Assuming the activity was confirmed successfully, the process proceeds to generate a message as an internal action, which is then sent. Again, the clear separation of creation of message as internal action and actually sending the message is visible. The internal process is completed by the administrative stop action.

Note that management and control of the internal processes is handled by the responsible AAO. As discussed in the previous section, the AAO instantiates and terminates the internal process, as well as providing the interface between the synchronous channels defined for the actions/operations in the net components and the management facilities in the PAFFIN entity.

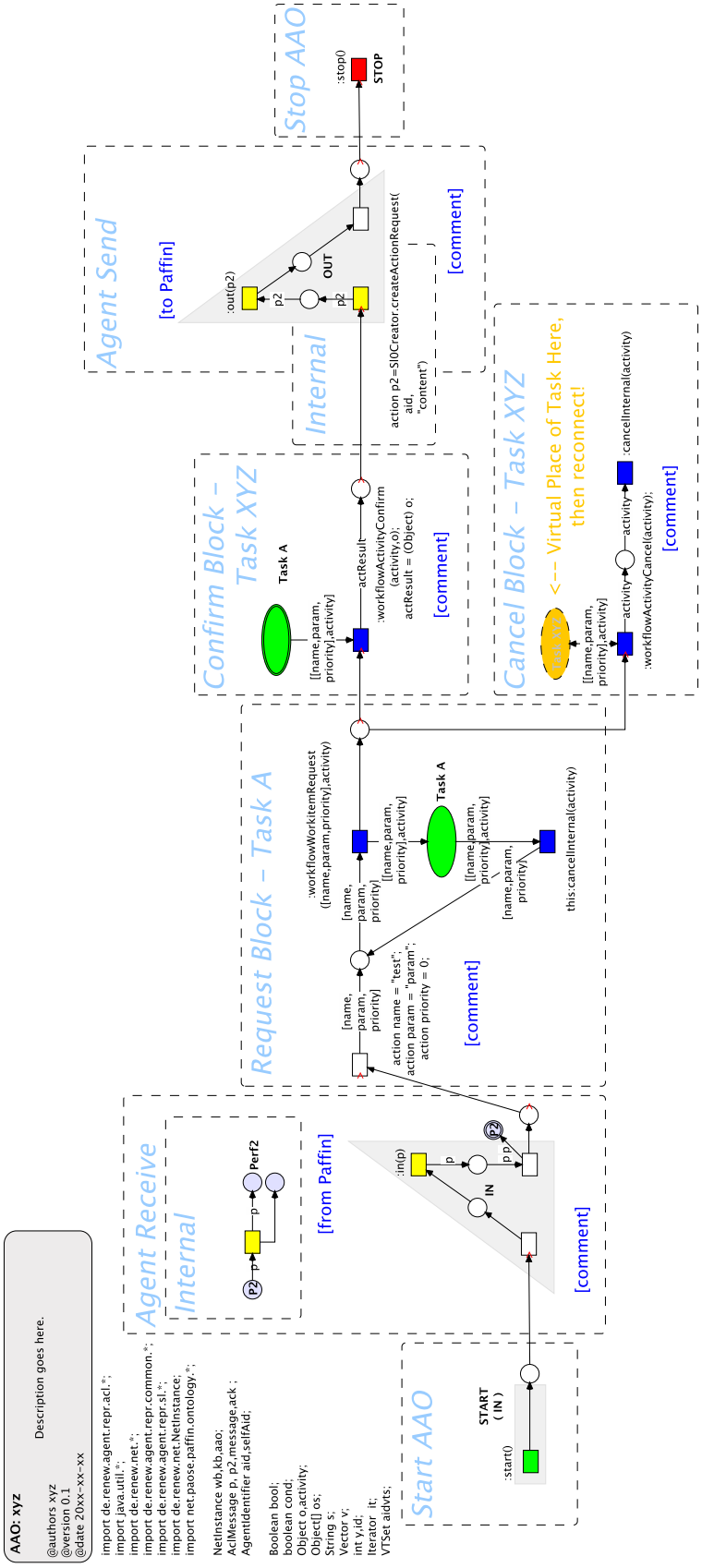

Figure 10.8: Example usage of net components in an internal process

Further net components for modelling process-protocols were also implemented in this prototype. The net component for the Agent-Activity net structure can be seen in Figure 10.5. Others pertain to administrative actions in the process-protocols and are not further discussed here.

A final implementation note concerns splitting tasks among multiple Agent-Activities. While use cases for this are feasible and it should generally be supported, it was decided not to implement special net components for it. It is realised by using the activity/task data of a activity as the result of the Agent-Activity it was started in. That result is passed to the process-protocol where it is used at a later point as the input parameter for the Agent-Activity where the activity should be completed. That later Agent-Activity needs to process the input parameter and inform the management facilities of the new execution location for the activity (all as internal actions). It can then be confirmed in that Agent-Activity. Note that cancellation is limited when using this feature. From a management perspective the activity can be cancelled and removed, but since the process in which it was started has been completed and is gone, no rollback can occur. This is a limitation that can't be handled automatically. It is only feasible to manually model a rollback within the process-protocol to the original Agent-Activity. An example of splitting a workflow task is provided and discussed in the application chapter.

Conclusion The internal process of an Agent-Activity is one of the key elements of the Agent-Activity approach. Ultimately, the internal process is what describes the behaviour of an integrated entity as an agent, a workflow or a hybrid. It is largely responsible for describing the degree and scope of an integration an entity uses. Consequently, its shape and form as well as modelling support for it are highly important. This prototype defines that shape and form, while also providing modelling tools that have been established and proven in Capa and Paose. Through net components it is possible to create an outline of the internal process, which only needs to be enriched with further detailed inscriptions and data/control flow. Distinct net components make the individual actions and operations instantly recognisable and prescribe a common structure throughout systems that utilise them. Therefore, the requirements made of this prototype regarding purpose and functionality have been fulfilled.

Concerning future work, additional net components are feasible. However, such net components would not add any new functionality, since the fundamental agent actions and basic workflow operations already encompass the entirety of agent and workflow behaviour. Rather, additional net components could be used for commonly used combinations of agent actions and workflow operations. For example, net components for receiving a message and creating a task from the received data could be provided.

10.2.4 Paffin Modelling

Purpose and Functionality Modelling with large net structures, like the Agent-Activity net structure seen in Figure 10.5, is quite cumbersome. Net layouts increase in size and readability suffers due to the large number of standardised and therefore modelling-wise needless additional transitions and places. Rearranging and editing nets with such repeating patterns can also be quite frustrating, even with grouping mechanisms.

Furthermore, while the net structure described in Section 10.2.2 correctly implements the life-cycle of an Agent-Activity it doesn't quite capture the concept on the modelling level. An Agent-Activity combines different agent actions and workflow operations into *one* abstract activity. From a modelling standpoint, the Agent-Activity is supposed

to be an elemental building block. A representation as a complex net pattern does not capture this aspect at all.

Consequently, the purpose of this prototype is to improve modelling with Agent-Activities. Following the idea of the task-transition for Renew workflow nets (see Section 2.3.2), the Paffin modelling prototype realises the representation of an Agent-Activity as one transition in a process-protocol, the so-called Agent-Activity-Transition. The Agent-Activity-Transition obscures the full, complex Agent-Activity net structure for system modelling and monitoring. It aims to remedy the issues of readability of nets and conceptually atomic representation.

Key Term Definition C.12 (Agent-Activity-Transition)**.** The Agent-Activity-Transition implements the Agent-Activity concept and Agent-Activity construct as a single transition in a process-protocol of the Paffin-System. The Agent-Activity-Transition is translated for execution into the full and complex Agent-Activity net structure of the Paffin-System, although it maintains the singular transition representation in the GUI.

Design After modelling a reference net in Renew that net is compiled into a shadow net that abstracts from the graphical representation and adds technical details irrelevant to modelling but required for execution and simulation. In that compilation step, the Agent-Activity-Transition is automatically compiled and translated into the Agent-Activity net structure. This compilation retains all incoming and outgoing connections from the Agent-Activity-Transition and connects them correctly in the shadow net. Renew is then capable of executing the shadow net containing the complex Agent-Activity net structure, while representing the original net with the Agent-Activity-Transition. The representation net always displays the correct state (including firing) of the shadow net.

One of the main challenges of the Agent-Activity-Transition is the realisation of the local rollback in case the Agent-Activity is aborted. For this challenge, the net structure connects the abort transition with all preconditions and inscribes the variable originally removed for triggering. The Agent-Activity-Transition creates these abort-arcs in the shadow net automatically.

Since the task-transition for workflow nets already implemented similar functionality, it was decided that using that functionality as a basis for the implementation was the best approach. It conserved implementation effort and reused already established functionality.

For the Agent-Activity-Transition inscription an approach similar to that of the task-transition was chosen as well. The parameters for an Agent-Activity were already discussed in Section 10.2.2. These parameters represent all data that is required for the execution of an Agent-Activity. Therefore, these parameters need to be inscribed onto an Agent-Activity-Transition. As with the task-transition the parameters are inscribed as a tuple. A tuple can be easily processed by Renew and presents the data in a structured way. The form of the chosen tuple is `[AgAcName, AAOName, param, result, trigger]`.

Implementation As described before, the prototype uses the functionality of the task-transition for Renew as its basis. The source code from the `WFNet` (Workflow net) Renew plugin was duplicated into a new plugin called `PaffinModelling`. After refactoring the code to its new location in the Renew plugin architecture, work began on adopting the task-transition source code for the implementation of the Agent-Activity-Transition. While the implementation utilised existing code, the implementation effort was still considerable.

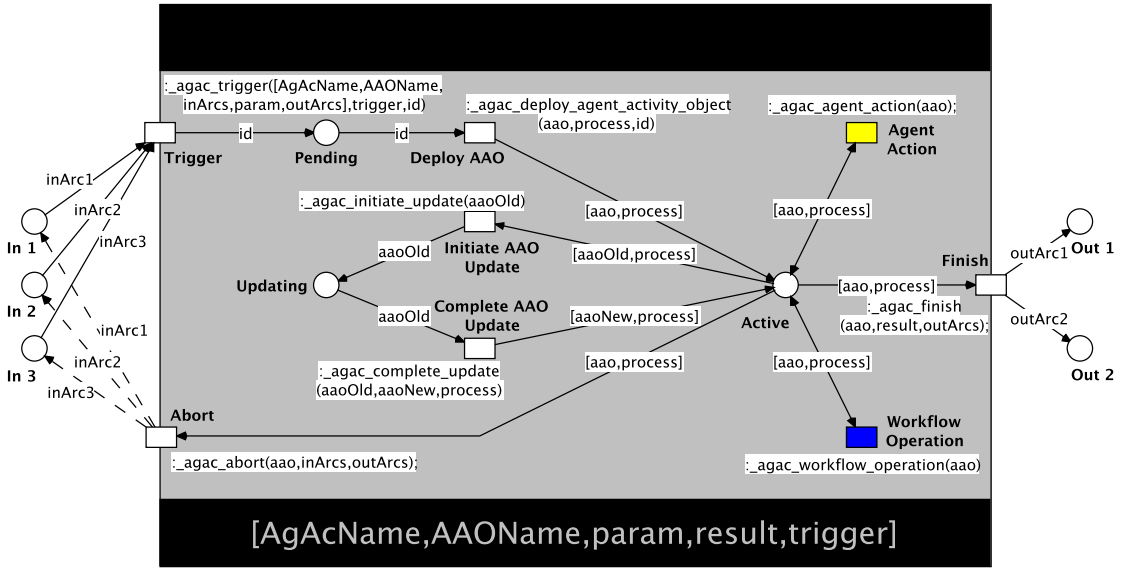

Figure 10.9: Internal net structure of the Agent-Activity-Transition

The task-transition, with three transitions and one place, has a much simpler structure than the Agent-Activity net structure with eight transitions and three places. Representing and implementing the translation in Java code was consequently more difficult.

The prototype encompasses nine Java classes with roughly 1500 lines of code. The source code encompasses the entire `PaffinModelling` plugin. The core of the functionality can be found in class `SingleProcessProtocolCompiler`, which actually compiles Agent-Activities.

Generally, all elements of the net are examined by the compiler. For all elements except the Agent-Activity-Transition the regular Renew Java net compiler is invoked. Then, each Agent-Activity-Transition is transformed, as Java code, into the Agent-Activity net structure. The basic process is as follows:

1. Check if the provided inscription complies to the prescribed form. If the check fails abort the compilation with an error message.
2. Read the individual objects from the inscription
3. Create the eight transitions of the Agent-Activity net structure
4. Determine the expressions that need to be inscribed to the arcs from the precondition to the triggering transitions, from the finish transition to the postconditions and from the abort transition back to the preconditions. These expressions are also required for the transition inscriptions.
5. Create the inscriptions for all transitions using the previously determined arc inscriptions and data obtained from the overall Agent-Activity inscription.
6. Create the three places of the Agent-Activity net structure
7. Create the inner arcs of the Agent-Activity net structure and inscribe them
8. Create the outer arcs (i.e. arcs to pre- and postconditions of the Agent-Activity) and inscribe them

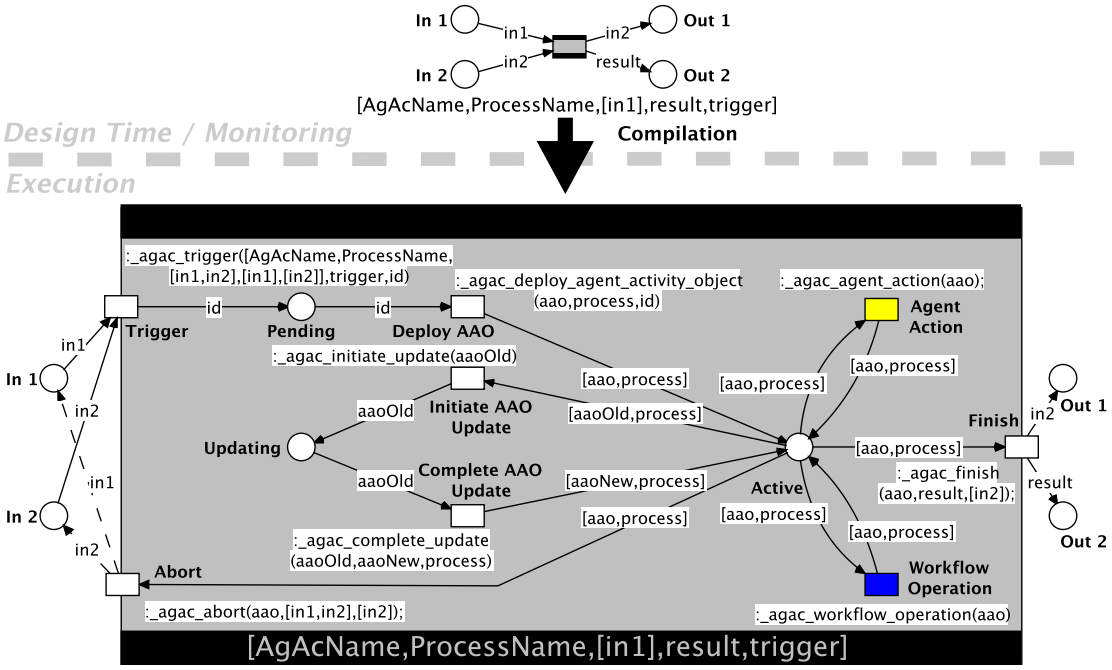

Figure 10.10: Agent-Activity compilation example (from [Wagner et al., 2016b])

The result of the previous process is represented in Figure 10.9. This net shows the generic net structure that is created by the class `SingleProcessProtocolCompiler`. An example, which correctly translates the given inscriptions and shows both the modelling and shadow net views is shown in Figure 10.10. Note that the compiled shadow net does not feature or require any graphical representation of net elements. Consequently, no net layout information is required or used during the compilation. Also note that any representation in figures showing the compiled Agent-Activity-Transition uses the layout of the Agent-Activity net structure, not the (non-existing) layout of the actually compiled code.

When the plugin is loaded, the Agent-Activity-Transition can be used in Renew by selecting the Paffin modelling formalism. The Agent-Activity-Transition then appears as a modelling tool in the main Renew window. It can then be used to draw Agent-Activity-Transitions in any Renew drawing. To distinguish between an Agent-Activity-Transition and a regular or task-transition, a representation using two horizontal thick bars has been chosen. A right-click on an Agent-Activity-Transition creates a default inscription. Handles for connecting places to and from the Agent-Activity-Transition are provided as well.

Conclusion The purpose of this prototype was to improve modelling and capture the Agent-Activity as an indivisible building block of process-protocols. The Agent-Activity-Transition obscures the Agent-Activity net structure while maintaining its behaviour during execution. For system modelling the Agent-Activity is just one, single transition, suitably representing the spirit of the concept. By eliminating the additional transitions, places, arcs and inscriptions of the net structure (at least for the modeller) it also reduces the complexity of the process-protocol nets without sacrificing any application-specific content. It also reduces possible errors by obscuring necessary technical details that might be (unintentionally) changed by modellers.

The concealment, though, can also be an issue. While it hides the details of the Agent-Activity net structure to provide a more concise and manageable representation of an Agent-Activity, it also complicates the inspection of that net structure and access to the AAO and the internal process. The latter was already alleviated by representing the content of the central active place, i.e. the tuple of AAO and internal process, as an inspectable token on top of the Agent-Activity-Transition. The former, however, remains an issue. To solve this issue a graphical switch would have to be provided, which allows for the representation of both the Agent-Activity-Transition and the Agent-Activity net structure when necessary. Currently, this is technically impossible to implement due to the limitations of the graphical user interface of Renew. Still, since inspecting the net structure is a rare demand, the Agent-Activity-Transition's benefits outweigh this issue.

10.2.5 Paffin Entity

Purpose and Functionality This prototype's purpose is to provide an implementation of a construct representing a Paffin entity (see Definition C.11) in the Paffin-System. The required functionality of this prototype is relatively simple, yet vital to the overall implementation of the Paffin-System. What is required is a construct that represents a Paffin entity on the platform and management level of the Paffin-System. That construct must provide the basic frame for the execution of process-protocols containing Agent-Activities. The internal management facilities of a Paffin entity, the so-called technical backend (see Definition C.9), are **not** covered by this prototype, but are rather described in the following ones.

Design The entity construct is a reference net, based on the standard Capa agent net (see Figure 2.11). That reference net is instantiated and managed on the platform and management level of the Paffin-System. It provides the following features and functionality:

- It can execute the technical parts of the fundamental agent actions send and receive message. Only the initiation of sending a message and receiving a message are modelled in the internal process. These actions must then be passed through the technical backend of the Paffin entity, before being actually executed by the Paffin (net) itself.
- It supports the execution of workflow operations and internal agent actions. While the majority of the support for the workflow operations is handled by the technical backend, the Paffin construct net itself must still support these operations by forwarding them to the management on the platform and management level. Additionally, internal actions might require access to internal components of the Paffin entity, like the knowledge base (KB) or DCs. The Paffin construct net needs to enable that access.
- It provides the environment for the technical backend, as well as access to all required components within the Paffin construct.
- It can represent itself as both agent and workflow on the platform and management level of the Paffin-System.
- It instantiates and manages *all* internal components and processes of the Paffin entity.
- It supports factory and knowledge base components.

A number of important design decisions were made during the development of the Paffin entity net.

Maintaining legacy compatibility: One of the first open questions that arose, once adaptation of the standard Capa agent started, was whether Capa legacy code should be supported. On the one hand this would enable utilising most tools, systems, applications and mechanisms built in the previous decade with Capa (and still compatible with the current version) to work with the Paffin-System. On the other hand, support of legacy code would require large parts of the Capa functionality to remain unchanged limiting the ability to introduce changes better suited for the integration. In the end, the decision was to fully support Capa legacy code. The sheer amount of systems this support enables Paffin entities access to was the deciding factor. WebGateway functionality, monitoring tools like the MulanViewer and MulanSniffer and other existing plugins provide a wealth of available functionality that outweighs the disadvantages. Furthermore, the limitation on changes was initially suspected and later, after the prototypes were finished, confirmed to be insignificant. Most mechanisms that should/had to be updated for the integration, like the internal message routing within the Paffin net, could transparently simulate the original Capa functionality.

Additionally, this means that the process-protocol interface includes the full agent protocol interface. Consequently, process-protocols can incorporate any agent functionality directly without being contained in Agent-Activities. This, in turn, means that agent actions can be defined outside of Agent-Activities. While this represents a slight break with the concept of Agent-Activities, it does not limit or interfere with it either. It just adds an additional way of defining agent actions in the Paffin-System. It was decided to neither restrict this option, which would also restrict the ability to execute Capa legacy code, nor endorse it. Agent-Activities should be used to describe and define the behaviour of Paffin entities, but normal agent behaviour is tolerated.

Retaining Capa *components:* Another important design decision regarded how to approach the internal components of the Capa agent, namely the protocol factory and the knowledge base. The question revolved around whether to reimplement their functionality elsewhere or keep them as-is until adaptions were necessary. Since Paffin entities needed both a knowledge base and (process-)protocol factory, the latter was decided. In the end, adaptions to the protocol factory turned out to be unnecessary, while the knowledge base only needed to add an additional knowledge access related to proactive Agent-Activity triggering (see Section 10.2.7).

Interface to platform and management WFMS: An important design decision revolved around the interface between the Paffin entity and the workflow management on the platform and management level. Agent management was taken care of via the standard Capa interface, with agent messages between the Paffin entity and the AMS and DF agents (now also implemented as Paffins) on the platform and management (see Section 10.2.13). Workflow management, however, needed to be implemented from scratch. Options included providing workflow management via reference nets with an interface of synchronous channels, as Java objects with a standard Java method interface or encapsulated as an additional component of the platform and management level. The latter option was ultimately chosen. Workflow management is implemented in a special Paffin entity on the platform and management level. The main justification was to provide a uniform way of realising interactions and management components. Since the AMS and DF are already Paffin entities, the platform and management

WFMS should also be realised as a PAFFIN entity. More details about the platform and management WFMS are discussed in Section 10.2.14. The consequence for the PAFFIN net and its interface are that agent messages are used to communicate with the platform and management WFMS. These administrative messages are handled completely and transparently by the management facilities within the PAFFIN entity, ensuring the conceptual clarity of workflow operations.

Implementation The implementation of this prototype encompasses only the PAFFIN net shown in Figure 10.11. That net is located in the `PaffinEntity` plugin. Instances of that net represent PAFFIN entities in the PAFFIN-System. Clearly, the basic MULAN and CAPA structure of the net is still recognisable. Main additions and adaptions are marked in light blue and mostly relate to the technical backend.

The following summarises the changes between the PAFFIN net in Figure 10.11 and the MULAN and CAPA nets discussed in Section 2.2.3.

- In addition to the knowledge base and protocol factory, the main backend control net (discussed in Section 10.2.6) is also instantiated during the initialisation of the PAFFIN entity.
- The technical backend control net possesses access interfaces (as transitions) to the knowledge base, the active process-protocols and the decision components.
- The message reception and routing is enhanced. Instead of just checking for an existing conversation ID in the *in-reply-to* field of the agent message, the entire message is checked for certain keywords indicating an existing target. If none of these are found the target is null and a the message meant to initialise a new process-protocol. If one of the keywords is found the message is handed to the backend, which internally routes it to a process-protocol, an AGENT-ACTIVITY or a workflow activity (as an administrative message). Finally, a message is discarded if the corresponding target is no longer available.
- As a management component the backend can create messages on its own and send them out. This is realised as a shortcut to the outgoing message interface of the PAFFIN entity net.

Conclusion The net shown in Figure 10.2.5 encapsulates and represents a PAFFIN entity in the PAFFIN-System. This net serves as the container for the management functionality contained within its components (backend, knowledge base, protocol factory), as well as the application functionality contained in its process-protocols, AGENT-ACTIVITIES and DCs. Consequently, the purpose of providing a construct to represent and execute a PAFFIN entity at runtime has been completely fulfilled. The interfaces between the components defined in this net provide all necessary access for the execution of all application and administrative processes. Further discussions relate to the internal components or the external role of the PAFFIN entity on the platform and management level. These are continued in the appropriate prototypes.

10.2.6 Paffin Backend Control

Purpose and Functionality The technical backend (see Definition C.9) represents the management facilities inside a PAFFIN entity, which are responsible for enabling the execution of all agent and workflow aspects of AGENT-ACTIVITIES. Those management

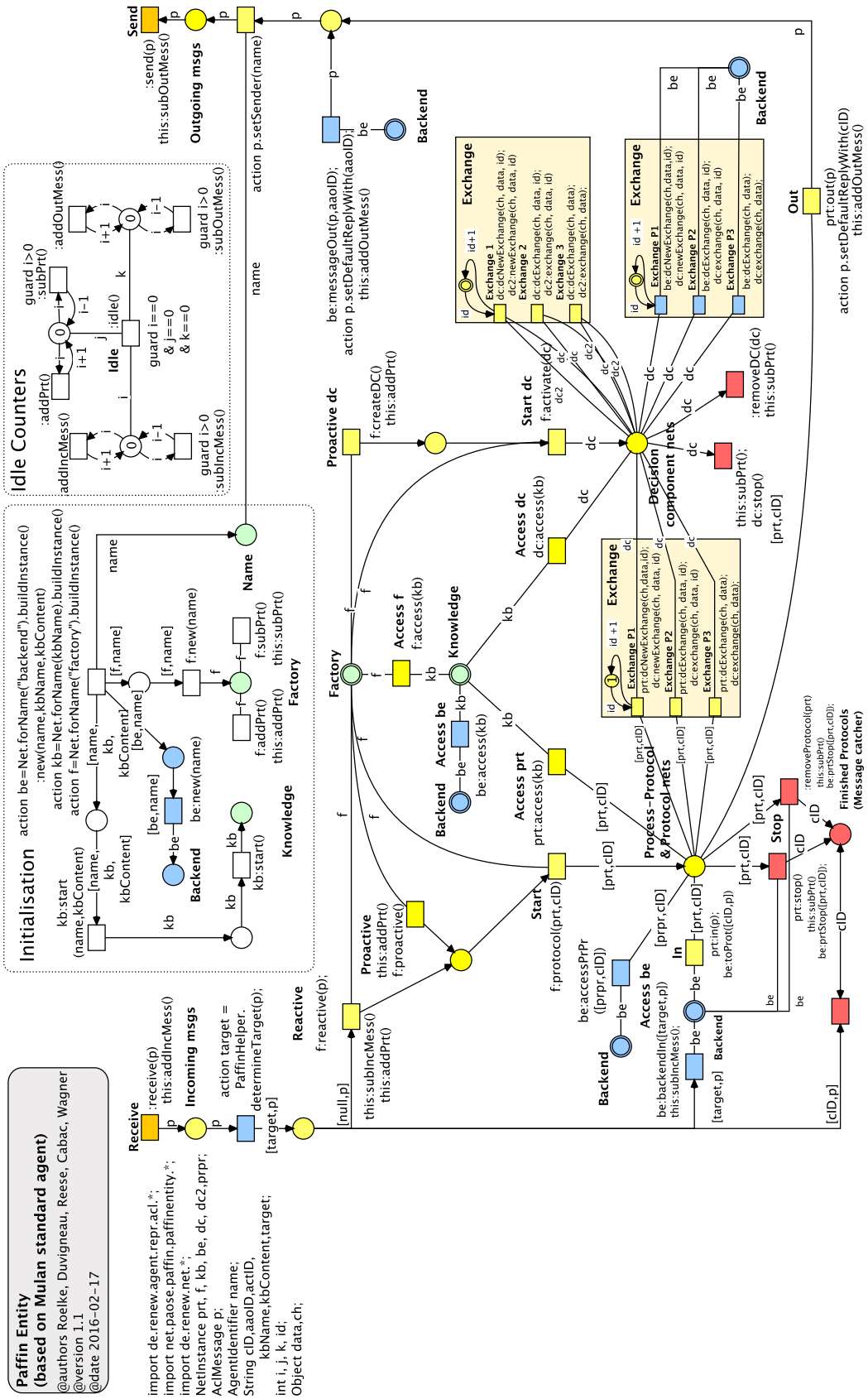

Figure 10.11: Paffin net

facilities can be grouped into four parts: Administration, agent, workflow engine and workflow resource. This current prototype deals with the administration. Its purpose is to enable the control and management of Agent-Activities. It is the net instantiated during the initialisation of the Paffin entity net (see previous prototype) and also serves as the container and interface to the outside for further backend nets implementing the remaining management functions. Consequently, the management of those nets is also part of the required functionality.

Design As the backend control net is the main administrative part of the backend management facilities, its functionality doesn't deal directly with agent actions and workflow operations, but rather with providing the supporting basis for them. Consequently, there are a number of specific functions that need to be provided by this part of the technical backend:

- It routes incoming messages to the correct Agent-Activity, process-protocol or workflow activity.
- It routes outgoing messages to the Paffin net, which then actually sends out these messages.
- It manages Agent-Activities, including triggering, finishing, aborting and updating them. In other words, it controls the life-cycle defined through the Agent-Activity net structure (see Section 10.2.2).
- It serves as a container for the remaining management facilities and provides them with an interface to the rest of the Paffin entity. This last functionality stems from the fact, that the backend should represent itself as a uniform component. That way the functionality is encapsulated better, which in turn make it easier to administrate and adapt it in the future.

The following design decisions were made during the development of the backend control prototype:

Communication control: During the development of the prototypes it became clear that message routing would be more complex than in the standard Capa agent. Before, there were two options. Either a message was intended for an already running protocol or a message would initiate a new protocol. These two options are still valid in the Paffin-System. However, there are three more options. A message can trigger an Agent-Activity reactively, a message can be intended for an Agent-Activity inside a process-protocol (instead of for reception directly in the process-protocol) and it can be an administrative message concerning a workflow activity. All of the five options need to be distinguished and handled differently. As the backend control has an administrative function in the Paffin entity it was decided to move the routing here. The backend control handles all routing, except for the initiation of a new process-protocol. That case is intercepted in the Paffin net and routed directly to the process-protocol factory.

Splitting up the backend functionality: Originally, there was only one backend net that included the entirety of the management facilities. However, it simply got too big to be technically and practically manageable. In fact, Renew, running on a high-end computer, was unable to simulate it and display the simulation at the same time. Consequently, the backend functionality was divided onto four nets. The backend control net serves as the container and controller for the three further, subordinate nets, which are discussed in the following prototypes.

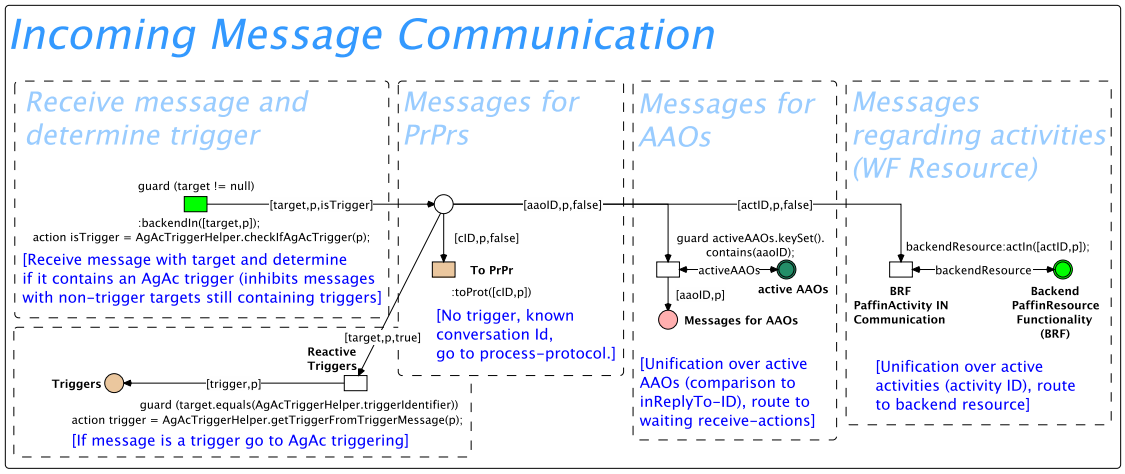

Figure 10.12: Backend control: Incoming messages routing

Implementation The implementation of this prototype consists of one reference net, supported by auxiliary Java helper classes, all located in the `PaffinEntity` plugin. The backend control net is the net that is instantiated by the PAFFIN entity net during its initialisation and ultimately put onto the **Backend** place in the PAFFIN entity net in Figure 10.11. Even though the backend control net is only one net, that net is too big to be reasonably presented as a whole in this thesis. Rather, all functional parts of this net are discussed, but only the most important ones illustrated as figures that show excerpts of the overall backend control net. The functional parts are:

Initialisation and State/Data Access: This functional part initialises the backend control net by providing default or empty data objects used throughout the net. This includes maps and sets of AGENT-ACTIVITY management data.

Incoming Message Communication: As described above, messages for four different targets arrive at the backend control net and need to be routed to the correct net parts, subordinate nets or external nets. The functional part implementing that message routing is shown in Figure 10.12. Messages arrive from the PAFFIN net through the upper, right transition where they are directly checked for containing AGENT-ACTIVITY triggers. If a message contains such a trigger, the trigger is extracted from it and is moved to the lower, right place. Triggering from there is further discussed below. If a message contains no triggers it is either routed to a process-protocol, an AGENT-ACTIVITY or the resource backend net where it is processed for a workflow activity. This routing is realised by utilising the unification in reference nets. The transitions routing the messages are all connected to places[9] containing the known identifiers for each of the three options. They can only fire if the identifier provided by the message matches one of the identifiers in the storage place. Unification ensures that messages are routed correctly to their target.

Outgoing Message Communication: Outgoing message communication involves messages sent by the associated agent action in AGENT-ACTIVITIES or for management and administrative purposes. The implementation is fairly simple providing a place which can be accessed by other transitions in the net. Any functional part or subordinate

[9]Not all these places are visible in Figure 10.12, since they are in different target nets or net components.

net that needs to send a message puts that message into this place, from which it is then removed and transferred to the PAFFIN net to be technically sent out.

AGENT-ACTIVITY *Triggering:* Possibly the most essential functional part of the backend control net is the triggering of AGENT-ACTIVITIES, shown in Figure 10.13. The ready transition on the left contains a token once initialisation is complete, to ensure that no triggering can be compromised by incomplete data. Once a token is available in that place the three concurrent transitions are ready to fire. They realise (from top to bottom) reactive, direct and proactive triggering.

All of the three concurrent transitions work similarly. They possess the downlink for the synchronous channel connecting to the AGENT-ACTIVITY net structure in the AGENT-ACTIVITY-TRANSITION. The first parameter of the downlink, `AgAcInfo`, is a tuple object containing all of the parameters inscribed to the AGENT-ACTIVITY. The second parameter defines the type of triggering, while the third one represents a unique ID that is created for this call and used to correctly match an answer later. Differences between the triggering types relate to the additional preconditions each type requires. Direct triggering requires no additional data. As soon as the preconditions of the AGENT-ACTIVITY are satisfied it can fire. Reactive triggering requires a corresponding incoming trigger message while proactive triggering requires a specific knowledge base entry. Reactive and proactive triggering are described in more detail in the next section.

After one of the three triggering transitions is fired the remaining processing is equal for all triggering modes. First, the base AAO net, shown in Figure 10.6, is created. Then all the AGENT-ACTIVITY parameters, as well as a unique identifier for the AGENT-ACTIVITY, are provided to that net to initialise it. The parameters also contain the identifier for the internal process, which can be instantiated with that identifier. Additionally, information about the AGENT-ACTIVITY is stored here for administrative purposes. Finally, once the internal AAO initialisation is completed, the AGENT-ACTIVITY is deployed, via downlink, back to the AGENT-ACTIVITY net structure in the AGENT-ACTIVITY-TRANSITION.

AGENT-ACTIVITY *Termination:* When an AGENT-ACTIVITY is successfully finished or unsuccessfully aborted the administrative data of it must be cleaned up in order to maintain correct and efficient system behaviour. All entries relating to an AGENT-ACTIVITY must be removed from all maps and sets used throughout the different backend nets. The output of the AGENT-ACTIVITY termination must be accessed and returned to the AGENT-ACTIVITY net structure in the AGENT-ACTIVITY-TRANSITION. Additionally, all of this has to happen in one atomic firing step to ensure no information is changed between data access steps. Consequently, this functional part of the backend control net consists of two separate transitions with large and complex inscriptions realising the finishing and aborting of AGENT-ACTIVITIES.

Clean Up Finished Process-protocol: Due to the distributed nature of the PAFFIN-System it is possible that a process-protocol can be stopped, even when not all of its workflow activities and AGENT-ACTIVITIES are finished. This can happen if, for example, a process-protocol is started by a PAFFIN entity in the context of a workflow activity for which the PAFFIN entity itself is the resource. If the engine of that activity directly confirms or cancels that activity, the resource PAFFIN entity needs to abort the entire process-protocol. The corresponding functional part in the backend control net manages this clearance by instructing the backend engine net to cancel all activities for all AGENT-ACTIVITIES of the prematurely ended process-protocol.

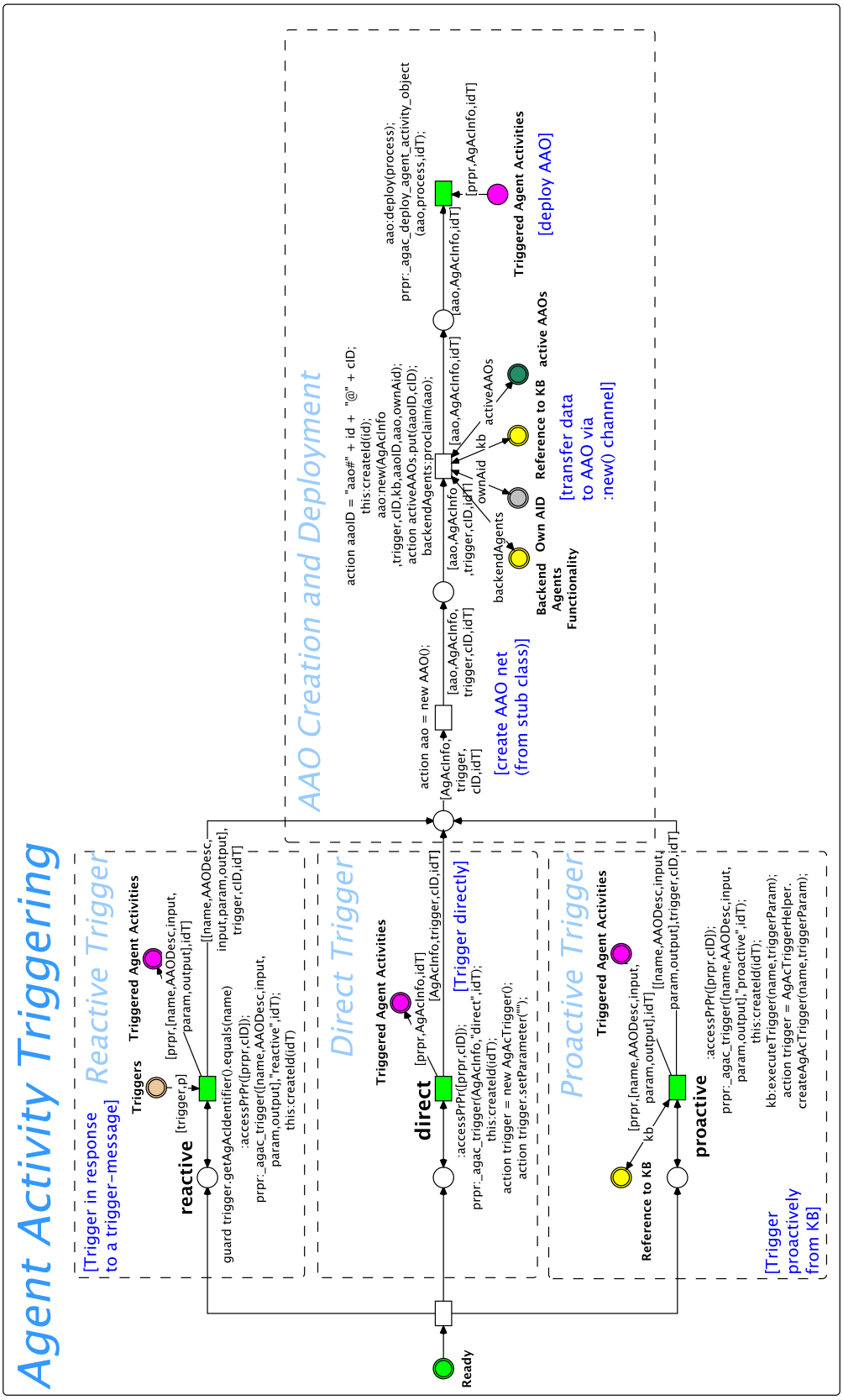

Figure 10.13: Backend control: AGENT-ACTIVITY triggering

Update AAO: Updating an Agent-Activity by removing the AAO from the central place of the Agent-Activity net structure and making changes to it before putting it back, is a desirable feature in the Paffin-System. For the proof-of-concept prototypes the simplest variant of this update mechanism was implemented, the reset of the AAO. The AAO is removed from the Agent-Activity net structure, all its data read and then a new AAO instantiated with the data and put back. Further update mechanisms are planned as future work.

Backend Subordinate Net Instances: The remaining functionality of the management facilities in the Paffin entity is not contained in the backend control net. Rather, it is implemented in three additional backend nets, *backend engine*, *backend resource* and *backend agents*. Ultimately, the backend control net controls these nets. It instantiates them and provides access for and to other components of the Paffin entity. Figure 10.14, for example, shows the functional part for the *backend agents* net.

The upper part of that net structure merely instantiates the net and puts it onto a specific place. The lower part represents the interface. To the left, the interface to the incoming and outgoing communication facilities are realised. The upper transitions realise the interface to the process-protocols, while the right transitions realise the interface to the DCs of the Paffin entity. Through the interface established by these transitions the *backend agents* net is capable of accessing all components it requires for its execution. The *backend resource* and *backend engine* functional parts have analogous net structures.

In terms of Java code, this prototype utilises auxiliary Java helper classes, which are shared between the different backend nets. The classes contain methods used in the backend control net to, e.g., determine if an incoming message contains an Agent-Activity trigger or determine sets of workflow activities for one Agent-Activity or for all Agent-Activities of one process-protocol. Additionally, the classes contain constants for synchronous channel names that allow using the standard reference net syntax check to find false inscriptions more easily.

Conclusion The backend control net serves a vital function in the Paffin entity. It is directly responsible for the administrative management of Agent-Activities and also controls the subordinate nets for the remaining management facilities. The implemented net features all of the required functionality and consequently fulfils the purpose of this prototype.

One aspect of the backend control that has direct future work implications is the update of Agent-Activities. Currently, only resetting an Agent-Activity is possible validating that the update mechanism generally works as intended. However, resetting an Agent-Activity is a rather simple use case with limited applications. Examples of possible extensions to this mechanism are:

- Exchanging the internal process of the Agent-Activity. An application example would be changing a payment type from credit card to direct debit. The internal process would be reset and then reinstantiated with the new payment type requiring a slightly different process.
- Adaptively changing the internal process. An example here would be adding additional operations or actions or removing ones the current situation doesn't require. This update mechanism would require an implementation of adaptive reference nets in some way, which is currently not available.

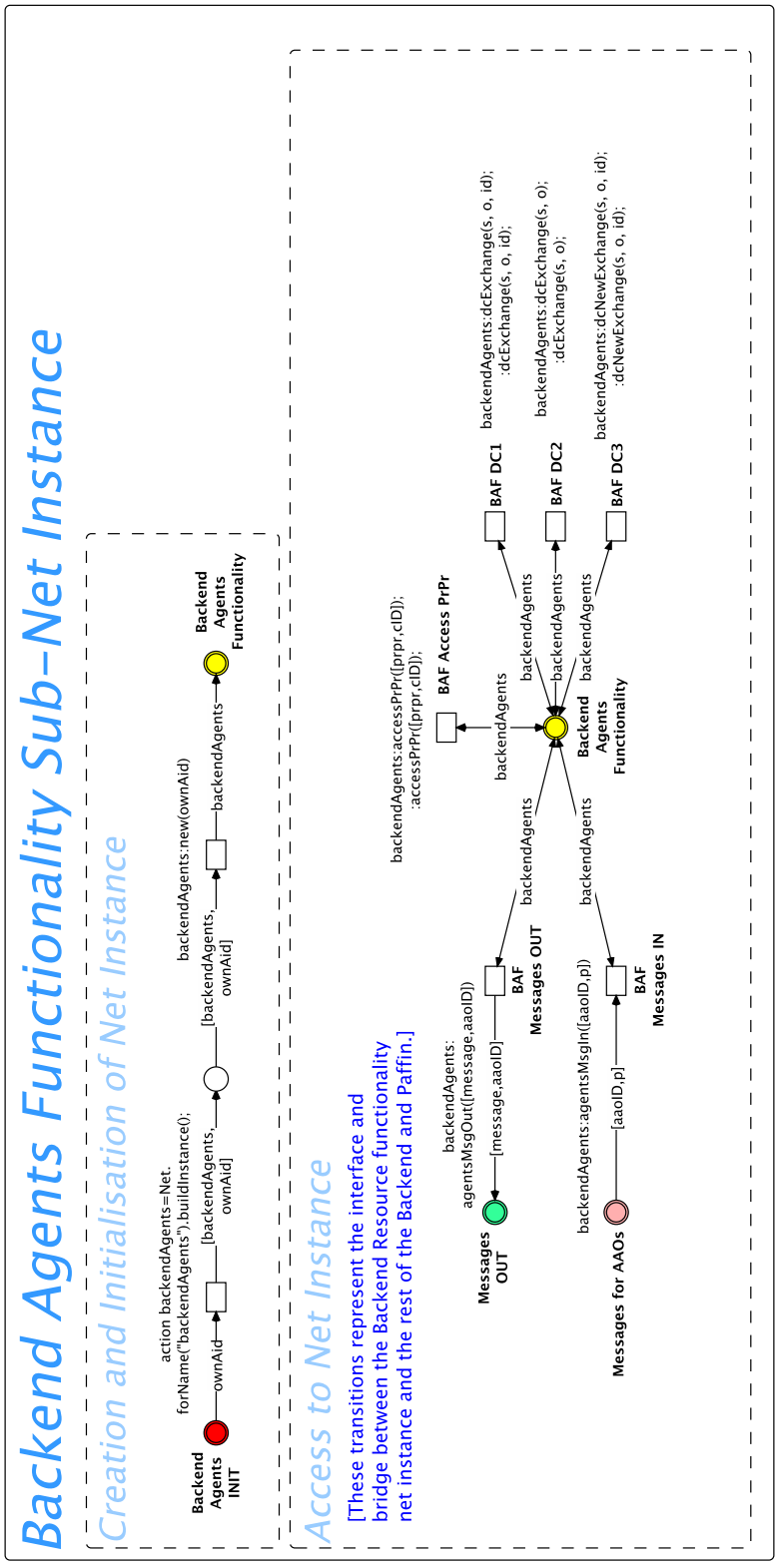

Figure 10.14: Backend control: Instantiation and interface for backend agents net

– Changing Agent-Activity parameters. This might become necessary if an Agent-Activity runs for a longer period of time and the circumstances change insofar that the process can continue, but requires up-to-date data throughout its execution. This kind of update can be simulated by the internal process of an Agent-Activity by interspersing it with repeating update constructs. The difference to the envisioned update mechanism is that that mechanism would update whenever new data was available, instead of at fixed times (push versus pull mechanic).

The topic of future work on and extensions to this prototype is picked up again in the discussion sections later in the thesis.

10.2.7 Reactive and Proactive Triggering of Agent-Activities

Purpose and Functionality Originally, the Paffin-System was developed with only the possibility to trigger Agent-Activities directly. Proactive and reactive triggering were planned, but only implemented in this current prototype in the late stages of development. The functionality of this prototype revolves around providing the means to delay triggering of Agent-Activities until certain conditions are fulfilled. Reactive triggering delays the triggering of an Agent-Activity until a trigger message from outside the Paffin entity has arrived. Proactive triggering delays the triggering until the Paffin entity itself decides to trigger the Agent-Activity. Both of these triggering modes improve the flexibility of the Agent-Activities by providing additional means with which to control when the triggering occurs.

Design Basically, the functionality to support reactive and proactive triggering was implemented as an extension to the existing mechanisms for direct triggering (see Section 10.2.6). In fact, only the first step of firing needed to be extended with two additional transitions with additional preconditions. These preconditions describe how the triggering delay is implemented. Reactive triggering requires a trigger message to have arrived, while proactive triggering requires an internal (knowledge) decision by the Paffin entity itself.

The following major design decisions were made during the development of this prototype:

Triggers arriving before Agent-Activity *is ready:* One design decisions dealt with reactive triggers arriving before their Agent-Activity is otherwise ready to trigger. This can happen if the process-protocol is not finished with another Agent-Activity, while the external trigger has already been sent. The two options were to either store incoming trigger messages for later use or discard them. Storing them can be exploited by spamming trigger messages to circumvent the reactive delay. Discarding them might lead to correct triggers being discarded because the distributed, asynchronous nature of the Paffin-System allowed them to arrive too early. Clearly, the ability to exploit the mechanism is less severe, since modellers can just use the direct triggering to circumvent reactive delay anyway. Loosing correct triggers could potentially deadlock the system. Consequently, the decision was made to store all triggers until they were needed. This in turn means that a reactive Agent-Activity might fire directly and without delay because the trigger was already present.

Relation between reactive triggers and agent actions: Reactive triggering means that an agent message arrives that contains a trigger object that causes the Paffin entity to trigger one of its Agent-Activities. An open question regarded the relation between reactive triggers and the seemingly equivalent agent action of receiving a message. While triggers and received messages do appear equivalent, their role in the life-cycle

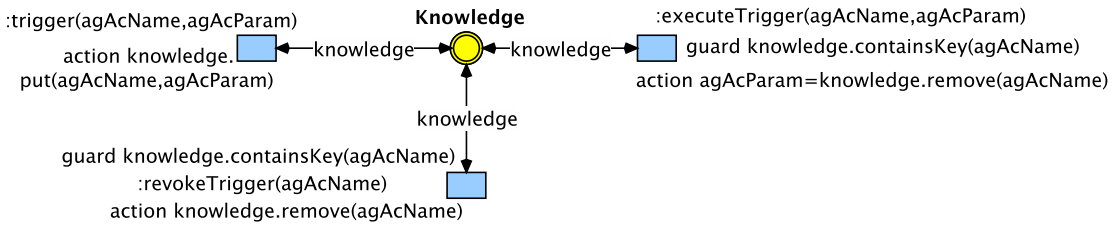

Figure 10.15: Paffin knowledge base: Triggering extension

of Agent-Activities distinguishes them. Receiving a message is part of the internal process of the Agent-Activity, which it executes after being deployed and before finishing or abortion. The reactive trigger arrives (from the perspective of the Agent-Activity) exactly when the Agent-Activity triggers. The trigger message is an administrative and management mechanism that is used for the Agent-Activity. It is consequently not part of the internal process and, from a modelling point of view, not an agent action. Therefore, the reception of the trigger message should not be explicitly modelled in the internal process. Rather, the trigger message should be stored as a parameter of the Agent-Activity that can be accessed via internal agent action.

Realisation of the proactive decision within the Paffin*:* Another decision relates to how the Paffin entity can make decisions that trigger Agent-Activities proactively. In general, decision making shouldn't be limited in any way. Paffin entities should be able to make decisions in process-protocols, Agent-Activities and in DCs. Consequently, the component, in which the decision is proclaimed, needs to be a central, globally (within the Paffin) available and accessible component. That component is the knowledge base. Making a decision on proactive triggering therefore means that somewhere in the Paffin behaviour a specific entry into the knowledge base is made that can be interpreted by the backend as the proactive trigger.

Implementation The implementation of this prototype does not feature any new nets, but rather extends existing ones. Consequently, this prototype is also part of the `PaffinEntity` plugin. Basically, three additions had to be made for this prototype:

AgAcTrigger ontology concept: To implement the functionality of this prototype, the first technical requirement was an object to represent trigger messages. That object needed to be sent as an agent message between Paffin entities and then be automatically distinguished for message routing. By implementing the Agent-Activity trigger as an ontology object called `AgAcTrigger` inheriting from the Capa agent action concept, all of these requirements are directly fulfilled. The trigger can be directly sent and can be recognised by any Paffin entity. It can also serve as the container for any kind of Java object as parameter. To match trigger and Agent-Activity the object also contains the Agent-Activity identifier (its name) as a field. The concept definition and its relation can be seen in the overall ontology in Figures 10.2 and 10.3.

Addition to the KB: In order to proactively trigger an Agent-Activity the Paffin has to be able to proclaim that trigger somewhere in its knowledge base. For that reason, the standard Capa knowledge base was extended with the interfaces shown in Figure 10.15. The left transition allows any behaviour of the Paffin to trigger an Agent-Activity identified by *agAcName* with an arbitrary Java object *agAcParam* as parameter. The

right transition then executes the trigger (see below). To round out the functionality, a trigger once given may be revoked as well. This might be necessary if the proactive trigger models behaviour valid only in certain circumstances. If the circumstances change, the PAFFIN may wish to revoke the trigger, which is implemented here.

Realisation in backend control net: The former two parts of the implementation provided the basis for reactive and proactive triggering. Going back to Figure 10.13, it is now possible to discuss how reactive and proactive triggering actually occurs.

Reactive triggering is implemented in the upper of the three concurrent transitions. That transition has an additional precondition with the *Triggers* place. The *Triggers* place contains all the received, and not yet used, AgAcTrigger objects the current PAFFIN has received. Reactive triggering can only occur if the *Triggers* place contains an AgAcTrigger object containing the name of the current AGENT-ACTIVITY in one of its fields. If that precondition is fulfilled, the triggering occurs with the trigger being removed from the Triggers place and stored as an additional parameter for the AGENT-ACTIVITY.

Proactive triggering is implemented in the lower one of the three concurrent transitions. That transition is connected to the knowledge base. It can only fire when the knowledge base transition to execute a trigger with the name of the current AGENT-ACTIVITY can fire. That firing is disabled by the right transition in Figure 10.15 through the inscribed guard until a corresponding element exists. During the firing of the triggering transition a new AgAcTrigger object is created with the parameter from the knowledge base. That way, the triggering parameter is accessible in a uniform way, regardless of the mode of triggering[10].

From a Java point of view this prototype utilises only auxiliary helper classes. These classes provide methods to check for triggers in incoming messages and to easily create AgAcTrigger objects.

Discussion For AGENT-ACTIVITIES, reactive and proactive triggering improve versatility. By being able to delay the triggering until preconditions outside of the process-protocol are met, the concept can be used directly in more varied use cases. Reactive and proactive triggering may both be simulated in regular AGENT-ACTIVITIES by having another, preceding AGENT-ACTIVITY implement the trigger condition. This is, however, cumbersome and inaccurate modelling. It adds a second AGENT-ACTIVITY that has no real purpose but to delay the original AGENT-ACTIVITY. The direct incorporation of this technique allows modellers to utilise it more easily and mire naturally. The purpose of this prototype has been fulfilled by enabling the description and support of both new triggering modes.

Regarding future work, one possible extension relates to the way the proactive triggering is handled in the knowledge base. Currently, it is only possible to proclaim a trigger and then revoke it or use it. In some use cases it might be beneficial to modify a trigger with updated data. Currently, this can only be simulated by revoking the trigger and reproclaiming it with new data. To support this extension, the revocation should also be extended to return the original parameter object for adaption. Another possible extension relating to reactive and proactive triggering is a combination of the two. It is feasible to desire an AGENT-ACTIVITY that requires both a reactive and proactive trigger, indicating a collaborative decision from within and without the PAFFIN. Overall, none of these

[10] Direct triggering also creates an AgAcTrigger object, although it is completely empty. This is done to ensure the AAO net does not deadlock due to a missing token or a null reference.

extensions change the raw capabilities of the triggering modes, which is why their priority in future work is only secondary.

10.2.8 Paffin Backend Agents

Purpose and Functionality As discussed, the technical backend (see Definition C.9) provides the management facilities for executing Agent-Activities and is partitioned into four nets responsible for different areas. The backend agents prototype is one of those nets and supports the execution of agent actions. The functionality of this prototype has to support the execution of the three fundamental agent actions.

Design The coarse design sees this prototype as an interface between the agent actions and the technical facilities in the Paffin entities. Outgoing Messages sent from internal messages are received and forwarded to the Paffin entity net. Incoming messages are received and forwarded to the appropriate Agent-Activities. Synchronous calls from internal actions to DCs and vice versa are put straight through (in the same firing).

Due to the small extent and straightforwardness of the functionality of this prototype, no major design decisions need to be discussed. Related to this prototype, though, the handling of DC access for internal actions is a major design decision, which is discussed in its own prototype in Section 10.2.9.

Implementation Like all parts of the technical backend, the backend agents is implemented as a reference net. That net is instantiated and controlled by the backend control net (see Section 10.2.6). The backend agents net is the smallest backend net, which makes it possible to present it in its entirety in Figure 10.16. Reasons for the net being so small lie in the fact that the Paffin-System prototype builds on Capa, which already incorporates all of the agent functionality. Therefore, the backend agent net merely needs to act as a proxy interface between the agent actions in the Agent-Activities and the technical facilities in the Paffin net. It receives data from each side and forwards it correctly to the respective other one.

The upper part of the net shown in Figure 10.16 shows the initialisation and data access parts. This place contains a copy of the identifier for the Paffin entity. Additionally, when deploying Agent-Activities, a reference to each active AAO net is put in a place here. That reference is required for the DC access, which is discussed further below. When an Agent-Activity is finished or aborted, the reference to the AAO is automatically removed.

The lower part of the net implements the interface for the agent actions. The interface for sending messages starts from the left. Here, the backend agents net receives a message from an Agent-Activity, which is put in a place and then, with the next firing, removed to the backend control net where it uses the functional part for outgoing communication. Receiving messages starts from the right. A message is received via the incoming communications functional part of the backend control net, put onto the place **Messages for AAOs** and then forwarded to the correct Agent-Activity. For both of these mechanisms reference net unification is used to ensure that messages arrive at the correct, intended target.

Finally, the interface for DC access for internal actions merely unifies over the AAO net, so that all transitions in the hierarchy, from the internal action transition in the internal process, over the AAO net, this current net, the backend control net, the Paffin net and finally the target DC net, fire synchronously. Three transitions here are necessary to realise the three types of DC access defined in the Paffin net.

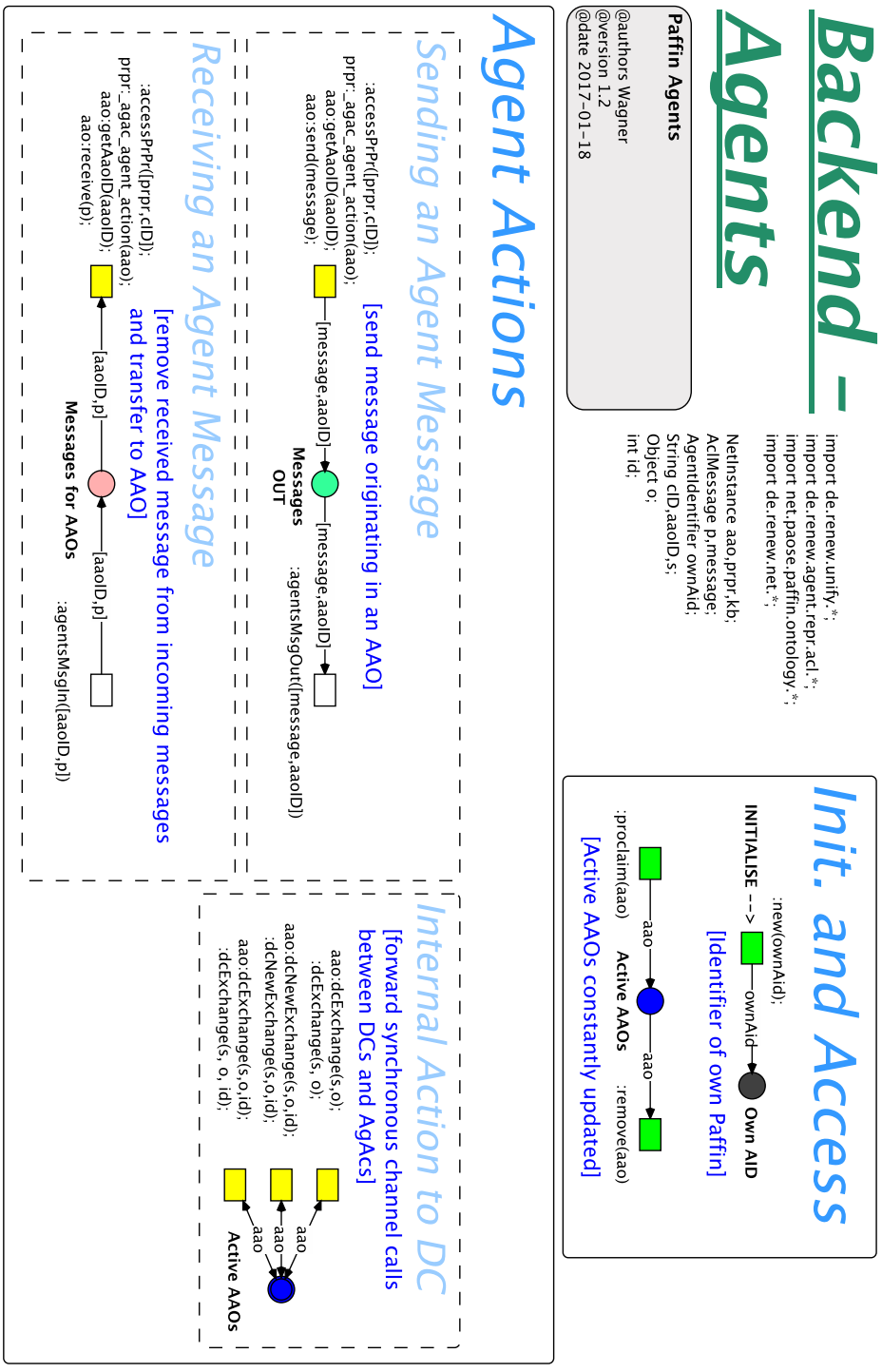

Figure 10.16: Backend agents net

Discussion The backend agents net provides a vital function within the PAFFIN entity, namely the interface between agent actions in internal processes and the technical PAFFIN entity itself. Without it, agent messages could neither be sent nor received and internal messages could not access decision components. Therefore it fulfils its purpose of enabling the execution of the three fundamental agent actions. It is a rather small prototype, due to the extensive CAPA base functionality. Still, besides its vital role it also serves as an extension point for any future upgrade to the agent functionality within the PAFFIN entity.

10.2.9 Paffin Decision Components

Purpose and Functionality Decision components (DCs) are an established mechanism in CAPA and PAOSE modelling (see Section 2.2.3). DCs model continuous, fully internal agent behaviour in CAPA agents. Incorporation of the DC mechanism into the PAFFIN-System is the purpose of this prototype.

Design Since DCs are directly available in the CAPA standard agent, they are also directly available without change from process-protocols (i.e. outside of AGENT-ACTIVITIES). However, due to the technical and conceptual abstraction introduced by the AGENT-ACTIVITY, they are not available to the internal processes of the AGENT-ACTIVITY without adaptions to the PAFFIN entity. This means that the scope of implementation of this prototype is not the reimplementation of the already existing DC mechanisms, but rather the adaption and enabling of the interfaces between internal processes and the DCs. That interface is included in the backend agents net, which was already discussed in the previous prototype.

The only major design decision of this prototype regards how DCs are considered in the PAFFIN-System. As discussed in Section 5.2, DCs can either be seen as fully internal components of an agent (or PAFFIN in this case), or they can be seen as an internal, but still autonomous agent (or PAFFIN in this case). In the former case all DC access would be considered as internal actions, while in the latter case an asynchronous interface using messages sent and received from the DCs would be considered.

Both options have their merits. Considering DCs as internal PAFFIN entities allows for a holonic view of PAFFINS of PAFFINS. This could be extended in future work into a full-fledged holonic system were the integration of agents and workflows can be utilised over different, explicit hierarchy levels. DCs as internal PAFFIN entities would require asynchronous communication though. While that is desirable in a distributed system, such as in the environment of the PAFFIN entity, it is less desirable in a quasi-centralised container PAFFIN. Here, synchronisation can be fully utilised as the availability of internal components is ensured. Synchronisation, incidentally, is the communication mechanism that would be required when considering DCs as purely internal components.

Another aspect to keep in mind relates to usability. Using internal actions to implement DC access means doing so in a uniform way of calling a specific channel and transmitting data that is accessible bidirectionally. Using send and receive message actions means that there are two ways in which to send messages only unidirectionally. This means that modellers have to decide more explicitly, carefully and with more foresight on the flow of the communication. Furthermore, distinguishing between sending messages to external PAFFIN entities or internal ones could become an issue. Requiring more modelling tools and net components would also increase the already substantial modelling complexity.

Overall, the arguments for treating the DC access as internal actions outweigh the arguments for treating it as asynchronous communication with internal, yet autonomous

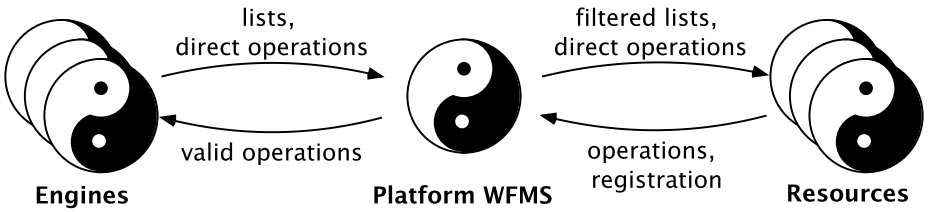

Figure 10.17: Basic workflow management architecture in the Paffin-System

partners. Synchronicity and ease of use offer concrete advantages, while the asynchronous option mostly only serves as preliminary groundwork for holonic Paffin entities.

Implementation The implementation of DC access was already discussed in the scope of the backend agents net in Section 10.2.8.

Discussion DCs are an established and proven tool for modelling continuous, internal behaviour of agents. Enabling this mechanism for the Paffin-System is desirable, as internal and continuous behaviour are useful in the same use cases as in pure agent-orientation. By implementing the DC mechanism, this prototype fulfils the purpose laid out for it.

The idea of implementing DC access in the form of asynchronous communication was rejected at this point of research into the integration of agents and workflows. Nonetheless, it represents an interesting first step into holonic ideas for the Paffin-System. Considering Paffins of Paffins or even Paffins of Paffin management systems would allow for the creation of highly complex hierarchic systems. While this is outside of the scope of this thesis, it is one possible extension to the integration.

10.2.10 Paffin Backend Engine

Purpose and Functionality The backend engine prototype is part of the technical backend (see Definition C.9) and is concerned with one of the four separate backend areas. Its purpose is to enable the support of the execution of workflow operations from the engine side. This means that the focus of this prototype lies in enabling a Paffin entity to act as a workflow (engine) and provide workitems and activities to other Paffin entities as resources. Therefore, the functionality of this prototype encompasses the support of the three basic workflow operations on the engine side. To enable that support, the backend engine has to gather and manage all the workflow execution-related data of its process-protocols and Agent-Activities, disseminate that data to the other workflow participants in the system and control the actual execution of the workflow operations.

Design This prototype concerns a Paffin entity's workflow engine role in the global workflow management of the Paffin-System. In order to understand that role, the overarching global workflow management architecture must first be specified. The basic model can be seen in Figure 10.17.

The basic workflow management architecture originates from the WorkBroker prototype, described in Section 10.3.2 and presented in [Wagner and Moldt, 2015b], and the corresponding principle (see Definition C.13). While that prototype was developed before the Paffin-System prototypes, and was thus not directly incorporated into the system, it

prepared the basis and general approach of workflow management between agents. That general approach is applied here for workflow management between Paffin entities.

Basically, all Paffin entities of a system may serve as both engines and resources. A Paffin as an engine is equivalent to the workflows it executes. Consequently, the Paffin as an engine is aware of and continuously keeps track of all workitems and activities and their states. Gathering them in lists, the data about workitems and activities is sent to the platform WFMS. The platform WFMS is a special type of Paffin that is automatically instantiated on each platform and management element (see Section 10.2.14). It has the same significance and status as the agent management components AMS and DF, which are also Paffin entities in the Paffin-System.

Paffin entities as resources register themselves at the platform WFMS. That way they declare that they are available for the execution of workflow tasks. The platform WFMS then filters the workitem and activity lists according to the permission sets for each resource and sends these lists to the resources. Lists are constantly updated.

Paffin entities as resources can use the lists to initiate workflow operations. When they receive a list with available workitems can request these workitems. These requests are not sent directly to the engine Paffin, but to back to the platform WFMS. The platform WFMS then checks the permissions and status of the workitem and only forwards valid requests. Validity can be compromised, e.g., if the resource does not have the correct permissions or if the workitem is no longer available. The valid requests are processed and the operations executed by the Paffin as an engine. Administrative data and updated workitem and activity lists are then disseminated to all workflow participants. Once a Paffin entity as a resource has successfully requested an activity it can initiate the confirmation or cancellation in analogous ways.

Finally, the Paffin as an engine may also initiate workflow operations. These direct operations have been discussed before in the context of the internal process of an Agent-Activity. Direct operations are performed on the authority and autonomy of the engine itself. However, the engine still has to inform the platform WFMS, which in turn informs the resource of the operation.

More details of the workflow management architecture of the Paffin-System are added in the descriptions of the respective prototypes. For now, the basic view of the other components provided here is sufficient. From the overall workflow management architecture the following concrete requirements of workflow engines and therefore the backend engine prototype are derived:

- The backend engine is aware of all currently available workitems, activities and their status.
- The backend engine is able to receive initiations of valid workflow operations and then perform these operations.
- The backend engine is able to execute direct workflow operations and then inform the platform WFMS of the direct operation.

Apart from the overall design decision on the management architecture of the Paffin-System, there were a number of design decisions regarding the backend engine prototype and its role in the architecture specifically.

Decoupling of engines and resource via the platform WFMS: From the description of the overall workflow management architecture of the Paffin-System it is directly clear that the platform WFMS possesses a centralised role between engines and resources. This design decision should be clarified. While it affects further prototypes (i.e. the

backend resource in Section 10.2.11 and the platform WFMS in Section 10.2.14), it is suitable to discuss here since the question is already valid.

There were two options for implementation: Directly connect engines and resources or use a global WFMS between to decouple them. The former prescribes a more complex, distributed approach. Paffins as engines and Paffins as resources would need to be aware of and keep track of the respective other group. Each engine would have to know all relevant resources, their permissions and what they were doing. Likewise, the resources would have to keep track of all engines that provide them with workitems and activities. The administrative and management overhead would be large. Paffin entities would be engaged in continuous update and processing behaviour relating to management and administrative data, in addition to their actual functionality. The benefit of this approach, though, would yield a fully distributed workflow management, where individual engine or resource failures would only affect the directly associated Paffin entities.

The other option represents a more traditional, simpler approach. Both Paffin engines and resources only have one contact point, the central WFMS. Engines do not need to be aware of resources and vice versa. This significantly reduces their operating effort for workflow management. However, if the central WFMS fails the connection between *all* engines and resources is severed, resulting in a total failure of workflow management.

Ultimately, the traditional WFMS option was chosen. The Paffin-System is intended to be a proof-of-concept for the integration approach. A simpler implementation of the workflow management is therefore viable. Furthermore, through the clear separation and encapsulation of the central platform WFMS, it is relatively easy to exchange it for a more distributed WFMS at a later time. Feasible solutions include the distribution of functionality onto multiple Paffin entities and even onto multiple platforms working in collaboration. That way the failure of one platform WFMS entity could be balanced out by other entities and platforms.

Additionally, the implementation of the platform WFMS as a Paffin entity allows for some interesting possible future extensions. The platform WFMS Paffin can be extended to forward its known workitems and activities to other platform WFMS Paffins on other platforms. In fact, it is possible to use this mechanism to create WFMS hierarchies with platform WFMS Paffins on higher levels disseminating their lists to all subordinate platform WFMS. This mechanism would implement a distributed workflow management which would partially alleviate the centralisation issues. Engines and resources could be designed to use other platform WFMS in the hierarchy in case their local platform WFMS fails.

In general, the choice of a quasi-central platform WFMS enables a number of extension options for future work while maintaining a rather simple approach for now. The decoupling between engines and resources and the clearly specified interface allow for exchanging and adapting the inner workings of the platform WFMS quite easily. For engines and resources the platform WFMS is a kind of black box that provides them with data for their own execution. How that black box works internally, if it is central, distributed or if it is part of a hierarchy, etc., is completely irrelevant to them as long as they receive the data they need.

States of workitems and activities: A design decision directly relating to the backend engine concerns how workitems and activities are tracked, or rather in which state they are tracked. For workitems this question is simple to answer. Once a workitem is

available, i.e. the transition to request it in the internal process of an Agent-Activity is activated, it needs to be tracked and data about it provided to the platform WFMS. If it is not yet available it does not yet need to be tracked and if it is successfully requested it no longer needs to be tracked.

For activities this question is slightly more difficult. After a workitem is successfully requested it becomes an activity, which means that a resource is actively executing it. However, since the internal process allows additional agent actions and workflow operations to be nested between a pair of coupled request and confirm operations the activity might not be confirmable directly after the request. Similarly, the composition of the internal process may inhibit or delay the cancellation of the activity at certain points. Consequently, activities must be tracked in different ways. On the one hand, the engine must track all of its active activities in order to correctly inform the platform WFMS and also be aware of what is currently being executed. On the other hand, it must keep track of the *confirmable* activities. An activity is confirmable only if, from the view of the internal process of an Agent-Activity, a confirm transition for that activity is activated. That information must also be sent to the platform WFMS and forwarded to the resource, since it needs to be aware if it can't confirm its activity yet. For a resource, an unconfirmable activity could indicate that it hasn't produced all the results yet or that there might be an error in the system. If a resource attempts to confirm an as-of-yet unconfirmable activity that confirmation fails and the activity remains active.

Regarding cancellation, if a resource wishes to cancel an activity it should always be able to do so. However, due to the free form of the internal process, modellers might choose not to allow cancellation at any point. Cancellation is explicitly modelled and should be available from any state, but this is not enforced. Initiating a cancellation therefore sets a flag that either cancels the activity immediately or as soon as the internal process reaches a state in which cancellation is possible. Consequently, cancellable activities do not need to be tracked.

Implementation Like all parts of the technical backend, the backend engine is implemented as a reference net as part of the `PaffinEntity` plugin. That net is instantiated and controlled by the backend control net (see Section 10.2.6). One thing of note is that the interface to the platform WFMS is provided by a standardised DC in the Paffin entity. The backend engine communicates with the engine DC, which in turn communicates via process-protocols with the platform WFMS. The engine DC is described along the platform WFMS in Section 10.2.14.

Before describing the backend engine net the Java parts of this prototype need to be explained to create an understanding of how the backend engine is capable of tracking available workitems and confirmable activities. Besides Java helper classes providing mostly convenience access to often reused functionality, the backend engine prototype incorporates two highly important classes. The `WorkitemRequestChannelWatcher` and `ActivityConfirmChannelWatcher` classes implement the functionality to recognise requestable workitems and confirmable activities. Both of these classes work analogously for their respective context, which is why they are discussed here together.

During the instantiation of the internal process net, the AAO net also creates an instance each of both ChannelWatcher classes. The functionality in that class connects the AAO net instance with the process net instance. Each instance contains a ChannelSupervisor object from the package `de.renew.watch`. In short, the ChannelSupervisor object monitors a specific net instance (the internal process) for a specific channel (either the request workitem

or confirm activity channel). The monitor checks for partial, possible unification bindings on that channel in that net instance. These are partial but for the specific parameter of the channel, meaning that the net instance needs to fulfil all preconditions of the firing except for that specific one, which is kept variable. In practice, this means that the binding could fire, as long as it receives the missing parameter as a concrete object. For the workflow operations here, this missing parameter is always the data object sent from the engine when the operation is actually executed. In other words, finding a binding indicates that, from the process perspective, the operation can be executed and is only missing the input from the management facilities. Since the management facilities can always provide the missing data object, a binding indicates to them that they can initiate the execution of the workflow operation.

Once a binding is calculated by the ChannelSupervisor the ChannelWatcher object is provided with the binding, which it processes and forwards to the AAO net. The connection between Java and net is realised as a so-called net stub. The stub defines Java methods for a net instance, which correspond to firing transitions with specific synchronous channels attached to them. That way the calculated bindings are transferred to the AAO net, which processes them further into lists of requestable workitems and confirmable activities. From the AAO net, these lists are reported to the backend engine net. This ultimately creates the ability for the backend engine to track workitems and activities.

As with the backend control net, the backend engine net is too big to fully present in this thesis. The following provides a description of all functional parts of the net, including the actual net structure for the most important functional parts.

Initialisation and Access: This functional part receives initialisation data, makes it available for access and also creates initial, empty or default tokens of lists and maps that are required for workflow operations. The latter are used to track workitems and activities.

Update Requestable Workitems: Whenever a workitem becomes available it is reported to this functional part of the backend engine net. The report itself is handled by the AAO net which recognises the new workitem and creates an updated list of its own workitems. That list has the form of a HashMap with the workitem ID as a key and the workitem object as content. In the backend engine net that HashMap is put into another HashMap which uses the Agent-Activity identifier as a key for the transferred HashMap of workitems. Whenever that HashMap of workitems per Agent-Activity is changed, an updated version is handed over to the engine DC, which handles the update to the platform WFMS.

Request Workitem: Figure 10.18 shows the part of the backend engine net responsible for requesting a workitem. In order to ensure correct operations, this is an exclusive loop, meaning that workitem requests are handled one at a time. Requests beyond the first are delayed but never discarded. The loop waits with the upper place and starts with a new request incoming at the left-most transition. In the first subsequent transition, the backend engine reviews if the workitem of the request is still available. Workitems may become unavailable between the time a request is made by the resource and it is processed by the engine. This can happen, e.g., if another request for that workitem was successful in the meantime or if the internal process removed precondition tokens with another agent action or workflow operation. If the workitem is no longer available, only the lower branch of the net structure can fire. Here, a failure object is created and then reported as the result of the operation. If the workitem is available, only the upper branch can continue. Here, in the extensive transition inscription, all data

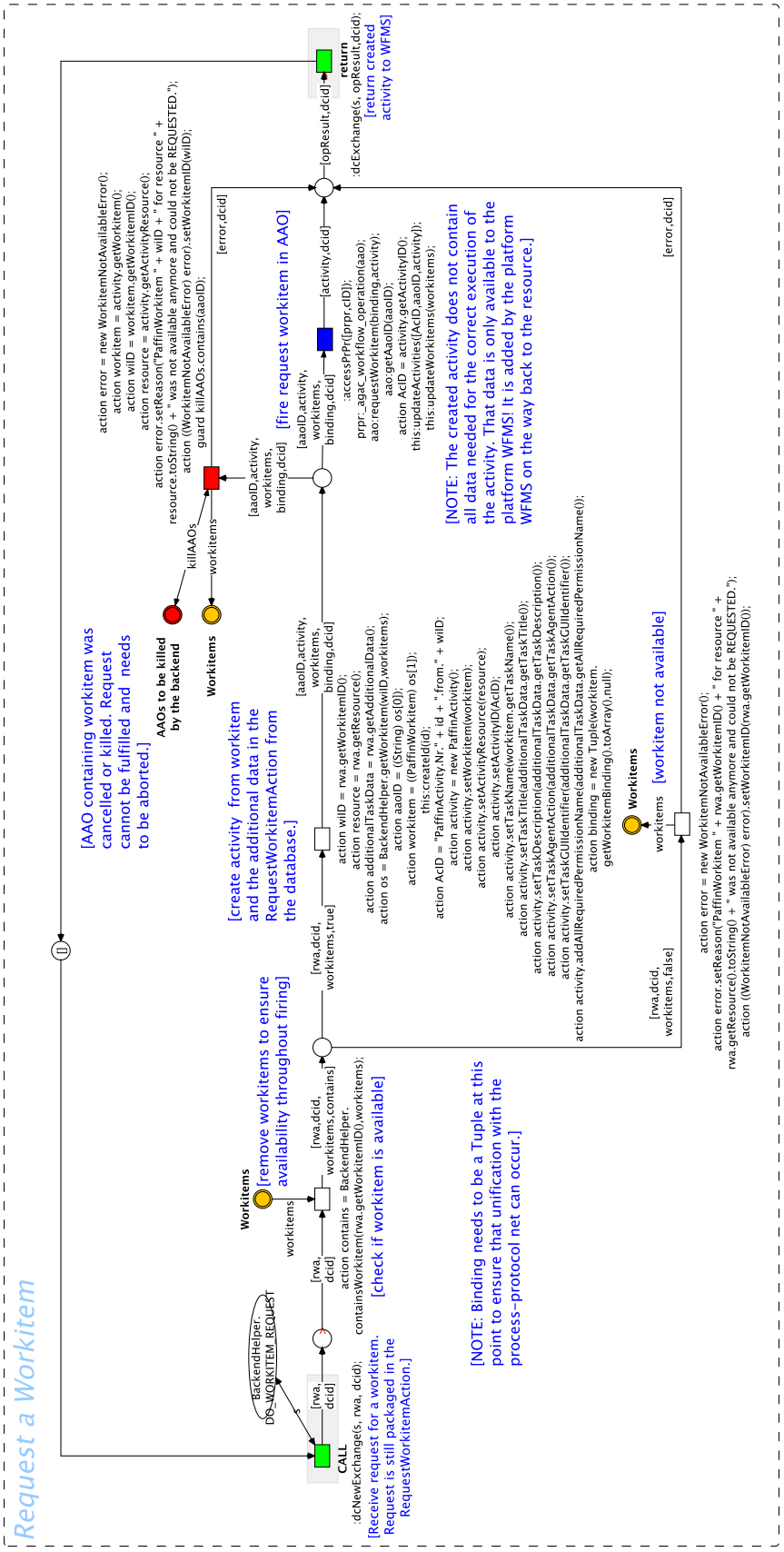

Figure 10.18: Backend engine: Request workitem

is read from the original request and workitem and a new activity created with the available data. This has to happen atomically to ensure consistency. Afterwards, the blue transition can fire, which synchronises up to the internal process. Only this firing in the internal process actually executes the workflow operation and finalises the workitem request. The activity object created during these steps is returned to the platform WFMS as the successful result of the request workitem operation.

The minor branch before the blue transition models the behaviour that is required when a process-protocol or Agent-Activity was terminated prematurely. In that case, workitem requests may be stuck at the point, because no unification can be found to satisfy all the conditions. To deal with such situations, the backend control net informs the backend engine net of the Agent-Activities that were prematurely terminated. Via unification of the Agent-Activity identifiers the red transition can fire, which skips the blue request step and reports back an error to the platform WFMS.

Update Valid Activities and Confirmable Activities: These two functional parts track the currently active/valid activities, as well as the confirmable activities. The basic functionality follows analogously to tracking workitems, with the AAO net recognising changes and informing the backend engine net. Particularities for activities include keeping two separate lists to ensure all updates are sent to the platform WFMS, even those which are fleeting, and an option to move control of specific activities from one Agent-Activity to another to realise workflow activities split between Agent-Activities.

Confirm Activity: The net structure implementing the confirmation of an activity is shown in Figure 10.19. It is an exclusive loop similar to the request workitem part, although more complex. It can start at either the left-most transition with a confirmation initiated by a resource or at the upper left blue transition modelling the direct confirmation initiated by the engine (in the internal process) itself. The handling for both options is almost the same, with minor differences in branching.

First, the engine checks if the activity is still valid. If it is not valid a failure message is created and returned as the result. If it is valid, confirmations initiated by a resource are checked for confirmability of the activity. Direct confirmations skip this check as a direct confirmation can only be initiated in the internal process if the activity is confirmable. If the activity is not confirmable another failure message is created and returned as the result. In the next step the confirmation is prepared by reading all the necessary data from the different data objects. This also checks if the resource initiating the confirmation is actually the one that is also recorded in the activity. If that is not the case another failure message is created and returned as the result. If all checks returned valid the activity can be confirmed by firing the blue transition which synchronises to the internal process of the Agent-Activity. Afterwards, the activity object is cloned with the result obtained for storage. This ensures that maps and lists containing that object are not corrupted (a technical inconvenience in Renew). Finally, the result is returned to the platform WFMS, either as an answer to the initiation by the resource as an inform in case of a direct confirmation.

As with the request workitem operation, confirm activity operations may get stuck if the process-protocol or Agent-Activity containing them is prematurely terminated. The same mechanism and similar shortcut in the net (minor branch above the blue confirmation transition) are used to handle these situations.

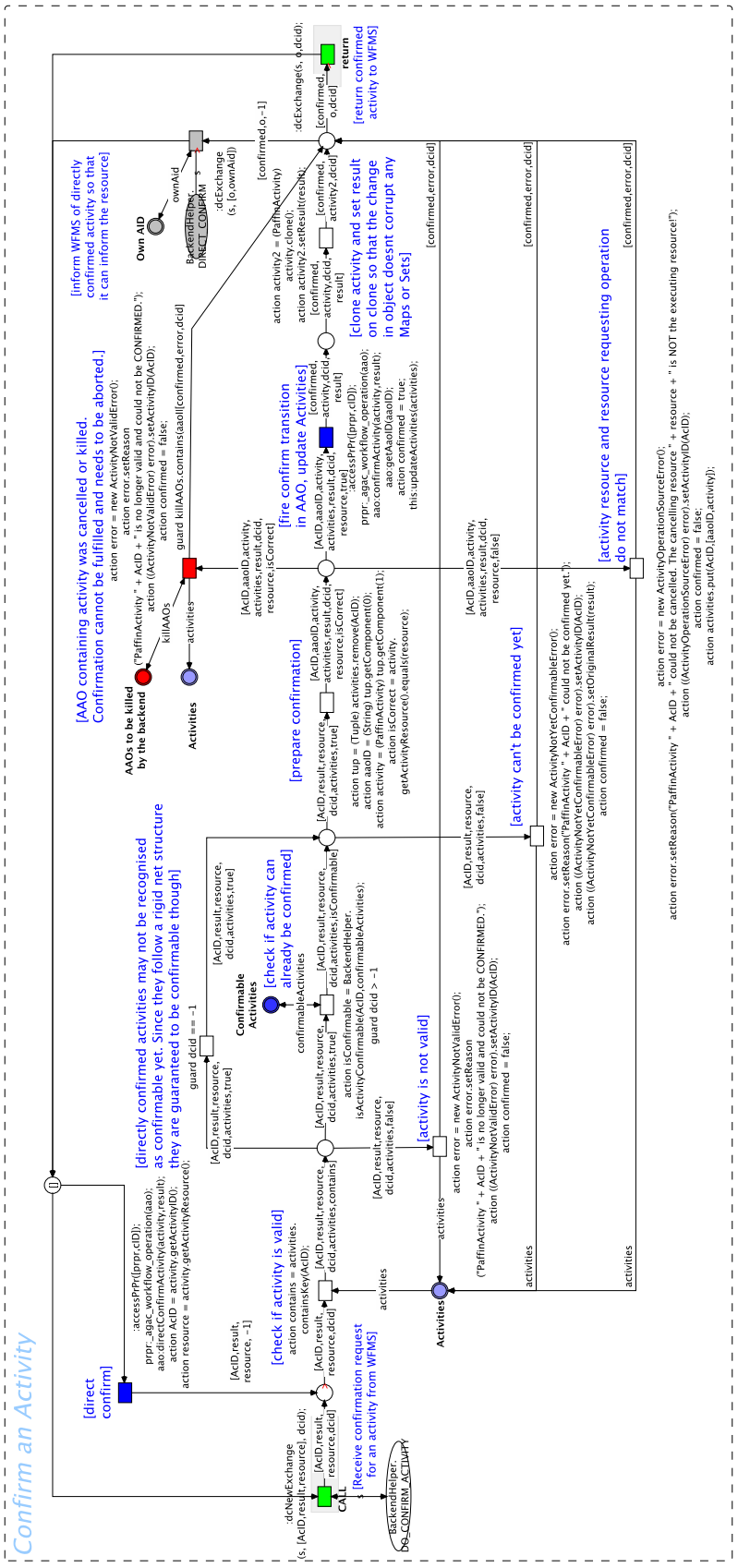

Figure 10.19: Backend engine: Confirm activity

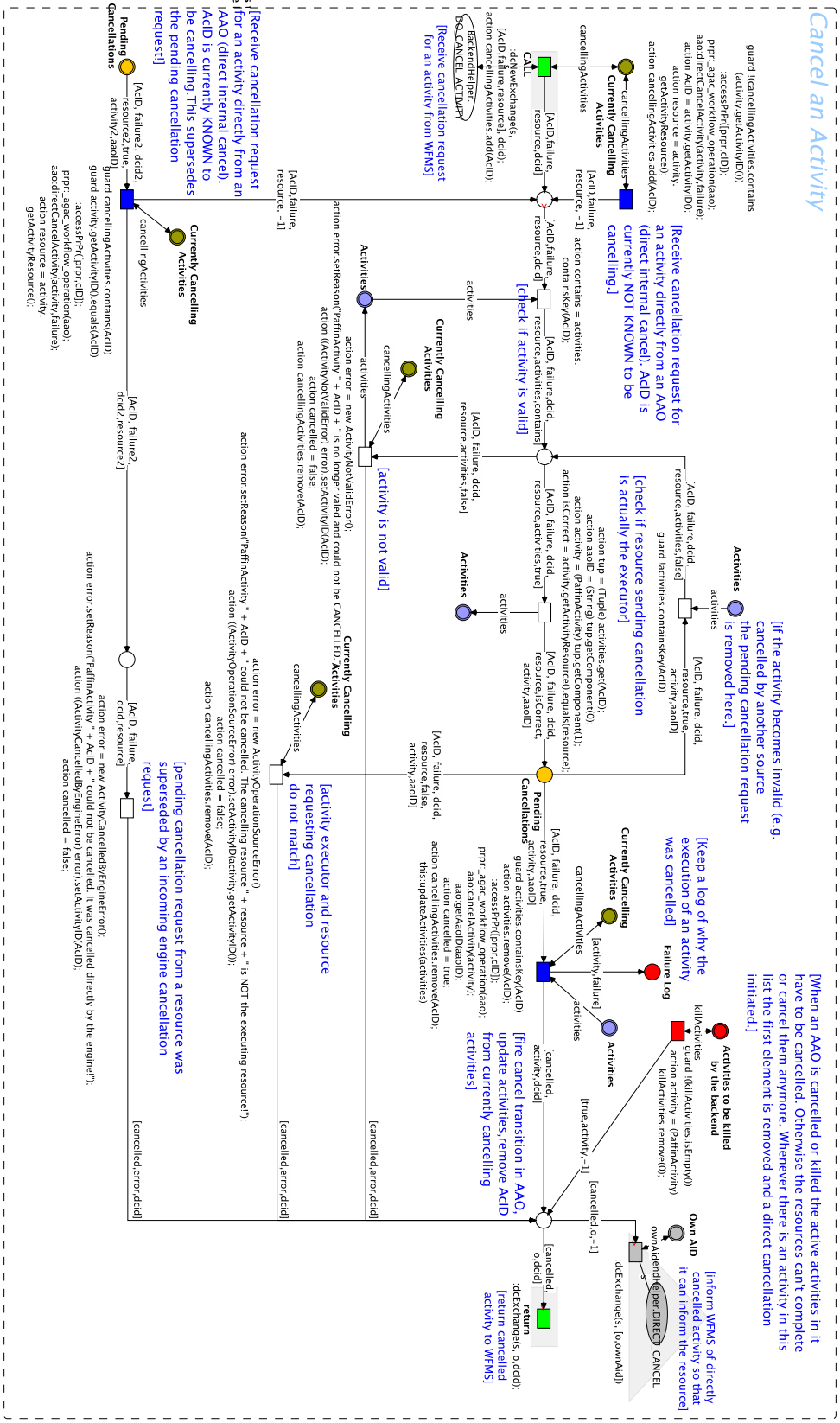

Figure 10.20: Backend engine: Cancel activity

Cancel Activity: The net structure implementing the cancellation of an activity is shown in Figure 10.20. It is markedly different from the functional parts for requesting a workitem or confirming an activity, because it allows for multiple cancellations to be processed at the same time and needs to keep track of additional side conditions.

Again, cancellations can originate from either a resource or directly from the engine itself. Both arrive at the upper left-most part of the net. As an exception, the lower left-most part models the behaviour when a cancellation arrives directly from the engine while a cancellation from a resource is already pending. In that case the engine cancellation supersedes the resource cancellation. The resource cancellation is aborted with a failure message and the engine cancellation processed from the start. The decision to have engine cancellations supersede resource cancellation was made to ensure that the PAFFIN entity as an autonomous workflow (engine) has full and ultimate control.

Processing of a cancellation request starts by checking activity validity and the origin of the cancellation. If the activity is valid and the cancellation was initiated by either the engine directly or by the correct resource the cancellation becomes pending. This means that a corresponding cancellation net component in the internal process of the AGENT-ACTIVITY can fire as soon as it is activated. In that case the blue transition fires which synchronises to the internal process and actually cancels the activity restoring the local state in that net before the original workitem request. Afterwards, the result of the cancellation is sent to the platform WFMS.

Of course, since the cancellations might be pending for a period of time the activities might become invalid at any other time. In that case the backwards loop from the pending cancellations place fires, which in turn causes the failure route to become activated.

For cancellations, cleaning up after prematurely terminated process-protocols and AGENT-ACTIVITIES is markedly simpler. Pending cancellations are removed as those activities become invalid. Cancellations that occurred successfully, but were not sent to the platform WFMS yet only need to be unified via the AGENT-ACTIVITY identifier and removed from the net.

Conclusion The workflow engine support for workflow operations is vital for the AGENT-ACTIVITY concept. This prototype fully supports the prescribed engine functionality, therefore fulfilling its purpose wholly. It is aware of all necessary states of workitems and activities, disseminates them to the platform WFMS and is able to perform and control the actual execution of all three basic workflow operations, initiated by either resource or engine. Since the engine support of workflow operations represents a basic prerequisite of the AGENT-ACTIVITY concept there is little to discuss regarding future work. Extensions of functionality are not applicable here, since the operations are already fully supported by the current prototype. The only feasible future work concerns improvements to efficiency. For example, adapting the mechanisms of this prototype to handle concurrent workitem requests or activity confirmations is a reasonable goal. However, it is not a focus of immediate future work.

10.2.11 Paffin Backend Resource

Purpose and Functionality The backend resource prototype is part of the technical backend (see Definition C.9). The backend resource functionality contains the support of

the three basic workflow operations from the resource side. Its purpose is to provide the facilities for a PAFFIN to receive and process data about tasks, workitems and activities and also initiate desired workflow operations.

Design The backend resource's role in the overall workflow management architecture is to represent a workflow resource as a management entity. Basically, the backend resource needs to be able to receive data about workitems and activities from the platform WFMS, process and present that data to the resource it represents and initiate or process workflow operations on behalf of the resource. Registration and deregistration of the resource are also handled by the backend resource.

One of the challenges of this prototype is the diversity of resources a PAFFIN might represent. A resource can be anything that can perform the work associated with a workflow task. Often, human users act as workflow resources, but automated machines and devices (e.g. printers) are also possible. In the PAFFIN-System a PAFFIN itself may also be a resource which executes automated functionality within the context of a workflow task. The backend resource has to distinguish between the different types of resources, because task and related data have to be provided to and processed from them differently.

Human users need to be provided with workitem and activity data, as well as a (graphical) user interface. Their decisions happen completely outside of the system while their work is usually supported by it and results entered into the system. Automated machines and devices, like printers, usually don't have complex decision making mechanisms. Workitems are more likely to be simply assigned to them. Their behaviour is fully automated and happens within the scope of the system. PAFFIN entities represent a mixture of those two extremes. They can possess decision making mechanisms and can (as agents) autonomously decide which workitems to request and when or how to complete activities. Their behaviour is automated in the way that it is completely defined and executed within the system.

The above can be simplified when considering that automated machines and devices need to be controlled in some way by the PAFFIN-System. If this is done through the PAFFIN entities themselves, the behaviour of the automated machines and devices becomes, conceptually, part of the behaviour of the PAFFIN entities. The PAFFIN becomes the resource, which has control of the machine or device.

Altogether, the backend resource can represent two kinds of resources: (human) users and automatic resources. Users are provided with the relevant workitem and activity data in a (graphical) user interface that lets them initiate and process workflow operations. Automatic resources are always the PAFFIN entities themselves, which are provided with the workitem and activity data directly and decide, through decision making mechanisms, which ones to execute. The behaviour executed is determined by the task and can involve the control of external automated machines and devices.

Note that this prototype only contains the core resource functionality. The graphical user interface provided to human users is discussed in Section 10.2.12.

The following describes the concrete requirements of the backend resource. Each of these needs to be provided for both resource kinds, which means that, if the requirements from that resource kind are incompatible, the backend resource has to implement different variants.

- The backend resource can process lists of workitems and activities for its resource.
- The backend resource can present the available workitems and active activities to its resource.

- The backend resource can initiate the three basic workflow operations on available workitems and active activities.
- The backend resource can register and deregister itself with the platform WFMS.
- The backend resource can handle and process incoming reports of direct operations from engines.

The major design decisions concerning this prototype are discussed here:

Multiple resources per Paffin *entity:* Within the overall workflow architecture some Paffin entities represent resources. One design decision made in the course of the backend resource prototype was to allow one Paffin to represent multiple resources. The decision was made to allow for related resources to be captured in one Paffin, unifying them in a single construct. Now, one Paffin entity may represent a human user, as well as all the devices that belong to that user, like printers or mobile devices. Of course, the management information (e.g. workitem and activity lists), as well as the provision of information between the resources needs to be clearly separated.

Subordinate resources: Another design decision revolved around the question, whether the resource a Paffin represents always needs to be singular or if it might be an abstraction of a group of resources. While similar to the previous design decision, this one does not ask if a Paffin can represent multiple resources, but rather if a resource can represent multiple resources. For the Paffin-System it was decided to allow the representation of groups of resources. The reasoning behind this was based mostly on the use case of having a Paffin entity (as a resource itself) control another group of Paffin entities to distribute the workload among them. The resource Paffin would be responsible for the workflow tasks while the so-called subordinate resources (i.e. the controlled Paffin entities) would do the actual work. Similar use cases are also feasible for automated devices (e.g. a Paffin controlling multiple printers) and even human users (e.g. a Paffin representing a department head that represents all other workers in his or her department).

Paffin *workflow task management:* When a Paffin entity represents itself as a resource it needs to be able to start some behaviour with the provided parameters of a workitem, control that behaviour and, when finished, receive back a result before confirming or cancelling an activity. To start the behaviour associated with a task, the regular mechanism from Capa to start protocols was adapted. Each task intended for a Paffin contains an object inheriting from the ontology class ResourceTaskAction (see Section 10.2.1). That object must be modelled specifically for that task and contains the task parameters. Due to its superclasses, it can be turned into a message that the Paffin can send to itself. Receiving that message reactively starts an instance of process-protocol containing the behaviour for that task. Sending that message is a transparent administrative action that is not explicitly modelled in an Agent-Activity.

During the task-related process-protocol the Paffin has control over the execution as it does over any other process-protocol, meaning it can terminate and abort it. The task-related process-protocol can have an arbitrary content of Agent-Activities or regular agent behaviour, but must also contain some standardised net components. Those components allow for the additional requirements of cancelling and confirming the activity. Cancelling terminates the process-protocol and informs the management of the cancellation while confirming also reports any achieved results. The report of

confirmation and cancellation is again handled via administrative, transparent agent messages that are automatically processed by the backend resource.

Handling direct operations: Direct operations happen when the engine decides to confirm or cancel an activity. For resources a decision had to be made on how to handle such direct operations. Since the control of the workflow and, consequently, the activities contained within it lies, as previously discussed, with the engine, the resource can only accept the information about direct operations. However, it was decided that the resource would still process the information of the direct operation and, in the case of a confirmation, would adopt the result of the confirmed activity into its own tracking of activity results. Additionally, in the case of a cancellation it was decided not to prohibit or prevent a new request of the workitem from the resource side. The engine can still decide to deny a request from a previously failed resource.

Implementation The net implementation of the backend resource comprises one main backend net located in the `PaffinEntity` plugin, which is instantiated by and managed in the main backend control net. As with most backend nets, the backend resource net is too large to suitably depict wholly as a net figure in this thesis. Because of this, the following describes the functional parts of the backend resource nets, with highlighting figures depicting the most important and representative net parts.

Initialisation and Access: This functional part receives initialisation data from the backend control net, makes that data available and also creates initial default tokens of lists and maps that are required for the resource functionality. It also initialises the registration of resources, which is then performed in the corresponding functional part.

Incoming Message Communication: Here, incoming messages routed from the backend control regarding currently active activities are received and further routed to the appropriate functional parts. For each routing, unification via IDs is utilised to ensure correct message assignment.

Outgoing Messages: The backend resource sends out some administrative agent messages to the platform WFMS itself, for example when initiating the Paffin behaviour associated with a workflow task. Whenever it needs to do so, it puts the message on a place in this functional part, which is then removed to the backend control net, which in turn moves it to the Paffin net, which actually sends out the message.

Resource Initialisation: This functional part prepares, initiates and processes the registration of all resources represented by a Paffin entity. For automatic resources the resource data, including credentials, is defined in the knowledge base of the Paffin. Whether a Paffin is designed to represent a human user is also defined in the knowledge base. However, this is only a flag that enables the Paffin to initiate a GUI in which the user can enter his or her credentials. Login and other GUI related mechanisms are discussed further in the GUI prototype in Section 10.2.12. Entered user credentials are returned to the current functional part. When all necessary data is available, the information is bundled and sent to the platform WFMS for the actual registration. The answer from the platform WFMS is always a resource object coupled with current versions of all applicable workitem and activity lists for that resource[11]. These are put onto the corresponding places, from which they are used throughout the rest of the backend. Note that each resource a Paffin represents is individually registered

[11] Resource object and lists are returned even if the registration failed. In that case the resource only has minimal default permissions, though. This is discussed further in Section 10.2.14.

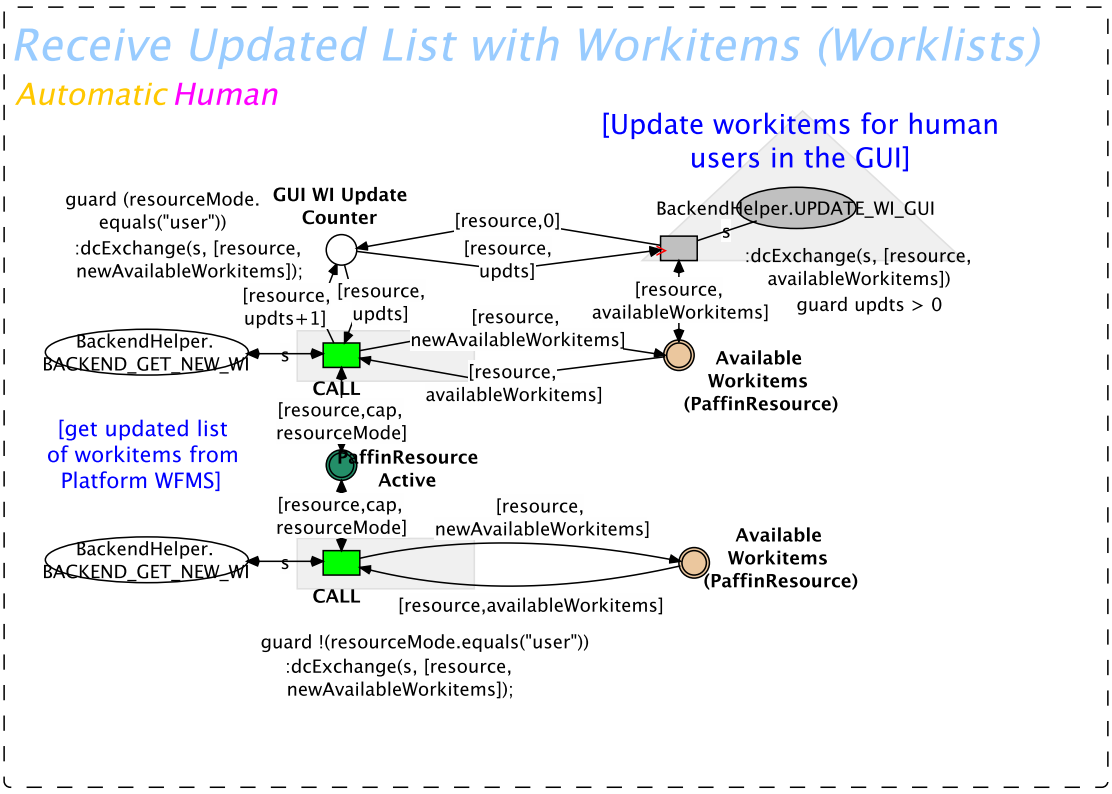

Figure 10.21: Backend resource: Workitem lists

and the data always separately stored. This ensures a clear separation between all resources represented by one PAFFIN entity. Each list or other data type associated with a resource is stored in a tuple with the unique identifier of the resource. That way it can always be accessed correctly using reference net unification.

Deregister Resource: The deregistration of a resource is implemented in this functional part. It is markedly simpler than the registration, only initiating the deregistration for one resource with the platform WFMS and, upon confirmation, removing all lists and data objects from the different places of the net. All runtime data associated with the resource is subsequently removed from the backend net, except for logging data.

Subordinate Resources: This functional part implements the support of subordinate resources (see discussion in the design section). Currently, the support is only trivially provided as preparation for future work. The only subordinate resource is the PAFFIN entity itself, which means that the work is always assigned to itself.

Workitem and Activity Lists: This is actually a group of functional parts. Each part contains and manages one type of workitem or activity lists for each resource of a PAFFIN:

- Available workitems (worklist)
- Active activities
- Confirmable activities (to indicate status of activities)
- Done activities (for monitoring and logging purposes)

All of these lists are managed in a similar way. Representatively, Figure 10.21 shows the functional part for available workitem lists. Each functional part manages one place containing the lists for each resource. That place is accessed by two transitions with synchronous channels, which receive lists and updates from the platform WFMS. The distinction here is between automatic and user resources. Automatic resources work directly on the lists in the places. User resources need to be presented with the information in the lists in a GUI. In Figure 10.21 the upper part receives and updates workitem lists for human users, while the lower part receives the lists for automatic users.

Request Workitem (Automatic Resource): This functional part realises the initiation of a workitem request for an automatic resource and is shown in Figure 10.22. Decision making mechanisms in the current prototype are kept simple, but can be extended. Each automatic resource has a maximum capacity of concurrent active activities it is allowed to request. This prohibits a resource from continuously requesting workitems without executing any work on them.

The choice of workitems is done in the left-most transition of Figure 10.22. Whenever workitems are available for an automatic resource and if the counter of concurrently active activities (stored in the yellow, central place) is less or equal to the maximum capacity for that resource the workitem with the highest priority is chosen by an auxiliary helper method. The workitem ID and resource object are then sent to the platform WFMS, which checks the request and, if valid, forwards it to the engine to actually perform the request. When an answer arrives at the resource backend there are a number of possibilities. If the request failed, e.g. when the workitem was not available anymore, the request loop is reset and can start again. If the request succeeded, the activity received from the engine and platform WFMS is checked for the required ResourceTaskAction object. Without that object, the Paffin can't initiate any automatic behaviour. If the ResourceTaskAction is missing the resource backend automatically cancels the activity and subsequently resets the request loop. If the ResourceTaskAction is available, the net structure in Figure 10.23 is called via synchronous channel. That net structure chooses a subordinate resource transforms the ResourceTaskAction into an agent message, stores the management data for the activity and finally sends the agent message to the chosen subordinate resource, which reacts with the desired behaviour. Lastly, back in the request loop in Figure 10.22, the successfully requested workitem is manually removed from the current workitem list of the resource. This is done to ensure that the loop, which is reset in the next firing step, does not attempt to request the same workitem. It is necessary, since the new workitem list is sent by the platform WFMS asynchronously and might not arrive before the request loop starts firing again.

Activity-related Message Handling: The activity-related message handling implements Paffin internal workitem and activity management for automatic resources. Here, messages relating to the activities of automatic resources represented by a Paffin are received, processed and any operations (confirm/cancel) initiated. Messages originate only from the subordinate resource performing the behaviour associated with the task. After the start message created from the ResourceTaskAction object is received by the subordinate resource and the corresponding process-protocol started, the special components in that process-protocol can send out the administrative activity-related messages. For each activity and process-protocol two messages are sent. First, a message is created right at the start, which acknowledges the start of the

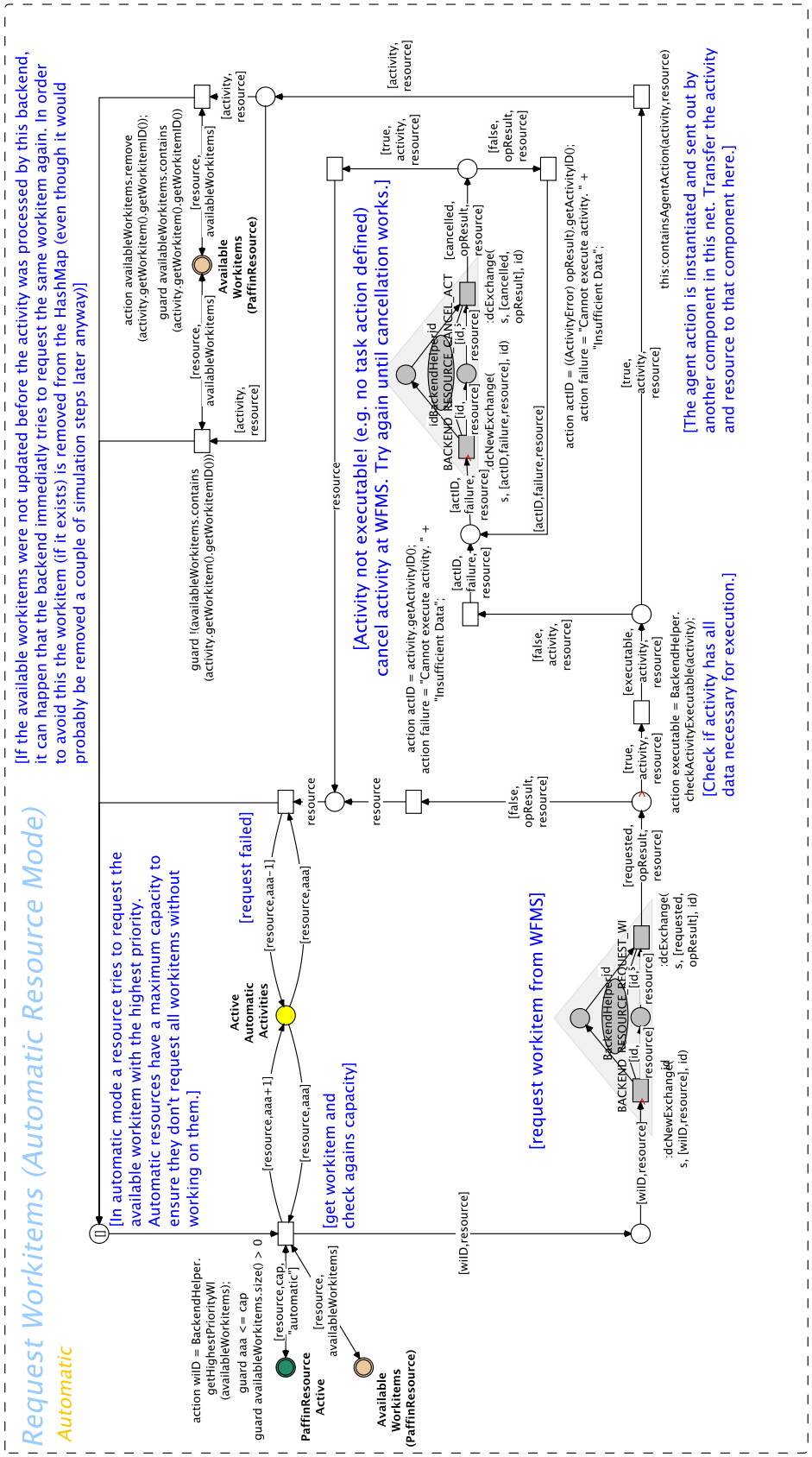

Figure 10.22: Backend resource: Request workitem (automatic resource)

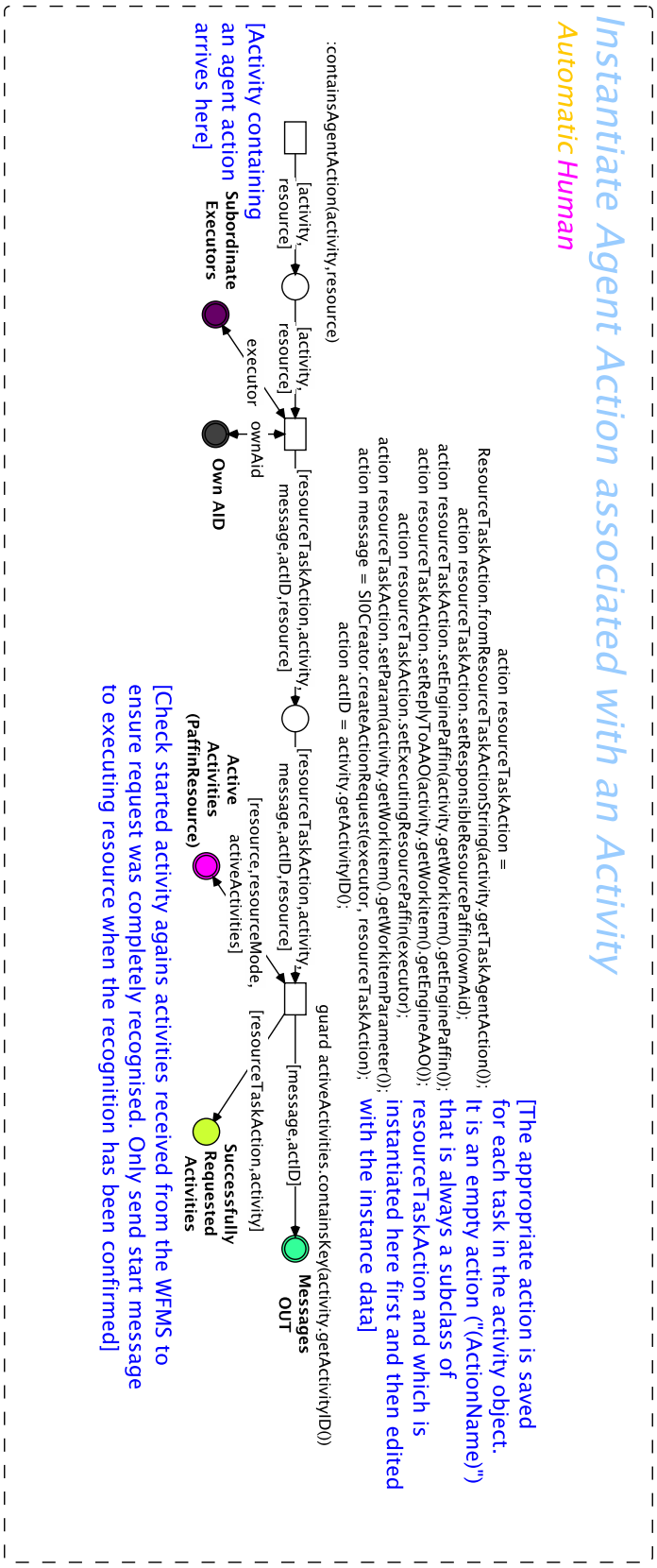

Figure 10.23: Backend resource: Start process-protocol through agent action

process-protocol. Second, a message is created when the activity and process-protocol are ended. Depending on whether the end is a confirmation or cancellation by the subordinate resource, different messages are sent out by the process-protocol and received by the backend resource. After processing of the received messages the backend resource initiates the confirmation or cancellation with the platform WFMS. Afterwards, the backend cleans up its management data and reattempts any failed operations if applicable.

Direct Operations Handling: This functional part receives information about direct operations performed by workflow engines via the platform WFMS. Since the control of the workflow lies with the engines, the resource can only accept the direct operations and clean up its management data. If the activity is executed by an automatic resource, that resource must be informed via administrative message to cancel the corresponding process-protocol as soon as possible.

Workflow Operations (Human User): This group of functional parts realises the initialisation of the three basic workflow operations for human users. They all follow similar net structures, which is why they are discussed here together and illustrated, representatively, by the workitem request shown in Figure 10.24.

Workflow operations for human users are always controlled by the human users via their GUI. The signals to initiate the operations arrive at the backend resource. First off, the credentials provided with the signal are checked against the credentials of the resources represented by the current Paffin. This ensures that no fraudulent signal causes a workflow operation. Next, the operation is performed by the platform WFMS and workflow engine, after which the answer of the operation is received. If the request failed, that failure is reported back to the GUI and the user.

If it succeeded, there are minor differences between the three operations. For workitem requests the result is checked if a special task GUI is provided or if the default GUI is used (see GUI prototype in Section 10.2.12). Special task GUIs are stored for later use and display. In addition, the result is also checked for a ResourceTaskAction that would indicate that some additional functionality for this user task should be executed by a subordinate (automatic) resource. For successful activity confirmations and cancellations the internal management data is cleaned up by removing the stored GUIs and references to the activity. Lastly, for all successful operations, the GUI is informed of the success. Updates to activity or workitem lists are not affected by this information, because they are handled by the functional parts managing those lists.

Conclusion The purpose of this prototype was to enable the functionality needed to have Paffins represent and function as workflow resources. The backend resource net fulfils this purpose completely. It allows a Paffin to receive workflow related data and process workflow operations on behalf of its resources. Those resources can be human users, automatic machines and devices, as well as the Paffin itself.

There are two major areas, in which future work may improve or extend the backend resource functionality. First, the idea of subordinate resources was discussed and prepared for during the design of this prototype, but was only rudimentary implemented. Currently, only the Paffin entity itself is a subordinate resource. Later on, the Paffin could serve as an automated workload distribution manager for all of its subordinate resources. That way, modularity provided through multiple Paffin entities becomes an implementation focus. This can be utilised to provide more flexible and adaptive application solutions. Another area for improvement is the decision making for automatic resources. Currently,

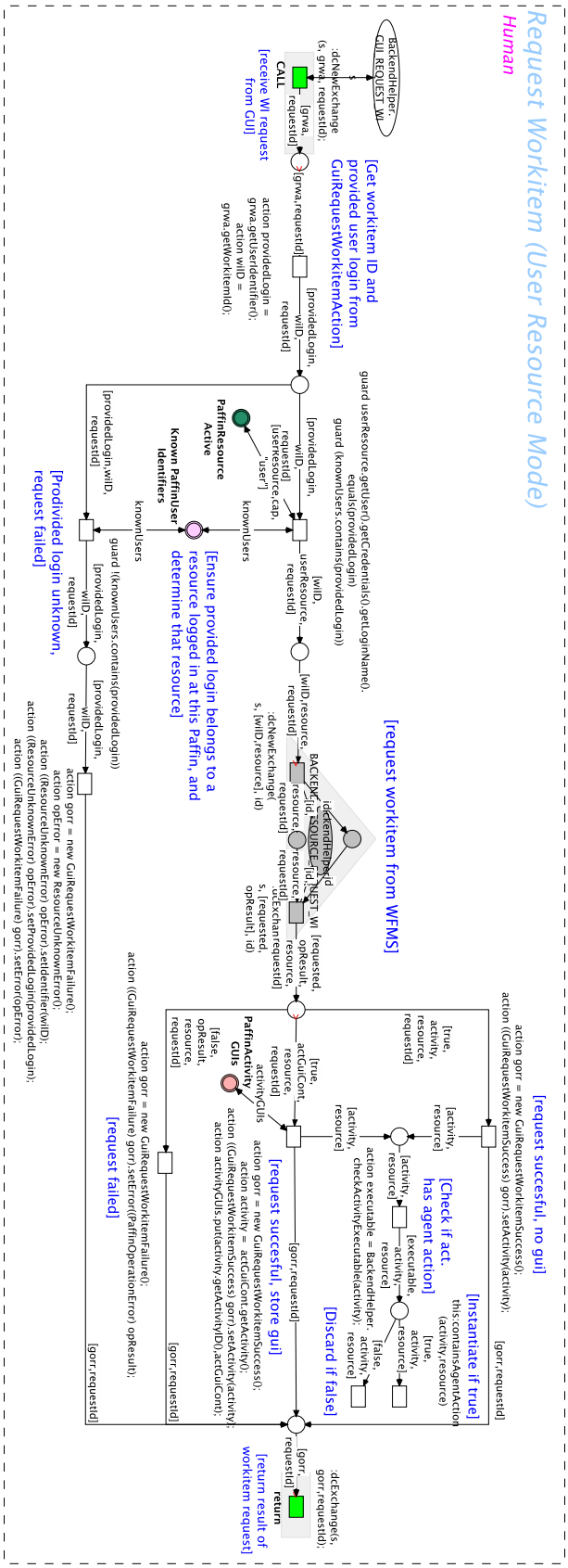

Figure 10.24: Backend resource: Request workitem (human user)

the highest priority workitem is always requested. While this is a reasonable approach, agent intelligence and decision making mechanisms could be introduced to better adapt to different situations. For example, analysing the workitem list might yield results about bottlenecks in the workflows of the overall system. If there are 20 workitems of the same task it might be better, under certain situations, for the overall system performance to work on these, rather than on one with a slightly higher priority.

Overall, these kinds of improvements go beyond the conceptual requirements for an integrated entity as a workflow resource. They would clearly improve the usability and power of Paffins, which is why they are being considered, but they don't add anything essential.

10.2.12 Paffin Web GUI

The Paffin Web GUI prototype is a collaboration with Dennis Schmitz. Implementation and design responsibilities were about equally shared. Roughly, Mr. Schmitz was responsible for the direct GUI parts, while the author was responsible for the incorporation of the GUI and its elements throughout the Paffin-System. Important implementations and design decisions were always made in close collaboration. Ultimately, both Mr. Schmitz and this thesis' author contributed about equally to this prototype.

Purpose and Functionality The purpose of the Paffin Web GUI prototype is to provide human users with a graphical user interface (GUI), allowing them to interact with the Paffin-System. The functionality of the Paffin Web GUI encompasses two major areas.

First off, the GUI functionality itself needs to be addressed. This includes the elements to display in the GUI, their (semantic) interactions/relations between one another and the scope/focus of the GUI. The latter is especially important, since it defines the orientation of the GUI w.r.t. agents or workflows.

Secondly, the GUI functionality needs to be connected to and incorporated into the Paffin-System. Mechanisms for providing the GUI with all necessary data need to be provided. This again strongly depends on the orientation of the GUI. Also, GUI administration and control from inside the Paffin-System needs to be implemented.

Design The core orientation of the Paffin Web GUI prototype is workflows, or more precisely workflow tasks. The reason for this is simply that workflow tasks are basic descriptions of work. While workflow tasks can be performed by any resource, including automatic ones, human users are the primary target resources. As this prototype aims to provide a GUI for human users, the orientation towards workflow tasks is only natural.

Given this core orientation, the starting point for the GUI can be given as a basic WFMS interface. This yields the following basic function requirements:

- Login mechanism
- Display of available workitems to the currently logged in users
- Selection of workitems and display of workitem information
- Mechanism to request the currently selected workitems
- Display of (currently) active activities
- Selection of active activities and display of activity information
- Support of work associated with currently selected active activity

- Mechanism to confirm the currently selected active activity
- Mechanism to cancel the currently selected active activity
- Display of done (i.e. previously confirmed) activities
- Selection of done activities and display of activity information including results
- Logout mechanism

The following major design decisions were made during the development of the Paffin Web GUI:

Web-based GUI: As the name of the prototype implies, the GUI is web-based. This means that users only require a web browser to access it. Simplifying access by providing an established and well-known type of interface for users was only one reason for this design decision. The other was related to the capabilities of the already available WebGateway plugin for Capa (see Section 2.2.3 and [Betz, 2011, Betz et al., 2014]), which could be used for the implementation of the web-based GUI due to the full support of Capa legacy code in the Paffin-System. The WebGateway allows Capa agents (and by extension Paffin entities as agents) to provide their functionality as web services to be accessed via browser based calls. It also provides an extensive and flexible toolset to build custom GUIs. Instead of reimplementing that functionality, which was continuously developed over a number of years at the TGI research group, the decision was made to utilise the available code. Additionally, during that time extensions to the WebGateway were being developed in [Müller, 2016]. The OgeeJS framework provided a modular toolkit with which to easily design templates for user-friendly web GUIs. This framework of templates was also chosen to be incorporated and used with the Paffin GUI.

Separation of GUI and other functionality: Separating the GUI implementation from the rest of the Paffin-System was another major design decision. The GUI implementation is modular by nature. As long as the basic interface, which is only accessed by the backend resource net (see Section 10.2.11), is adhered to, any other GUI implementation can function with the Paffin-System. This decision was made to clearly encapsulate the GUI from the rest of the system. Future work may also provide new and/or specialised GUI implementations more easily.

Task-specific GUI: Most workflow tasks can be supported by a hard-coded GUI for all tasks. Differences could be handled by configuration for the needs of each task. However, more advanced mechanisms would always require changes and extensions to the mechanisms within the framework. These kinds of extensions can be difficult and also precarious if they affect other parts of the system negatively.

A better option is to provide a flexible, task-specific GUI. Each task would bring with it its own GUI object, which would adhere to a specific interface. The GUI then would only provide a container in which that object could be placed and displayed. That way, a task could provide any kind of user support without changing or cluttering the framework. Modularity and simplicity could still be maintained by providing a default modular GUI object and basic toolset of the modules.

The latter option was chosen for the Paffin Web GUI prototype. It required more implementation effort due to the flexibility. However, besides the advantages described above it also helped to alleviate the issue of the workflow-orientation of the GUI to a degree. Any task can contain any kind of GUI object. That GUI object can freely

interact with agent functionality contained between the request workitem and confirm activity operations. This means that any arbitrary agent GUI can be incorporated into a workflow task in the PAFFIN-System.

External GUI: Another design decision regarded the status of the PAFFIN Web GUI. Its encapsulation ensures that it can be easily replaced. Its current status determines it as the *default* GUI for the PAFFIN-System. That does, however, not mean that it is the only one. The design decision here was not to force the GUI on applications, thus allowing any part of the system to provide and maintain other GUIs. This decision also further alleviates the issue of the workflow-orientation of the PAFFIN Web GUI. Any PAFFIN as an agent or a workflow can provide an alternative GUI. This GUI has to be completely handled in AGENT-ACTIVITIES and process-protocols, though, but can access the management facilities of the backend nets whenever necessary.

Implementation The `PaffinWebGui` plugin holds all nets, Java classes and JavaScript files of this prototype. Separating the `PaffinWebGui` plugin from the rest of the PAFFIN-System plugins ensures the encapsulation of the GUI and simplifies modification and possible replacement in the future.

The GUI functionality is accessed and initialised only when a PAFFIN entity wants to register a human user resource. In that case the resource registration is rerouted to another branch of the backend resource registration component. That component starts by instantiating a new DC, the so-called *GUI Connector DC*. That DC represents the interface between the PAFFIN-System and the `PaffinWebGui` prototype. It also starts another PAFFIN entity, the so-called *GUI* PAFFIN, which is responsible for providing, maintaining and communicating with the actual web GUI. Upon startup of the GUI PAFFIN, a login screen is provided to the user. Until a user accesses the GUI via a web browser and enters his or her credentials, the resource registration is delayed. Once the user has input his or her credentials, these credentials are received by the GUI PAFFIN and then transmitted to the original resource PAFFIN's GUI Connector DC. That DC then provides those credentials for the resource registration. After registration, the resource PAFFIN receives workitem and activity data for the user resource. That data is received as described in the backend resource prototype. Whenever an update occurs, that data is given to the GUI Connector DC, which transforms the data for readability and sends it to the GUI PAFFIN. The GUI PAFFIN then hands over the received data to the web GUI, which displays it. From the GUI, the user can initiate the workflow operations and other actions available in the task-specific GUI component.

The web GUI itself can be seen in Figure 10.25. The upper part is the menu bar. Here, the user can switch between the login screen and the workflow processing screen. In future work, more areas, like an administration area, can be added here.

The main part of the GUI is split. On the left-hand side the lists for available workitems, active activities and done activities can be seen. Here, users can select a workitem or an activity. The selection here controls what is shown on the right-hand side of the GUI. For workitems, the right-hand side is standardised. It shows the task details, including task title and description, as well as a request workitem button which initiates the request operation in the PAFFIN-System. For activities, the task-specific GUI is shown. That GUI is stored in the database and read by the platform WFMS during the request of the workitem. It is stored by the resource PAFFIN and provided to the GUI PAFFIN when the activity list is updated.

In Figure 10.25 the activity "RedTimer Gruppe" is selected. The task-specific GUI contains the task description, as well as an input field and a "Submit" button. Additionally

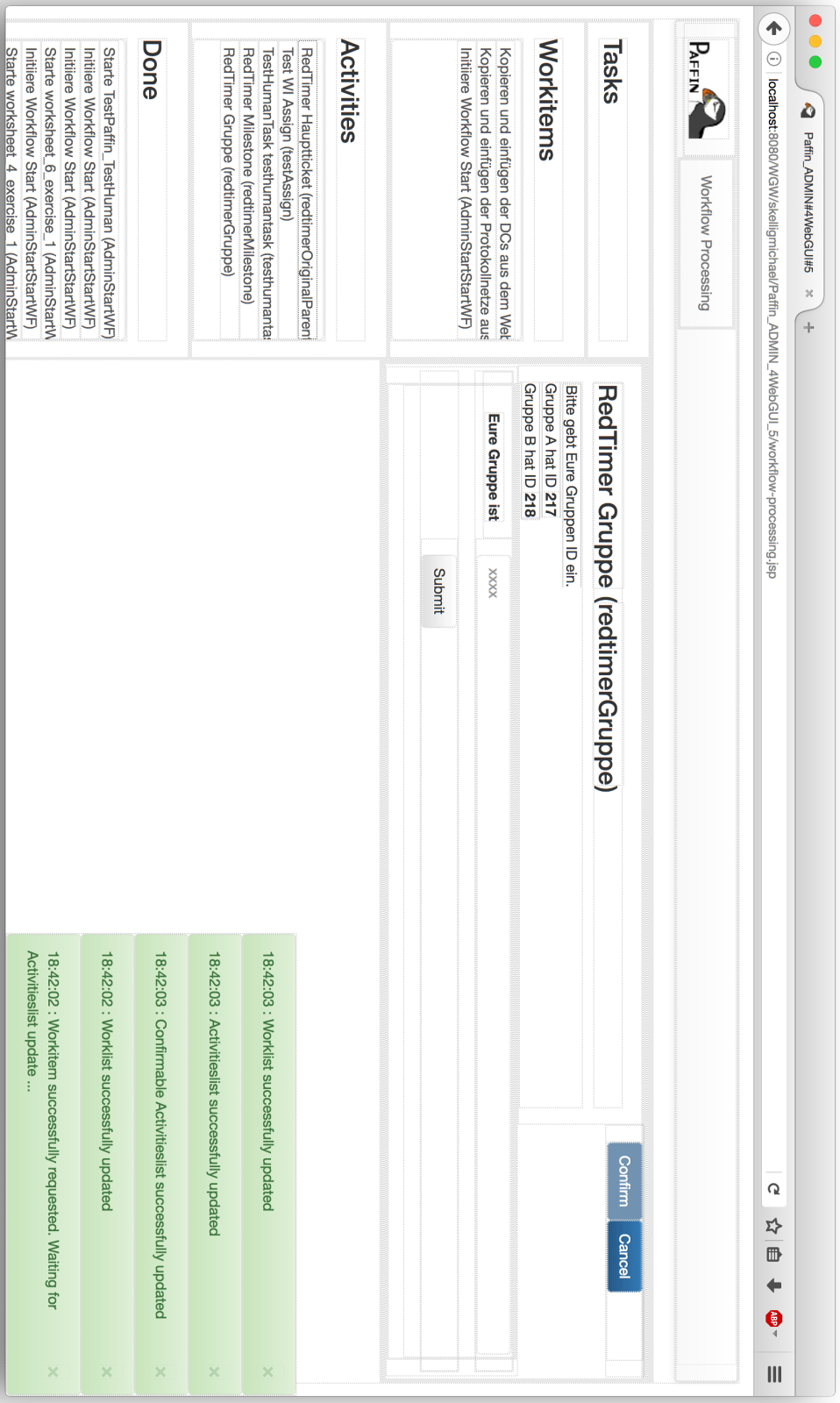

Figure 10.25: Paffin web GUI screenshot

the upper right-hand side contains buttons for confirmation and cancellation of the activity. The confirmation button in Figure 10.25 is currently disabled, as the "Submit" button supersedes it. On the lower right-hand side of Figure 10.25 an additional feature of the prototype is shown: status pop-up messages. These messages inform users of the results of the workflow operations they initiated, as well as of updates to workitem and activity lists. Green updates indicate successful operations, while red updates indicate and detail errors.

Internally, the Paffin Web GUI prototype is quite extensive. The following presents a rough outline of the implementation areas:

Extended ontology: The Paffin Web GUI required additional ontology concepts. These concepts can be roughly separated into two groups. The first one contains those concepts that are abstractly describing handling GUI objects within the Paffin-System. These concepts describe, without assuming GUI implementation details, which data is send to and received from the GUI. Due to the omission of implementation details, these concepts are independent of the actual GUI implementation and contribute to the interface. They can be reused for other GUI implementations. Examples include container concepts for workitem and activity lists, as well as concepts relating activities to their task-specific GUIs. These general concepts are maintained in the overall ontology of the Paffin-System.

The second group of ontology concepts contains concepts specifically for the Paffin Web GUI. These are maintained in a separate ontology in the `PaffinWebGui` plugin. This group includes the basic elements for the modular task-specific GUI, containers for results of tasks and internal list concepts for displaying workitem lists and activities.

Interactions: Since the GUI Paffin is a separate Paffin entity to the resource Paffin, interactions consisting of pairs of process-protocols needed to be implemented. These realise the exchange of data between GUI and resource. Included are interactions for starting a GUI Paffin, resource login, updating workitem and activity lists and handling workflow operations. The form for all of these interactions is similar. Either it is a simple inform (e.g. in the case of updating lists) or it is a request and wait for answer pattern (e.g. in the case of workflow operations).

Adaptions in the Paffin-*System:* Adaptions to the Paffin-System primarily related to the backend resource net. Here, the interface to the GUI is realised through synchronous channels connecting to the GUI Connector DC. These changes have already been discussed in the context of the resource backend prototype.

In addition to the backend resource, the platform WFMS needed to be extended. Reading, processing and handling the task-specific GUIs needed to be implemented. More details of this are provided in the corresponding prototype in Section 10.2.14.

GUI Connector DC: As described above, the GUI Connector DC is responsible for connecting the Paffin Web GUI subsystem with the rest of the Paffin-System. It is a standardised DC connected to the backend resource net of a Paffin, from which it receives updates from the resource and the platform WFMS and to which it forwards data and commands from the GUI. The functionality of the GUI Connector DC is kept intentionally simple. Most of the actual processing of data is done in the backend resource net. This way doesn't have to be reimplemented in case the GUI (including the GUI Connector DC) is replaced. Altogether, the GUI Connector DC contains net structures to instantiate the GUI Paffin, request credentials from the GUI, update the lists of workitems, active activities, confirmable activities and done activities, process

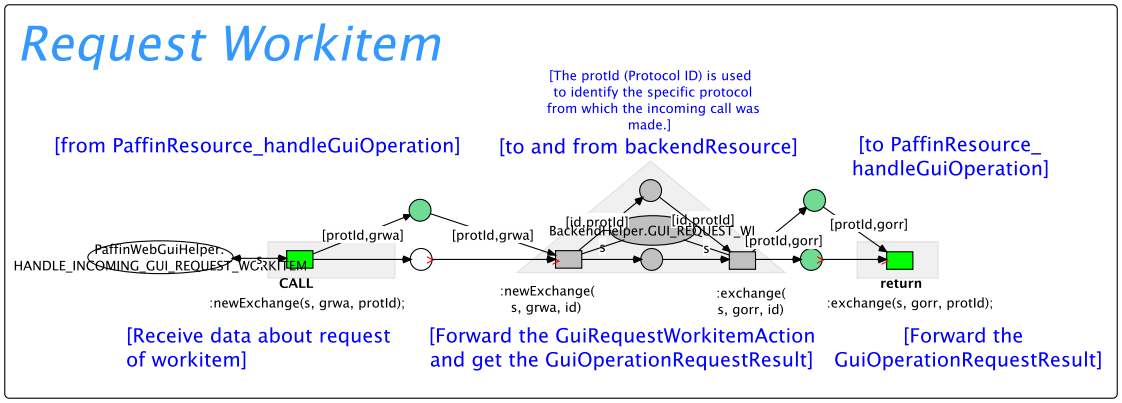

Figure 10.26: GUI connector DC: Request workitem

and forward workitem requests, activity confirmations and activity cancellations from the GUI and forward additional status updates from the GUI to the activity.

Representatively, Figure 10.26 shows the net structure to confirm an activity from the GUI. On the left-hand side the command to confirm is received from the process-protocol started upon reception of a corresponding message from the GUI Paffin. Next, the command is forwarded to the backend resource, which actually processes and performs it with the platform WFMS and engine. During that time, the GUI Connector DC waits for the answer. That result comes in the form of a GuiOperationRequestResult, which either contains a success or failure. That object is then finally returned to the original process-protocol, which in turn forwards the answer back to the GUI Paffin and the GUI. The net structures for request workitem and cancel activity are completely analogous.

Processor Agentlet DC: The Processor Agentlet DC is executed in the GUI Paffin. It realises the technical link between the agent/Paffin parts of the system and the WebGateway and web parts of the system. On the one hand it translates incoming commands from the GUI into commands that can be understood and processed by the Paffin-System. On the other hand it also translates the data coming from the GUI Connector DC of the associated resource Paffin into data and objects that can be processed and displayed in the web GUI. Regarding communication, the web GUI communicates directly via web services translated from the WebGateway mechanisms with the Processor Agentlet DC. The communication between the Processor Agentlet DC and its resource Paffin is handled via the above mentioned interactions.

The Processor Agentlet DC contains the following net structures that realise the required link:

- *GUI location:* Since it is possible to have multiple GUI agents in a system the Processor Agentlet DC must track which one it is in charge of.
- *Update lists:* In these net structures the data originating from the GUI Connector DC is processed for display. Each list has a timestamp. The timestamp of the last update is stored and compared against any new incoming lists. That way race conditions are avoided[12]. When the timestamp of an incoming list is processed in

[12] Even though the Paffin-System is a distributed system timestamps can be used here. The lists and timestamps always originate in the same source GUI Connector DC, so the timestamps are always valid and comparable.

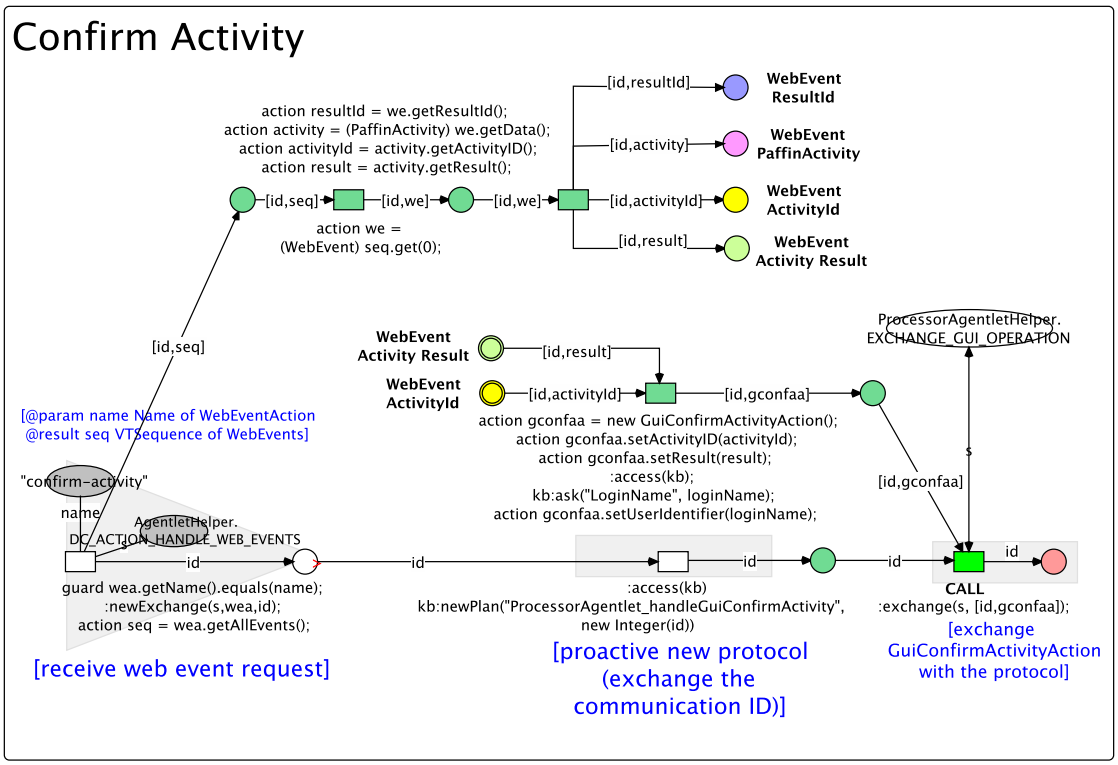

Figure 10.27: Processor agentlet DC: Confirm activity

the Processor Agentlet DC a new WebEvent object is created and handed over to the WebGateway mechanisms to display in the web GUI.

– *Provide credentials:* When the GUI is first started the Paffin-System expects the credentials of the user before it can continue. The input fields are prepared for and then initiated for display in this net structure, which also waits for the input and then forwards it back to the GUI Connector DC, which then in turn hands it over to the backend resource for resource registration.

– *Cached data access:* This net structure caches the most up to date content of the GUI. In case the user refreshes the GUI web page, the cached data can be accessed and restored. This mechanism avoids a full update and restoration from the Paffin-System.

– *Workflow operations:* In these net structures the commands from the GUI to initiate the workflow operations are processed. For all three basic workflow operations, the net structures are analogous and consist of two parts: Receive command from GUI and process result. Representatively, Figure 10.27 shows the net structure to process the command to confirm an activity.

On the left-hand side the command from the GUI is received. It is then processed in the upper branch, where all data is extracted from the command. The lower branch starts a process-protocol which waits for the processed data provided to it from the right-most transition inscribed with synchronous channel and sends that data the GUI Connector DC in the resource Paffin. The answer to the command is received in the related net structure to process the result, not shown in Figure 10.27. If the operation was successful a GuiConfirmActivitySuccess

object is received, otherwise the answer is a GuiConfirmActivityFailure object. Either way, the result is processed, packaged into a WebEvent object and handed over to the WebGateway for display.

Beyond the Processor Agentlet DC, the standard WebGateway mechanisms are used for the implementation. These connect the agent/Paffin parts of the system with the actual web GUI. For details about the WebGateway and utilised mechanisms please refer to [Betz, 2011, Betz et al., 2014, Müller, 2016].

In terms of non-net implementation the Paffin Web GUI prototype's most prominent features relate to the GUI provided in JavaScript[13] classes. The GUI itself is defined in a JSP file, containing the JavaScript code for the dynamic web page shown to the user. When the web page is called by a browser program, the corresponding JSP file and its controller object are initiated. That controller subscribes to certain events with the WebGateway from the Paffin-System. Whenever the Processor Agentlet DC hands over a WebEvent object to the WebGateway, the WebGateway transforms it into the corresponding event and sends out that event, via WebSocket, to the event's subscribers. The content within that event object consists of objects defined in the WebGateway ontology. When an event arrives at a controller for a web page, a predefined *callback* method is called and provided with the event as a parameter. The data contained in the event is processed and filled into an OgeeJS template, which can be directly displayed in the web page. In the other direction, active elements of the GUI (e.g. buttons) are defined with a specific handler, which calls a method when the element is activated. That method gathers data from the GUI (e.g. content of input fields), creates a container object within a new WebEvent and sends that towards the WebGateway, which processes it and forwards it to the correct GUI Paffin entity.

For the Paffin Web GUI, there are currently two main JSP files. The first one, *index.jsp*, contains the GUI for the login of users. The second one, *workflow-processing.jsp*, contains the overall workflow GUI shown in Figure 10.25. To switch between the different JSP files, users can either use the topmost menu bar or they are automatically redirected after login.

Apart from the JavaScript GUI parts, the prototype also features a number of auxiliary Java classes. These provide often-used methods that simplify net code and constants used throughout the prototype.

Conclusion The purpose of the Paffin Web GUI prototype was to provide a graphical user interface for human users to access the functionality of the Paffin-System. The prototype fulfils that purpose. It provides an interface structured by the workflow task concept. Users are presented with available workitems, can select and request them and then work on the emerging activities. Of the functionality laid out in the coarse design, everything has been implemented and works, except for the logout mechanism. The latter is considered future work until a more complex user management from the Capa context can be incorporated. The remainder of the functionality fully supports users. Each workflow tasks brings its own modular and extensible GUI. That GUI can contain standard workflow related input and description fields, but can also contain any arbitrary HTML compatible functionality defined within the agent parts of the process-protocols of the Paffin-System.

Regarding future work, the GUI provides the biggest potential for improvements in usability. The current prototype implements the necessary basics of functionality that are required for users to work with the Paffin-System. However, in terms of usability, there are a lot of possible extensions. Some of them have already been implemented. It is

[13] `https://www.javascript.com/` (last accessed May 28th, 2017)

already possible to use a specific syntax in task descriptions that translates specific string constants into variables from the task parameters. Text formatting in task descriptions in the GUI is also already implemented. The pop-up alerts shown in Figure 10.25 are another, already implemented additional feature. These kinds of features are not essential to using the Paffin-System, but together improve the usability greatly. Other possible and not yet implemented features include an interactive visualisation of the overall workflow in the GUI, system administration in the GUI and workitem/activity filters. These topics, however, are designated as future work.

10.2.13 Paffin Platform

Purpose and Functionality The Paffin platform prototype's purpose is to provide the runtime execution environment for Paffin entities. The functionality of the Paffin platform prototype can be directly drawn from that purpose. Required is a runtime execution environment that manages both the life cycle of the individual Paffin entities and all the auxiliary services required for the execution of all Paffin entities. The latter services include agent management services defined by the FIPA, global (at least w.r.t. the platform) workflow managing services and the infrastructure enabling the communication between Paffin integrated entities.

Design Roughly speaking, the challenges of this prototype can be separated into providing an execution environment for Paffins and incorporating management services into that environment. The base Capa platform already provides a solution following such a separation for regular agents. The platform itself is a distinct (agent) net that serves as a container and execution environment for other (agent) nets that provides agent management services (AMS and DF) for all agents in the platform. By adapting the mechanisms to Paffin nets instead of Capa agent nets and extending the management services for workflows the challenge of this prototype can be addressed. Choosing this approach for implementation in this prototype yielded the following required base functionalities that needed to be implemented:

- Paffin life cycle management
- Communication infrastructure provision
- Management services initialisation

Note that the management services themselves are not part of this prototype. The agent management services are already fully provided by the Capa AMS and DF agents, which could be directly reused. The workflow management services on the other hand were not available before. They needed to be fully implemented. Due to the large scope of the management services they were implemented in a separate prototype that is discussed in the following section.

Implementation This current prototype featured a limited scope of implementation areas. Using the Capa platform as a basis allowed for the utilisation of the entire agent management functionality of Capa. Only some nets had to be adapted in order to achieve the necessary requirements for Paffin entities. These nets included the Paffin platform net and the platform protocol net to start a new Paffin entity. However the changes to these nets were limited to changing inscriptions to indicate the different target domain and nets (e.g. instantiate the `paffin.rnw` net instead of the `agent.rnw` net). Additionally, the

platform protocol net that is responsible for setting up the platform services like AMS and DF needed to be extended to also start the platform WFMS services. Stopping already running Paffin entities required no adaptions, as it is handled via simple unification over net instances.

The communication facilities of the Capa platform could also be directly reused. Paffins communicate with the same interface and asynchronous message format as Capa agents. Furthermore, the overall workflow management architecture of the Paffin-System determined that the management services and participants would also communicate and interact via such (administrative agent) messages.

Regarding Java implementation, this prototype only uses a minor helper class that is used to define constants and locate necessary platform data files in the file system.

Discussion The decision to base the Paffin-System technologically on the existing Capa system allowed for an easy realisation of this Paffin platform prototype. The purpose of the platform prototype is completely fulfilled, since the existing mechanisms to start and stop agents, as well as enable inter- and intraplatform communications, could be easily adapted or directly utilised.

Future work regards the realisation of optional and extended Paffin management functionality. Mobility of Paffins, for example, is a desired feature that, regarding the implementation, would mostly affect the Paffin platform nets. However, mobility is only rudimentary supported in base Capa. An extension to Capa, called Mapa [Cabac et al., 2009], focuses on mobility and its representation. Incorporating mobility and Mapa into the Paffin-System is not essential to the proof-of-concept for the Agent-Activity approach and therefore considered future work.

10.2.14 Paffin Platform WFMS

Purpose and Functionality The purpose of this prototype is to implement the connection between the Paffins as engines and resources on the platform and management level. It is part of the Paffin platform management services.

The functionality of the Paffin platform WFMS is outlined in the overall workflow architecture shown in Figure 10.17. It receives workflow runtime information from all workflow engines and distributes that information to registered resources. In addition to that it informs resources of direct operations performed on activities by engines. In the other direction, the platform WFMS receives operation commands from the resources, checks them for validity and permissions and forwards the valid operation commands to the engines which actually perform the operations.

Design The functionality to connect and relate workflow engines and workflow resources constitutes the workflow management services required by a Paffin platform. Having the same significance as the agent management services AMS and DF, the platform WFMS was required to be implemented in a similar fashion as an agent subsystem[14].

In of itself, the platform WFMS subsystem represents an intermediary system in the terms of the WorkBroker principle (see Definition C.13 in Section 10.3.2). An intermediary system in the WorkBroker also connects engines and resources in the same way as the platform WFMS of the Paffin-System.

An important aspect of workflow management is persistent data storage. Persistent data includes resource credentials, additional task data not inscribed directly in the workflow

[14]Subsystem in this case relates to one or more agents providing the required functionality.

operations and the task-specific GUI. Providing for and enabling the Paffin-System access to that persistent data is also part of this prototype.

Given this coarse design, the detailed scope of functionality contains the following subject matters:

- Resource (de-)registration and management of registered resources
- Reception of all types of individual workitem/activity lists from engines and aggregation of those lists
- Filtering and distribution of all workitem/activity lists to resources
- Reception, validation and routing of commands to the correct engines
- Monitoring and logging of workflow operations
- Interactions for exchanging data between platform WFMS and Paffins as engines
- Interactions for exchanging data between platform WFMS and Paffins as resources
- Interface/reception mechanisms to backend resource net
- Interface/reception mechanisms to backend engine net
- Provision of additional (persistent) task data not directly inscribed in the Agent-Activity

Most design decisions affecting the platform WFMS have already been discussed in the context of the backend engine and backend resource. These applied to the role and shape of the platform WFMS and its relation to the engines and resources. Remaining major design decisions affecting only the platform WFMS included the following:

Encapsulation of interface to backend engine and resource nets: To further separate concrete workflow management concerns from the integration aspects of the Paffin-System an additional indirection between the backend resource/engine nets and the platform WFMS was introduced to the system. The platform WFMS interacts with Paffins as engines and resources via process-protocols. If those process-protocols would directly interact with the backend engine/resource nets, the backend and management would be directly coupled. Even though the main interface between the platform WFMS and the engines and resource consists of the administrative agent messages between them, possible later changes to the main communication mechanism are not excluded. A direct coupling would hinder or prohibit such changes. Additionally, there is a lot of data processing necessary for the communication between Paffins and platform WFMS. Including that data processing in the backend nets would complicate and enlarge those nets even more. For these reasons DCs were implemented for both the resources and engines. These DCs connect to the respective backends, take over data processing duties and also handle communication with the platform WFMS.

Workflow task permission system: Workflow tasks require some form of access control. Role based access control is a well-established access control mechanism [Ferraiolo and Kuhn, 1992, Sandhu et al., 1996]. The basic concept is to define roles to which users/resources are assigned. Roles define permissions, which describe which tasks a resource may work on. As it is an established, yet simple access control mechanism, role based access control was chosen for the Paffin-System.

Choice of database storage: It is clear that the platform WFMS requires a database connection. However, a decision needed to be made as to which data to store in that database. Ultimately, the following data was chosen for storage:

- Resource credentials
- Workflow roles
- Permissions
- Task data (e.g. descriptions, required permissions)
- Task-specific GUI

All of this data shares the same distinctions. It is needed globally (i.e. by the platform WFMS) and it is needed in each execution of an application built with the Paffin-System. Furthermore, moving all of this data to the database simplifies the inscriptions in the Agent-Activity internal processes. Only the absolutely necessary inscriptions are needed there, which improves readability and maintainability. Reusing data like the task-specific GUI for multiple tasks is also easier when that data is stored in a database.

Database access: Another design decision regarding the database concerned which Paffins of a system would have access to the database. Obviously, the platform WFMS Paffin needs access, as it is responsible for checking permissions, roles and credentials. However, the additional task data and task-specific GUI are more relevant for the resource Paffins. Still, the decision was made to limit database access to only the platform WFMS Paffin. While resource Paffins could benefit from directly accessing the task data and GUI, they could, conceivably, also gain access to data that should not be available to them. By limiting access to database, the platform WFMS can ensure that resources only get the data they are authorised for.

Implementation The Paffin platform WFMS prototype is part of the `PaffinPlatform` plugin. However, the interfaces between the backend nets and the workflow management, implemented as DCs, are located in the `PaffinEntity` plugin since they relate to the individual Paffin entities as opposed to the overall platform.

The platform WFMS itself is implemented as a single, special Paffin entity. That platform WFMS Paffin is started automatically during platform initialisation along with the AMS and DF Paffins. Its identifier is made known to all Paffin entities started on that platform during their initialisation, ensuring that all Paffins know how to interact with the platform WFMS.

The main implementation efforts of the prototype concern the platform WFMS Paffin. It executes a DC, which implements the functionality required for the prototype. The following descriptions are focussed on and structured by that WFMS DC. To realise the interface between the Paffin engine and resource backend nets and the platform WFMS, additional DCs and process-protocol interactions were implemented. Those DCs are the engine DC and resource DC. Descriptions of the DCs and interactions are incorporated into the descriptions of the main platform WFMS DC to provide a better structure to the descriptions.

The platform WFMS DC contains the following net structures:

Initialisation: The initialisation net structure prepares default and empty values for all lists and maps used for storing, accessing and processing data within the platform WFMS. Beyond that, it also initialises the database with the initial values class (see below) and the basic logging mechanism.

Register resource: When a Paffin wants to register one of its resources, it hands over all the resource data (e.g. user credentials received from the web GUI) to its resource

DC, which sends that data to the platform WFMS. That data is received in this net structure. Registration processing starts off with an assessment of credential validity. The provided credentials must be non-empty and non-default and they must be present in the database of known users. Even automatic resources must provide a known and valid name and password.

If at any point the registration encounters an error or false/invalid information, the resource is logged in with the default role instead of the role it intended. Using the default role is the general escape mechanism for resource registration. The mechanism simplifies the registration for automatic resources that don't require any special access to the workflow functionality. With the default role they are logged into the system and can receive basic information from it. Further details of the role based access control are provided further below.

After checking credential validity, the remaining resource data is read and a resource ontology object created. That object is then actually registered with the platform WFMS by adding it to the resource list. The result of the registration is a message to the resource PAFFIN containing the resource object and all initial lists.

Deregister resource: Deregistering a resource is initiated by the backend resource net and then sent to the platform WFMS via the resource DC and a process-protocol. The process is markedly simpler than the registration. If the resource is registered and if the resource does not have any active activities it is working on, the deregistration succeeds. Otherwise a failure message is returned.

Workitem lists handling: This net structure is responsible for receiving workitem lists from engines, aggregating them and then initiating the update of the workitem lists filtered for resources. Whenever new workitems are available in one engine, a list of all of that engine's workitems are handed over to the engine DC. The engine DC then starts a process-protocol that sends the list to the platform WFMS DC. The net structure for receiving workitem lists is shown in the upper part of Figure 10.28. New lists arrive at the left-hand side. The processing is always related to one engine. It starts by separating workitems already known to the platform from those that are completely new. The platform WFMS requires additional workitem data only available in the database. If the workitems are already known, that data was accessed before and is already present for the platform WFMS. Unknown workitems are handed over to the database functionality to retrieve their additional task data (lower branch in Figure 10.28). Once these workitems have been updated with the additional data, they are recombined with the already known workitems into a complete list of workitems for the current engine. Lastly, the list of workitems is stored in a HashMap that contains all lists for all workflow engines and the update flag (a simple counter) set, before returning a confirmation message to the engine PAFFIN, which is delaying the next update until the current one is finished. This completes the update process of workitems from the engine side.

The update of workitems for resources is handled by the lower part of Figure 10.28. When the flag for a pending update is set (i.e. the counter greater than zero), the update is initiated. The list of current workitems and resources is copied and handed over to a newly started process-protocol. That process-protocol iterates over the resources and creates, for each one, a filtered list of workitems. Filtering is done based on the resource permissions, which are stored along with the resources in the list provided to the process-protocol. The created lists are timestamped and sent to the resources, which receive them at their resource DC. The part of the resource DC

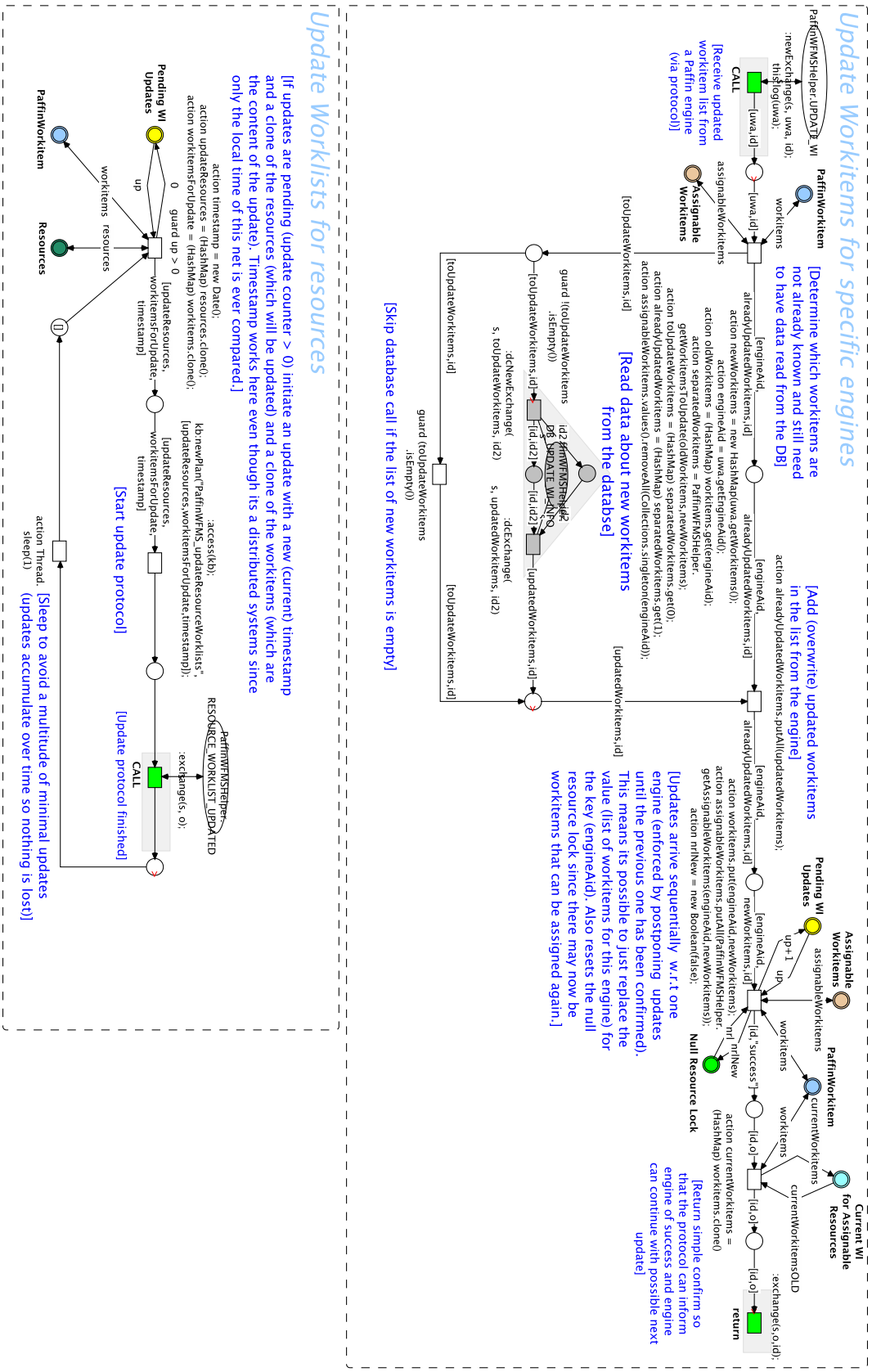

Figure 10.28: WFMS platform DC: Workitem list handling

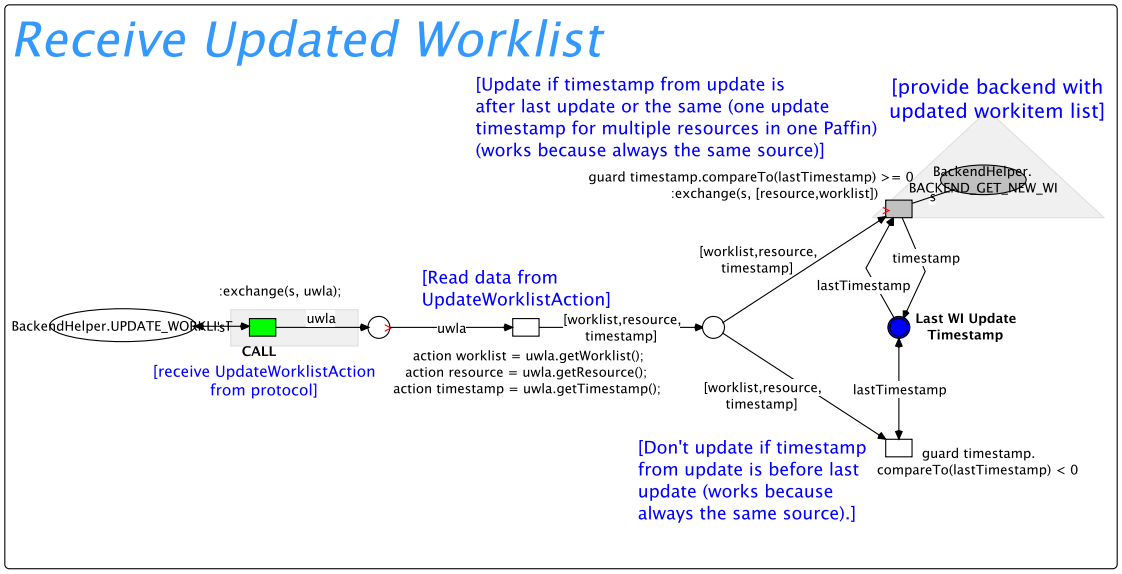

Figure 10.29: WFMS resource DC: Receive updated workitem list

responsible for receiving workitem lists is shown in Figure 10.29. Processing the new workitem list is kept deliberately simple. The workitem list, resource and timestamp are read from the incoming message. If the timestamp is newer than the last update timestamp (stored in the blue place), the workitem list is handed over to the backend resource net for the given resource. In the same firing step the last update timestamp is replaced with the timestamp of the current list. If the timestamp of the current list was older than the last update timestamp, which can happen in a distributed, non-deterministic Petri net system, the workitem list is discarded because more recent data is already present. The timestamp method is valid in this situation, since the source of the timestamps, i.e. the platform WFMS, is constant, which means it is not affected by distribution issues.

Activity lists handling: This net structure is responsible for receiving updated lists of active and confirmable activities and for forwarding them to the resources. The processes are analogous to the update of workitem lists. However, they are simpler since activities already contain all data (i.e. no additional database access and data retrieval) and each activity is only send to one resource.

Check operation commands: This net structure checks incoming workflow operation commands from a resource. To provide context, the following considers the full process of a workitem request command from a user. Coming from the GUI, the command arrives at the backend resource and is then handed over to the resource DC. The resource DC sends the command via process-protocol to the platform WFMS. There, it is received and handed over to the check operation command net structure. In multiple steps, that net structure checks the command:

1. Is the workitem still available?
2. Is the requesting resource registered?
3. Does the resource have the required permissions for this workitem?

If any of the checks fail, a failure message is returned to resource Paffin. If all of the workitem checks succeed, the workitem can be requested by the resource. Consequently,

the request is forwarded to the engine Paffin. There it is received by the engine DC with the net structure shown in Figure 10.30. Here, only one check is directly performed by the DC, namely if the message was forwarded by the known platform WFMS. This ensures that only the platform WFMS can forward workflow operations. If the platform WFMS sent the operation command, the command is handed over to the backend engine net, which attempts to perform the workitem request as described in the respective backend engine prototype section. The result of that attempted operation is handed back to the engine DC, which returns it to the platform WFMS, which recognises the result (see next net structure description) and finally forwards it to the original Paffin resource. If the operation was successful, the sets of workitems and activities change, causing updates that cascade throughout the system causing it to progress in its execution.

The previous described the process for workitem requests. For activity confirmations and cancellations, the process is analogous.

Determine engine: This net structure is responsible for determining the identifier of an engine given a workitem or an activity. It is required whenever a resource wishes to initiate a workflow operation and the platform WFMS needs to forward the initiation to the correct engine.

Recognise results of operations: The result recognition is the basic monitoring and logging mechanism for the platform WFMS. The mechanism works threefold. For one, it maintains a text file log in the current user's home directory. Here, everything that is handled in some way in the platform WFMS is logged, including all lists of workitems and activities and all operations. This persistent log is used for maintenance and analysis. The other two logging facets are not persistent and only used for runtime analysis. A log of all workflow operations is used to analyse issues resources might have with the operations (e.g. do the operations of a resource always fail). Another log tracks all finished and done activities of the system. This is used to analyse if the system's workflows proceed normally.

The result recognition also serves another, important function w.r.t. newly requested activities. If the platform WFMS DC recognises the successful request of a workitem, that result is specially routed. As described in the GUI prototype, each task has a specific GUI associated with it. That GUI is stored in the database as well. When a workitem is successfully requested, its GUI has to be retrieved from the DB. This happens when the request result is recognised by the platform WFMS. Here, instead of just recognising the result, the result is enriched with the additional GUI data before being forwarded to the resource Paffin, which then initiates displaying it in the GUI.

Database access for Capa agents was the topic of [Mosteller, 2016]. There, the `PersistentOntology` plugin for Capa was extended to provide the means for Capa agents to store and read any object defined in an agent ontology in a database. Due to the compatibility of Capa and the Paffin-System, this plugin could be directly utilised. The `PersistentOntology` allows agents (now Paffins) to utilise a provided persister interface. That interface connects to an existing database. By accessing that interface with special ontology objects, other ontology objects can be stored, read or modified from the database. Querying utilises an underspecified mechanism. This means that the database is queried with an only partially specified object and returns all stored objects that match the partial specifications. For example, the database could be queried for an empty resource object and return all resources or it could be queried for a role containing a specific username and return all roles in which that username occurs.

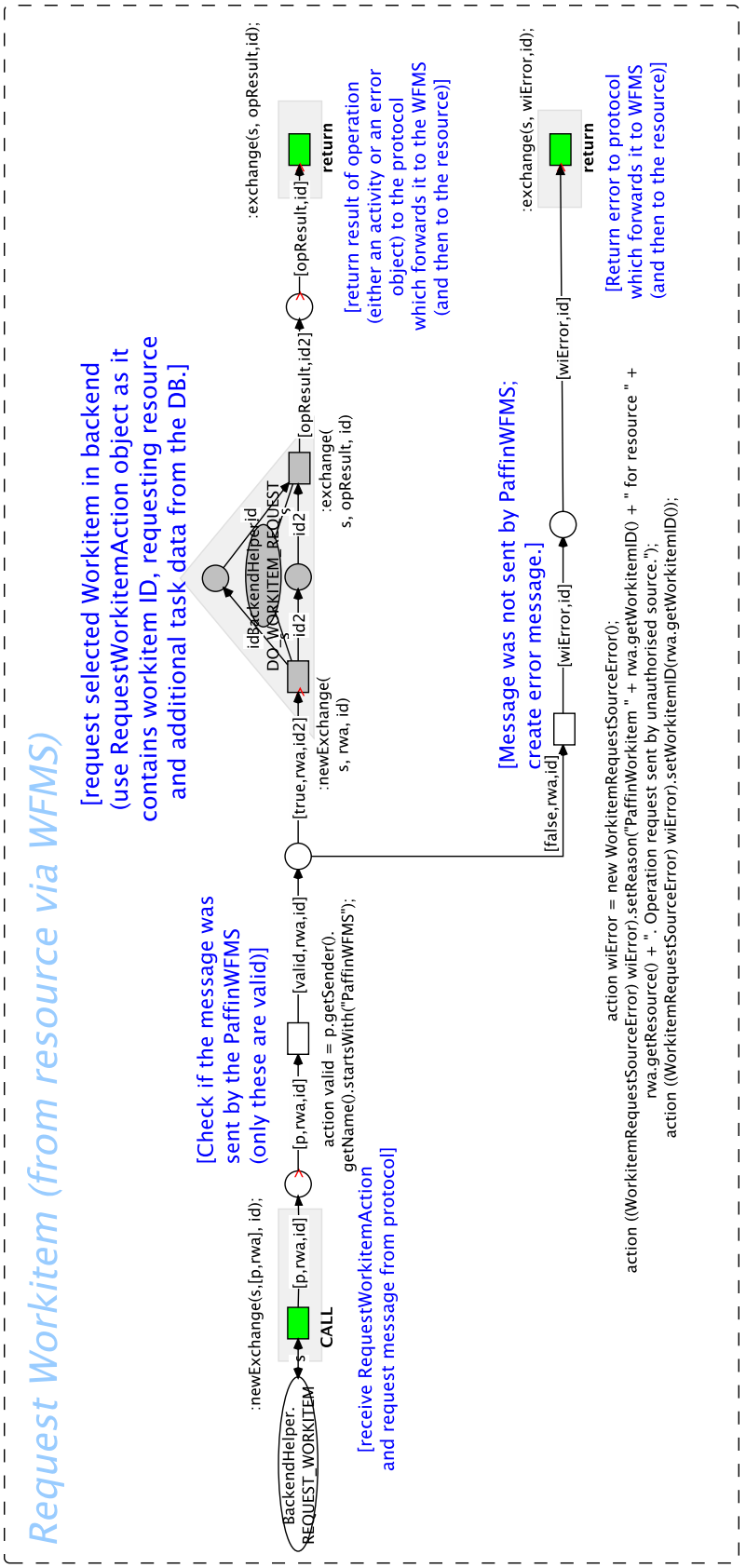

Figure 10.30: WFMS engine DC: Request Workitem

For the Paffin-System, only the platform WFMS Paffin utilises the persister interface. This interface connects to a db4o[15] database stored in the current user's home folder. Such a localised implementation is sufficient in this prototypical stage of the Paffin-System. As future work, the `PersistentOntology` allows extensions to other (distributed) database. That, however, is outside of the scope of this thesis.

Regarding the scope of database functionality, currently only read access is provided to the Paffin-System. The database can be accessed and modified via an auxiliary Java class that contains the initial values of the DB. That class is instantiated and compared against the existing database on every platform WFMS startup. If database entries are missing the database is restored to the new initial values. Future work deals with extending the capabilities of the database access, including enabling the Paffin-System to store and modify data at runtime.

As described above, the platform WFMS DC, in many places, accesses the database. That database access is realised through an additional DC, the persister DC, which has direct access to the persister interface provided by the `PersistentOntology`. In general, the process of querying the database is always similar. Representatively, Figure 10.31 shows the database access to determine the set of permissions for a given resource. Starting on the left, the persister DC is called by the platform WFMS DC. Here, the resource object representing the resource is provided. The name of the resource is read and packaged into an underspecified WFRole object, which, in turn, is packaged into a PersistableConcept object. That PersistableConcept object is handed over to the database interface, which takes over actually querying the database using the mechanisms defined in the `PersistentOntology`. The result of that querying call is either DBQuerySuccess object containing the found results or a DBQueryFailure object if the query failed. A failure is logged and returned to the platform WFMS DC as an error. A success object is further processed. It contains a sequence of all WFRole object matching the underspecified query. In the next step a helper method is called which iterates through the set of WFRoles and returns a set of all permissions contained in those roles.

The persister DC contains further net structures for querying resource credentials, workitem data and activity GUIs. These all work similarly to the example discussed above.

The most crucial, non-net implementation aspect of the platform WFMS prototype is the role-based access control system (RBAC) implemented for the Paffin-System. The system is defined in the ontology of the Paffin-System through a number of ontology concepts. These concepts are stored as Java objects in the database and read and compared at runtime.

Figure 10.32 shows an excerpt of the overall Paffin-System ontology with the RBAC concepts highlighted. The core concept is the `paffin-w-f-role`. That concept describes a role, which has an identifier, a set of assigned resources (identified by their login names) and a set of assigned permissions (identified by their names). To increase the flexibility of the system, each role can also contain a superrole identifier realising a hierarchic RBAC, where each role also has the permissions of all of its superroles. Besides the role concept, the system also features a `paffin-permission` base concept defining the name of the permission. All permission types inherit from that base concept. Currently, only the `positive-permission` concept is supported. A positive permission describes that a resource with this permission is allowed to do something. Using the base permission allows for future extensions with, e.g., negative permissions that prohibit resources from doing something.

[15] `https://sourceforge.net/projects/db4o/` (last accessed May 28th, 2017)

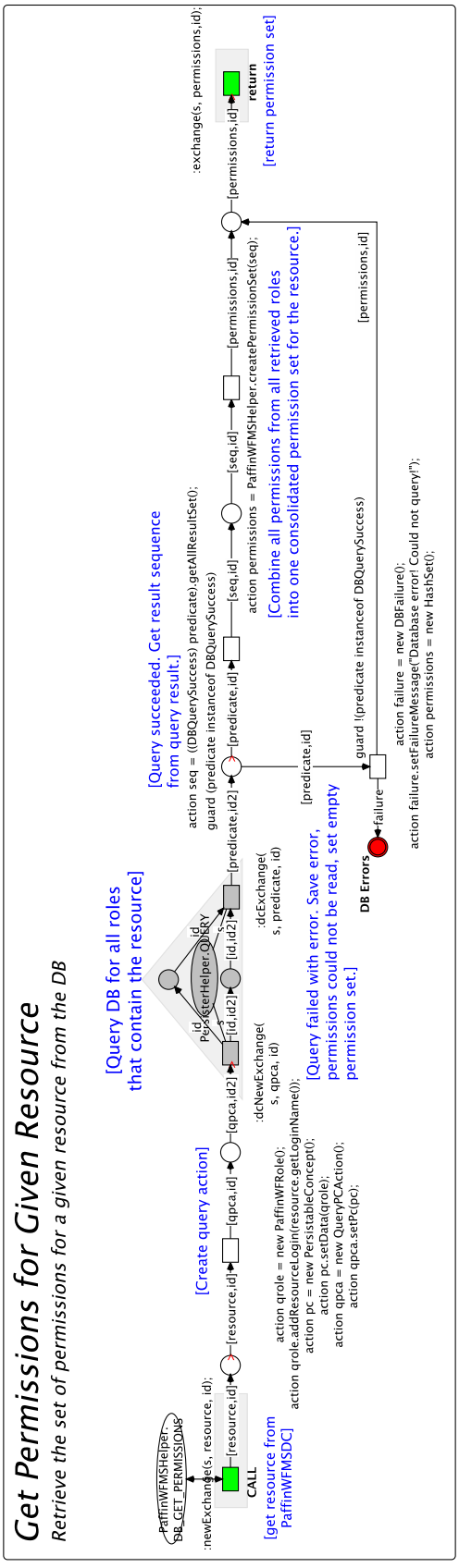

Figure 10.31: WFMS persistor DC: Resource permissions

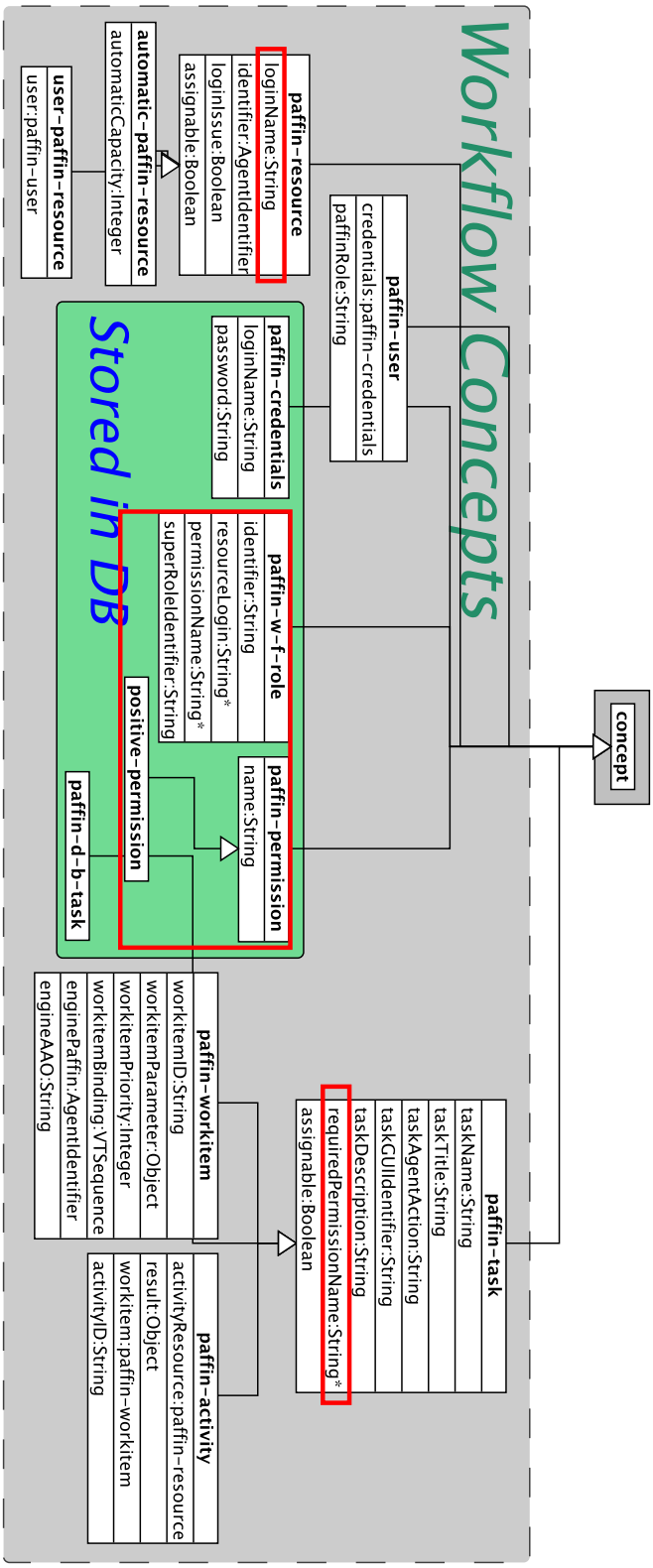

Figure 10.32: Paffin-System ontology excerpt: Role based access control

The overall RBAC works in the following way. When a resource registers with the platform WFMS, its permissions are calculated according to the roles it has been assigned to. Each task contains a set of required permissions. When a resource attempts an operation on a task (in workitem or activity state), the system only needs to check if the set of required permissions of the task is a subset of the permissions a resource possesses. If even one required permission is missing for the resource, it is not allowed to perform the desired operation. This rather simple RBAC system fully realises a necessary workflow security control mechanism for the Paffin-System.

Regarding the database, the platform WFMS prototype also defined the ontology objects for storage in the database. These can also be seen highlighted in Figure 10.32. In addition to the core RBAC concepts, which are all stored in the database, the `paffin-credentials` concept stores resource credentials and the `paffin-d-b-task` concept stores all task information. Not shown in Figure 10.32 is the concept for storing GUIs. That concept contains an identifier and a set of elements provided with a specific layout to be displayed in the GUI.

Due to the functionality limits currently found in the platform WFMS prototype, the initial database needs to be defined via a Java class that is instantiated and stored during startup of the platform WFMS. That class implements a specific database interface, which generates the desired objects from the given Java code.

Apart from the RBAC system and database concepts, the platform WFMS also features a large amount of auxiliary helper methods defined in Java. These help in processing the workitem and activity lists, checking for resource permissions etc. The extent of these helper methods is bigger than in the rest of the Paffin-System, because the platform WFMS involves more iterations over lists, which are cumbersome to handle and maintain in net implementations.

Conclusion The platform WFMS prototype completely fulfils the purpose laid out for it. It realises the connection between Paffins as workflow engines and Paffins as workflow resources. In terms of the WorkBroker prototype, it represents a fully-fledged intermediary system between engines and resources.

While the prototype fulfils its purpose there are a number of possible improvements and additional features. One of those, the active assignment of workitems, has already been implemented. Before, resources could only request workitems in a pull-mechanic. Now, the platform WFMS can assign workitems that are marked as assignable to resources that agree to take on all such workitems. The mechanism utilises the already present request mechanism. However, instead of the resource initiating the request, the platform WFMS itself does it. When the platform WFMS is informed of an assignable workitem and has resources available it attempts to automatically "request" the workitems on the resources behalf, thus realising a push-mechanic from the perspective of the resource. Other possible future extensions relate to the RBAC system, extensions to logging and monitoring and more advanced resource management.

One of the most obvious areas of improvement concerns the database. Currently, it is only possible to read data from an initial database (defined in a Java class) at platform startup. While this is sufficient for the current prototype and proof-of-concept stage of the Paffin-System, extending the database functionality has a high priority in future work. At the very least, runtime storage and modification of the already supported concepts needs to be implemented. Additionally, runtime data like workflow progress and results should also have the option to be stored in a database. Beyond that, providing the database non-locally, possibly using a cloud service, is a desired feature that is being considered.

10.3 Other Prototypes

The Paffin-System represents the main practical and technical result of this thesis. Other prototypes have also been developed and are discussed in this current section. While the technical results of these prototypes did not directly contribute to the Paffin-System, the results and lessons learnt nonetheless strongly influenced its implementation and thus should be discussed briefly.

10.3.1 Net prototype

The net prototype was the first, rudimentary realisation of a system following the Agent-Activity concept. It is independent of the technical context of Capa and does not feature any management mechanisms beyond the basic, technical backend. It is completely implemented with reference nets without any additional Java classes or methods.

The scope of the functionality was limited to instantiating a number of integrated entities that could execute the fundamental agent actions and basic workflow operations. The integrated entities were only represented by a net providing the base functionality of the technical backend, as well as interfaces to a system net. Within its limited scope the net prototype realised the basic Agent-Activity concept. Representatively, Figure 10.33 shows the technical backend net of the net prototype.

The net prototype served as an indicator and first attempt to investigate, if the envisioned, basic Agent-Activity approach could even be realised in reference net form. None of the functionality could be directly incorporated into the later Paffin-System. However, many ideas and basic principles could be reused in the Paffin-System. For example, the basic Petri net model of the Agent-Activity, described in Section 9.1.2, was developed and tested in the net prototype. The net prototype also defined the basic scope of the management facilities of the technical backend. Beyond that, some net structures could be technically enhanced and can be recognised. Take, for example, the base net structure of the triggering of Agent-Activities shown in Figure 10.13 and the net structure for triggering in the net prototype in Figure 10.33. The basic layout and process is quite similar.

As an indicator for the feasibility of an Agent-Activity realisation the net prototype served its purpose quite well. However, as soon as the feasibility was established, the prototype was largely abandoned due to its limited scope and restriction to pure reference nets. Work on the Paffin-System began soon afterwards with the net prototype serving as inspiration.

10.3.2 WorkBroker

The WorkBroker-system was developed in the context of the yearly AOSE (Agent-oriented software engineering) teaching project in the winter term of 2014/2015. It was developed by this thesis' author collaborating with and supervising the students Joanna Sieranski and Max Friedrich. Overall, the AOSE project aimed to realise a support environment for processes in the Paose development approach (see Section 2.2.3). The WorkBroker-system represents an independent subsystem, which allows agents to interact using workflow and task principles. The WorkBroker was presented at the Petri nets conference 2015 [Wagner and Moldt, 2015b][16].

[16] [Wagner and Moldt, 2015b] was revisited as an extended abstract at the EMISA 2016 workshop [Wagner and Moldt, 2016].

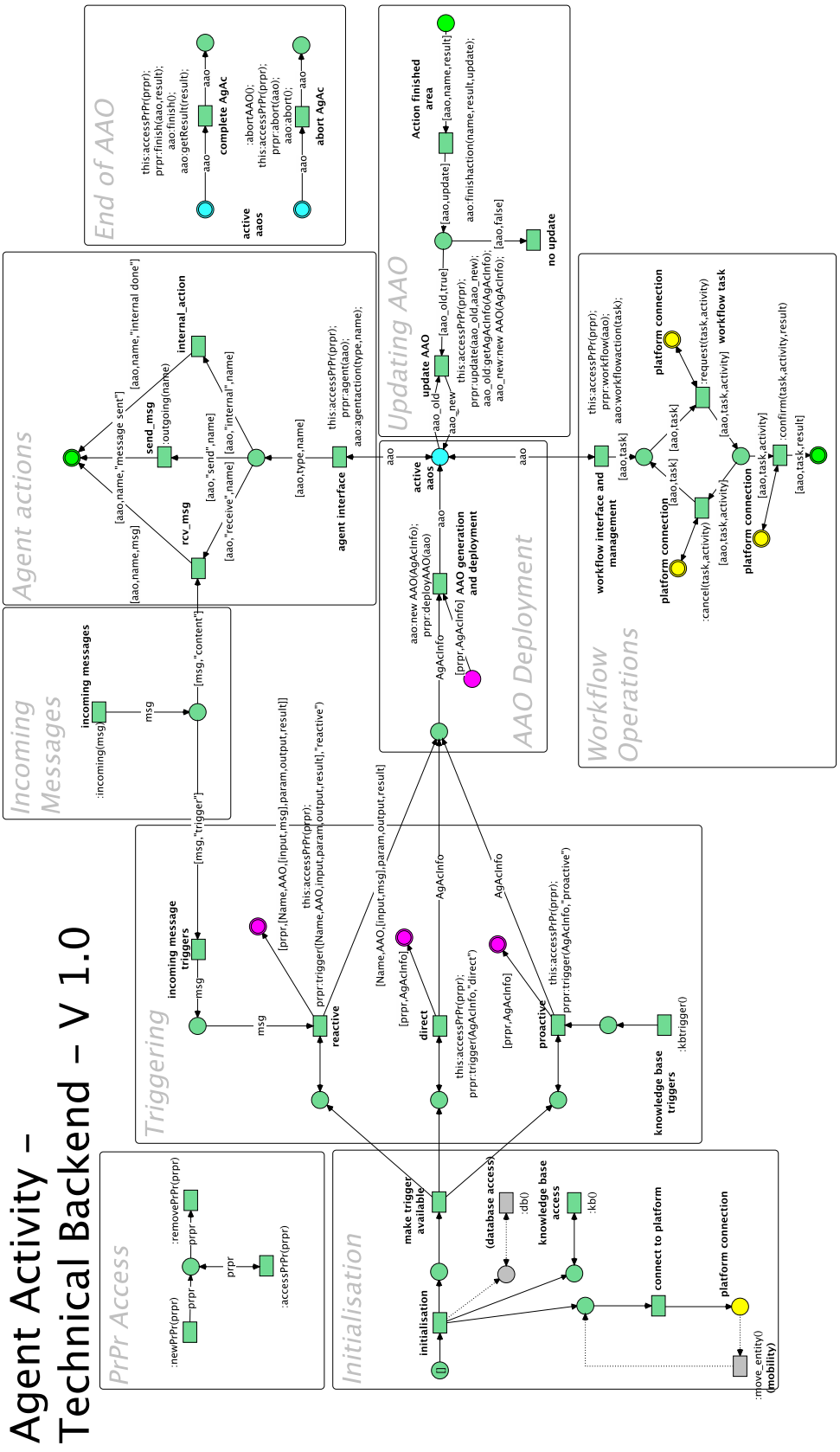

Figure 10.33: Net prototype: Technical backend

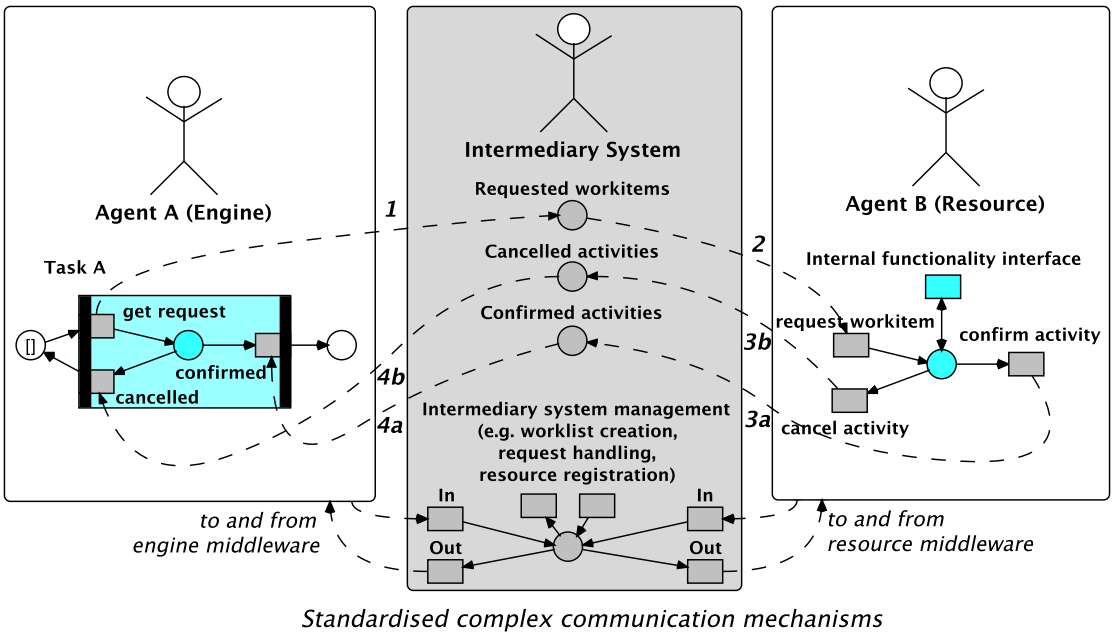

Figure 10.34: WorkBroker technical concept(from [Wagner and Moldt, 2015b, p. 11])

In short, the WorkBroker allows Capa agents to execute their behaviour as workflows containing workflow tasks. These agents are the engines to those workflows. Tasks in the workflows are executed by other agents, which serve as the resources. Communication and interaction between engines and resources is standardised and completely controlled by the so-called intermediary system. That system receives workitem and activity data from all engines and forwards that data to the authorised resources. Resources can then initiate (workflow) operations based on the workitem and activity data provided to them.

Key Term Definition C.13 (WorkBroker principle)**.** The WorkBroker principle allows agents to interact using workflow and task principles. Agents as engines execute behaviour containing workflow tasks that are intended to be executed by other agents as resources. The connection between agents as engines and agents as resources is realised through a special, platform-wide subsystem called the intermediate system.

Figure 10.34 illustrates the technical concept of the WorkBroker. Engines, on the left-hand side, execute workflows containing tasks in the style of the Renew task-transition. Data about available workitems and active activities is exchanged through standardised communication mechanisms, indicated in the lower part of the intermediary system. Workflow operations are pseudo-synchronised through asynchronous communication between engine, intermediary system and resource. After the workitem is requested (**1** and **2**), the resource agent accesses its internal functionality to execute the work/behaviour associated with the task. It can then either confirm the activity (**3a** and **4a**) or cancel it (**3b** and **4b**). The latter reinitialises the workitem in the engine.

An example of what an entire interaction between three agents can look like in the WorkBroker system can be seen in Figure 10.35. Clearly visible is the separation of agent behaviour as an engine and as a resource. The overall behaviour is separated into tasks, which are executed by some resource. Engines can also be resources for their own engine tasks. Hierarchic workflow behaviour is also supported, as agent B executes a full workflow while serving as a resource for task B1 for agent A.

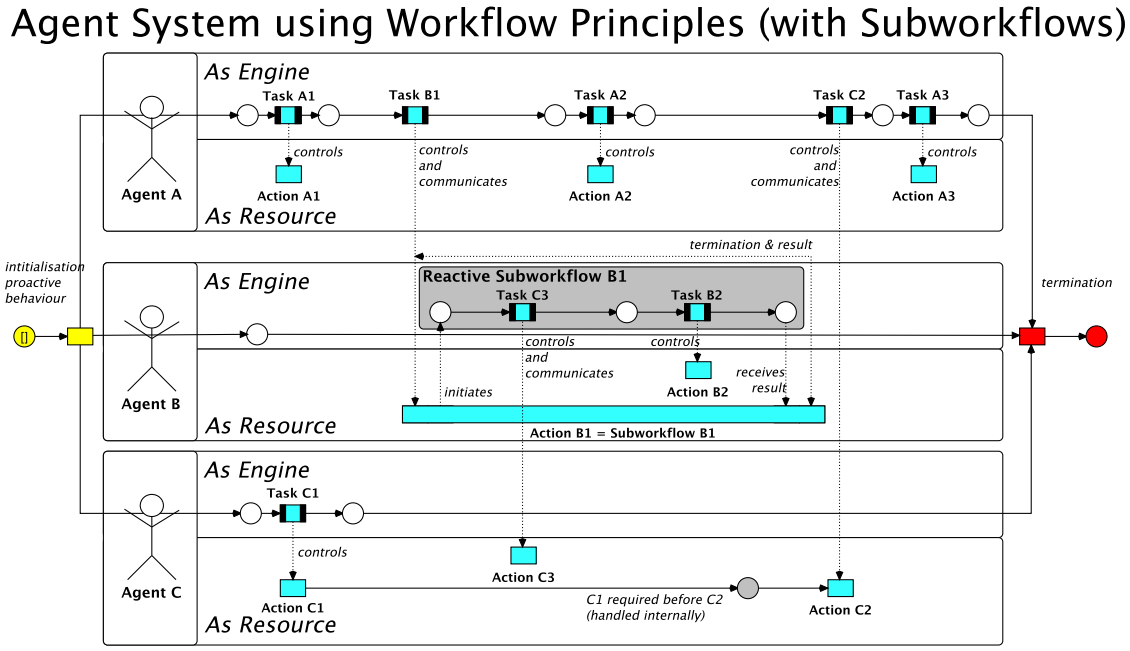

Figure 10.35: WorkBroker interaction principle
(from [Wagner and Moldt, 2015b, p. 17])

The prototype for the WorkBroker developed for the AOSE teaching project featured only limited functionality in resources and engines. Task choice and the interface to the internal functionality of the resource were severely limited to the form and function of the specific Paose support processes. Therefore, the WorkBroker prototype could not be directly incorporated into the later Paffin-System. However, the general principle of the WorkBroker system was adopted into the Paffin-System overall workflow management architecture (see Section 10.2.10 and especially Figure 10.17). In fact, the development of the platform WFMS of the Paffin-System used principles and lessons learnt from the WorkBroker system. Due to the specialisation and limitation of the WorkBroker prototype, the platform WFMS needed to be completely reimplemented from scratch, though. Still, the platform WFMS can be considered as a fully-fledged intermediary system as described in the WorkBroker approach.

10.3.3 PPB - Paose Process for Billboard

The Paose *Process for Billboard* (PPB) system was developed as an add-on project to the *Billboard* system of the teaching practical "Distributed Development of Distributed Software (DDDS)" in 2014. It was developed by this thesis' author collaborating with and supervising the students Christian Röeder and Dennis Schmitz. The overall Billboard system provided the functionality to create and post messages to a central billboard, from which other users could access, read and respond to those messages. The purpose of the PPB system was to extend that functionality of the Billboard w.r.t. defining and displaying processes. Through the PPB it is possible to allow users to define processes through textual input and have a graphical representation of that process be posted on the billboard.

The PPB system defined a basic syntax with which to describe processes. Processes in the PPB are basic Petri nets that consist of tasks represented as transitions connected through pre- and postcondition places. The syntax describes a process as a collection of an

arbitrary amount of four-element tuples. Each tuple contains a set of precondition places, a set of postcondition places, a task type (in preparation for possible later incorporation of the task-transition or other types of tasks) and set of task parameters. The parameters are dependent on the task type. For the prototypical bbtask (BillboardTask) the parameters were task name, task description and priority. The following shows an example of the textual representation of a PPB process.

```
#Process:Schedule Meeting;
(s*),(z1),bbtask,
      (setAgenda,Set an agenda date for the meeting,5);
(z1),(z2),bbtask,
       (proposeDate,Propose a date for the meeting,5);
(z2),(z1),bbtask,
       (dateRejected,The proposed date was rejected,1);
(z2),(z3),bbtask,
       (dateAccepted,All participants accept the date,1);
(z3),(z4),bbtask,
       (holdMeeting,Hold the meeting,60);
(z3),(z5),bbtask,
       (cancelMeeting,Cancel the meeting,1);
(z4),(z5),bbtask,
       (writeReport,Write a report about the meeting,1);
(z5),(z6),bbtask,
       (matterResolved,The matter has been resolved,5);
```

The PPB system receives this textual representation as input and transforms it. The process from the above listing is transformed into the net shown in Figure 10.36. The transformation first processes the textual representation and generates a process ontology object from it. That ontology object contains all the data of the process in a structured form. From that ontology object a Renew net drawing is generated by iterating over all tasks and adding or connecting corresponding transitions, places, arcs and transitions. Afterwards, a simple layout algorithm is applied to make the net readable. The resulting net drawing can be exported and shown on the billboard. However, it is also possible to instantiate that drawing as a net instance running in Renew.

The purpose of the PPB system in relation to this thesis was concerned with the internal process of an Agent-Activity. At that time, the author examined how to store and handle the description of the internal process and how to subsequently incorporate and instantiate it into the Agent-Activities. A text-based approach to process definition seemed feasible if a transformation from text to net (needed for the execution) could be implemented. This transformation was implemented in the PPB. However, usability of the PPB processes was too limited. With the developed syntax only simple processes could be designed. And even defining these simple processes as text was cumbersome. More complex processes would require a more complex syntax, which would make those processes almost impossible to properly model and maintain in textual form. The layout of the generated nets also proved to be cumbersome, impeding system monitoring and maintenance. As a consequence, the approach of textual process representation for Agent-Activities was abandoned in favour of the pure reference net modelling and representation used in the Paffin-System. Still, some of the lessons learned from generating nets in Java code could be applied to the Paffin-System, especially in regards to the translation between Agent-Activity-Transition and Agent-Activity net structure.

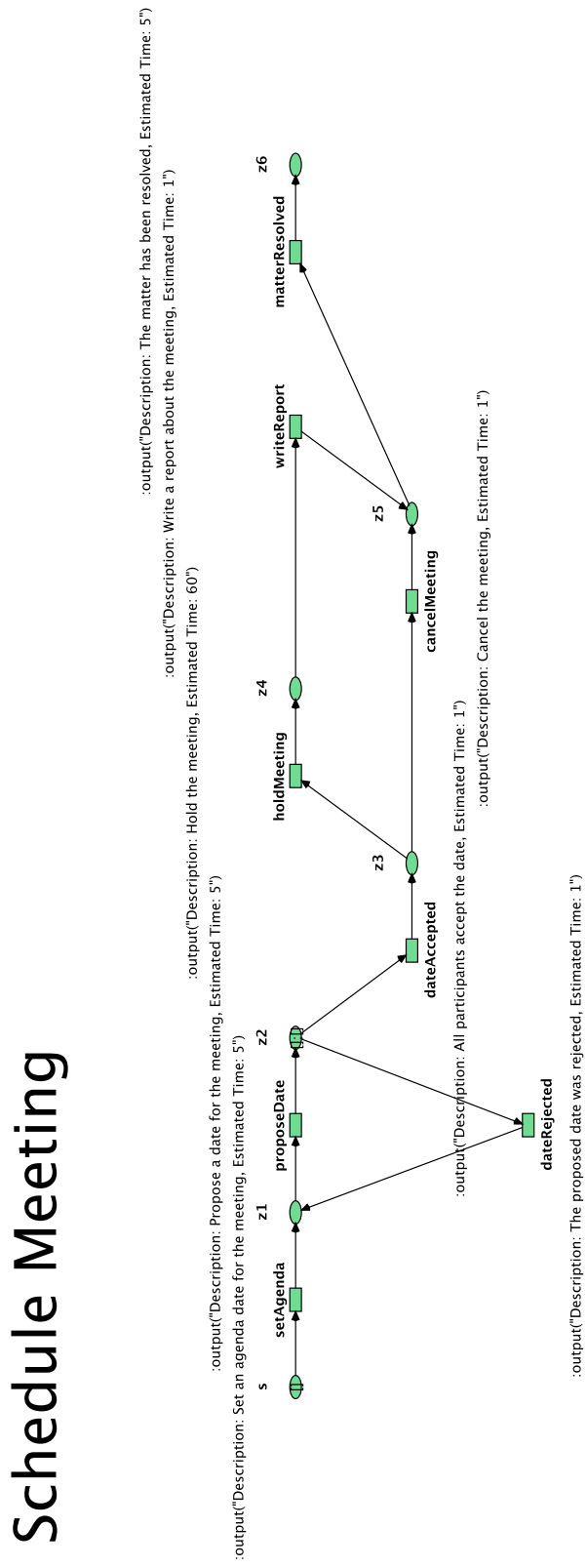

Figure 10.36: PPB: Example of generated process

10.4 Paffin-System Conclusion

This chapter described the prototypes created for this thesis, especially the Paffin-System. The Paffin-System is the technical proof-of-concept of the Agent-Activity approach and the main practical result of this thesis.

To conclude this chapter, this section discusses the prototype as a whole and begins to evaluate it on the technical level. For that purpose, Section 10.4.1 compares the conceptual Agent-Activity approach to what has been implemented in the Paffin-System. Evaluating the Paffin-System against the requirements of the conceptual approach validates it as the desired proof-of-concept. Afterwards, Section 10.4.2 examines the Paffin-System in regards to the previously established integration criteria. This explores whether the vision and specification of the integration, as laid out in Chapter 7, has been achieved as intended and desired. The difference between the two sections is that the first examines the Paffin-System when compared to the Agent-Activity approach, while the second examines the Paffin-System when compared to the general integration vision, thus further validating the assumptions made for the Agent-Activity approach.

Note that this conclusion section does *not* consider specific applications. Here, only the technical features and properties of the basic Paffin-System framework are taken into account. This provides a more technically-oriented evaluation of the Paffin-System, which is necessary as a foundation for later discussion chapters. An application based discussion and evaluation of the Paffin-System is provided in Chapter 11.

10.4.1 Comparison to the Agent-Activity Integration Approach

To validate the Paffin-System as a proof-of-concept for Agent-Activities, it has to be evaluated against the requirements of the Agent-Activity approach, laid out in Section 9.3. That way it is possible to determine if the Paffin-System fully implements the Agent-Activity approach.

The Agent-Activity *is the main modelling focus:* This requirement is fulfilled. Agent-Activities are available for modelling of process-protocols. While the design decision to support Capa legacy code does not enforce exclusive use of Agent-Activities, they are still the emphasised and focussed (i.e. main) element for modelling. Within each Agent-Activity, an AAO instantiates and executes an internal process consisting of fundamental agent actions and workflow operations. The process is implemented as a reference net supported by standardised net components. The order of the actions and operations is prescribed by the reference net semantics.

Agent-Activities *are executed by integrated entities:* Agent-Activities are combined into process-protocols, which are directly executed by Paffin entities. Paffin entities are integrated entities as defined in Definition B.7. They possess an agent interface inherited from the Capa framework and a workflow interface following the Work-Broker approach enabling Paffins as resources and engines. Both of these interfaces can be concurrently accessed and used. Consequently, Paffin entities can act as and be interacted with as agent, workflow, hybrid of both or something in between.

Dynamic state determination of the executing integrated entity: If a Paffin executes an Agent-Activity, it executes the abstract task defined through the agent actions and workflow operations of the internal process. As such, an Agent-Activity containing only agent actions means that the Paffin executes an abstract task of an agent. In that case, the Paffin is, for all intents and purposes, an agent. Analogous arguments can be

made for Agent-Activities containing only workflow operations and those containing mixtures of actions and operations. Concurrently executing Agent-Activities allow a Paffin to be both agent and workflow at the same time (full hybrid). Agent-Activities and the abstract task they represent end after all actions and operations have been executed. Beyond the end of an Agent-Activity it no longer affects the state of the executing Paffin. Only the currently active Agent-Activities affect the state. Altogether, this requirement is fulfilled as the state of the Paffin entity depends on the Agent-Activities it executes at any given time, meaning it is dynamic and fleeting as well.

Agent-Activities *are atomic management units:* Each Agent-Activity in a process-protocol corresponds to the Agent-Activity net structure described in Section 10.2.2. Even if the Agent-Activity is modelled as an Agent-Activity-Transition (see Section 10.2.4), that Agent-Activity-Transition is internally compiled by Renew into the net structure as a shadow net. Therefore, once an Agent-Activity is triggered the Agent-Activity net structure is entered in the process-protocol. There are only two exits to that net structure. Either the internal process and the Agent-Activity finish successfully, or they are aborted. An abortion resets the locality of the Agent-Activity by replacing the tokens removed by the triggering. This means that an Agent-Activity either happens (success-)fully, or not at all restoring the original state in its locality. This already fulfils this requirement in a basic sense. Beyond that, modelling the Agent-Activity as an Agent-Activity-Transition also emphasises the atomicity even more by abstracting from the net structure and life-cycle representation. With it, there is no feasible way to model an additional escape out of the Agent-Activity net structure, as it only exists in the hidden shadow net layer.

The Agent-Activity *life cycle is observed:* The life cycle of an Agent-Activity is defined through the Agent-Activity net structure described in Section 10.2.2. As described there, that net structure is equivalent to the basic Petri net model from Section 9.1.2 with two exceptions. Triggering and updating of the AAO are no longer instantaneous, but happen in two consecutive steps. While this adds two states to the life-cycle of the Agent-Activity, namely pending and updating, it does not violate the original life-cycle. A pending state can only be left by deploying the AAO leading to the active state and an updating state can only be left completing the update, also leading to an active state. This means that triggering-pending-deploy is equivalent to the original triggering and initiate-updating-complete is equivalent to the original updating. Consequently, the life cycle of the Agent-Activity is observed and extended to improve management capabilities.

Integrated entities are agents: A simplified way of considering Paffin entities is to see them as enriched Capa agents. This view fails to capture the significance and extent of the workflow and integration mechanisms incorporated into the Paffin entities, but on a technical level it is valid. The Paffin-System used the Capa framework as a technical starting point and still fully supports all Capa mechanisms and legacy code. By ignoring all workflow and integration aspects, a Paffin is indistinguishable from a Capa agent in its behaviour, capabilities, mechanisms and concepts. Since Capa agents follow the Mulan agent model, Paffins as pure agents also follow that model. Thus, the current requirement is fulfilled.

Integrated entities are workflow engines: The technical backend of the Paffin provides, in the backend engine net described in Section 10.2.10, all the local management

facilities needed for recognising, managing and reporting on workitems and activities in Agent-Activities the Paffin executes. This functionality qualifies the Paffin as the workflow engine for all the tasks described through workflow operations in its Agent-Activities. The engine functionality reports on workitems and activities to the platform WFMS, which distributes that data to the resources. Viewed from the platform WFMS and resources, the engine Paffin appears to be an aggregation of all of its tasks. That aggregation is a workflow filtering out all non-workflow related actions and covering all process-protocols. Consequently, the Paffin, as an engine, also represents itself as the workflow or workflows it is executing. This requirement is therefore fulfilled.

Integrated entities are workflow resources: This requirement is fulfilled. The backend resource net implements the functionality for Paffin entities to represent resources. It can register with the platform WFMS and subsequently receives workitem and activity data from it. It provides that information to the resources it represents and can initiate workflow operations on their behalf. Resources can be human users or the Paffin entity itself, which can control and manage other automatic resources, like devices, through process-protocol behaviour associated with a workflow task.

Local and global management: Local management is provided by the technical backend. Functionality is grouped into general Agent-Activity control and administration (see Section 10.2.6), agent functionality (see Section 10.2.8), workflow engine functionality (see Section 10.2.10) and workflow resource functionality (see Section 10.2.11). Each functionality group is implemented in a reference net. Those backend nets utilise the Paffin interface, i.e. the adopted Capa agent interface, to communicate with the global management facilities in the platform and management level. Paffins themselves are managed directly by the Paffin platform net. Agent-related management is provided by standardised AMS and DF Paffin entities, also adopted directly from Capa. Workflow-related management is provided by the platform WFMS Paffin, which implements an intermediary system as described in the WorkBroker concept. AMS, DF and platform WFMS Paffins are automatically started and by default available and known to all application Paffins of a platform. Together, these components handle all agent and workflow related, global, i.e. platform spanning and beyond, management issues of the Paffin-System. This separation and implementation of local and global management facilities fulfils this requirement.

The reference architecture is distinctly realised: The four levels of the reference architecture, as well as the relationships between them, are distinctly realised in the Paffin-System. Agent-Activities (or rather Agent-Activity-Transitions) are combined into process-protocols, implementing the first level of the reference architecture of the same name. The process-protocols are instantiated by the process-protocol factory inside the Paffin entity net representing a Paffin entity. Process-protocols are then executed and finally terminated fully within the Paffins. Paffins implement the second level of the reference architecture, the entity level. The technical backend within each Paffin also provides the entirety of the (local) management facilities for all Agent-Activities and the agent actions and workflow operations contained within them. The runtime environment for Paffins is provided by the Paffin platform, implementing the third level of the reference architecture, the platform and management level. Paffins are started and terminated on a Paffin platform and also communicate using the infrastructure provided by it. The Paffin platform also provides the (global) management facilities for Paffins in the form of standardised

Paffin management entities (AMS, DF, platform WFMS). Finally, the implicit system level is an aggregation of all Paffin platforms, the Paffins executed within them and the process-protocols executed by them. In summary, the Paffin-System fully complies with the reference architecture.

As the Paffin-System fully realises all the requirements set out by the Agent-Activity approach, it is confirmed as a proof-of-concept of that approach.

10.4.2 Integration Criteria in the Paffin-System

The previous section showed that the Paffin-System is in fact a technical proof-of-concept of the Agent-Activity approach. This current section now evaluates the Paffin-System against the original integration criteria laid out in Chapter 7. The goal of this examination is twofold. First, it examines if the actually, technically implemented system in fact realises an integration of agents and workflows as envisioned for this thesis. By doing so, this asserts that the specification and vision of an integration can actually be translated into a technical system. Second, it compares the evaluations of the criteria between the conceptual approach and the technical implementation. This showcases any areas, in which the implementation evaluates better or worse than the concepts due conceptual ambiguities eliminated by the implementation.

Criterion MC1: Integrated Entities: The discussion about the requirements of the Agent-Activity approach already established that Paffin entities are integrated entities according to Definition B.7. Even though the Agent-Activities represent the clear majority and emphasis of the modelling effort and focus, they "only" describe what happens within the greater context of each Paffin entity. Paffins as a whole, including their knowledge, DCs and technical/management facilities, are the core of the Paffin-System. Only by considering them fully, with all their capabilities and not just as their behaviour, is it possible to fully utilise modelling in the Paffin-System. Consequently, the Paffin integrated entities are the main modelling construct of the Paffin-System. In this criterion there are no changes to the conceptual Agent-Activity approach.

Criterion MC2: Entity Dynamic: As was established in for the requirements of the Agent-Activity approach, the technical Agent-Activities determine the dynamic state of their Paffin entities. Therefore, this criterion is completely fulfilled. Again, the evaluation of this criterion does not change between the concept and the implementation.

Criterion MC3: Logical Entities: The Paffin net itself provides only the basic framework that can be filled with any desired functionality. Therefore, it can be considered as a blank container, which can be filled with anything the system modeller desires. Paffins can be enriched to be as complex as conceivable and possible with net modelling. They can also be kept very simple. In fact, it is even possible to have a Paffin be and represent a simple data type. In order to implement this, the Paffin would simply return its data, like a number or a string, on each message it receives. Whether this is a reasonable design decision for an application is doubtful, but it emphasises that every element of the system can be considered as an integrated entity. The evaluation of this criterion does not change between the concept and the implementation.

Criterion SB1: Structure of the System: The structure of the system consists of the Paffin entities and the Paffin platforms. As discussed for the requirements of the Agent-Activity approach, Paffins are agents, since they follow the Mulan/Capa agent model and satisfy that agent interface. The Paffin platform has a specialised

form and function, but satisfies the same interface and agent model due to the Capa basis. Consequently, this criterion is completely fulfilled. The evaluation of this criterion does not change between the concept and the implementation.

Criterion SB2: Behaviour of the System: The behaviour of each Paffin is defined through the Agent-Activities it executes in its process-protocols. On an abstract level, these Agent-Activities as abstract tasks prescribe an abstract workflow that is being executed across the Paffin entity. That workflow describes the abstract behaviour of the Paffin, therefore fulfilling this criterion.

However, the design decision to support Capa legacy code slightly diminishes this. As Capa code outside of Agent-Activities is supported, behaviour outside of the abstract workflow defined through the Agent-Activities is possible. This is a slight conceptual limitation related to this criterion. However, the practical gain of enabling Capa legacy code outweighs this limitation. Since it is not enforced nor endorsed it does not violate the integration criteria, but rather circumvents it in certain situations. The support of Capa legacy code also explains the change in evaluation when compared to the conceptual approach.

Criterion SB3: Mutual Incorporation I: Interactions between agents can be modelled through workflow tasks. It supports agent interactions following the WorkBroker principle. A Paffin as an engine can provide a task for a Paffin as an automated resource. That resource Paffin then executes a process-protocol associated with that task and returns the result. By nesting tasks in the latter process-protocols, complex interactions between groups of Paffins are also supported. Since regular agent interactions are in no way prohibited, though, this criterion remains slightly limited. There is no change to the conceptual approach.

Criterion SB4: Mutual Incorporation II: As discussed for the requirements of the Agent-Activity approach, Paffins are workflow engines executing workflows. Since, also as discussed for the requirements, Paffins are also agents at any time they consequently are agents that execute workflows thus fulfilling this criterion. The evaluation of this criterion does not change between the concept and the implementation.

Criterion ToP1: Properties and Mechanisms: As discussed for the requirements, Paffins have the full capabilities and mechanisms of agents, workflow engines and workflow resources. Agent actions and workflow operations can therefore access those capabilities fully. When actions and workflows are combined into the internal processes of Agent-Activities the transfer of properties can happen. As there is no restriction to the form of the internal processes, transfer of all properties is wholly supported. In this criterion there are no changes to the conceptual Agent-Activity approach.

Criterion ToP2: Agent Actions and Workflow Operations: Agent-Activities consist of an internal process of agent actions and workflow operations. There is no restriction on these actions and operations, thus fulfilling this criterion directly without change when compared to the conceptual approach.

Criterion ToP3: Regular Multi-Agent Systems: By exclusively using Agent-Activities containing only agent actions, regular multi-agent systems can be modelled with the Paffin-System. Even more so, the support of Capa legacy code means that, potentially, the Agent-Activity mechanism can be ignored to build pure agent systems indistinguishable from Capa multi-agent systems. Conceptually, this only improves the evaluation of this criterion marginally when compared to the conceptual approach. Practically, the support of Capa legacy code enables years worth of plugins

and agent functionality to be available to the Paffin-System, which is why this design decision was made in the first place.

Criterion ToP4: Regular Workflow Systems: By exclusively using Agent-Activities containing only workflow operations, regular workflow systems can be modelled with the Paffin-System. The evaluation of this criterion does not change when compared to the conceptual Agent-Activity approach.

Criterion Mgt1: Functionality: Agent-Activities largely[17] define the functionality of the Paffin executing them. Therefore, the functionality is distributed amongst the Paffin entities in the system. Since the Agent-Activities also dynamically define the state of each Paffin, the functionality is likewise distributed amongst the different states. The criterion is fulfilled, with no changes when compared to the conceptual approach.

Criterion Mgt2: Hierarchies: Relationships between Paffins can be quite complex. A Paffin may be resource, engine and agent in different and shared contexts with other Paffins. Restrictions do not apply here. Logical hierarchies can therefore be found throughout the system and throughout all states of Paffins, depending on what an application prescribes. By supporting the boundary crossing of logical hierarchies between Paffins and states, this criterion is fulfilled with no changes when compared to the conceptual approach.

Criterion Pra: Practicability: A practical evaluation regarding feasibility of the Paffin-System is no longer applicable, since it has already been implemented. The usability is evaluated in the context of applications in the following chapter.

Table 10.1 summarise the evaluation of the Paffin-System w.r.t. the integration criteria. The evaluation is overall very good. Only Criterion SB2 and Criterion SB3 are slightly limited. However, as discussed, these limitations do not violate the integration vision. They are not enforced meaning that they represent optional ways of circumventing conceptual limitations to gain practical benefits, like Capa legacy support and the flexibility of freely choosing the interaction type.

The evaluation of the Paffin-System establishes that it provides a practical framework for the specification and vision of an integration desired in this thesis. The next steps are to evaluate and discuss the integration further with a more practically-oriented basis that is now available due to the Paffin-System. Evaluation will be based on general discussions, applications, modelling capabilities, synergies and comparisons to related work. All of these are presented in the next chapters.

[17] The remainder of functionality is provided through the DCs. As completely internal components, DCs do not affect this criterion and can be ignored for this discussion.

Criterion	Evaluation	Comment
Criterion MC1	full	
Criterion MC2	full	
Criterion MC3	full	
Criterion SB1	full	
Criterion SB2	slightly limited	Capa legacy code circumvents this criterion. Agent code outside of Agent-Activities otherwise not endorsed.
Criterion SB3	slightly limited	Possible, yet not enforced.
Criterion SB4	full	
Criterion ToP1	full	
Criterion ToP2	full	
Criterion ToP3	full	Support of Capa legacy code represents a slight conceptual, yet practically large, improvement over the Agent-Activity approach.
Criterion ToP4	full	
Criterion Mgt1	full	
Criterion Mgt2	full	
Criterion Pra	N/A	

Table 10.1: Evaluation overview for the Paffin-System

Part D

Application, Discussion and Evaluation

Part D discusses and evaluates the results produced in this thesis. Chapter 11 emphasises specific applications in the Paffin-System. This substantiates the practical usability and maturity of the Paffin-System, while also showcasing application prototypes and application visions in which the extended capabilities provided by the Agent-Activity and the Paffin-System can be utilised advantageously. Applications are discussed before the overarching and general discussion of the results in Chapter 12. The general discussion deals with aspects such as the general applicability of the results, the synergies, strengths and open points, the application area properties, future work directions etc. These discussions require an advanced understanding of the capabilities of the Agent-Activity and the Paffin-System which can only be established by examining specific application aspects first. Finally in this part, Chapter 13 evaluates the results of this thesis w.r.t. previous and related work. Here, the Agent-Activity as a modelling construct and the Paffin-System as its implementation are compared to existing and established research.

The purpose of this part of the thesis is to critically examine the conceptual and practical results presented in the previous part, as well as establish a better understanding of the newly established capabilities. By observing, examining and discussing the various facets of the results, their ultimate research contribution becomes clear. The discussions and evaluations of this part are also used to answer the research questions and whether the goal and result requirements have been achieved.

11 Application Discussion

The previous part of the thesis described the conceptual approach of the Agent-Activity and its proof-of-concept implementation in the Paffin-System. To start this discussion and evaluation part of the thesis, this chapter considers the context of practical applications. While small-scale use case scenarios for Agent-Activities have already been discussed, this chapter now examines practically implemented prototype applications. This represents a direct continuation from the previous chapter of framework prototypes, which is also why the application discussion has been brought to the front of the evaluation part.

The overall goal of this chapter is to discuss and evaluate the practical application aspects of the results of this thesis. Section 11.1 discusses how to model with the Paffin-System. The focus of this section is not in the methodology, but rather in the modelling artefacts that need to be modified to create applications. Afterwards, Section 11.2 details the overall maturity of the Paffin-System. This section provides an overview of what techniques, mechanisms and concepts modellers can utilise in the Paffin-System, as well as its current, technical limitations. Then, the main part of this chapter, Section 11.3, presents and discusses a number of larger application prototype examples built with the Paffin-System. Following this, Section 11.4 presents a number of application visions that have been envisioned for the Paffin-System, but have not yet been implemented. Next, Section 11.5 discusses inter-organisational contexts as an application area for the Paffin-System. Finally, Section 11.6 concludes this chapter with an overall discussion of the application aspects.

Please note that this chapter does *not* feature a general discussion and evaluation of application areas well suited for Agent-Activities and the Paffin-System. Such a discussion is provided later on in Section 12.5, as it requires the general discussion about strengths and open issues of the Paffin-System (in Section 12.4), which in turn requires the application descriptions provided in this current chapter.

11.1 Application Modelling in the Paffin-System

The Paffin-System presented in Chapter 10 represents a framework for the creation of applications. The first discussion step in this chapter is to clarify *how* to model applications in the Paffin-System.

Note that this section does not discuss modelling methodology. This section examines which modelling artefacts modellers need to provide or adapt in the Paffin-System in order to create an application. It basically distinguishes the Paffin-System framework from the Paffin applications discussed later in this chapter. Methodology, especially relating to the Paose approach, is discussed as future work in Section 12.6.3.

Overall, the artefacts and components of the Paffin-System that need to be provided for each application are shown in the architecture context in Figure 11.1. Modelling effort for each component can be classified into three groups indicated in Figure 11.1 by different colours.

Global modelling (configuration): These components are part of the framework and indicated by the colour white in Figure 11.1. In general, they are never directly adapted for

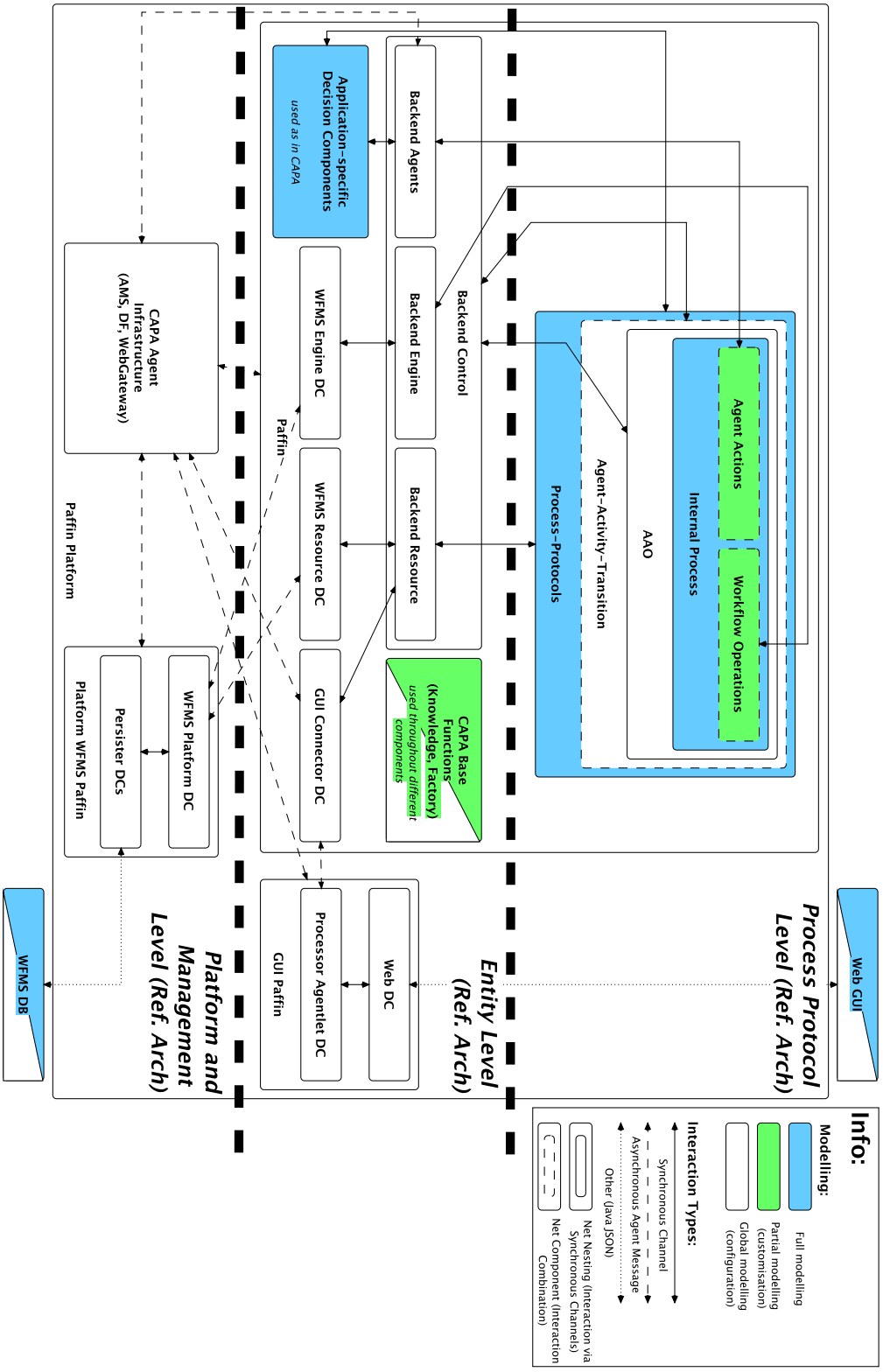

Figure 11.1: Paffin-System architecture modelling overview (modified from [Wagner et al., 2016b]

an application. They only need to be configured, i.e. provided with deployment and startup information relating to global, w.r.t. the application, technical issues. Examples of such configuration data are the specific PAFFINS to be started for an application and on which platform and management elements these PAFFINS are started.

Partial modelling (customisation): These components, indicated by a green colour in Figure 11.1, are fixed and preconstructed parts of the framework, but need to be customised and filled with application data for each application. This means that the basic frame of these components is predefined, but the content within them needs to be tailored to an application.

Full modelling: These components are indicated by a blue colouring in Figure 11.1 and need to be fully modelled for an application. They are not part of the framework at all, but are fully specialised for applications. Reusable patterns, like predefined net components (see Section 10.2.3), may support the modelling of these components.

Some components in Figure 11.1 are marked both white and either green or blue. This indicates that this component has both framework and application-specific parts. The framework parts usually provide the basic mechanisms and containers, while the application-specific parts contain content relevant to the application.

The global modelling components are largely irrelevant for modelling an application, because they are only configured and used. The remainder of this section discusses the remaining partial and full modelling components individually:

Internal processes: What happens within the AGENT-ACTIVITIES of an application is described in the internal processes. In fact, they can be regarded *as* the AGENT-ACTIVITIES from an application standpoint, as they are the only application-specific part of the AGENT-ACTIVITY that is provided for an application. The AAO and the AGENT-ACTIVITY-TRANSITION are both standardised parts of the framework.

Clearly, the internal processes of the AGENT-ACTIVITIES of an application are completely dependent on that application and thus need to be fully modelled. However, the internal processes consist of agent actions and workflow operations. These are modelled in the form of net components (see Section 10.2.3). Net components are a modelling tool and semi-standardised for the PAFFIN-System. They represent templates and patterns for agent actions and workflow operations that need to be customised after being added to an internal process. Therefore, the agent actions and workflow operations are part of the partial modelling components, while the internal process itself is part of the full modelling components.

Process-protocols: Process-protocols combine AGENT-ACTIVITIES into larger, connected behaviours of PAFFINS. As the behaviour is fully dependent on the application, the process-protocols are also part of the fully modelled components for an application.

PAFFIN *knowledge:* PAFFIN knowledge is stored in the knowledge base of each PAFFIN entity. It is part of the CAPA Base functions in Figure 11.1. The knowledge base net itself and parts of the knowledge (e.g. platform service locations) are standardised or automatically generated parts of the framework. However, the application-specific knowledge needs to be modelled for each application. It needs to be provided into the frame of the knowledge base, which is why it is classified as a partially modelled component.

Application-specific DCs: Application-specific DCs (as opposed to DCs belonging to the framework, like the WFMS Engine DC) describe completely internal behaviour of a

PAFFIN. As with the other behaviour artefacts, this is completely dependent on the application and must therefore be fully modelled.

Database content (including application-specific Web GUI): The database content contains all the persistent workflow information for the PAFFIN-System. While the basic frame of the database, as well as some default data entries, is predefined and part of the framework, the data for an application needs to be specifically and fully modelled. This includes the data for the workflow tasks, roles and resources.

Task-specific GUIs are also stored in the database. These are the parts of the GUI which need to be specifically modelled for each application. The overall Web GUI, i.e. the workitem and activity lists, login and basic layout, are not stored in the database, but are part of the framework. To indicate the separation of framework and application GUI, the Web GUI component in Figure 11.1 is marked in both white and blue, even though the full modelling parts are actually stored in the database.

In summary, to create a PAFFIN application a modeller needs to fully model the entire behaviour of PAFFIN entities, including individual AGENT-ACTIVITIES, process-protocols and DCs. Modellers also need to provide the knowledge base content for each PAFFIN. Furthermore, the supporting workflow data including tasks, roles, resources and GUIs needs to be fully modelled. In addition to these PAFFIN-specific system elements, modellers also need to provide any functionality that is *used* by the PAFFIN application. This includes custom Java classes, specialised user interfaces, external programs, etc.

The PAFFIN-System uses the established RENEW plugin mechanism. An application is either built in an existing plugin or a new plugin is created for it. These plugins should be located in the PAFFIN subproject of RENEW, but through global net and class path variables there is no technical prescription for this. All modelling artefacts of an application are stored in the plugin folders, including any custom Java or other non-PAFFIN-specific code. The suggested structure of the plugin folders, e.g. a source folder containing the sources ordered through roles and interactions, is adopted for the PAFFIN-System.

PAFFIN behaviour artefacts are nets. These nets are modelled directly in RENEW, being supported by mechanisms like net components. They are saved in the standard RENEW file format *.rnw* and saved somewhere in the plugin folder. As long as the location of the nets is included in the RENEW net path variable the net can be built along with the plugin and executed.

As PAFFIN knowledge is compatible to CAPA knowledge, the PAFFIN knowledge is modelled in the standard CAPA ARM (Agent Role Model) tool [Mosteller, 2010]. That tool allows for the description and automatic generation of knowledge base contents for different agent (here PAFFIN) roles. Knowledge bases generated in an ARM are automatically saved into the plugin folders so that they are directly usable for execution.

Finally, workflow data including GUIs are Java objects. They are modelled externally from RENEW in any Java or basic text editor. For this thesis the Eclipse[1] IDE (Integrated Development Environment) was used. All objects in the database are, in the current PAFFIN-System prototype, modelled and saved in database classes. These classes are provided to a PAFFIN-System instance during startup via parameter.

In conclusion, the modelling artefacts necessary for creating an application in the PAFFIN-System are relatively straightforward. When compared to regular agent modelling in CAPA, the only real difference is the addition of workflow data in the database. Note that this similarity only extends to the *kinds* of artefacts. The content of those artefacts,

[1] `https://eclipse.org/ide/` (last accessed May 28th, 2017)

of course, differs greatly as not only agent functionality can be created, but also workflow and hybrid functionality. One difference between Capa artefacts and the Paffin-System artefacts is that the behaviour for Paffins contains additional abstraction levels introduced through the Agent-Activities. Still, the general modelling efforts is similar. Knowledge modelling is virtually the same, while GUI modelling is even improved, as a default, basic GUI framework is provided for Paffins, while Capa (without the WebGateway extension) provides no such thing. The current way of database modelling and handling exhibits, as discussed before, potential for improvement. Here, future work and tool support can improve modelling highly.

11.2 Paffin-System Maturity

The Paffin-System fulfils its intention as a proof-of-concept, proving that an integration approach based on agent actions and workflow operations can indeed be realised in the proposed way. With previous descriptions focussing on the implementation, the concretely implemented modelling mechanisms and tools provided by the Paffin-System were not heavily emphasised. Likewise, technical limitations were only described along with the technical details. Both of these areas are discussed in this section. Section 11.2.1 describes the feature set of the Paffin-System, while Section 11.2.2 describes its technical limitations.

11.2.1 Paffin-System Feature Set

In general, the Paffin-System allows for the creation of agent systems, workflow systems and any hybrid combination of the two. Agent actions and workflow operations can be freely mixed and nested into one another, in order to utilise and exploit the different mechanisms provided by the traditional concepts in novel ways. General modelling is discussed in the context of application scenarios and prototypes in the following sections. Here, the discussion is focussed on the specifically implemented features of the Paffin-System, which enhance and extend the basic modelling options provided by the base capabilities of the Agent-Activity concept. In a way, the following list contains all the additional and auxiliary features that improve upon the basic Agent-Activity concept in one way or another and go beyond the basic proof-of-concept.

Note that the following list does **not** contain features provided by Renew, like the basic capability to model reference nets, or by Capa, like the WebGateway. Only features are listed that were actually and explicitly implemented for and/or incorporated into the Paffin-System. The list of features does not follow any particular order.

"Push" workitem assignment: It is possible for the platform WFMS to assign workitems. In contrast to the regular request workitem, which constitutes a"pull" mechanism, the assignment "pushes" the workitem to the resource. Assignment doesn't violate the workflow operation of requesting a workitem. It is conceptually the same operation with the difference being the initiator. Regular workitem requests are initiated by the resource. Workitem assignment simply allows initiation by some other element of the system (here, the platform WFMS). Workitem assignment in the Paffin-System is implemented in two variants. The first is based on the workitems, i.e. the workitem is assigned automatically to an eligible resource, and the second one is based on resources, i.e. the resource is automatically assigned to all workitems it is eligible to.

AAO Reset: Updating an Agent-Activity during its execution is included in the Petri net model of the Agent-Activity approach. How and in what form that update

happens, however, is not specified. The PAFFIN-System currently only implements the simplest AGENT-ACTIVITY update mechanism, the AAO reset. By initiating an AAO reset, a new instance of the AAO is created with the same parameters and internal process as before. This can be useful when external parameters, e.g. knowledge base entries, have been changed and a calculation should be restarted.

AGENT-ACTIVITY *Inscription Checking:* When implementing the inscription of an AGENT-ACTIVITY-TRANSITION, great care was taken to ensure that only valid inscriptions were possible. The compiler used for AGENT-ACTIVITY-TRANSITIONS checks the transition at modelling and at compile time and creates an error if the inscription is not valid.

Variables in AGENT-ACTIVITY-TRANSITION *inscriptions:* The inscription of an AGENT-ACTIVITY can contain variables in all parts of the tuple. For results a variable is even prescribed, while for parameters it is most often used to bind input tokens. Besides that, the identifiers of the AGENT-ACTIVITY name and the AAO can also be given as variables, bound to input tokens at firing time. This allows for a certain degree of flexibility in the execution of process-protocols and AGENT-ACTIVITIES. By using a variable for name and AAO, it is possible to make the execution of an AGENT-ACTIVITY dependent on previously achieved results. For example, one AGENT-ACTIVITY may model a customer choosing a payment method for his or her order. The result of that AGENT-ACTIVITY would be used as the AAO variable in the following AGENT-ACTIVITY implementing the actual payment. The AGENT-ACTIVITY would still model payment, but each AAO variant would model the internal process according to the slight differences in each payment method. Exchanging the whole AGENT-ACTIVITY by keeping the main identifier variable would lead to completely generic process-protocols. If the result of one AGENT-ACTIVITY determined the following one, a workflow user could create a completely adaptive overall process from a modular construction kit. Each component would feature the application content and a second element choosing the following step. In that way, adaptivity can be simulated in the PAFFIN-System without resorting to flexible Petri net mechanisms.

PAFFIN *Web GUI:* A GUI is important for any application involving human users. It is, however, not an essential part of the AGENT-ACTIVITY approach. The AGENT-ACTIVITY just assumes that the resources can somehow operate the workflow operations. How this is done is irrelevant to the conceptual approach. Therefore, the PAFFIN Web GUI described in Section 10.2.12 is considered as one of the additional features of the PAFFIN-System.

Task formatting: The PAFFIN Web GUI uses HTML[2] elements to display task GUIs in a web browser. It can therefore interpret HTML code for text formatting, e.g. bold/italic fonts, font size. Any element of a workflow task that is displayed in the GUI can utilise this formatting. In addition to this basic formatting, specific String constants were implemented for the GUI. Before task information is sent to the GUI, a special parser is used on task title and task description. That parser transforms each of the String constants into a corresponding data field from the dynamic task data at runtime. That way it is possible to have a task description dynamically incorporate current runtime data like the current task parameters or the identifier of the engine executing that task. All in all, there are 15 constants that can be used. They are shown in Table 11.1. If data for a constant is not available it is not parsed. The availability of data depends

[2]Hyper Text Markup Language

Constant	Description	Availability
#name#	Name of the task	Available everywhere
#title#	Title of the task	Available everywhere
#titleraw#	Raw Version of title of the task	Available everywhere, no variables parsed
#description#	Description of the task	Available everywhere
#descriptionraw#	Raw Version of description of the task	Available everywhere, no variables parsed
#agentaction#	Agent Action of the task	Available everywhere
#guiid#	GUI Identifier of the task	Available everywhere
#workitemid#	Workitem ID	Available in workitems and activities
#param#	Parameters of the workitem	Available in workitems and activities
#priority#	Priority of the workitem	Available in workitems and activities
#binding#	Workitem binding	Available in workitems and activities
#engine#	Name of workitem engine Paffin	Available in workitems and activities
#aao#	AAO Identifier of workitem	Available in workitems and activities
#resource#	Currently assigned resource	Available in activities
#activityid#	Activity ID	Available in activities

Table 11.1: Overview of constants parsed in task titles and descriptions

on the state of the task. Some static task information is available everywhere, while dynamic workitem and activity information is only available during or after that state.

Workflow management facilities: The entirety of the platform workflow management facilities (described in Section 10.2.14) and the Paffin internal workflow management facilities in the backend (described in Sections 10.2.10 and 10.2.11) are, just like the GUI, important but non-essential parts of the Agent-Activity approach. Again, the Agent-Activity just assumes there are engines and resources and that these two can handle workflow operations. Workflow management is implicitly required, but not an actual part of the concept. Therefore, the management facilities that connect these are additional features beyond the basic proof-of-concept. More so, additional extensions like the role-based access control system and the basic database support have been added as well.The general management facilities also ensure a clean execution, i.e. they feature mechanisms to remove references to invalid workitems, activities and Agent-Activities.

Enduring tasks: Application tests revealed use cases, in which workflow tasks should endure beyond singular Agent-Activities (see Section 11.3.3). Therefore, an additional feature was implemented allowing a workflow task to be started (i.e. the workitem requested) in one Agent-Activity and terminated (i.e. confirmed or cancelled) in another. Automatic local rollback here is severely limited due to the original request being unavailable, yet modellers can accommodate for that explicitly.

Channel watchers: In order to support Paffins as workflow engines, sophisticated channel watchers have been implemented. This way Paffins can recognise available workitems and confirmable activities in their Agent-Activities and provide that information to the resources (via the platform WFMS).

Decision component support: Decision components are part of the Capa standard, yet are not explicitly required by the Agent-Activity approach. Incorporating them into Paffins (see Section 10.2.9), considered as internal agent actions, enables modellers to utilise these continuous behaviours, which have proven their usefulness over the years in Paose. Through implementation efforts it was possible to provide them in the same way as in Capa (using flexible synchronous channels), even though the added abstraction of the Agent-Activity and AAO complicated these efforts.

Direct knowledge base access: Knowledge base access in Capa is handled via a standard synchronous channel. Similar to the decision components, using this channel within an Agent-Activity was initially impossible due to the modelling and abstraction levels added by the Agent-Activity and AAO preventing direct synchronisation between the nets. Therefore the direct access to the Paffin knowledge was implemented by interconnecting synchronous channels in all the nets in the nesting hierarchy between AAO and Paffin knowledge base.

Capa *legacy support:* Representing more of an implementation constraint, rather than an implementation feature, the Capa legacy support also needs to be discussed in the scope of the feature set. The design decision to enable all Capa legacy code was already discussed in Section 10.2.5. Maintaining all original Capa interfaces, by either implementing compatible transformations or keeping the original interface in place, complicated development somewhat, yet the added benefit of being able to use years worth systems, like the WebGateway, outweighed the effort easily.

WorkBroker *interactions:* The Paffin-System allows for Paffins to serve as automatic resources in workflow tasks of (other) Paffins. This realises an interaction variant for agents based on the WorkBroker principle (see Definition C.13 in Section 10.3.2). Paffins as agents can utilise behaviour based on tasks, which are distributed to (automatic) resources by the platform WFMS. The resources, again, are Paffins as agents. While this relates strongly to Criterion SB3 concerning the general integration, the principle is heavily emphasised and extended in the Paffin-System and therefore warrants inclusion in this feature set.

Direct workflow operations: As described in Section 10.2.3, Paffins have the ability to confirm or cancel the execution of workflow tasks for which they serve as the workflow engine. This mechanism allows Paffins to confirm or cancel the execution of tasks autonomously and independently from the resources. Examples of use cases for this are quality control (the engine checks the quality of the result before confirming or cancelling), enforcement of constraints (the engine cancels activities that take too long) and flexibility of results (the engine can receive intermediate results and choose if these are already sufficient). Direct workflow operations strongly emphasise the integration of agents and workflows. Paffins as workflows become autonomous agents that can independently decide how best to proceed in their (workflow) behaviour.

Agent-Activity *trigger management:* The incorporation of explicit Agent-Activity trigger management mechanisms allows for their easy use. The Paffin-System provides means to create a trigger message and directly transform it into an asynchronous message. It also provides means to set and revoke proactive triggers.

AGENT-ACTIVITY *update from GUI mechanisms:* The PAFFIN Web GUI features a mechanism with which to send updates to the AGENT-ACTIVITY containing the task currently being executed in the GUI. Through this mechanism it is possible to provide the AGENT-ACTIVITY with current up-to-date information about the execution happening in the GUI. This information can then be used for further actions and operations within the AGENT-ACTIVITY, like, for example, direct workflow operations or data processing as agent actions.

Message routing options: In general, message routing in individual PAFFINS has been improved when compared to the standard CAPA agent. Before, only messages for reactive protocol start or for existing agent protocols could be distinguished. In the PAFFIN-System, routings for reactive process-protocol start and existing process-protocols are still available. In addition to that, message routing for existing AGENT-ACTIVITIES, reactive AGENT-ACTIVITY triggering and for active workflow activities have been implemented.

Accessibility of parameters: All parameters for AGENT-ACTIVITIES, workflow tasks and agent behaviour are implemented as Java objects represented by reference net tokens in some reference net. These parameters are either directly accessible in the AGENT-ACTIVITIES or are provided access to by synchronous channels. This means that all runtime data is available to both workflow operations and agent actions. A PAFFIN can, for example, easily access its workflow activity data and perform agent actions based on that. It can also create workflow tasks based on data it received from asynchronous messages.

PAFFIN *net components:* PAFFIN net components (see Section 10.2.3) provide predefined modelling elements with default values for the basic workflow operations and fundamental agent actions. They improve modelling efforts and readability of internal processes of AGENT-ACTIVITIES.

Logging and monitoring: The PAFFIN-System features several logging mechanisms. A logfile is created in the current user's home folder that features all events happening within the platform WFMS. In addition to that the platform WFMS also keeps done activity lists and distributes these to the resources for display. For monitoring, the different nets of the backend and platform WFMS feature a number of logging places. Here, tokens are deposited during the execution that represent and contain data about the events that have occurred in the system. These can be accessed using the established MULANVIEWER tool

Messages cleanup: With the increased amount of administrative messages in the system, it soon became clear that, for efficiency and monitoring purposes, invalid messages had to be more thoroughly cleaned up than was the case in standard CAPA. Due to the distributed nature of the system, messages relating to AGENT-ACTIVITIES or workflow activities could arrive after the corresponding target was already terminated. Thus, the PAFFIN-System features several mechanisms to discard messages for invalid (i.e. terminated) targets.

GUI cache: The PAFFIN Web GUI features a mechanism that stores the current content of the GUI (i.e. the list of available workitems, current activities and done activities). This cache is used when a user refreshes the web browser or accidentally leaves the GUI page. It allows restoration of the GUI state without requiring a new update from the PAFFIN-System.

This feature set represents the current status of the Paffin-System at the completion of this thesis. The maturity highlighted by the previously described features emphasises that the Paffin-System goes beyond the desired proof-of-concept for the Agent-Activity approach. Ongoing development and research will add more and improved features. However, the prototypes described in Chapter 10 with the feature set described here are the core Paffin-System and represent the technical and practical main result of this thesis.

11.2.2 Limitations of the Paffin-System

The Paffin-System is a prototype. It fulfils its duty as a proof-of-concept of the Agent-Activity integration approach and even goes beyond that, as described in the previous section. However, its prototypical nature means that there are still some technical limitations that should be discussed here. Some functionality is only implemented in a basic way. Most of the limitations can be manually circumvented, but should be addressed in future work.

Reasons for the limitations are most often lack of time and manpower during the implementation. Except for the Paffin Web GUI, the entirety of the Paffin-System was implemented by the author of this thesis alone. For the Paffin Web GUI and the Paose support application prototype (see Section 11.3.3), the development team grew to include Dennis Schmitz. Still, two people developing a full-fledged modelling framework aren't enough to address every mechanism and feature. Additionally, the goal of the Paffin-System was always to provide only a proof-of-concept and not a system that could compete with established, commercial systems, which are outside of the scope of this thesis. It needs to be stressed again that the Paffin-System *fully* accomplished its goal as a proof-of-concept, even if some limitations remain.

The following lists notable technical limitations in the feature set of the Paffin-System. Each limitation will also discuss a potential solution or current workaround. General open issues, such as performance or complexity, are discussed later in Section 12.4. Likewise, apparent extra improvements to the agent or workflow infrastructure, such as agent mobility, are not considered current technical limitations. Such features are considered supplemental and are addressed as future work.

Runtime database access: The database in the Paffin-System is currently only used to read persistent management data at runtime. Only the platform WFMS Paffin has access. This represents two limitations. First, the platform WFMS Paffin should be able to write data as well. This would allow it to change permissions at runtime and write administration data (see next point). Second, other Paffins of the system should be able to read and write runtime application data to a persistent storage. This way workflow and/or agent results from applications could be stored and reused in later executions. Storing and restoring Paffin states is also an interesting scenario. For security reasons, application Paffins should not be able to access the management data that includes permissions and workflow roles. Therefore a second database is feasible.

Currently, the only way to (write) access the database is to edit the initial database. Database access uses the PersistentOntology plugin for Capa. That plugin also contains all necessary code to address both of the limitations described above, meaning it provides means to read and write data to a database for any Paffin. That functionality can already be implemented as an application-specific solution, thus circumventing the limitations. However, a framework solution is preferable. By

providing Paffins with a standardised database interface, persistent data storage could be better and more easily utilised in applications.

Runtime administration: Related to the runtime database access is runtime administration. Since administration data is stored in the platform WFMS database, the database limitation has to be fixed in order to allow for runtime administration. Beyond that, an interface needs to be provided as well. Section 11.4.3 presents a vision of a workflow administration based on workflows that would address this limitation. However, a traditional administration interface in the Paffin Web GUI is also possible. The only current workaround to administration is to edit the initial database class that is instantiated with the Paffin platform.

Agent-Activity *update mechanisms (beyond AAO reset):* As described in the feature set, the only update mechanism currently available is the AAO reset. Other mechanisms could exchange an AAO for another one, thus exchanging the internal process. This could be used to completely change the execution of an Agent-Activity or to simply change small details like delete a workflow task or some agent actions. Yet other update mechanisms could restore previous states or inject new or changed parameters. Use cases for update mechanisms are varied. The basic framework for this mechanisms already exists in the backend, but they still have to be implemented.

Mutually exclusive task choice: One limitation of workflow tasks in the Paffin-System is actually introduced by the Agent-Activity abstraction. Mutually exclusive workflow tasks need to be modelled in the same Agent-Activity. If they are modelled in different Agent-Activities, there is no automatic way to ensure mutual exclusiveness as the AAO nets are separate from one another. This may limit certain complex scenarios of task dependencies. A straightforward circumvention is careful modelling that translates and simplifies such scenarios so that the tasks in question are modelled in the same Agent-Activity. Some kind of automated support, though, would be desirable. Feasible solutions include a special workflow request operation variant that aborts mutually exclusive Agent-Activities, token removal controlled by the backend or synchronisation between AAO nets.

Manual connection of workflow operations to activities: A modelling limitation relates to workflow operations. When a workitem is successfully requested the tuple of input parameters, as well as the activity object are stored in a place for that task for unification purposes. It is currently not possible to automatically generate that place for the confirmation or cancellation. Rather, the modeller has to create a virtual copy of the place and connect it to the confirmation and cancellation. This manual connection is error-prone. An automated solution would require an extension of the modelling tools, e.g. saving a reference to the last place created for a request operation and automatically creating a virtual copy for the confirmation and cancellation operations.

Automatic resource intelligence: Currently, automatic Paffin resources choose the available workitem with the highest priority for request. While this strategy is sufficient in basic scenarios, it is a wasted opportunity not to utilise agent intelligence and decision making here. By adding additional data processing functionality to the backend resource net, it would be possible to analyse the available workitems for more than just the priority. Possible checks may include checking if there are lots of workitems from a single engine possibly indicating a bottleneck in the system that may be alleviated by choosing (and thus reducing) the workitems from that engine. Comparative checks against a history of done activities could also be used.

GUI global process view: Currently, users are presented with a list of their done activities as the only indication of the progress they made in the process so far. A graphical representation of the current process-protocol and/or Agent-Activity would benefit the users. Currently this is only available as a static image, but a dynamic (and possibly interactive) representation is preferable.

From the previous listing it should be clear that none of the limitations described above are truly critical, in the way that they prohibit any usability of the Paffin-System. For future work it may be essential to address some of them, like the database access, but others, such as the automatic connection of activities to workflow operations, may be ignored without much disadvantage. It also needs to be noted that none of the limitations described above really require any major or incalculable implementation effort. Given time and manpower, all of the limitations can be addressed with relative ease.

In conclusion, while the limitations present some obstacles for the usability of the Paffin-System, they are not vital to it. As the following sections will show, it is already possible to build advanced and complex integration applications with the Paffin-System. By addressing the challenges described in this section, the Paffin-System does not gain any integral capabilities, it is only improved in the already existing ones.

11.3 Application Prototypes

This section presents a number of practically implemented application prototypes. Each of these prototypes serves a specific function in the application discussion and evaluation for this chapter.

The producer-store-consumer prototypes in Section 11.3.1 highlight the flexibility of the Paffin-System. Here, three different prototypes are presented that implement the same, simple producer-store-consumer example in three different ways. Each of the prototypes emphasises another aspect of the integrated Paffin entity's agent, workflow or hybrid nature.

Next, the pizza collaboration prototype in Section 11.3.2 highlights the differences between the capabilities of the Paffin-System and regular agent or workflow management. Here, a simple example, taken directly from the BPMN context, is presented as a Paffin application. The discussion showcases the advantages and disadvantages different implementation options.

Both the producer-store-consumer and pizza collaboration prototypes lack a practical test context for the Paffin-System. Therefore, the Paose teaching support prototype in Section 11.3.3 highlights a real-world application of the Paffin-System. The Paose teaching support prototype was developed for and actually used by students in the yearly AOSE teaching project operated by the TGI group in the winter term of 2016/2017. This provided a real-world test of the Paffin-System which is discussed and evaluated here.

11.3.1 Producer-Store-Consumer Prototypes

This section presents the producer-store-consumer (PSC) application prototypes. These three prototypes implement a simple scenario in three different ways. The first implementation is agent-oriented, the second workflow-oriented and the third one mixes agents and workflows where they fit best. All three implementations use the same mechanisms provided by the Paffin-System, albeit in different ways. Although the scenario is deliberately simple, the differences in the implementations show how flexible the Paffin-System and

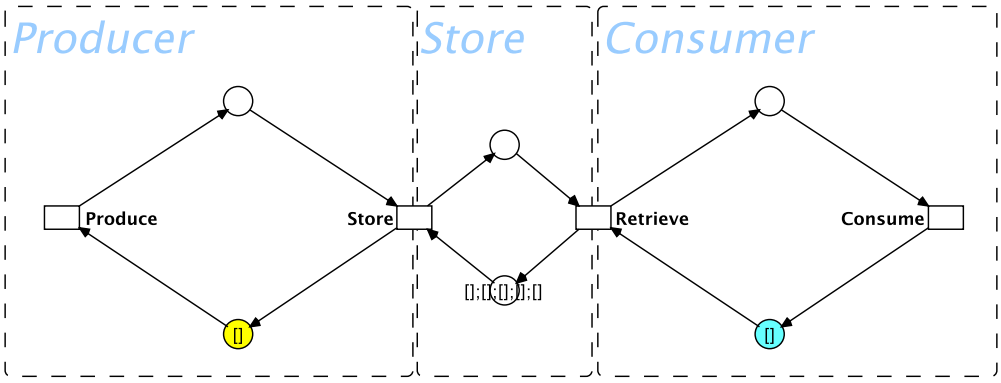

Figure 11.2: PSC scenario as a Petri net (based on [Reisig, 1987, pp. 119ff])

especially the Agent-Activity can be utilised. This flexibility can best be presented in such a simple scenario.

Scenario and General Implementation

The scenario implemented in all three prototypes is the same. It consists of producers, stores and consumers. Producers produce products, which they then hand over to a store. After a product is handed over to a store the producer can produce another product. The store has a maximum capacity of five products it can store. Once at least one product is in a store, a consumer can retrieve a product from the store. This removes the product from the store and hands it over to the consumer. The consumer then consumes, i.e. uses, the product after which another product can be retrieved from a store. The producer-store-consumer scenario is well known and established. It originates from [Reisig, 1987, pp. 119ff].

Modelled as a Petri net, this scenario can be seen in Figure 11.2. The simple Petri net version emphasises the process of one producer, one store and one consumer. The later implementations have no restrictions on the number of producers, stores and consumers.

The scenario has three basic interconnected loops, as can be clearly seen in Figure 11.2. A producer produces and stores in a loop. A store stores a product from a producer, which is then retrieved from a consumer. This is a loop that can happen five times concurrently, modelling the five product capacity of the store. Finally, a consumer retrieves a product and consumes in a loop. The interconnection between the loops models the interaction between a producer and a store and a consumer and a store. The loops are initialised in the Petri net model through tokens in the lower places of each loop.

Regarding the overall implementation, a new plugin in the Paffin project was created for this application. The `PaffinExamplePSC` plugin contains the code for each variant.

Each component of the scenario, i.e. producer, store and consumer, is implemented as a Paffin role. A Paffin role is modelled in the ARM tool [Mosteller, 2010] and defines the knowledge and capabilities of a Paffin entity inhabiting that role. Each Paffin role in the implementation possesses a number of process-protocols. These process-protocols implement their component of the scenario for the given variant. Each process-protocol contains a number of Agent-Activities. These Agent-Activities correspond, as abstract tasks, to the four basic transitions in Figure 11.2. The *Produce* Agent-Activity is associated with a producer component. The *Store* Agent-Activity is associated with both a producer and a store component. Likewise, the *Retrieve* Agent-Activity is

associated with both a store and a consumer component. Finally, the *Consume* Agent-Activity is associated with a consumer component. Depending on the variant, the distribution of these Agent-Activities in process-protocols can be markedly different. However, the basic process, execution order and Agent-Activity association remains constant as defined in the Petri net model in Figure 11.2.

Regarding the product that is exchanged in all three variants, a simple semi-randomised String object was chosen. That String consists of a random choice of three selections, a base String for each variant and the timestamp at the creation of the String. Product consumption is implemented as the consumer Paffin reading the String and writing it in its knowledge base. Overall, a String was chosen for representation and ease of use.

Storage is implemented in all three variants in a DC in the store Paffin role. That DC simply takes products from a store Agent-Activity and gives them out to a retrieve Agent-Activity. In variants A and C, the store DC is the same, shown in Figure 11.6. Variant B only differs from Figure 11.6 in the removal of the capacity check, which is handled elsewhere in variant B.

Implementation A: Agent-oriented

The first implementation variant for the producer-store-consumer example takes an agent-oriented approach. Each Paffin represents one of the three components as an autonomous agent-like entity. Communication and product exchange between the components is handled exclusively through asynchronous messages. The store Paffin provides a *Store* service, which is made known to the producer and consumer Paffins via the platform DF. Internally, each component uses internal agent actions to produce, store, retrieve and consume products.

Both the producer and consumer Paffins proactively start their process-protocols, seen in Figure 11.3 and 11.4 respectively, which have an analogous form. The *Start PrPr* and *Stop PrPr* net components are administrative components and not relevant for the application scenario. The produce and consume cycles in the respective upper right parts implement the cycles for the producer and consumer. After process-protocol initialisation, the producer can start the *produce* Agent-Activity and the consumer can start the *retrieve* Agent-Activity. Neither Agent-Activity uses a parameter, but both return a product as result. The product from the *produce* Agent-Activity is produced fully through internal actions and then returned as the Agent-Activity result. The product from the *retrieve* Agent-Activity is retrieved via interaction with the store Paffin. The internal process of that Agent-Activity can be seen in Figure 11.7. First, the consumer looks up the store Paffin identifier in its knowledge base and then sends out a request message to that Paffin. When that message arrives at the store Paffin, the process-protocol shown in Figure 11.5 is reactively started. This process-protocol contains only the counterpart Agent-Activity for retrieving from a store. That Agent-Activity uses internal agent actions to retrieve a product from the store DC shown in Figure 11.6 and then creates an answer message to the original request message containing the product. That product is received by the original *retrieve* Agent-Activity in the consumer (Figure 11.7), where the product is extracted from the message and returned as the result of the Agent-Activity.

Back in the main process-protocols for producer and consumer in Figures 11.3 and 11.4, the produced or retrieved product is ready for use. In the consumer process-protocol the product is used as the parameter for the *consume* Agent-Activity. That Agent-Activity reads the product and consumes it as internal agent actions. After consumption, the consumer loop can start again with another product retrieval. In the producer process-

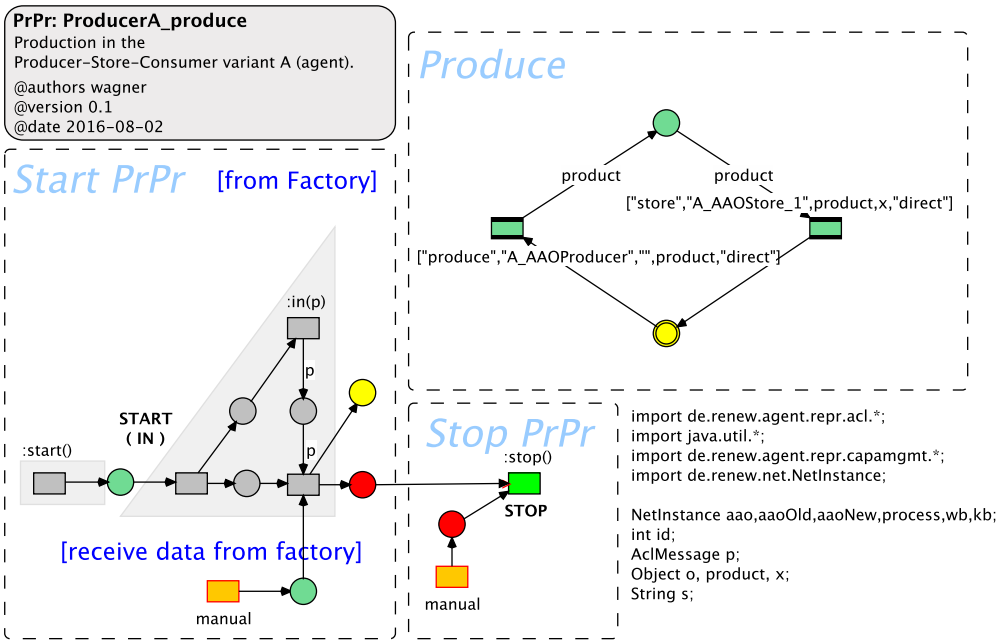

Figure 11.3: PSC variant A: Producer process-protocol

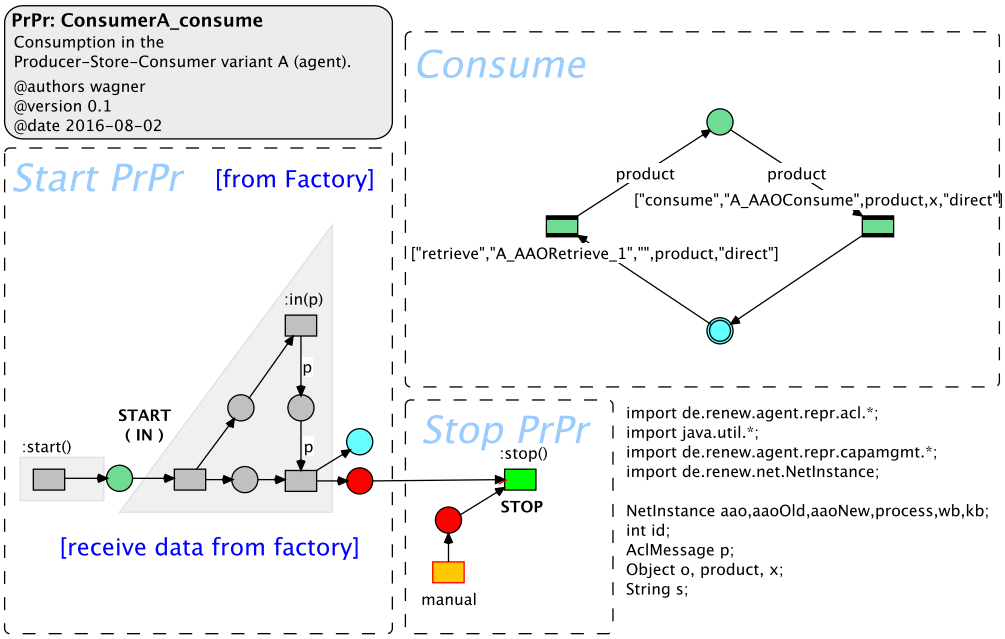

Figure 11.4: PSC variant A: Consumer process-protocol

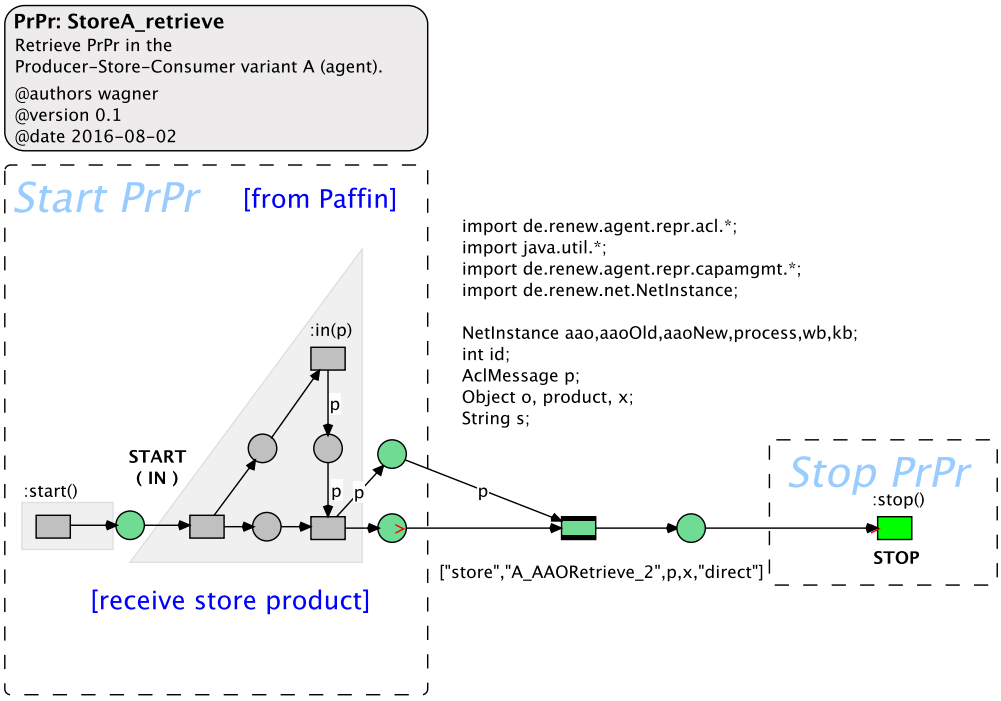

Figure 11.5: PSC variant A: Retrieve process-protocol for retrieving

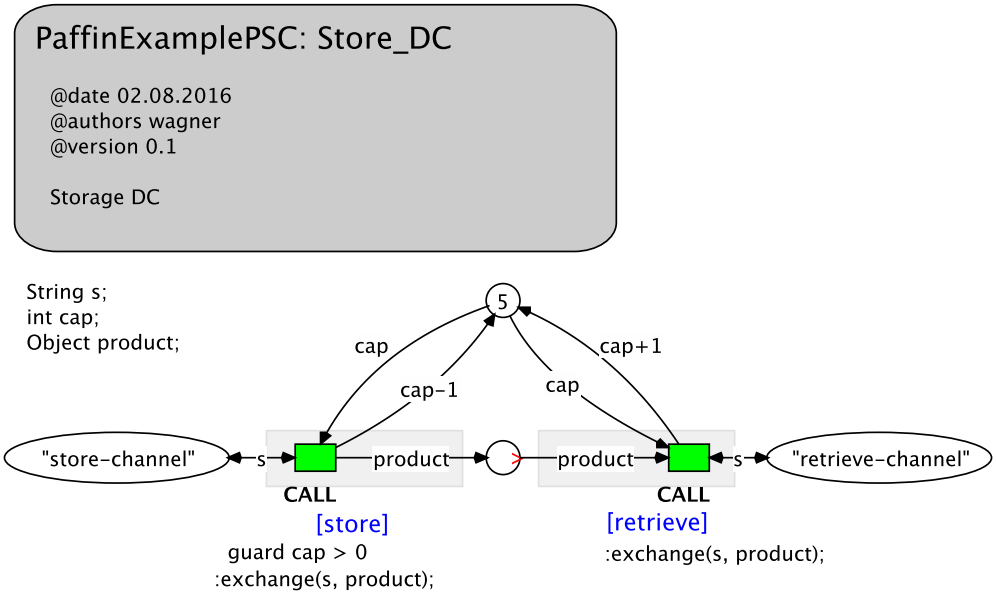

Figure 11.6: PSC variant A: Store DC

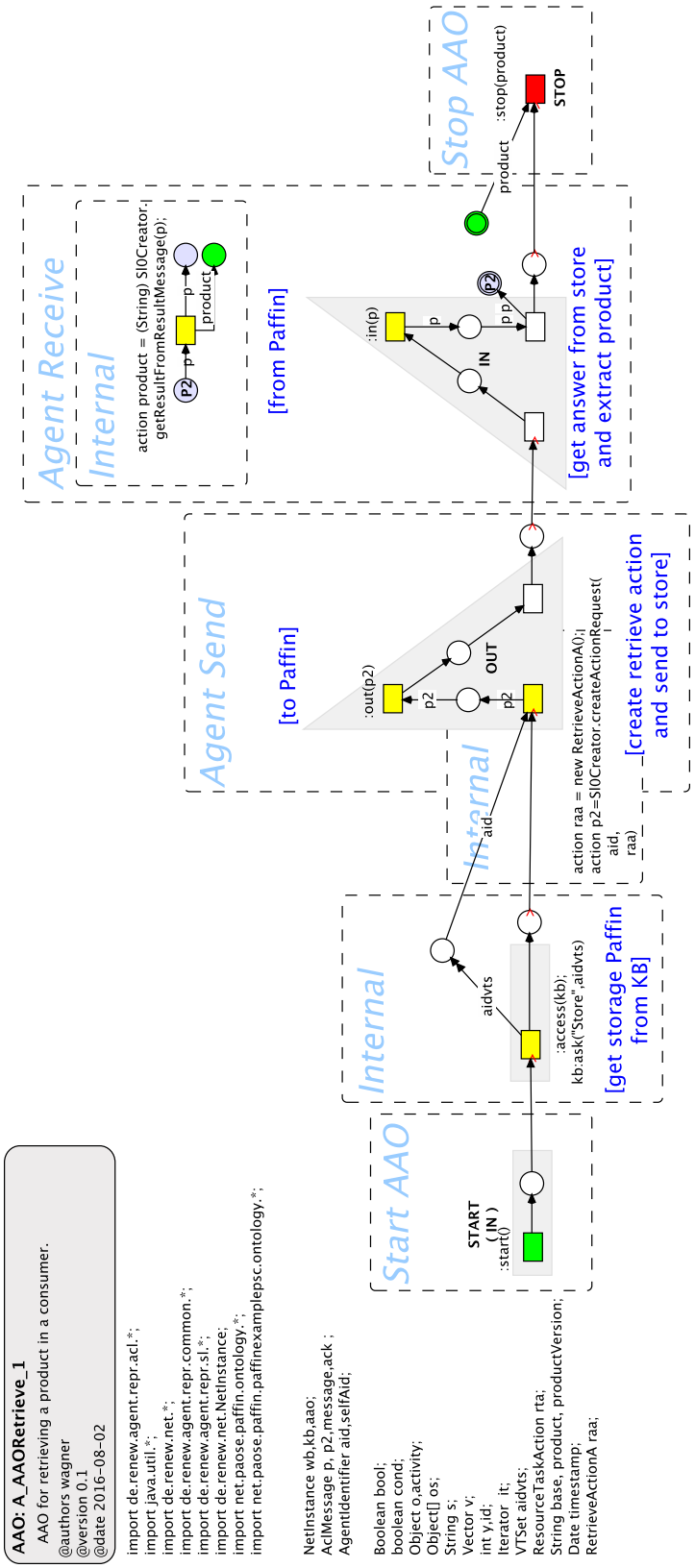

Figure 11.7: PSC variant A: Agent-Activity for retrieving in a consumer

protocol the produced product is used as a parameter for the *store* Agent-Activity. This Agent-Activity interacts with the store Paffin to store the product. It is similar to the retrieve Agent-Activity and not shown as an additional figure here. The producer simply reads the product from the parameter, the store identifier from the knowledge base and creates a store request message, which it sends to the store. The store reactively starts a process-protocol on message reception, which only contains one Agent-Activity. That Agent-Activity merely reads the product, stores it in the store DC and then sends a confirm message to the producer, which can then finish its own Agent-Activity. Finishing the store Agent-Activity reinitialises the producer loop.

Discussion For all intents and purposes, this variant implementation uses the Paffin-System to emulate a regular multi-agent system. Paffins only use the fundamental agent actions to achieve their purpose. They use agent mechanisms (services and the DF) to advertise interaction partners, use agent messages to transport information and products and use internal agent actions for all processing steps.

Consequently, the role of the Agent-Activity and integration is, from a practical perspective, somewhat reduced. Its most obvious practical advantage lies in the modularity and dynamic. It is easy to exchange the internal processes of the Agent-Activity, which could be used to realise different producer, store or consumer behaviour. For example, if the consumer should no longer write the product into the knowledge base to consume it, but rather print it to the system command line, modellers would only have to model that behaviour, save that net and then change the inscription of the *consume* Agent-Activity in the consumer process-protocol. As that inscription accepts variables, switching between different internal processes could be implemented dynamically depending on additional factors.

Considering not only the practical level, though, the Agent-Activity provides another, more important, benefit. The abstraction introduced by the Agent-Activity contributes strongly to the process perspective in the otherwise agent-oriented system. Each individual loop of the scenario is clearly distinguishable. The producer loop is the main part of the producer process-protocol in Figure 11.3, the consumer loop is the main part of the consumer process-protocol in Figure 11.4 and the store loop can be found in the store DC in Figure 11.6. The concrete behaviour, modelled through the different agent actions in all Agent-Activities, remains basically the same as if it were implemented as a regular agent system. However, perceiving the basic process from those rather large sequences of actions distributed in different protocols and DCs is more difficult. By using Agent-Activities to summarise and abstract from related sets of actions, a concise and correct process view, similar to that of the basic Petri net model from Figure 11.2, is constructed. That process view is still distributed through three nets, but each of these nets describes the process much more clearly than the combination of all agent actions into a set of protocols would. In that way, even though the integration of agents and workflows is not directly or practically utilised in this implementation, the application still benefits from it. Through the Agent-Activities a kind of abstract workflow is established, consisting of the abstract tasks described by the Agent-Activities and implemented through the agent actions. While such a process view is available in other agent views e.g. in an AIP, it is usually not coupled with the execution and only available statically as a modelling artefact. Here, the process view provided by the Agent-Activities directly and dynamically reflects the actual execution happening at any time.

Implementation B: Workflow-oriented

The second variant implementation of the producer-store-consumer scenario is workflow-oriented. Each of the three components of the scenario is implemented as a Paffin entity serving as an automatic resource in an overall workflow. That workflow contains four tasks, which correspond to the abstract tasks from the Petri net model in Figure 11.2. Communication between the different components is handled exclusively via the platform WFMS and the administrative messages exchanged therein. The workflow itself is implemented as an additional, separate Paffin entity, the process Paffin. Producing, storing, retrieving and consuming products is implemented in the automatic resources as agent actions. All three components correspond to workflow roles of the same name and each component Paffin registers itself as an automatic resource at the platform WFMS at startup. Tasks, roles, permissions and resources are all modelled as database objects and stored in the platform WFMS database.

At startup, the process Paffin proactively starts the process-protocol shown in Figure 11.8. Besides the administrative components, that process-protocol contains the basic workflow implemented with Agent-Activities. The net structure of the Agent-Activity process in Figure 11.8 mirrors the basic Petri net model, except for the technical inscriptions. After initialisation, only the *produce* Agent-Activity can trigger.

The internal process of that Agent-Activity can be seen in Figure 11.9. It consists only of a workflow task modelled through a request workitem and confirm activity net component. Since none of the components in the scenario has the ability to cancel its abstract tasks, no cancellation of activities is added to the Agent-Activities.

The production task is named *produceB* and is automatically, when activated, reported to the platform WFMS. There, it is processed, the required permissions computed and provided as an available workitem to all registered workflow resources that have the produce permission, i.e. all producer Paffins. When a producer Paffin receives a workitem list containing such a workitem, it requests it and, upon successful request starts the process-protocol seen in Figure 11.10. That process-protocol contains additional administrative components that realise its association with a workflow task. The upper administrative component allows for task management from the engine, while the extended task finish component creates an activity result.

Apart from the administrative components, the process-protocol in Figure 11.10 contains only one Agent-Activity. That Agent-Activity actually produces the product and its internal process can be seen in Figure 11.11. Basically, it reads the product base from the Agent-Activity parameter and uses it to create the product using (at random) one of the three templates provided in the enlarged place. It then returns the product as the result of the Agent-Activity.

Back in the produce process-protocol in Figure 11.10, the product is packaged into an activity result message and sent, via the technical backend of the producer Paffin, to the platform WFMS, initiating the activity confirmation in the process. The result arrives back at the process Agent-Activity in Figure 11.9 as the *actResult* token and is then handed back as the Agent-Activity result to the overall process in Figure 11.8. Here, the next Agent-Activity for storing becomes activated with the product as parameter. Each of the Agent-Activities and workflow tasks follow analogous behaviour, which is why they won't be discussed in detail here further. When the *store* Agent-Activity in the process is finished, the producer loop is reinitialised and can produce again. Additionally, since there is a product now available, the *retrieve* Agent-Activity can also retrieve the product for the consumer, which, when completed, initialises the *consume* Agent-Activity. Finishing that Agent-Activity reinitialises the consumer loop. Through the

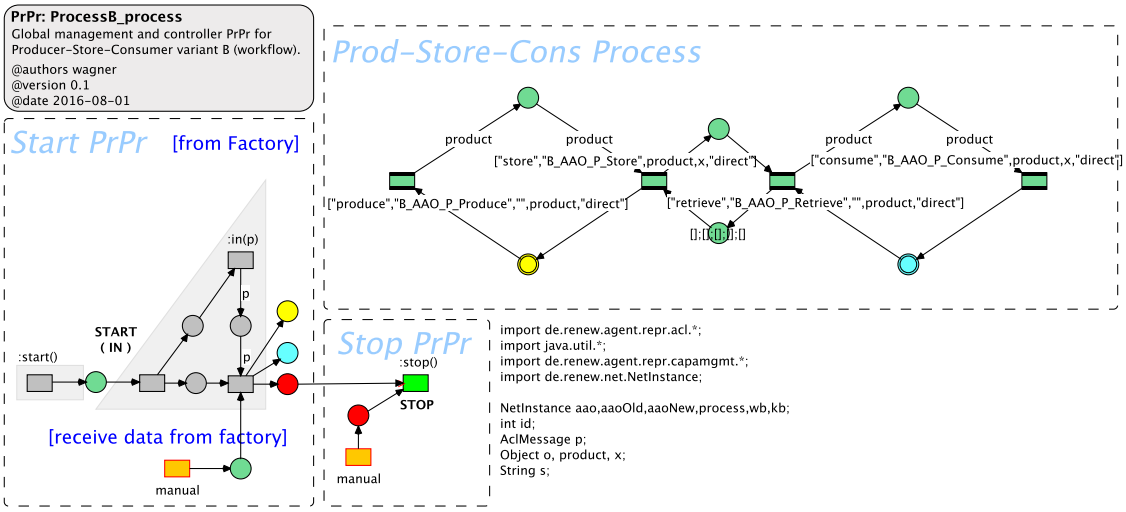

Figure 11.8: PSC variant B: Process process-protocol

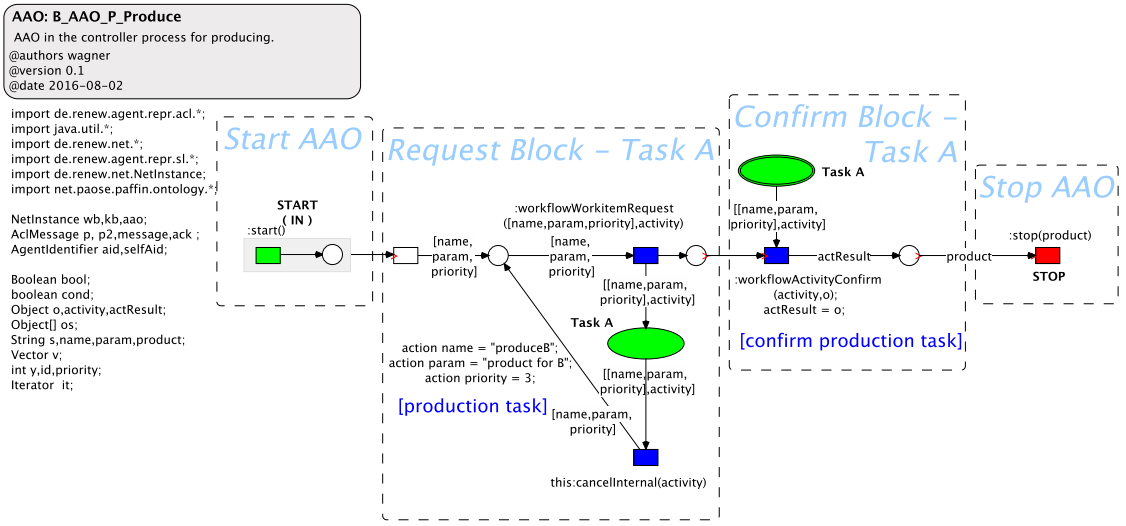

Figure 11.9: PSC variant B: Process Agent-Activity for producing

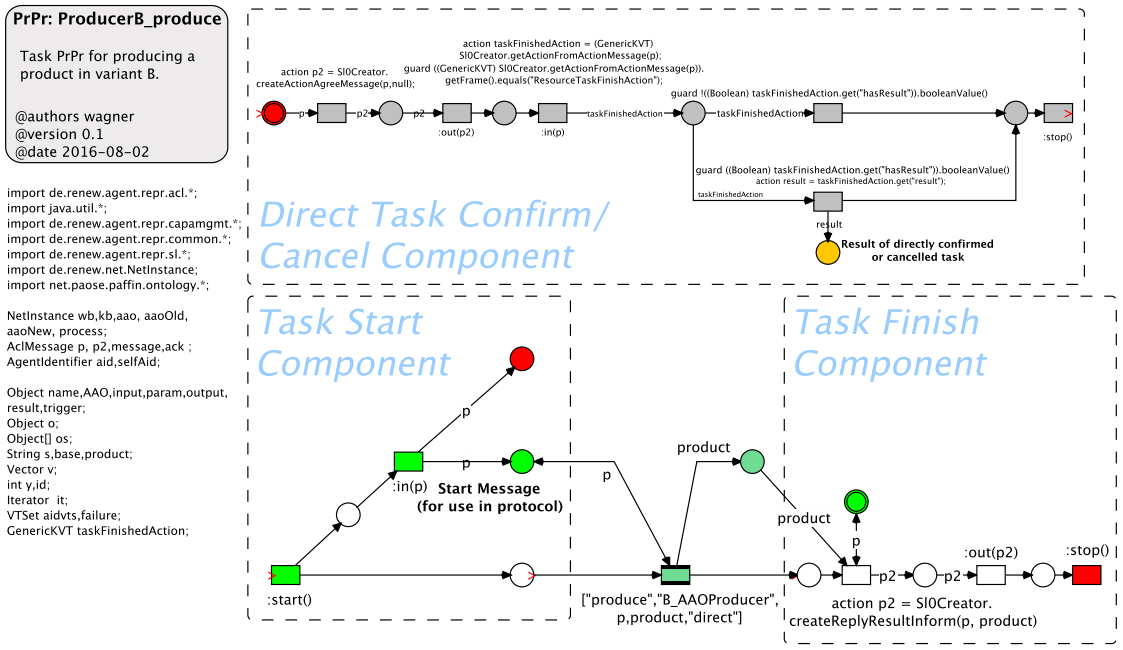

Figure 11.10: PSC variant B: Producer process-protocol for producing

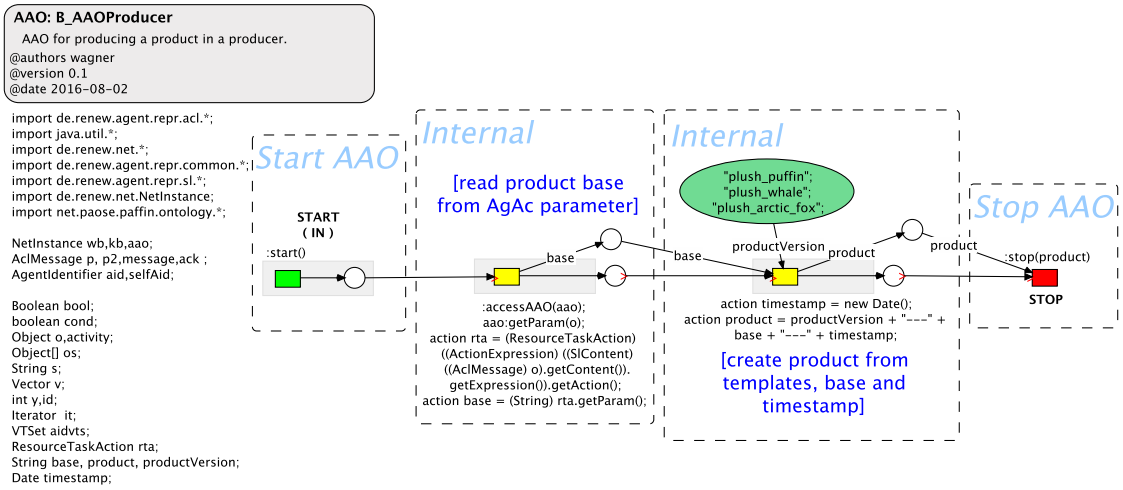

Figure 11.11: PSC variant B: Producer Agent-Activity for producing

interplay of preconditions of the different Agent-Activities, the producer-store-consumer process is realised.

One thing of note for this variant is, that the store DC works slightly differently here. The overall process controls the execution of producers, stores and consumers. Consequently, it also controls the capacity of the storage, which is why the capacity is explicitly modelled in the process and removed from the store DC. The store DC is identical to the one from variant A, shown in Figure 11.6, but has the capacity and related guard inscription removed. The store can still only hold five products at any time however, even if this is controlled outside of the store.

Discussion In this implementation variant, the Paffin-System works very similar to a regular WFMS. It utilises a process-protocol using Agent-Activities that only contain workflow operations to effectively model the producer-store-consumer process as a classical workflow. Agent actions and other aspects are only used administratively in the background or to implement the actual work associated with the different workflow tasks. For all intents and purposes, when ignoring the functionality of the resources, the Paffin-System here acts as a regular WFMS controlling automatic (agent) resources.

However, this is only a limited perspective. The process in Figure 11.8 does not just control the execution of work in the component Paffins. It **also** completely handles the communication. Seen as autonomous agent actors, the component Paffins are completely decoupled from each other. Their only connection is indirect via the overall process. All communication is either from the process or to the process, both via the platform WFMS. This means that the Agent-Activities, or rather the workflow tasks within them, control and manage the work and interaction between the involved actors. This correlates to the WorkBroker principle (see Definition C.13) for utilising task principles in agent interactions. The Paffins as agents, benefit from, e.g., task atomicity (although not utilised in this simple scenario without activity cancellations) and resource decoupling (i.e. they don't need to know beforehand who to interact with). And while the component Paffins are controlled through the tasks in the overall process, they don't lose any degree of autonomy, since they themselves decide to request the corresponding workitems, execute the associated work and return a result. Overall, this additional type of interaction between Paffins opens up many possibilities for modellers to apply workflow ideas to otherwise agent-oriented areas.

As with variant A, the role of the Agent-Activity in this implementation is slightly diminished. Its modularity and dynamic flexibility characteristics can still be used much in the same way. However, in this prototype implementation, the use of Paffins to represent resources provides the additional benefit. During execution, the Paffins active in the system represent the structure of the system. Each component of the system and what it does is easily recognisable, and via inspection more details can be obtained. As the system here consists of a workflow, the Paffins create a structural view on that workflow, which is something that is otherwise difficult to obtain in classical WFMS. That view is dynamic and updates itself constantly. Together, the structural view of the Paffins and the process view obtained by the Agent-Activities in the overall process, which correlate here directly to the tasks in the workflow, create a more comprehensive view on the overall system, as either agents or workflows could do on their own. While the resources in this scenario are automatic, the previous result can be directly applied to Paffins representing human users or any other kind of resource in a workflow in the Paffin-System.

Implementation C: Hybrid

The third and final implementation variant of the producer-store-consumer scenario mixes agents and workflows. It takes mechanism from agents, workflows and the integration and applies them were they best fit in the scenario.

In its implementation it follows the same basic approach as variant A. Three PAFFIN roles implement the three components. No explicit process PAFFIN is used. This was chosen as a process controlling both producers and consumers at the same time is less fitting for the scenario. Each component should be more independent.

As in variant A, the producer and consumer PAFFINS proactively start their process-protocols at startup. These are shown in Figure 11.12 and 11.13. A direct difference between the producer and consumer process-protocols of variant A is that the explicit Petri net loop is gone. Rather, the *produce* and *retrieve* AGENT-ACTIVITIES are now triggered proactively. This captures that producers and consumers are themselves aware of (in their knowledge) that they are ready to produce or retrieve. The trigger is initially set and then reset after the product has been stored for the producer or consumed for the consumer.

After initialisation, the producer can start the *produce* AGENT-ACTIVITY. This is the same one as in variant A, using only agent actions. An agent-oriented AGENT-ACTIVITY was chosen here since the producer internally does not need to set itself a workflow task, thus avoiding management overhead. Following the production, the *store* AGENT-ACTIVITY can be triggered with the product as parameter. That AGENT-ACTIVITY corresponds to the one from variant B, using a workflow task. For the scenario this realises the more reasonable view that storing a product is a task given to the store by the producer. The workitem is provided to and worked on by the store in the same way as in variant B. The difference is, that the store again has control of and manages its own capacity, which is again more reasonable than having an external process manage the capacity. When the store has confirmed the store task, the producer resets its proactive trigger for production, thus reenabling the producer loop.

In the consumer, the execution starts proactively with the *retrieve* AGENT-ACTIVITY. That AGENT-ACTIVITY also uses a workflow task, as a model in which the consumer's "desire" for a product is best modelled as a task for the store to retrieve a product. However, it doesn't use the same AGENT-ACTIVITIES as variant B. The current AGENT-ACTIVITY internal process for the consumer is shown in Figure 11.14, while the one for the store is shown in Figure 11.15. Until after the workitem has been requested, the basic procedure it is the same. In response to successfully requesting the workitem, the store starts a task-related process-protocol, which only contains the AGENT-ACTIVITY shown in Figure 11.15. Here, it first gathers the data about the executing workflow engine for the current activity, which in this case is the consumer PAFFIN. It reads the ResourceTaskAction associated with the task from its parameters and reads both the identifier of the engine and the identifier of the (retrieve) AGENT-ACTIVITY in the engine from the ResourceTaskAction, in addition to its own identifier from its own knowledge base. It then retrieves the product normally. Afterwards, however, it does not return the product as the result of the task and have the store PAFFIN confirm it itself, but rather uses the identifiers gathered before and packages the product into a message it sends to the engine AGENT-ACTIVITY in the engine PAFFIN. It then finishes the AGENT-ACTIVITY without a result. When the message containing the product arrives at the consumer retrieve AGENT-ACTIVITY in Figure 11.14, the product is unpacked and only then is the activity confirmed *by the consumer*. This direct confirmation by the engine better captures the fact that the retrieval of the product is only finished when the consumer has received it and is content with it. After retrieving a product, the *consume* AGENT-ACTIVITY in the consumer process-protocol in Figure 11.12

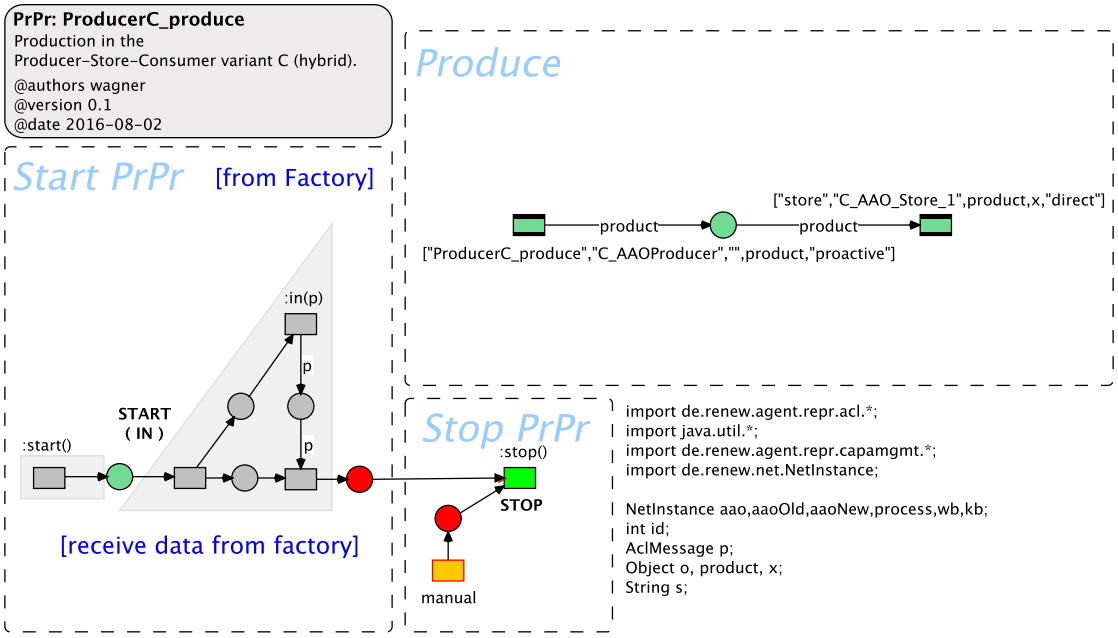

Figure 11.12: PSC variant C: Producer process-protocol

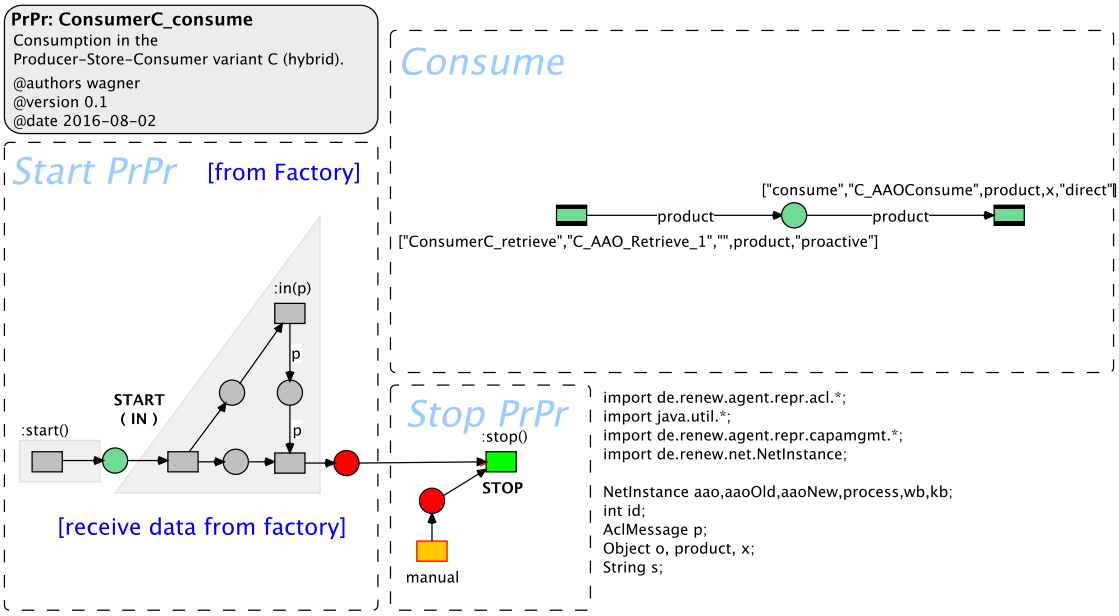

Figure 11.13: PSC variant C: Consumer Agent-Activity for producing

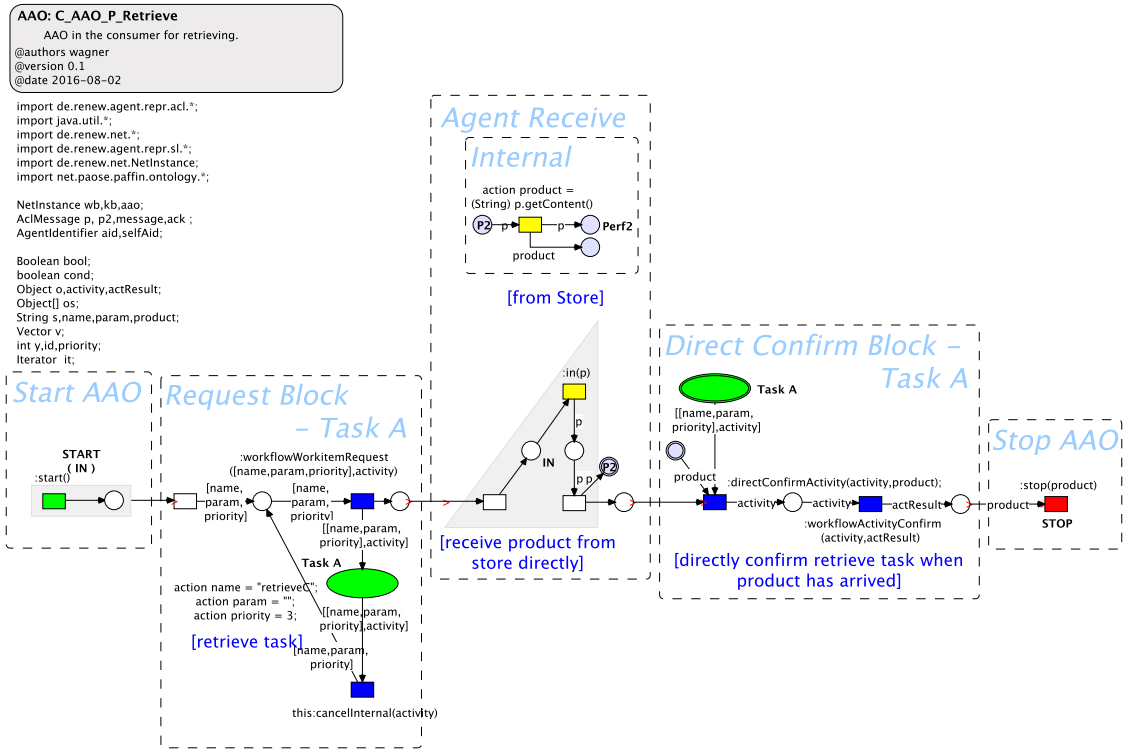

Figure 11.14: PSC variant C: Consumer AGENT-ACTIVITY for retrieving

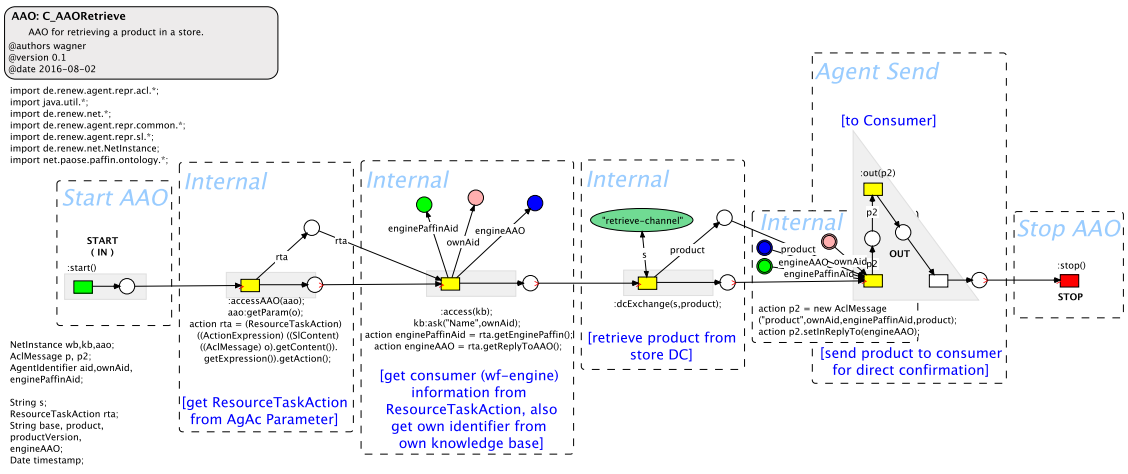

Figure 11.15: PSC variant C: Store AGENT-ACTIVITY for retrieving

can trigger. This one is, again, almost identical to variant A using only agent actions. The only difference is that at the end of consumption the trigger for proactively retrieving a product is reset, thus reinitialising the consumer loop.

Discussion Even though the simplicity of the prescribed scenario limits the options w.r.t. integrations and combinations of agents and workflows, the hybrid prototype still manages to illustrate some integration mechanisms. These can be transferred to more complex scenarios easily.

Proactive triggering of Agent-Activities is here only used to realise the producer and consumer loops. In extended scenarios, any other internal component of the producer or consumer can use the triggering mechanism to either reset or revoke the trigger. For example, if the scenario featured a payment system the an accounting part of the producer may revoke the trigger if not enough funds were available for production.

Direct confirmation of workflow tasks is also only used in a limited way, confirming when a product arrives at the consumer. Here, a condition could be incorporated that the consumer Paffin only wants specific products and (also directly) cancels the activity if it received a wrong one.

Ultimately, the purpose of this prototype does not lie in highlighting the integration features. Rather, it showcases the flexibility of the Agent-Activity and Paffin concepts. It uses the best-fitting, i.e. most reasonable, agent, workflow or hybrid mechanism to implement the specific requirements of the scenario. Producing and consuming happen directly within the producer and consumer respectively and are thus best captured in the agent-oriented variant. Storing and retrieving are tasks the producer and consumer set for the store and are thus best captured in the workflow-oriented variant. As an extension, when retrieving a product the consumer should have ultimate control over whether it accepts the product, thus the hybrid direct confirmation is best suited. Finally, the decision to produce or retrieve a product should happen proactively, not in a rigidly prescribed loop. The aspects of flexibility of the Agent-Activities and Paffins are picked up again in the following overall evaluation discussion of these prototypes.

Concluding Evaluation

This version of the producer-store-consumer scenario is among the simplest. The scenario itself can be arbitrarily extended. A payment system, distinct products, market competition, custom product orders or specific consumer needs are just some of the many possible extensions. However, the choice of the simplified version was intentional. It allows focussing the discussion and evaluation of the prototypes not so much on additional features of the scenario, but rather on the different ways of implementation.

The Paffin-System combines and integrates agents and workflows. Implementation variant A showed how the producer-store-consumer scenario can be implemented in the Paffin-System in an agent-oriented way ignoring any of the workflow or integration mechanisms. It uses Agent-Activities only with agent actions and has Paffins that are effectively agents. Even though it did not directly use anything other than agent mechanisms, this variant still benefited from the overall integration. The process view on the otherwise structurally-focussed agent-like system provided through the Agent-Activity abstraction creates a distributed, but global view on the abstract behaviour of the system. In this simple scenario the gain is negligible, but if the same concepts are applied to a more complex scenario, involving many more components and sub-behaviour, the gain of an abstract behaviour overview can't be underestimated. That benefit is increased, even in the simple scenario, by the fact that the Agent-Activities abstractly represent

the behaviour, but are actually directly coupled with the behaviour (agent actions) being executed. This means that the process-view is dynamic and constantly up-to-date.

Implementation variant B then showed how to implement producer-store-consumer in a workflow-oriented way in the Paffin-System. Except for the functionality of the automatic resources, this implementation ignored agent and integration mechanisms. One central workflow controls and manages the execution of work within each of the components of the scenario. Again, the implementation still benefited from the integration in the background. It realised a workflow-based way of managing the interactions between agents and also provided a structural view on the components. By having Paffins represent the resources (here, themselves), the Paffins constitute the structure so that a representation of the Paffins, as given in the MulanViewer tool, is a representation of the structure of the overall system. That representation is dynamic, constantly updated and directly coupled with the workflow, thus creating a bridge between behaviour and structure.

Finally, variant C considered where best to utilise agent, workflow and integration mechanisms. It used pure agent actions in some areas, workflow operations for other areas and finally hybrid integration functionality in yet other areas. In of itself it showcased what this entire group of prototypes showcases. Paffins are a versatile tool. They can make use of what best fits the needs of each given situation without then becoming rigidly stuck in a concept. Producers are agents when they produce and then turn into workflows when they want to store the product. The same is true for consumers, which are workflows when they retrieve products and turn into agents to consume them. Stores are only agents in this implementation, but acting as automatic resources in the context of workflows. Additionally, the abstract views on structure and behaviour discussed for variants A and B also apply here.

Overall, the three prototypes presented in this section illustrated the flexibility of Paffins in regards to modelling. The discussions made above about variant C apply to the overall scheme of the three prototypes. It is possible to realise producer-store-consumer in any number of ways. With the given scenario and the context of considering what is most reasonable for that scenario, the hybrid variant C was developed with the subjectively best-fitting mechanisms. By modifying the scenario or changing the context, that subjective way of considering what the best mechanisms are can drastically change. If a central control of workflow was preferred, a hybrid version more akin to variant B could have been implemented. Here still, agent mechanisms could be of use in taking care of storage capacity or product transfer. The Paffin-System could still realise this new hybrid variant, without changing the basic building blocks of Agent-Activities.

In conclusion, the main result of these prototypes is that the Paffin-System is, in fact, capable of implementing agent, workflow and hybrid applications. Not only that, but it also manages to implement all three types of applications without changing tool- or modelling construct-sets. With the same modelling construct, namely the Agent-Activity, the Paffin-System can implement the two extremes of agent-orientation and workflow-orientation, as well as a combination and integration of the two in between.

11.3.2 The Pizza Service

This section presents the pizza collaboration application prototype. Based on an established scenario from the BPMN context (see Section 3.2.1), this prototype highlights some features of the Agent-Activity and Paffin-System that relate to how integration mechanisms can be used to improve upon basic, traditional agent or workflow modelling.

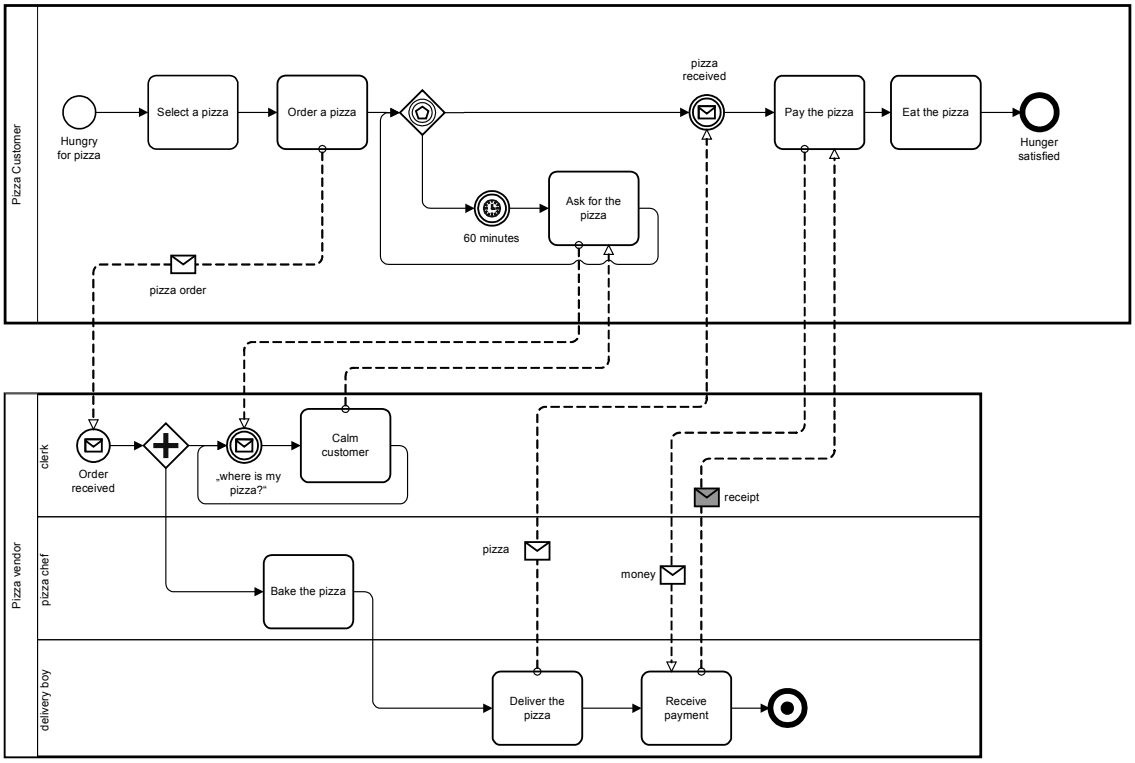

Figure 11.16: BPMN pizza collaboration (from [BPMN10, 2010, p. 4])

Scenario

The scenario implemented in this prototype is based on the pizza collaboration example in [BPMN10, 2010, pp. 4–5]. Roughly, the scenario describes how a customer may order a pizza from a pizza delivery service. The BPMN model for this scenario is shown in Figure 11.16.

The model description is provided: " *If we step through the diagram, we should start with the pizza customer, who has noticed her stomach growling. The customer therefore selects a pizza and orders it. After that, the customer waits for the pizza to be delivered. The event based gateway after the task "order a pizza" indicates that the customer actually waits for two different events that could happen next: Either the pizza is delivered, as indicated with the following message event, or there is no delivery for 60 minutes, i.e., after one hour the customer skips waiting and calls the vendor, asking for the pizza. We now assume that the clerk promises the pizza to be delivered soon, and the customer waits for the pizza again, asking again after the next 60 minutes, and so on. Let's have a closer look at the vendor process now. It is triggered by the order of the customer, as shown with the message start event and the message flow going from "order a pizza" to that event. After baking the pizza, the delivery boy will deliver the pizza and receive the payment, which includes giving a receipt to the customer.*" [BPMN10, 2010, pp. 4–5]

This scenario is used in this thesis to highlight some inter-organisational aspects relating to the AGENT-ACTIVITY in a simple scenario. The customer and pizza delivery service are both considered as organisations[3] that collaborate in order to achieve their goals. The

[3]For this example the view of the customer as an organisation is reasonable, due to the explicit modelling of its internal and external process. [BPMN10, 2010] also considers this scenario as an example of business-to-business collaboration.

customer wants to satiate his or her hunger, while the pizza delivery service wants to sell a pizza. In addition to that, the pizza delivery service organisation employs multiple employees that are all explicitly modelled in the scenario. What is important for the purposes of the following discussions is how the two organisations, as well as the individual (employee) actors interact using mechanisms and properties from agents, workflows or a mixture of both.

Please note that the following implementation is only based on the model provided in [BPMN10, 2010]. That model misses some specifics that are required for an implementation, like details about how the individual tasks are performed and how the data flow is between them. The basic, overall inter-organisational process of the model, as seen in Figure 11.16, is completely captured by the implementation. It is however extended in its details regarding the tasks and data flow.

Implementation

The implementation of the pizza collaboration is provided in the `PaffinExamplePizza` plugin in the Paffin project. To highlight the capabilities of the Paffin-System regarding user input and GUI, the basic premise was implemented as follows: The customer is an actual human user in the Paffin-System using the default Paffin Web GUI. The remaining actors, i.e. the three employees of the pizza delivery service, are "simulated" by automatic Paffins.

All in all, the prototype contains four main Paffin roles realising the four types of actors, of which an arbitrary number of Paffin entities can be instantiated. The `PizzaCustomer` Paffin role is defined in user mode, so that it automatically starts a web GUI Paffin. It also proactively starts the `PizzaCustomer_hungry` process-protocol, which models the main process within the customer. The remaining main roles are `PizzaClerk`, `PizzaChef` and `PizzaDeliveryBoy`. In addition to the main roles, there are a number of supplementary roles dedicated to realising multiple pizza delivery services. These roles contain WFMS login data for the clerks, chefs and delivery boys of currently two delivery services implemented in the prototype. Through these roles and mechanisms within the Paffin-System it is ensured that a pizza order is completely worked on by clerk, chef and delivery boy of the same delivery service. Mechanisms involved in this are discussed further below. Note that the supplementary roles are only necessary for the automatic pizza delivery Paffins. Customers, as human users, provide their logins and passwords manually via the web GUI.

All exchanged objects are provided as data objects, either standard ones, mostly Strings, or specifically created ontology objects. The ontology for this prototype contains six concepts. Three resource task actions are used to automatically start process-protocols associated with workflow tasks for receiving an order and baking and delivering pizzas in the automatic pizza delivery Paffins. The remaining three ontology objects are the `Pizza`, the `PizzaBox` and `Receipt`. A Pizza object contains fields identifying what kind of pizza it is and the ingredients (as Strings) used by the chef baking the pizza. Pizzas in the system are delivered in PizzaBox objects. This is an extension to the BPMN model. When the customer orders a pizza, it is delivered to him or her inside an additional wrapper object called a pizza-box. This design decisions was made to better capture and include administrative data like addresses and identifiers to facilitate communication between the different active Paffins and Agent-Activities. The pizza-box contains the pizza and all of the required administrative data. Finally, the Receipt object summarises the payment data of the pizza order.

Regarding the database, it contains the standard required data. WFMS resource credentials for all human and automatic resources, task data for all workflow tasks and

RBAC data including permissions and workflow roles are stored. In addition to that, this prototype also contains task-specific GUI definitions. For each task worked on by a human user, the database contains an `ActivityGuiModel` object created using helper methods from the PaffinWebGui plugin. These are invoked to create buttons, text fields and radio button selection areas in the web GUI.

Before describing the implementation of the BPMN model in detail, it is worth highlighting the dynamic permissions mechanism of the Paffin-System. This mechanism is used to ensure a correct resource to task allocation beyond the static permissions defined in the database. Each step of the pizza delivery, from selecting to eating the pizza, is implemented as workflow tasks for the human user. The dynamic permissions are used here to ensure that the customer that selected a pizza in the first workflow task also subsequently receives it, pays for it and so on. A second area in which the dynamic permissions are used is to ensure that a pizza clerk only forwards the order to a chef and delivery boy of their shared delivery service. In both cases the dynamic permissions work similarly. For the customer, the dynamic permission relating to its WFMS login is added to all tasks intended for him or her. This ensures that not all (human) customers are provided with the task, but only the current one with the correct login. For the delivery service Paffins, one of the supplementary Paffin roles is used. Each delivery service Paffin is instantiated with one of the delivery service roles, which only contains the identifier of the delivery service. The clerk reads that identifier first and adds it as a dynamic permission to the task it provides to the chef. The chef does the same for the task it provides to the delivery boy. The dynamic permission is added in the internal process of an Agent-Activity at the workitem request operation. Here, instead of the usual three-tuple of task name, parameter and priority, a four-tuple containing an additional list of String objects is handed over to the technical backend. From that tuple, the workitem object is created that is used throughout the platform WFMS. Whenever a task is checked for permissions, not only the static permissions are checked, but also the dynamic ones.

Since the overall BPMN model is relatively sequential, the Agent-Activities, tasks and process-protocols are discussed in rough sequence in the following as well. Some implementation mechanisms are highlighted with figures, yet due to the size of the prototype not all created nets can be shown here.

The proactive start of the prototype is provided by the `PizzaCustomer_hungry` process-protocol, shown in Figure 11.17. It basically captures the BPMN pool of the customer, with slight alterations. Each Agent-Activity in this process-protocol contains, amongst other actions and operations, at least one workflow task for the (human) customer. To ensure that the same customer and only that customer is provided with related tasks, the dynamic permissions mechanism is utilised.

The first Agent-Activity is *SelectPizza*, which corresponds to the BPMN task of the same name. This Agent-Activity is kept relatively simple. It reads the user login from the knowledge base to add it as a dynamic permission. Then it provides a workflow task with that dynamic permission to its logged in user. The GUI for that task contains a text field, in which the user can enter the kind of pizza he or she would like to order. When confirmed, the input from the GUI is processed and returned as the result of the Agent-Activity.

In the process-protocol of Figure 11.17, the next Agent-Activity is *Order the pizza*, again corresponding to the BPMN task. This Agent-Activity again starts by reading the user login for dynamic permissions. This is followed by another workflow task for the customer, which uses the user login as a dynamic permission and the pizza selection from the previous task as the parameter. In the GUI, a radio button group is shown, in which

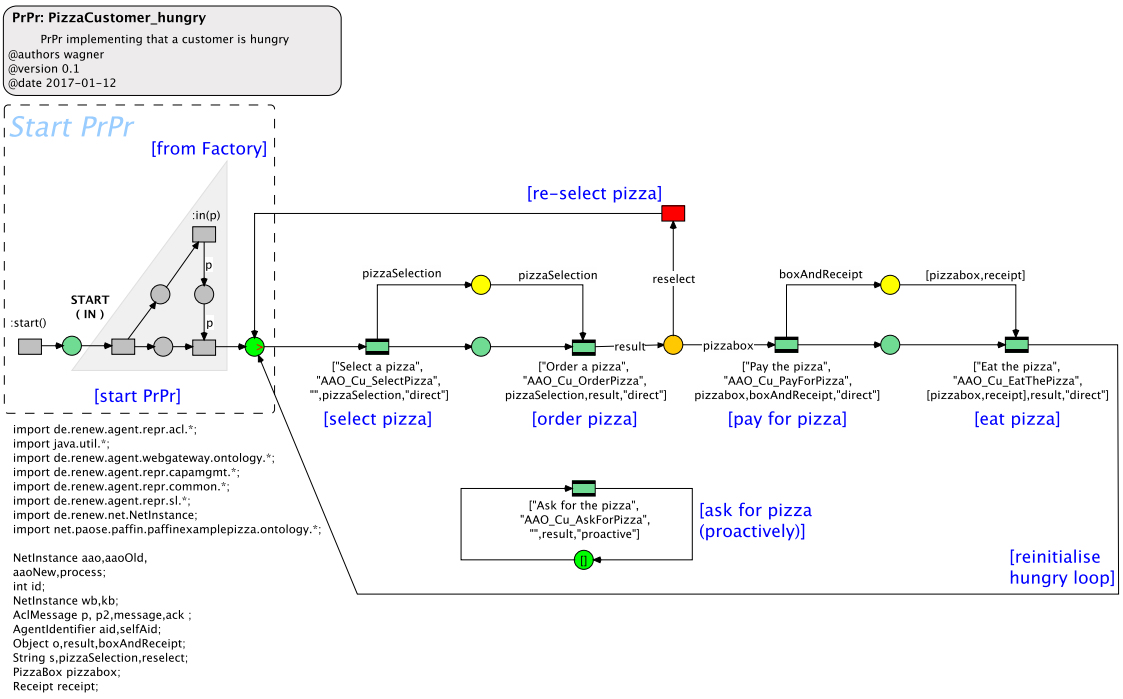

Figure 11.17: Pizza collaboration: Customer process-protocol

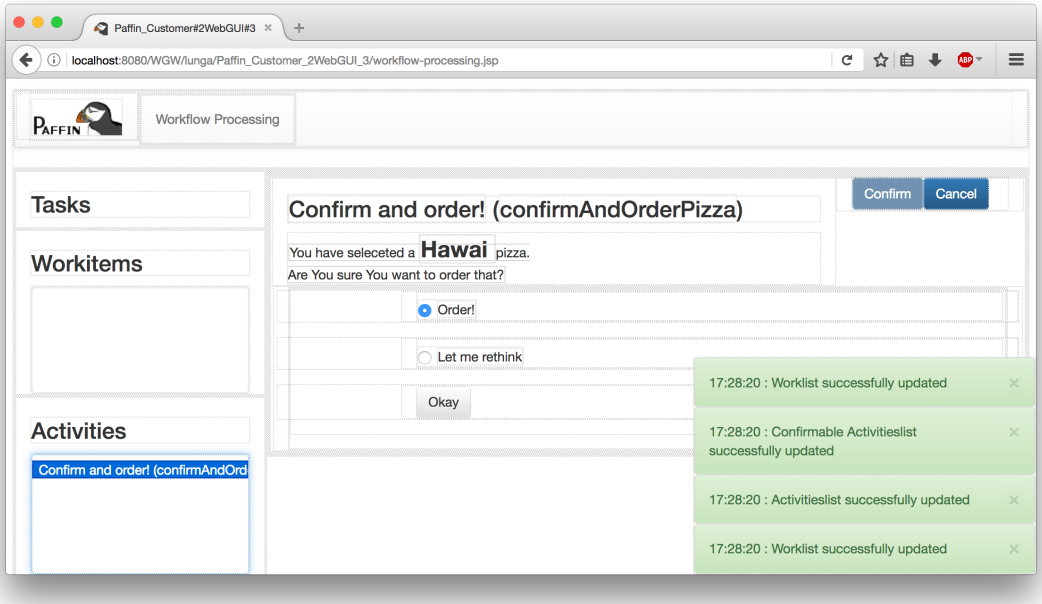

Figure 11.18: Pizza collaboration: Screenshot of the confirm order task

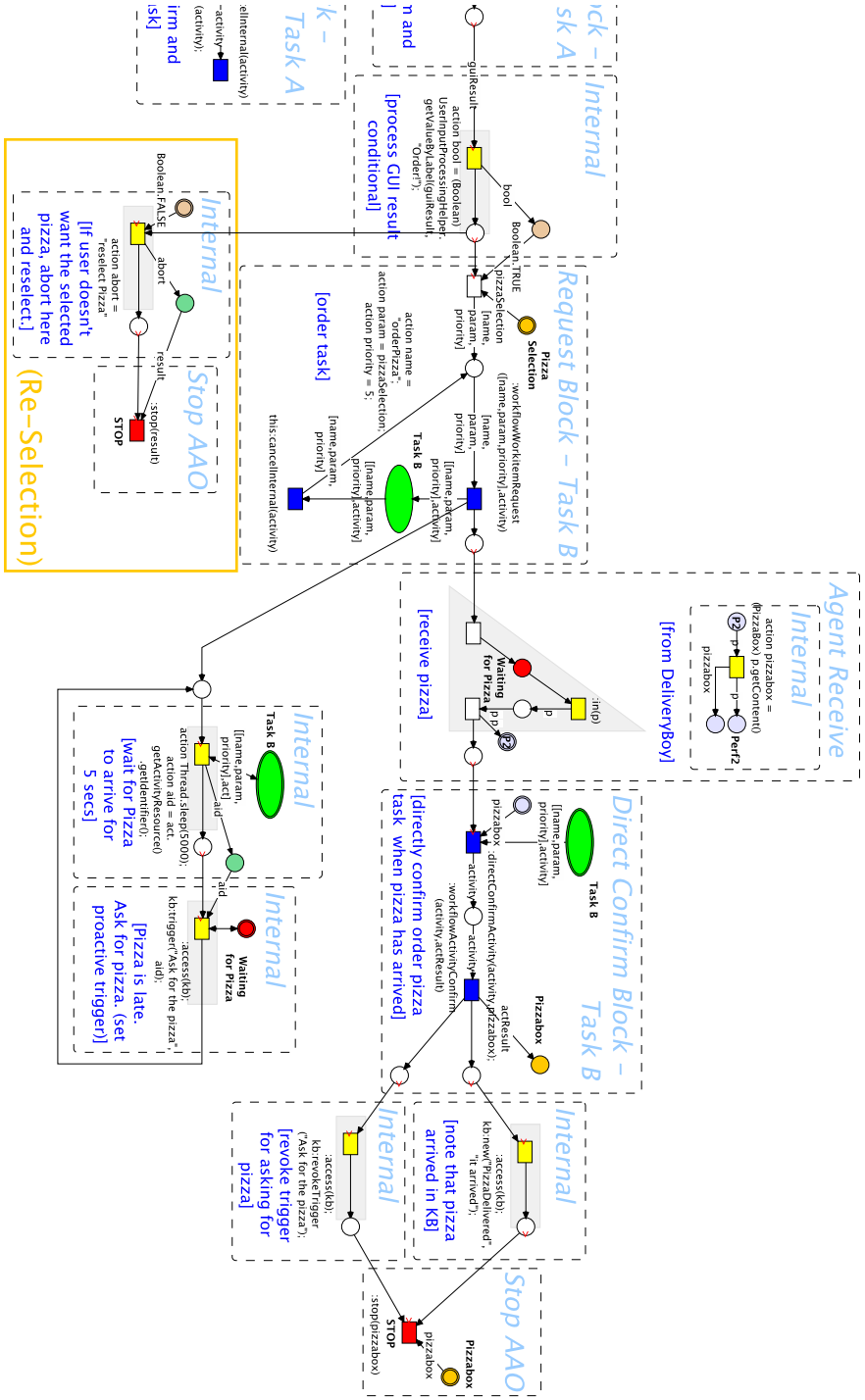

Figure 11.19: Pizza collaboration: AGENT-ACTIVITY for confirming and ordering (excerpt)

the customer can confirm the selection or choose to reconsider. A screenshot of that GUI can be seen in Figure 11.18.

When the selection in the GUI is confirmed, the input is processed. This is the first internal action on the left side of Figure 11.19, which is an excerpt of the internal process and shows part after this first workflow task. If the customer wishes to reconsider, the Agent-Activity is finished with that result. This part is marked by an orange border in Figure 11.19. In the process-protocol of Figure 11.17, the result is unified against the backwards loop reinitialising the original pizza selection Agent-Activity. This is an extension to the original BPMN model that was added to allow users to confirm their pizza selection.

Back in the ordering Agent-Activity, if the customer confirms his or her pizza selection, the upper branch unifies and another workflow task follows the internal action to process the GUI input. That task is intended for a pizza clerk. Once the relating workitem is successfully requested by a pizza clerk, the customer waits for an agent message containing the pizza box. This is the same as in the BPMN model. Additionally, as an internal agent action, the Paffin begins to wait a set amount of time. When that time has passed, the Paffin reads the identifier of the pizza clerk Paffin and stores that identifier as a proactive trigger for the *Ask for the pizza* Agent-Activity. At this point, the customer is waiting for a pizza box to arrive. For readability these descriptions assume this pizza box arrives now. What happens between the pizza clerk requests the workitem and the pizza box is delivered is discussed further down below. Once the pizza box arrives, the engine, i.e. the customer, autonomously and directly confirms the order pizza workflow task. The pizza box is the result of the task and handed over back to the customer process-protocol as the result of the overall Agent-Activity as well. Before that happens, though, any proactive triggers for the *Ask for the pizza* Agent-Activity are revoked and a new entry added to the knowledge base signalling that a pizza has arrived. The former prevents any more inquiries about the pizza from being started, while the latter aborts any running Agent-Activity asking for the pizza. In the customer process-protocol the result is recognised as a pizza box, thus enabling the *Pay for pizza* Agent-Activity. Before moving on to that Agent-Activity, the pizza order process in the delivery service and the asking for the pizza subprocess are described.

As stated before, while the customer is waiting for his or her pizza, a timer waits for a specified amount of time before creating a proactive trigger for asking for the pizza. In the process-protocol of Figure 11.17, asking for a pizza is modelled as a proactive Agent-Activity in a separate unique loop. The loop in the net is only necessary to ensure that a customer can only have one active inquiry.

The internal process of the *Ask for the pizza* Agent-Activity can be seen, in part, in Figure 11.20. The process starts off by reading the user login for dynamic permission, before providing a workflow task in which the user can enter his or her question in the GUI. That input is processed concurrently with reading data from the knowledge base and Agent-Activity parameter. This can be seen in the upper right side of Figure 11.20. That data contains the address of the pizza clerk that requested the original "order pizza" workflow task. From that data, the customer Paffin creates a message that is received by the clerk in question. The clerk, upon reception of the message, reactively triggers the Agent-Activity to calm the customer. This Agent-Activity merely reads the received data, randomly chooses from a set of answers and sends the chosen answer back to the customer. That message is received in the net from Figure 11.20, where it is processed for the answer. To conclude asking for the pizza, another workflow task is provided to the customer, which merely displays the answer of the delivery service in its description. The

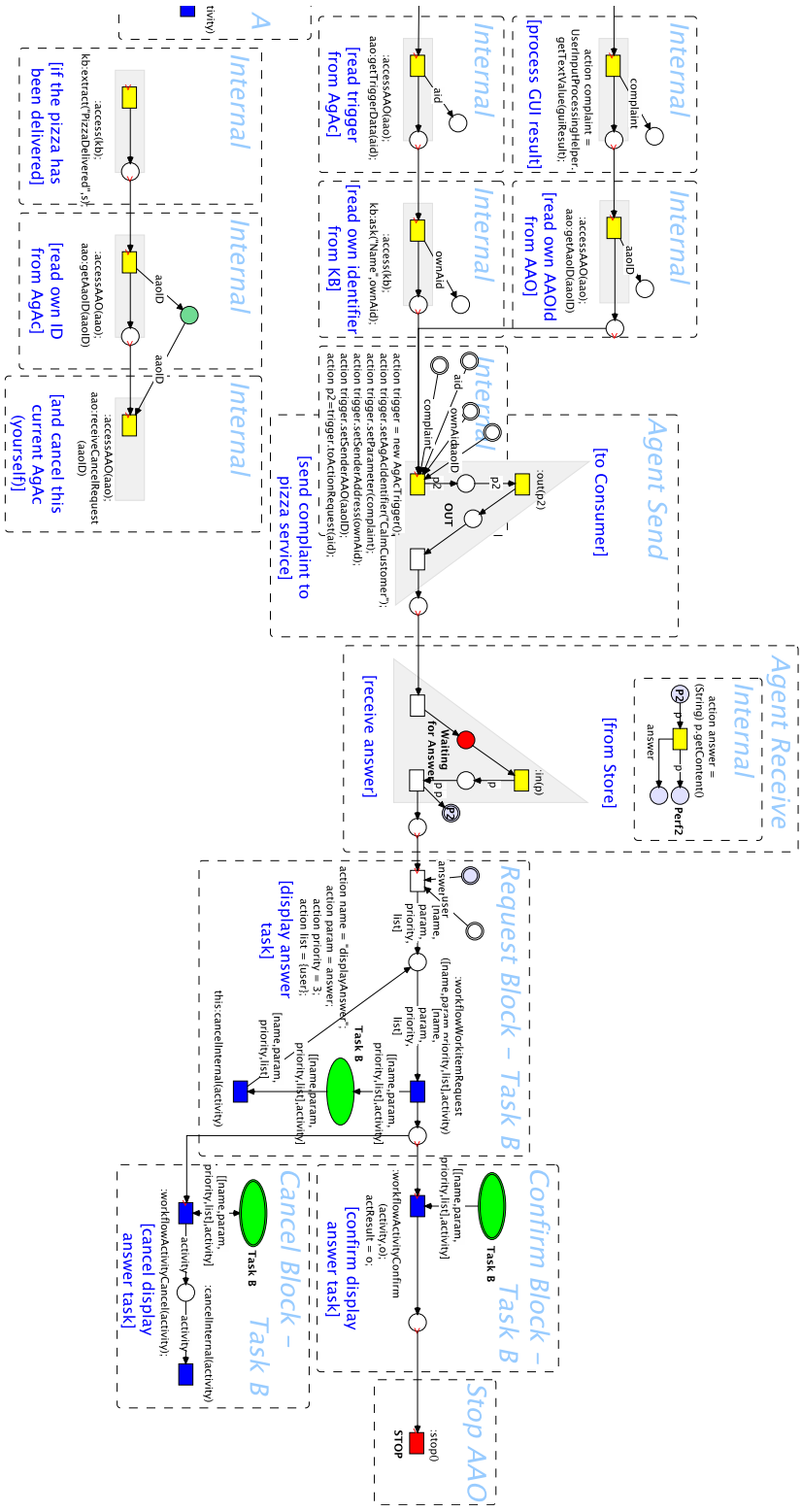

Figure 11.20: Pizza collaboration: AGENT-ACTIVITY for asking for the pizza

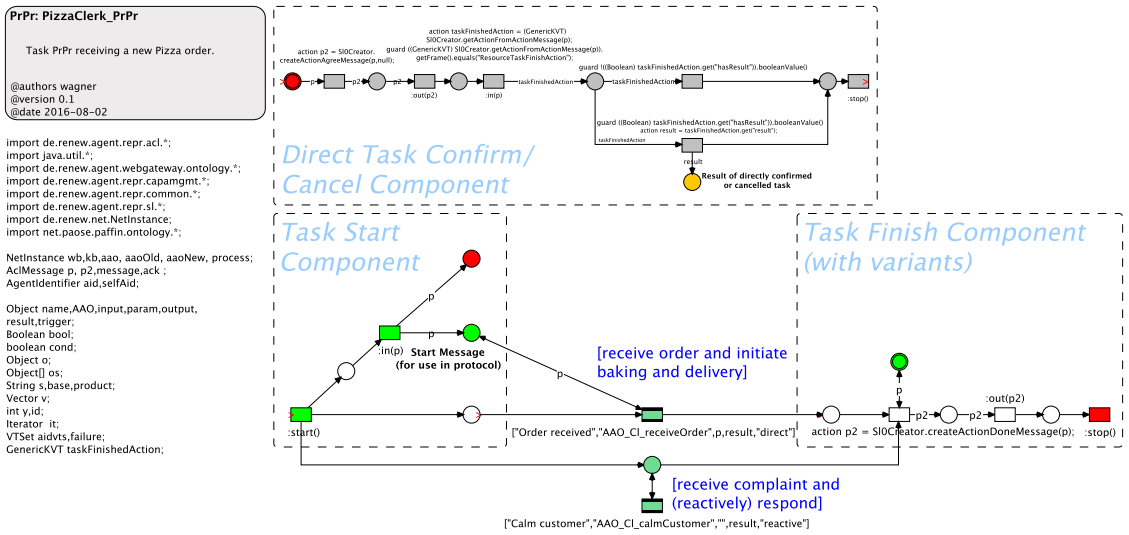

Figure 11.21: Pizza collaboration: Pizza Clerk process-protocol

display is dynamic, meaning it always shows the current answer, using one of the variables available for task descriptions. Additionally, that task is also automatically assigned to the customer, thus modelling the display of the answer directly without the need for another manual request. Once the answer is confirmed by the customer, the AGENT-ACTIVITY can be finished.

The AGENT-ACTIVITY for asking for the pizza can also be aborted. As described above, when the pizza arrives, the ordering AGENT-ACTIVITY sets a flag in the knowledge base. If that flag is available, the lower right side of Figure 11.20 becomes active. It extracts the flag from the knowledge base, reads its own AGENT-ACTIVITY ID and aborts itself by calling the corresponding synchronous channel with its ID. This terminates the AGENT-ACTIVITY fully. As the flag is only set in connection with any proactive triggers being revoked, any other inquiries about pizzas are impossible until the customer decides to order another pizza.

Once the task to order a pizza is provided by the customer, any pizza clerk in the system can request it. When it is successfully requested, the process-protocol shown in Figure 11.21 is started reactively. This process-protocol contains only two AGENT-ACTIVITIES. The first models the reception of the pizza order and subsequent forwarding to pizza chef and delivery boy. The second one models the reaction to a customer asking for his or her pizza described above.

The reception of a pizza order AGENT-ACTIVITY is kept simple. The clerk merely reads both the parameter, i.e. the customer's pizza selection, and its own pizza service identifier. It then proceeds to provide a "bake pizza" workflow task for pizza chefs using the pizza selection as parameter and the pizza service identifier as a dynamic permission. The latter ensures that only chefs of the same delivery service can see, request and work on that task. Until the chef confirms the task, the clerk waits at this point.

When a pizza chef requests a "bake pizza" task, a simple process-protocol containing only one AGENT-ACTIVITY is started. That AGENT-ACTIVITY models the baking of a pizza. In the first step, the required pizza type is read from the task parameter. Following that, the pizza chef PAFFIN uses an application DC, called the `Kitchen_DC` to bake the pizza. Depending on the selection, different pizzas are created. The baking also takes a semi-randomised time. In the end, a pizza object containing different ingredients is created

and handed back to the Agent-Activity for baking the pizza. The pizza chef then only packages the pizza into a new pizza box object and then uses that pizza box as parameter for a "deliver pizza" task for a delivery boy of the same delivery service. Again, until the delivery boy confirms that task, the chef waits at this point.

A delivery boy Paffin requests a "deliver pizza" task and also starts a simple process-protocol containing only one Agent-Activity. The process of that Agent-Activity contains only agent actions. First, the pizza box (from the parameter) and additional delivery data (from the knowledge base) are read. Then, using that information, the pizza box is sent, as an agent action, to the customer. Here it is received, as previously described by the ordering Agent-Activity, which can then finish its execution. After sending the pizza box, the delivery boy Agent-Activity waits for the payment from the customer. The delivery boy expects that payment as an agent message. That agent message is sent out in the *Pay for pizza* Agent-Activity in the customer process-protocol from Figure 11.17.

The internal process of *Pay for pizza* can be seen in Figure 11.22. It starts by reading the user login for dynamic permissions and the received pizza box. Then, a workflow task is provided using the pizza from the pizza box as a parameter. In the GUI associated with the task, the customer can enter what he or she wishes to pay for the pizza. A screenshot of that GUI can be seen in Figure 11.23. Once the user confirms, the payment input it is processed in the Agent-Activity. The delivery boy identifiers are read from the pizza box object and the payment sent to that Paffin.

Once the delivery boy Paffin receives the payment in its Agent-Activity, it processes it and creates a receipt object, which it sends back to the customer. Afterwards the delivery boy finishes its Agent-Activity, thus completing its overall process-protocol and also confirming the task provided by the pizza chef. The pizza chef in turn can then finish its own Agent-Activity and process-protocol, confirming the original task from the pizza clerk. The pizza clerk can then also confirm its task, Agent-Activity and process-protocol. Thus, the confirmation of the delivery boy task and subsequent (successful) termination of the process(-protocol), cascades up to the pizza clerk. In terms of the BPMN model from Figure 11.16, this cascading termination conforms to the terminate end event after the pizza is delivered.

While the pizza service actors and Paffins are terminated at this point, the customer must still finish its process. It receives the receipt created by the delivery boy and finishes the *Pay for pizza* Agent-Activity with both the pizza box and receipt as the results (wrapped in a tuple object). The customer process-protocol from Figure 11.17 can then start the final *Eat the pizza* Agent-Activity with pizza box and receipt as parameters. That Agent-Activity starts off with a workflow task (with dynamic permissions for the current user), which displays the receipt. When that task is confirmed the pizza object is processed, i.e. eaten, by the Paffin entity, before the Agent-Activity is finished.

As a final minor change to the original model, finishing the *Eat the pizza* Agent-Activity reinitialises the *Select a pizza* Agent-Activity, thus enabling a loop from selecting a pizza to eating it. This was done only for testing and demonstration purposes.

Discussion and Concluding Evaluation

In general, the example implemented here is a simple inter-organisational scenario. Customer and pizza delivery service collaborate modelled as organisations. The delivery service employs three employees that themselves collaborate to handle all tasks involved in delivering a pizza. The prototype implementation of that scenario showcases how such inter-organisational scenarios may be handled by the Agent-Activity. Both agent and workflow mechanisms are used throughout the prototype to realise both internal logics and interactions between the different involved actors.

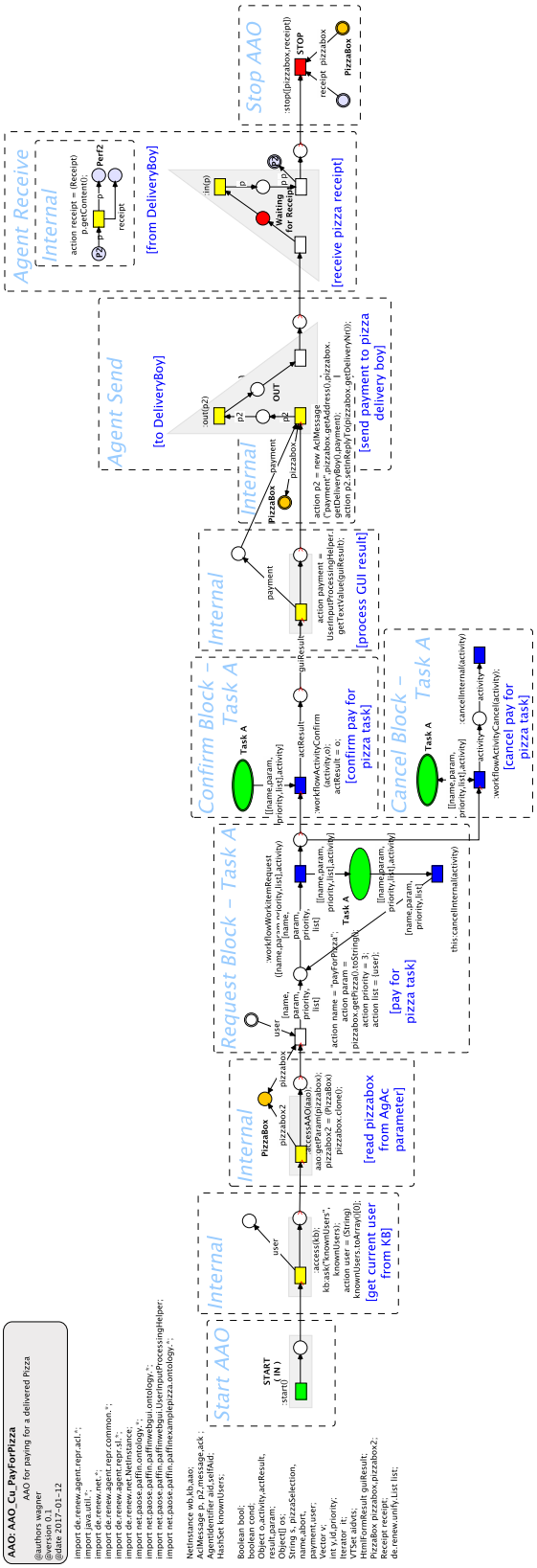

Figure 11.22: Pizza collaboration: AGENT-ACTIVITY for paying for the pizza

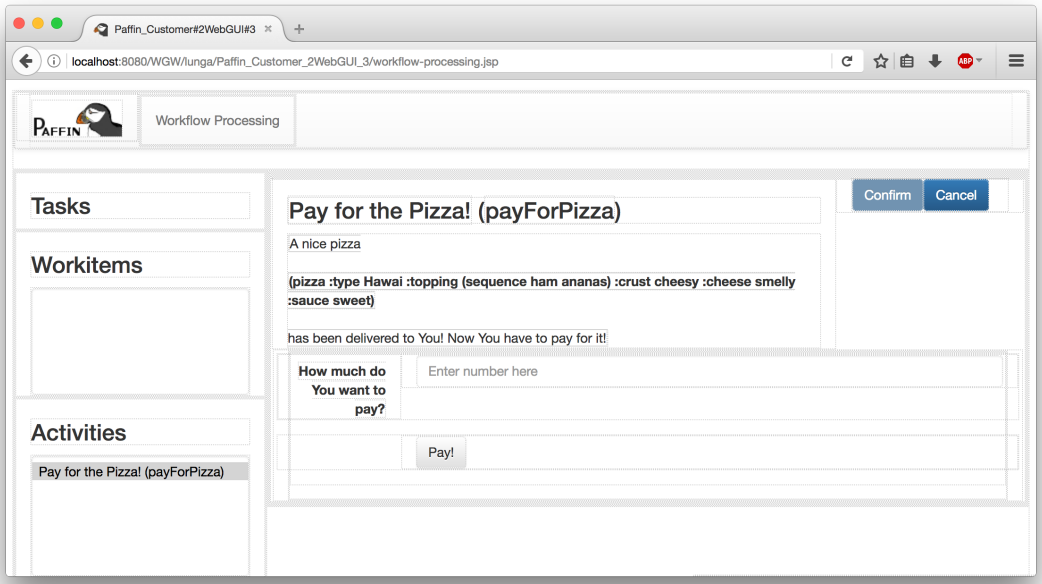

Figure 11.23: Pizza collaboration: Screenshot of the pay for pizza task

One of the most important aspects of inter-organisational collaboration is the encapsulation of the individual organisations and processes. In the prototype this encapsulation is realised directly through the Paffin entities. Each Paffin (role) in the prototype represents one of the actors of the inter-organisational BPMN model. The three delivery service actors are united via a shared supplementary role. Each Paffin then encapsulates the processes and functionality required by it. Only the customer can order pizzas, only the clerk can receive orders and calm the customer, only the chef can bake a pizza and only the delivery boy can deliver the pizza and receive payment. If a Paffin doesn't have a specific role it doesn't have access to the processes, functionality or data associated with that role. This ensures a complete encapsulation of critical data. In more complex, real-world applications this encapsulation still needs to be supported by security mechanisms, yet these are outside of the scope of this thesis.

Communication is another important aspect of inter-organisational contexts. Here, the prototype chooses from agent and workflow mechanisms wherever they fit best. Interactions with the user are naturally always modelled as workflow tasks. By using the feature of dynamic permissions, these tasks are always associated with the correct user. Interactions within the delivery service are also modelled as workflow tasks. Here as well, dynamic permissions are used to ensure that collaboration happens only within one organisation (delivery service). Agent-like message interactions would have been technically feasible here as well, but considering the scenario as a task-based collaboration between the employees of the pizza service is more reasonable. The involved Paffin entities represent different actors. Managing the work of those actors is far better captured by workflow tasks, then by rigid agent interactions.

Regarding dynamic permissions, these are controlled by the internal logic of the Paffin and are, as discussed, often associated with internal agent actions. Of course, in this simple scenario these agent actions relate to accessing the Paffin knowledge, but in more advanced scenarios this mechanism is feasible to utilise more complex agent actions

and agent intelligence. Consequently, the dynamic permissions can be classified as an integration mechanism, combining agent intelligence and workflow operations into an advanced task-related mechanisms. That mechanism ensures, based on dynamic PAFFIN knowledge, that additional restrictions to workitem and activity execution can be applied beyond the static ones defined in advance at modelling time. This is a clear example of an integration mechanisms practically applied to an inter-organisational application scenario providing directly observable benefit.

Other communications and interactions are also handled via agent communication. The customer asking for the pizza and the clerk calming him or her are good examples of situations, where a workflow task would not quite fit as an implementation. In the real-world the customer would simply call the clerk and ask directly. There would be no task to be processed. Another example is the delivery of the pizza and subsequent payment. Here, as well, tasks would not fit what is, in essence, a direct interaction between the actors.

Overall, the ability to choose agent, workflow or integration mechanisms emerges as a very beneficial capability of the PAFFIN-System at this point. It allows for using the best tool in any given situation. It is also available without switching between views, perspectives, artefact types, models or anything of the like. Agents and workflows are seamlessly interconnected, thus keeping the model and implementation clean and understandable. Take, for example, the actual delivery of the pizza. Here, an agent action delivers the pizza, which leads the customer PAFFIN engine to directly confirm the task, ask the user for payment information as a workflow task and send that payment to the still waiting delivery boy PAFFIN as an agent message. That PAFFIN then internally creates a receipt and sends that, again as an agent message, back, thus completing both the overall process of the delivery service and finishing the AGENT-ACTIVITY in the customer. In this complex back and forth between agent, workflow and hybrid in two PAFFIN entities the main description level remains the same: A combination of agent actions and workflow operations within different AGENT-ACTIVITIES.

This aspect is also one of the points where the PAFFIN-System and the AGENT-ACTIVITY are set apart from traditional agent and workflow modelling. The scenario could be implemented using either of the two traditional ways. However, the workflow side misses details about the implementation, while the agent side misses the process perspectives.

The BPMN workflow model from Figure 11.16 shows the overall process. However, details about the functionality of the tasks and the concrete data flow between some of the tasks are missing. The model is also not executable directly, meaning that it would have to be implemented in another kind of system. The PAFFIN-System maintains the process overview to a degree. The main customer workflow can be recognised in the customer process-protocol in Figure 11.17. While there is no completely global view, each other subworkflow in the delivery service actors can be regarded individually (even though these are quite simple in this scenario). Each task of the BPMN model can be recognised in the AGENT-ACTIVITIES of the prototype. With the exception of the reception of a new pizza order, also no further AGENT-ACTIVITIES are necessary. The latter is, however, necessary as the forwarding of the order to a pizza chef and the preparation for calming a customer need to be implemented somewhere.

In contrast to the BPMN model, the PAFFIN-System adds, however, the actual execution and support of the scenario. It contains all the functionality in a clearly distinguishable way, i.e. within the AGENT-ACTIVITY processes. That functionality, besides the actual workflow tasks, explicitly contains all the processing and communication (agent) actions. These actions are what actually realises the work associated with the overall workflow and are not available within the BPMN model.

On the other hand, when compared to the Paffin-System implementation a potential agent implementation would also miss a number of things. As previously discussed, agents do not possess any abstract process view. This is available in the Paffin-System and provided by the Agent-Activity arrangement in the different process-protocols. An agent system would also have to manually incorporate a subsystem to distribute the work between the different actors. This is handled, in the Paffin-System implementation, by the workflow management mechanisms within the individual Paffin entities and the platform WFMS.

Basically, the Paffin-System combines the process perspective and task distribution mechanisms from a workflow model with the functionality and communication implementation and details from an agent implementation. To that it adds additional integration mechanisms, like the dynamic permissions, the direct engine operations and the ability to reactively and proactively trigger Agent-Activities, which are both utilised in the scenario. In conclusion, the implementation of this scenario has shown not only how to approach inter-organisational settings in the Paffin-System, but also how and where the Paffin-System has advantages over traditional agent or workflow modelling. Both of these points are picked up again in later discussions.

11.3.3 Paose Teaching Support Prototypes

This section presents the Paose teaching support prototypes. This group of prototypes was developed for and deployed in the yearly Paose teaching project hosted by TGI in the winter term of 2016/2017. In the first part of the teaching project the students work on a set of worksheets designed to teach them the basics of Petri net modelling, agent-orientation in general, Paose and the related toolset. The worksheets were previously updated, redesigned and modelled as workflow nets by Dennis Schmitz [Schmitz, 2016, Schmitz et al., 2016]. The prototypes presented here are implementations of parts of those workflow nets in the Paffin-System. Developing, deploying and supervising the use of these prototypes was a collaboration with Dennis Schmitz that followed the creation of the `PaffinWebGui` plugin (see Section 10.2.12). As with the creation of the Web GUI, the work on these application prototypes was about equally shared between Mr. Schmitz and the author of this thesis. Parts of these prototype examples were also discussed in [Wagner et al., 2016a].

The purpose of these prototypes was not so much testing the integration features and mechanisms available. Rather, it represented a chance to test the Paffin-System in an actual real-world setting, where users, i.e. the students, who were never involved with the development would be "let loose" upon the system. These users possessed a fresh and impartial view of the system and their feedback helped the developers improve upon the system in general. The deployment on multiple computers concurrently with multiple users also exposed technical bugs and shortcomings that could be directly fixed and retested by the developers.

The teaching support prototype group contains three prototypes overall. Each prototype realises one exercise from a worksheet from the Paose teaching project. Each prototype is, in that way, a test case of the Paffin-System in the Paose teaching project. Through the implementation, the workflow contained in that exercise was supported by the Paffin-System. The implementation's main goal was to guide the students through the exercises and provide them with all the information they required to successfully complete them. The prototypes were developed in such a way that for three consecutive weeks, the students would have one exercise of a worksheet be supported by the Paffin-System.

Test Case: Paose Worksheet 4, Exercise 1

One of the goals of the PAOSE teaching support prototypes was always to test the prototype in the PAOSE teaching project as soon as possible. Testing the system in the application environment it was being designed for would validate already made and implemented design decisions, identify issues and improvement options and generally allow the developers to adjust the current and future development of the system accordingly. The first test was executed once the base GUI functionality required for the PAOSE worksheets was implemented in the PAFFIN-System. Within the teaching project this corresponded with the beginning of the fourth worksheet.

The exercise of the worksheet modelled for this test dealt with the introduction of the MULAN WEBGATEWAY (see Section 2.2.3). The students should use the DCs and agent protocols from an earlier worksheet that implemented a chat application between agents. In the first exercise these DCs and agent protocols should be prepared and adapted for use with the WEBGATEWAY. The process-protocol, shown in Figures 11.25 to 11.27, models the workflow and error handling for this exercise. It contains thirteen tasks from the worksheet, as well as five tasks representing advice and hints from the textual version of the worksheet (see the description about workitem assignment below).

Test Development The core task in developing this test case was modelling the workflow for worksheet four exercise one as a process-protocol and providing access to the system for the students. Besides modelling the actual process-protocol, the different AGENT-ACTIVITIES for the task types had to be designed, as well as the initial database filled with the specific workflow tasks for the exercise. To provide students access to the system a bash script for the deployment on the project computers was created, which checks out the PAFFIN sources from the repository and builds them on the local machines.

During the modelling of the core content of this test, a number of supplemental features were identified and implemented. Note that the specific functionality and internal workings of these features were already described in the description of the prototypes. The following presents only the basic descriptions and reasoning behind the changes in the application context they were designed in.

HTML form support: The overall functionality for modelling and (at runtime) creating HTML forms (i.e. buttons, text input fields, etc.) was improved during the creation of this test case.

Radio button GUI support: To better capture tasks that involve user input of predefined type and exclusivity the support of radio buttons in the GUI was added.

GUI activity result classes: Up until this test case, an activity executed by a human user received the GUI ontology object as a result. This resulted in increased overhead and was cumbersome to deal with during modelling for processing the results. New ontology objects representing results obtained from the HTML GUI were created. A sequence of these result objects is now given as the activity result, instead of the entire GUI. This addresses the previously mentioned issues.

Variables in task title and description: Static task titles and descriptions prevent efficient reuse of tasks. In some cases, tasks only differ in their parameters. Incorporating variables for parameters and other runtime data allows for a more flexible reuse of basic tasks and also helps users distinguish multiple identical tasks with different parameters. Variables for most task data fields have been added for this test case and can now be used in task titles and descriptions.

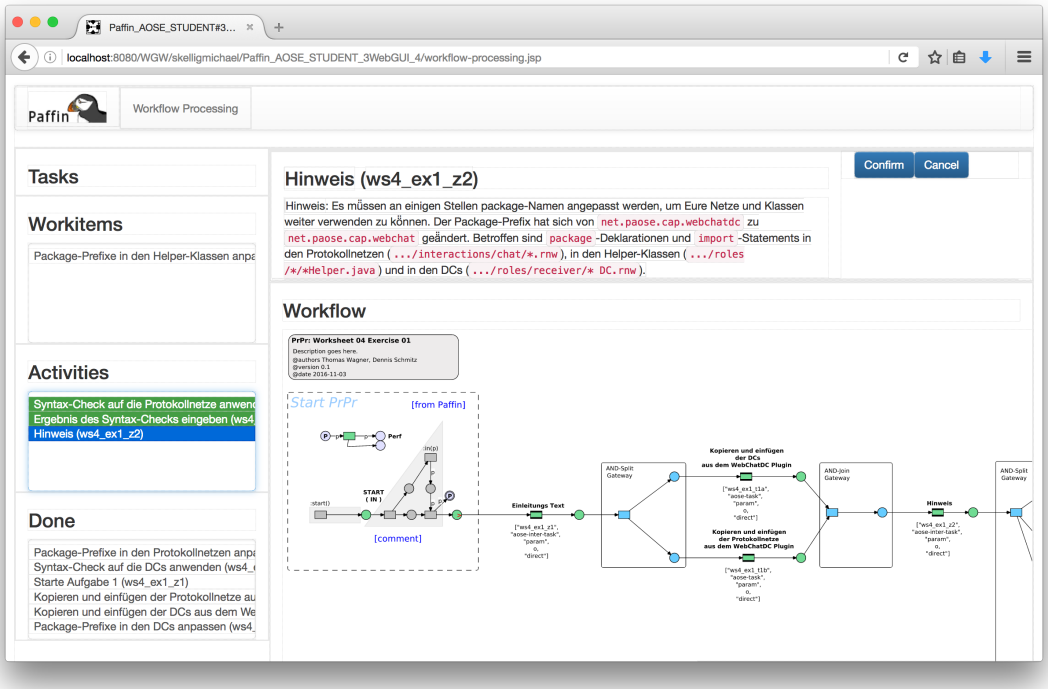

Figure 11.24: Worksheet 4, Ex. 1: Screenshot of the GUI

Automatically assign workitems (workitem push): While modelling the process-protocol for the exercise, a discrepancy between the textual and workflow representations of the worksheet were noticed. General text passages between tasks, as well as hints or pieces of advice were not represented in the workflow. However, these text passages are still needed to successfully process the worksheet. They would, however, be needed to be handled differently from the actual tasks. It was decided that they should not appear as workitems for users, but should automatically appear as activities. That way, they would be distinguished from the regular tasks, while still being available for the students at runtime. In order to realise this, the workitem push mechanism was implemented, which automatically assigned these workitems to the students at runtime (as opposed to the students "pulling" or requesting the workitems). Figure 11.28 shows an AGENT-ACTIVITY process for such an advice task. When the request becomes activated, the WFMS automatically recognises the task to be assignable, determines the correct resource and assigns the task. Since these advice tasks are not really tasks, they do not need to be able to be cancelled. Consequently, only a confirmation block is added after the request.

Logging mechanism: In order to evaluate and trace any technical issues with the prototype some kind of logging mechanism became necessary. This logging was implemented in the platform WFMS and logs all workflow related events going through the WFMS between engines and platforms (e.g. the basic workflow operations, worklists, activity lists, etc.). These events are now written into a logfile in the current user's home folder.

GUI beautification: A number of improvements to the general presentation of the PAFFIN-System to human users were made. These can partially be seen in Figure 11.24. For examples, a logo was created and added to the upper left corner. Also, HTML

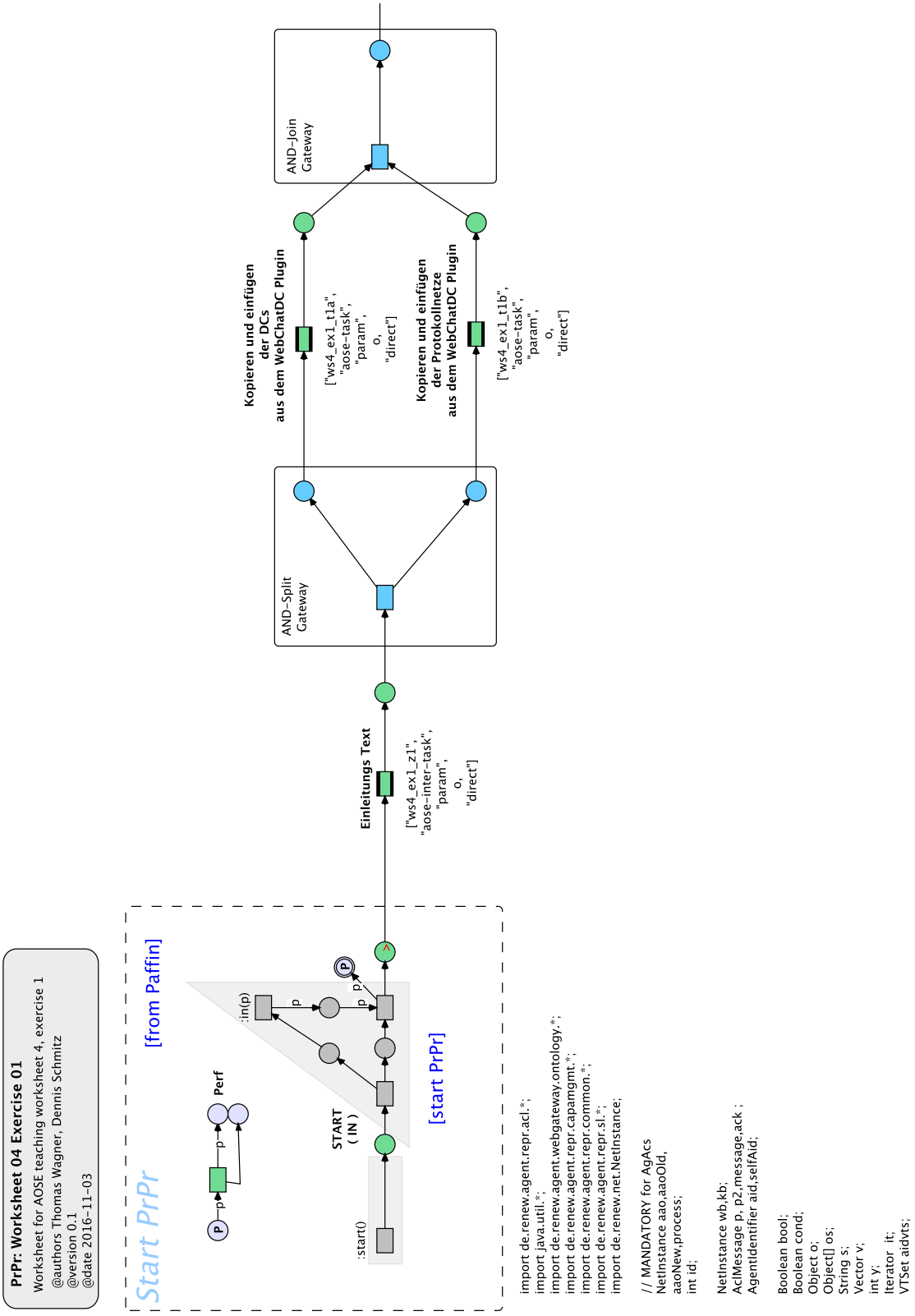

Figure 11.25: Worksheet 4, Ex. 1: Process-protocol for test deployment (part 1)

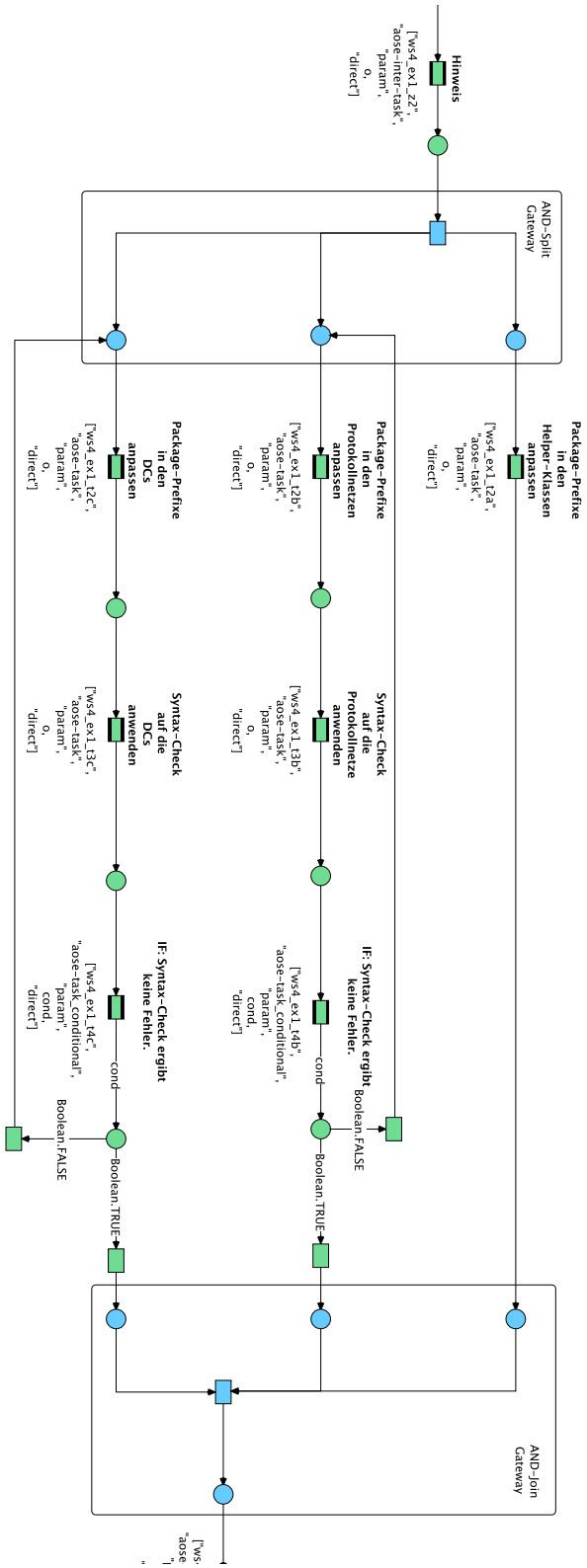

Figure 11.26: Worksheet 4, Ex. 1: Process-protocol for test deployment (part 2)

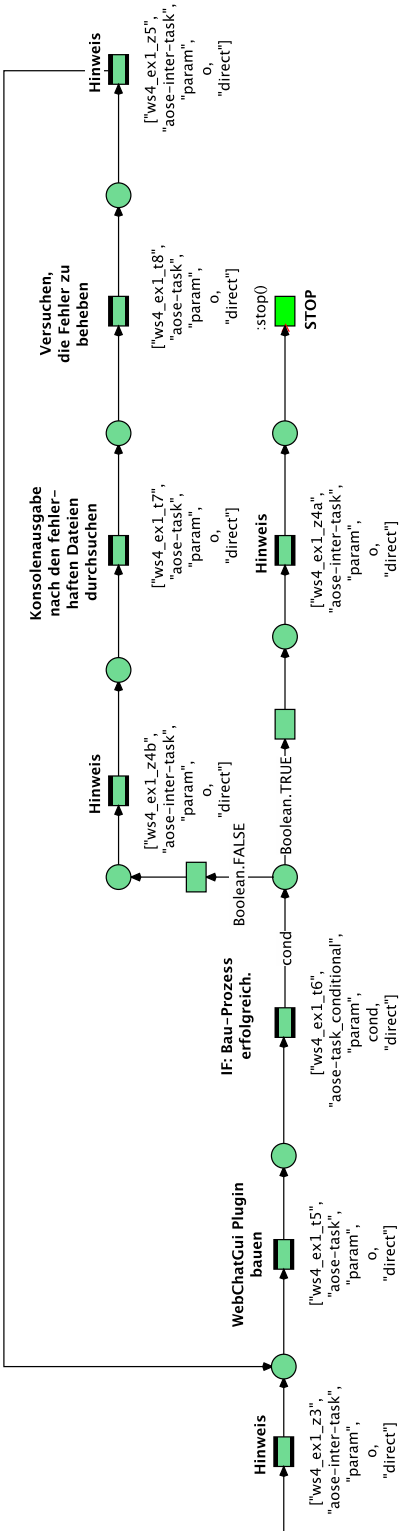

Figure 11.27: Worksheet 4, Ex. 1: Process-protocol for test deployment (part 3)

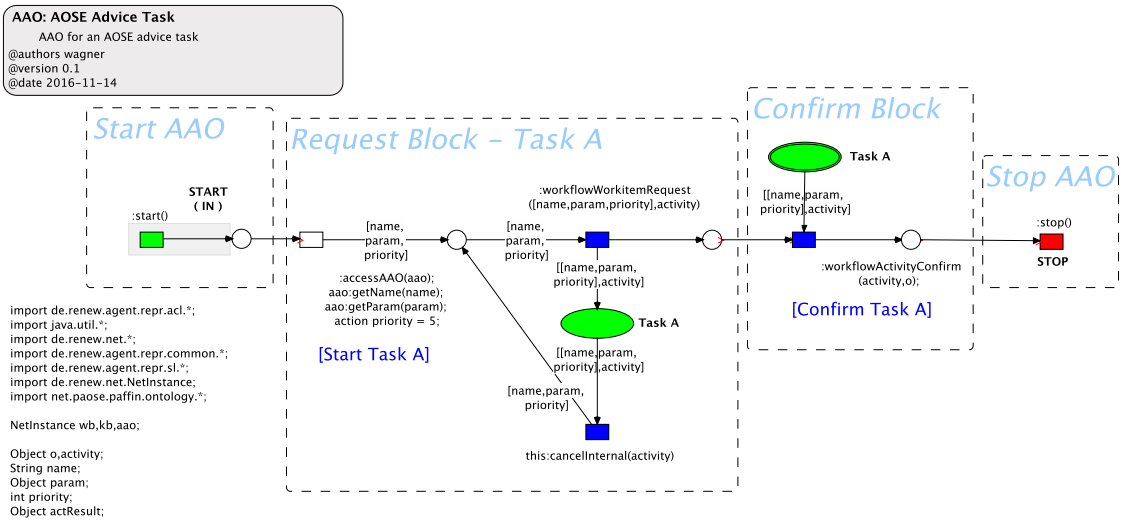

Figure 11.28: Worksheet 4, Ex. 1: PAOSE advice task process

text formatting was tested in task descriptions and incorporated in the tasks of the worksheet (e.g. the text highlighted as `<code>` in Figure 11.24).

Test Execution The test was executed when the students began working on worksheet four. Two teams of two students each worked on the worksheet on separate computers. Additionally, the system was tested by colleagues and other interested students present in the project room. The PAFFIN-System development team supervised the students and explained how to use the PAFFIN-System. When questions or issues arose, it was possible to quickly react and provide an answer or fix.

The application was set up to use a script that started the PAFFIN platform, as well as one PAFFIN executing the exercise process-protocol and one PAFFIN representing the user and providing the GUI. After starting the system with the script, the students could access the GUI in a web browser (address `http://localhost:8080/`). There, the students could navigate the WEBGATEWAY plugin to the PAFFIN providing the GUI. Login credentials (Username: *Student*, Password: *Student*) were provided to the students on a wiki-page in the PAOSE project Redmine[4] management environment. Once logged in, the students would receive workitem and activity updates from the PAFFIN executing the exercise process-protocol. The students worked with the system until they were finished with all tasks from exercise one. Afterwards they would switch back to the textual, paper version of the worksheet for the remaining exercises.

Test Evaluation Generally, the PAFFIN-System was well received by the students. After some initial confusion as to how workflow management works (e.g. what is an activity and what do I do with it?) the students could process the exercise. The students said that being supported by such a system was helpful, though a number of possible improvements were directly suggested by them.

User guidance: As mentioned, some of the students had issues with generally working with a WFMS. They had not yet worked with or studied this particular context and didn't

[4] `http://www.redmine.org/` (last accessed May 28th, 2017)

know what to do. For future tests the modelling team decided to better guide users with more instructive task descriptions, as well as changing GUI behaviour in some areas. It is, for example, feasible to highlight confirmation buttons for confirmable activities or create alert messages during long periods of inactivity.

Assign workitem based on resource: During discussions about the assignment of workitems (workitem push) it was noted that a workitem push based on the resources could also be an interesting and useful feature. A user or automatic resource might set a flag that is available for any workitems it would be allowed for. The WFMS would then attempt to assign any workitem that resource is eligible for. This functionality would mostly use mechanisms implemented for the regular assignment of workitems and was consequently implemented shortly after the test.

Workflow overview: One issue the students directly noticed was that, while helpful in general, the system lacked an overview of the overall progress within the workflow. The students and users in general, are only presented with the available workitems, the current activities and the activities they have finished so far. An indicator of how many more tasks are in the workflow or how different concurrent tasks are related to one another (e.g. how many and which concurrent tasks need to be finished before the next bigger step can be executed) was desired. As a direct, but short-term solution, the modelling team added a static illustration of the workflow to the GUI. This can be seen in the lower right part of Figure 11.24. In future work the display of the workflow should also include the current status of the workflow, as well as further information to help students/users.

Enduring activities: One group of students noted that the advice tasks, i.e. the tasks representing text passages from the textual worksheet not directly associated with actual tasks, would be more helpful if they were displayed longer. A number of them provided information that would be useful in following, later tasks. However, in the implementation of this current test, they disappeared from the activity lists after being confirmed.

To improve these advice tasks a number of options were discussed. It would be possible to model these tasks as concurrent to the ones they supported. However, the issue here was that users would have had to manually confirm the advice/hint when it wasn't needed anymore. Automatically firing this task from the engine once other tasks were completed would also have been possible, however, this behaviour would have been counterintuitive for modellers/monitors to observe as the task would suddenly be completed.

The option that was chosen to be implemented was to allow a task to be started in one Agent-Activity and be finished in another one. This way it could be directly confirmed by the engine, while maintaining a clear control flow in the process-protocol. Created during the test so that it could be directly retested by the students, the implemented change uses two Agent-Activities instead of one for the advice tasks. The resulting process-protocol is shown in Figure 11.29, which shows the same part of the original process-protocol as Figure 11.26. Red borders highlight the two Agent-Activities for the new advice task. The first one starts the task and provides the activity information as its result back to the process-protocol. The internal process is shown in Figure 11.30. The second on, whose process is shown in Figure 11.31 updates the activity data in the backend so that it can fire the confirmation. This update is the only functionality that needed to be implemented in order for the enduring tasks to work in the Paffin-System.

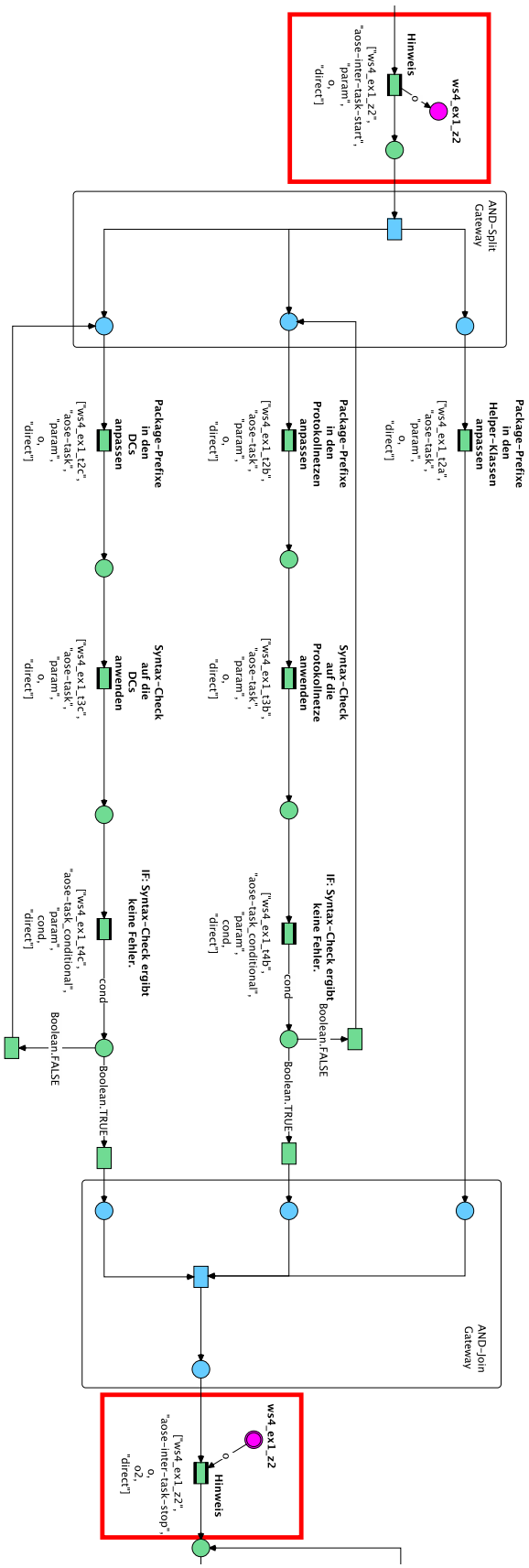

Figure 11.29: Worksheet 4, Ex. 1: Process-protocol after test deployment (excerpt)

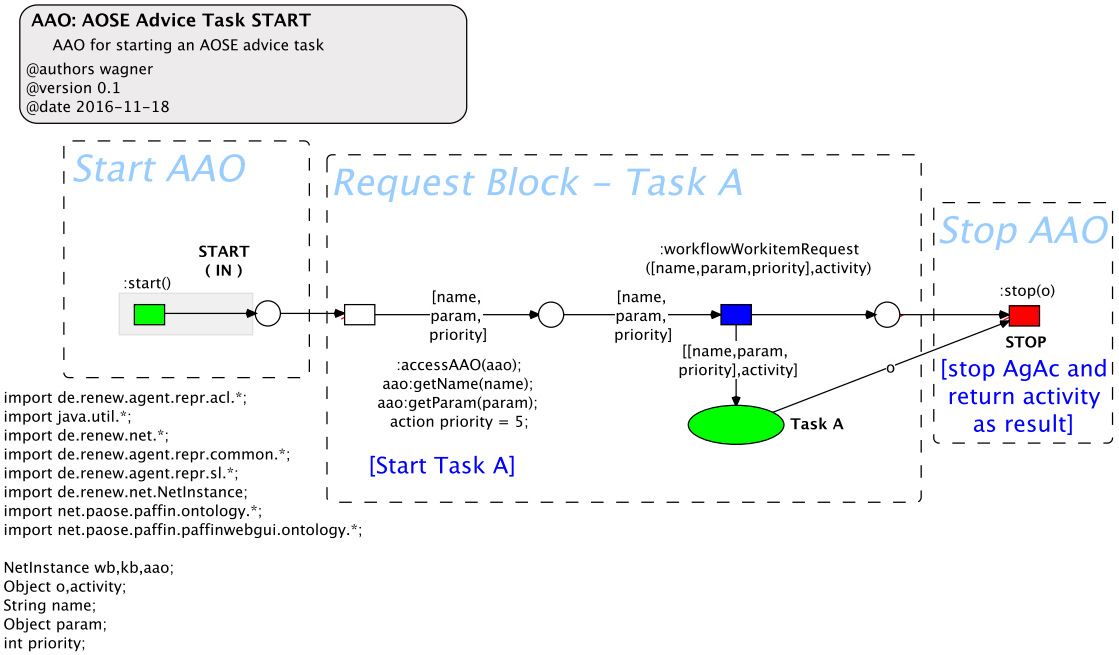

Figure 11.30: Worksheet 4, Ex. 1: PAOSE advice task start process

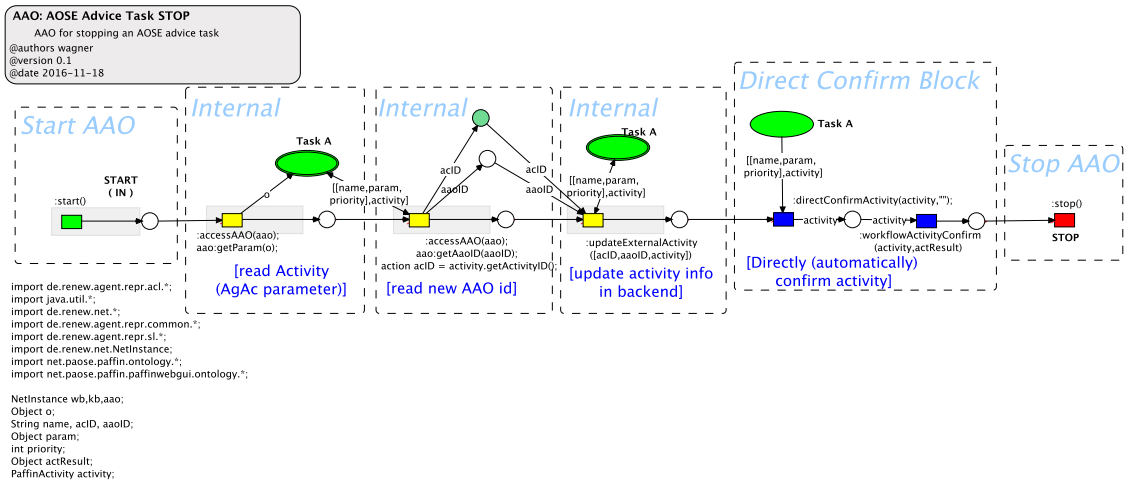

Figure 11.31: Worksheet 4, Ex. 1: PAOSE advice task stop process

> One issue with this implementation of enduring tasks relates to cancelling tasks. In general, the activity can be cancelled, but since the request block, which would be reactivated in case of cancellation, is in another Agent-Activity that no longer exists it can't be reset. The only possibility here is that the process-protocol reinitialises the original Agent-Activity, which then creates a new workitem/activity. This would have to be modelled manually by the application modeller. In the Paose teaching support application, this behaviour is not needed as the advice tasks don't need to be cancelled at all. An automated handling of enduring task cancellations represents a special use case and is part of future work.

From a technical standpoint, the test was also a success. No major technical problems were discovered and the system could be tested and executed by all interested participants. While developing and executing the test a number of minor bugs were discovered and fixed. This included activity lists that were not handled correctly and could be corrupted under certain circumstances, as well as status updates in the GUI that were erroneous.

Test Conclusion Overall this test and first deployment of the Paffin-System in the teaching project can be considered a success. The system was overall well received by the students and the modelling team received valuable feedback about design decisions and future improvements.

Regarding integration mechanisms, this test case was very workflow-oriented. It basically realised the existing workflow of the exercise in the Paffin-System. The only areas, in which advanced and integration mechanisms were utilised, were the enduring activities described above. Here, the agent state and knowledge of a Paffin was used for advanced task control and management. The mechanism allowed a task to be extended beyond the scope of a singular Agent-Activity and automatically cleaned up after. However, this is only a minor improvement w.r.t. traditional workflow management. Since the focus of these test cases was more on a praxis test, the realisation of a more traditional workflow setting was acceptable for the time being.

Test Case: Paose Worksheet 5, Exercise 1b

The second test case was developed for the fifth worksheet. While the first test case already supported the execution of workflows associated with exercises extensively, one thing the students also wished for was an improvement to the project management support.

The students are required to log the time they spend on the exercises in the project Redmine. Exercises and tasks correspond to features and tickets in Redmine. To support the students in their project management tracking, Frederick Thomssen was, at the time of the tests, developing the so-called RedTimer tool in the context of his master thesis [Thomssen, 2017]. That tool provides the students with a simple application GUI containing a timer mechanism. When the students start the work on a Redmine ticket, they use the RedTimer to start the timer mechanisms. When they stop or pause working on the ticket they stop the timer in the RedTimer. At that point, the RedTimer automatically logs the spent time for the ticket in Redmine. Screenshots of the RedTimer GUI can be seen in Figures 11.34 and 11.37 (both taken in the context of the following test case).

During the first test case the week before, the students noted that it would be nice if the Paffin-System automatically executed RedTimer actions, as the tasks in the workflow correspond to the tickets in Redmine. For the current tests the modelling team coordinated with Frederick Thomssen and agreed upon a command line interface for the RedTimer that could be called from the Paffin-System process-protocols.

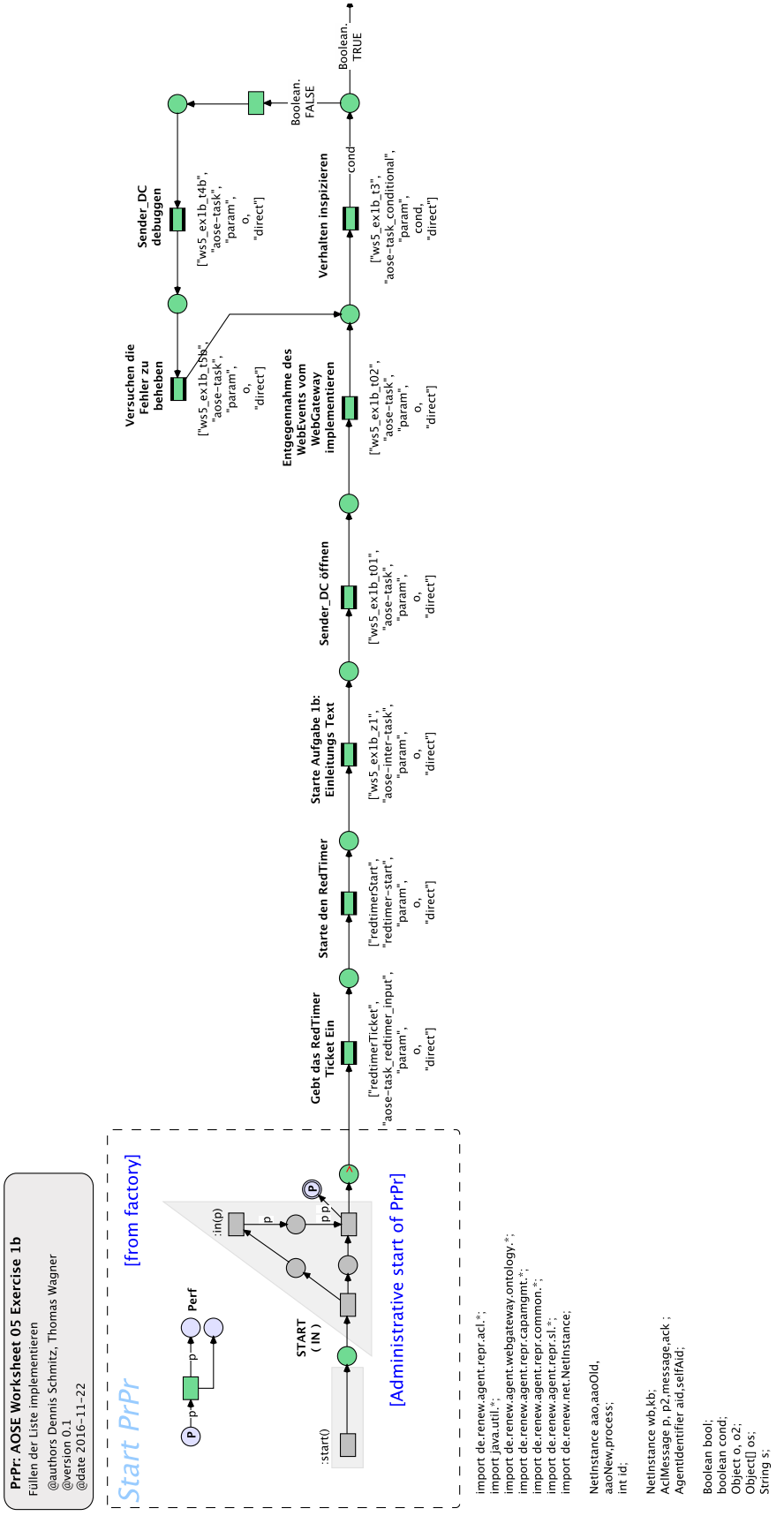

Figure 11.32: Worksheet 5, Ex. 1b: Process-protocol for test deployment (excerpt)

Test Development The technical goal of this prototype was to enable the communication between the RedTimer tool and the Paffin-System. On the RedTimer side, Frederick Thomssen provided a command line interface, as well as the technical deployment to incorporate the RedTimer into the command line. On the Paffin-System side, Dennis Schmitz and the author of this thesis worked on ways to call upon the command line from an Agent-Activity. The challenge here was not about improving the Paffin-System. Calling the command line command is an internal agent action. Rather, the challenge was how best to call the command line command from a reference net, which the internal process of the Agent-Activity is.

In the end, the simplest version on both the RedTimer and Paffin-System sides was chosen. The RedTimer interface merely required an (existing in the Redmine) ticket number. A call to that interface was forwarded to a running RedTimer instance, which was already configured regarding user name and context. On the Paffin-System side, the internal action merely called the current Runtime object and executed the call as a simple command with the ticket number as parameter.

In addition to the technical RedTimer incorporation and provision, an exercise for the fifth worksheet had to be modelled as well, of course. The topic of the fifth worksheet is still a simple chat scenario utilising the Mulan WebGateway, albeit now with a focus on using and editing the web GUI components. The second part of exercise one, dealing with the update and display of chat partners, was chosen for implementation as it is the most extensive exercise on the fifth worksheet. The resulting process-protocol can be seen, in part, in Figure 11.32.

Figure 11.32 shows that the process-protocol for the exercise starts off with two Agent-Activities not directly associated with the exercise workflow. The first Agent-Activity *redTimerTicket* prompts the students to enter the Redmine ticket number for the current exercise. That ticket number is stored in the Paffin knowledge base. Afterwards, the second Agent-Activity *redtimerStart* starts the timer and tracking with the RedTimer tool as described above. The internal process of that Agent-Activity can be seen in Figure 11.33. It is quite simple, only reading the ticket number from the knowledge base and then calling the command line interface with it. The rest of the test deployment configuration and parameters are unchanged from the first test case and are not described further.

Test Execution The test execution was almost identical to the first test. When the students began working on exercise one b, they switched to the Paffin-System. The RedTimer was already deployed and configured so that it could be used directly. The development team and Frederick Thomssen stood by for support during the execution of the test.

Test Evaluation As with the previous test, the students generally received the support provided to them by the Paffin-System as positive. Considering the improvements, as well as the increased experience the students themselves now had with the system, the Paffin-System was able to provide more meaningful support to them. The basic incorporation of the RedTimer was highly appreciated by the students. As a target for the next test case, this incorporation was set to be improved even more. The main problem the development team and the students identified was that the configuration and control was still completely handled in the Redmine (e.g. ticket creation) and the RedTimer (e.g. user status). Moving more of this control into the Agent-Activities directly was set as the main target for the next test case.

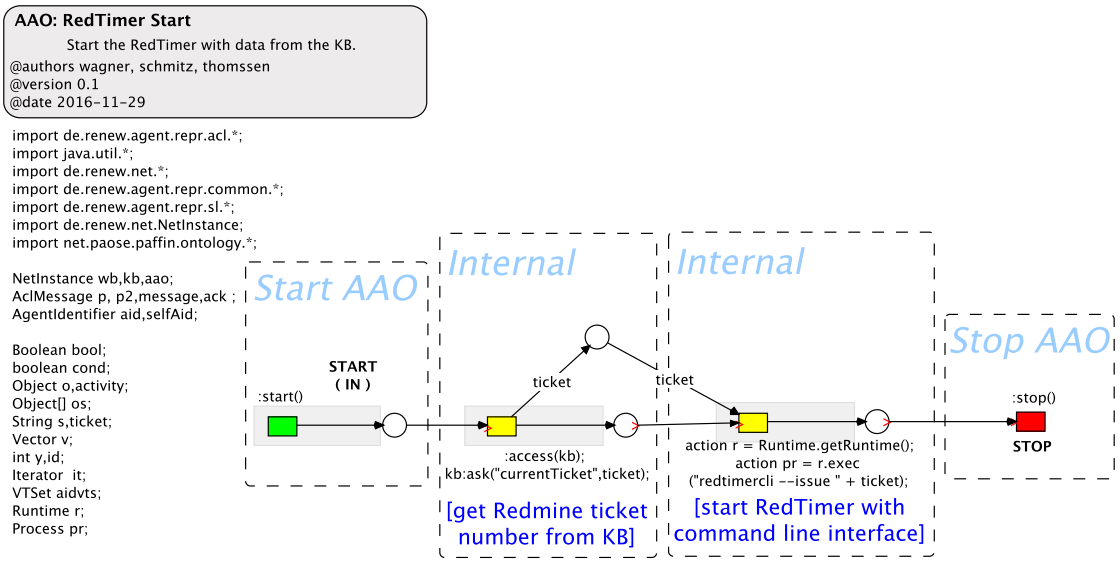

Figure 11.33: Worksheet 5, Ex. 1b: RedTimer start task

Apart from this, the test also continued to confirm the technical robustness of the Paffin-System. No major technical issues arose nor bugs found. Minor issues could either be fixed quickly and directly or could be traced back to user errors (e.g. wrong credentials).

Test Conclusion Overall, the second test of the Paffin-System in the Paose project can also be considered a success. While limited in scope, the incorporation of the RedTimer and the automatic control of it through Agent-Activities proved helpful to the students. After the initial uncertainty and lack of workflow knowledge from the first test was overcome, the students could appreciate the guidance provided to them by the Paffin-System and more clearly recognise the benefits of the digital support of the worksheet exercises as opposed to the classical paper variant.

From an integration point of view, this prototype begins to add more agent functionality. While the prototype is still very workflow centric, the workflow, represented by a Paffin entity, now autonomously controls an external tool, the RedTimer. Even though the interaction between Paffin-System and RedTimer is still very simple in this test case, it showcases that the Paffin as a workflow is no longer just a representation of the process. Rather, it is an active component of the system that has extended functionality with which to improve upon the process. Here, that improvement is merely reading from its knowledge and calling an additional command, but the principle can be extended arbitrarily. Advancing upon that principle is the focus of the third and final test case.

Test Case: Paose Worksheet 6, Exercise 1

The third test was developed for worksheet six of the Paose project. Following directly from the second test case, this test case aimed to improve upon the RedTimer incorporation into the Paffin-System even more.

While the second test case only started the RedTimer on a new exercise logging a predefined ticket, this current test would allow the RedTimer to create tickets for sub-exercises and log time depending on what the students have selected in the GUI. This would better support multiple tasks and allow for a better and more accurate time tracking.

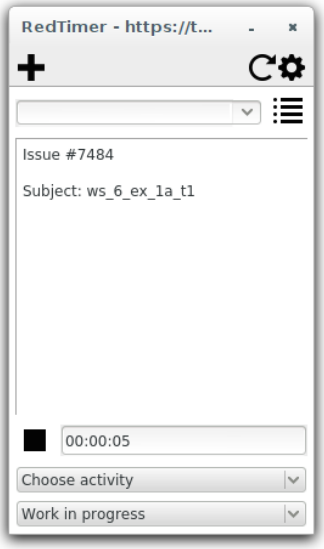

Figure 11.34: Worksheet 6, Ex. 1: GUI screenshot of the RedTimer

Test Development Building on the second test case, the development plan for this one was clear. On the RedTimer side, the command line interface needed to be extended. Instead of just providing the ticket to which the tracked time should be added, the command call would have to include additional parameters in order to move the control further towards the Paffin-System. Apart from that, the GUI, seen in Figure 11.34, remained unchanged.

On the Paffin-System side, the extensions were twofold. On the one hand, the system had to provide the parameters to the command call. Some of these parameters could be generated automatically, but others were still required by the students. In the end, this simply meant prepending additional user input tasks in the GUI for the students to enter data into. On the other hand, the dynamic switching needed to be implemented. As users switching between activities in the GUI is normally only a question of display, this proved to be more of a challenge. Ultimately, a new mechanism was implemented that allows the GUI on specific occasions to send updates as agent messages directly to an Agent-Activity. In this case the Agent-Activity would contain a task controlling a Redmine ticket. Whenever a user selects another activity a message is send to the corresponding Agent-Activity, which recalls the RedTimer command. As the RedTimer only tracks one ticket at a time, a recall of the command causes the RedTimer to stop tracking another ticket and resume tracking the other one. The mechanism to inform an Agent-Activity of updates is currently either turned on or off for all activities in an application in the GUI.

Ultimately, the process-protocol for worksheet six, exercise one can be seen, in part in Figure 11.35. Worksheet six deals with testing for nets, i.e. providing test components and DCs. The main difference to previous test cases, besides the exercise content, is that it starts off with three concurrent user input tasks. Here, the students enter their group ID, the parent Redmine ticket and the current milestone of the project. The group ID identifies the students for the RedTimer, while the parent ticket and milestone provide an anchor point to which to attach the remaining tickets. These parent tickets are also the only tickets that need to be created manually inside the Redmine. The development team and Mr. Thomssen could find no alternative to implement these parameters automatically.

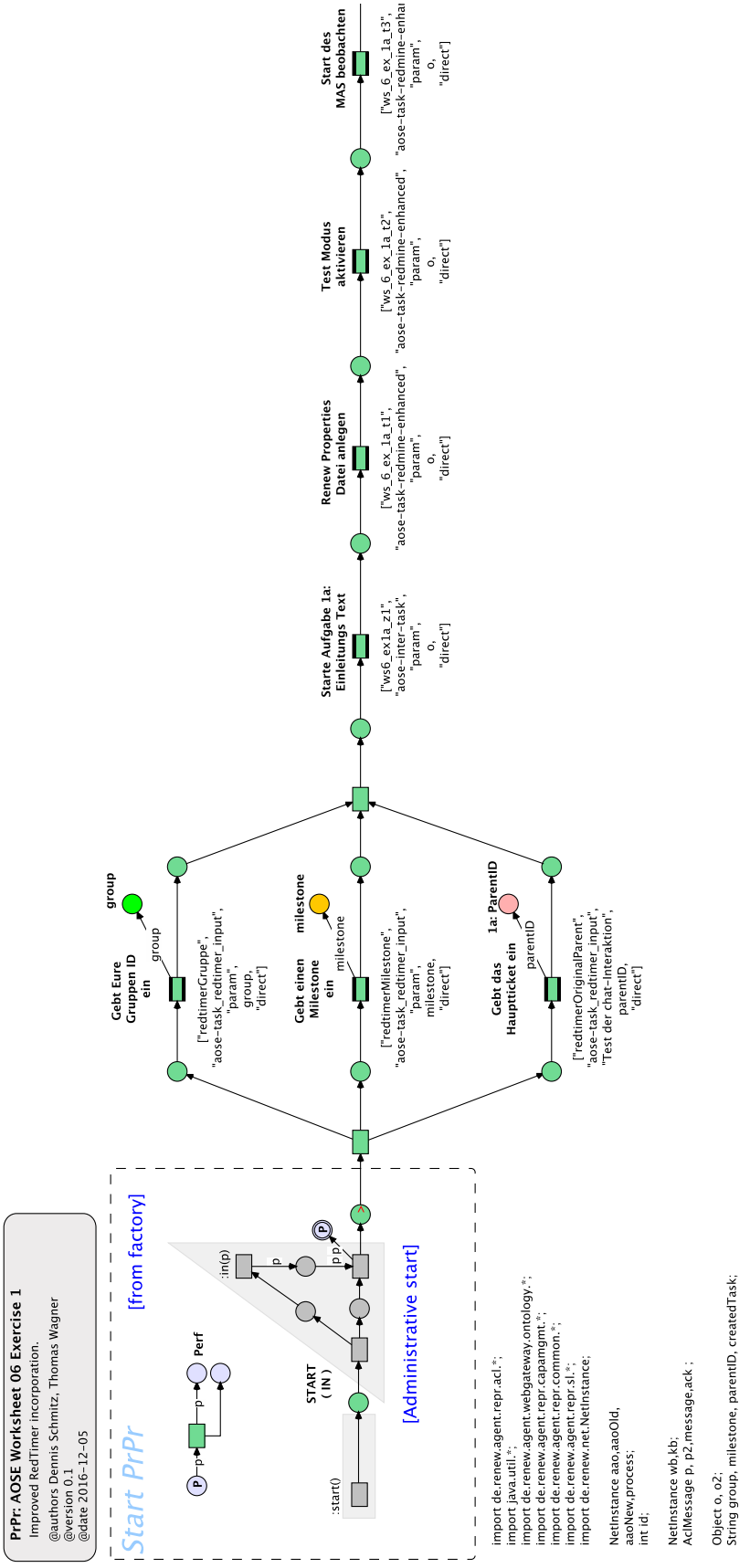

Figure 11.35: Worksheet 6, Ex. 1: Process-protocol for test deployment (excerpt)

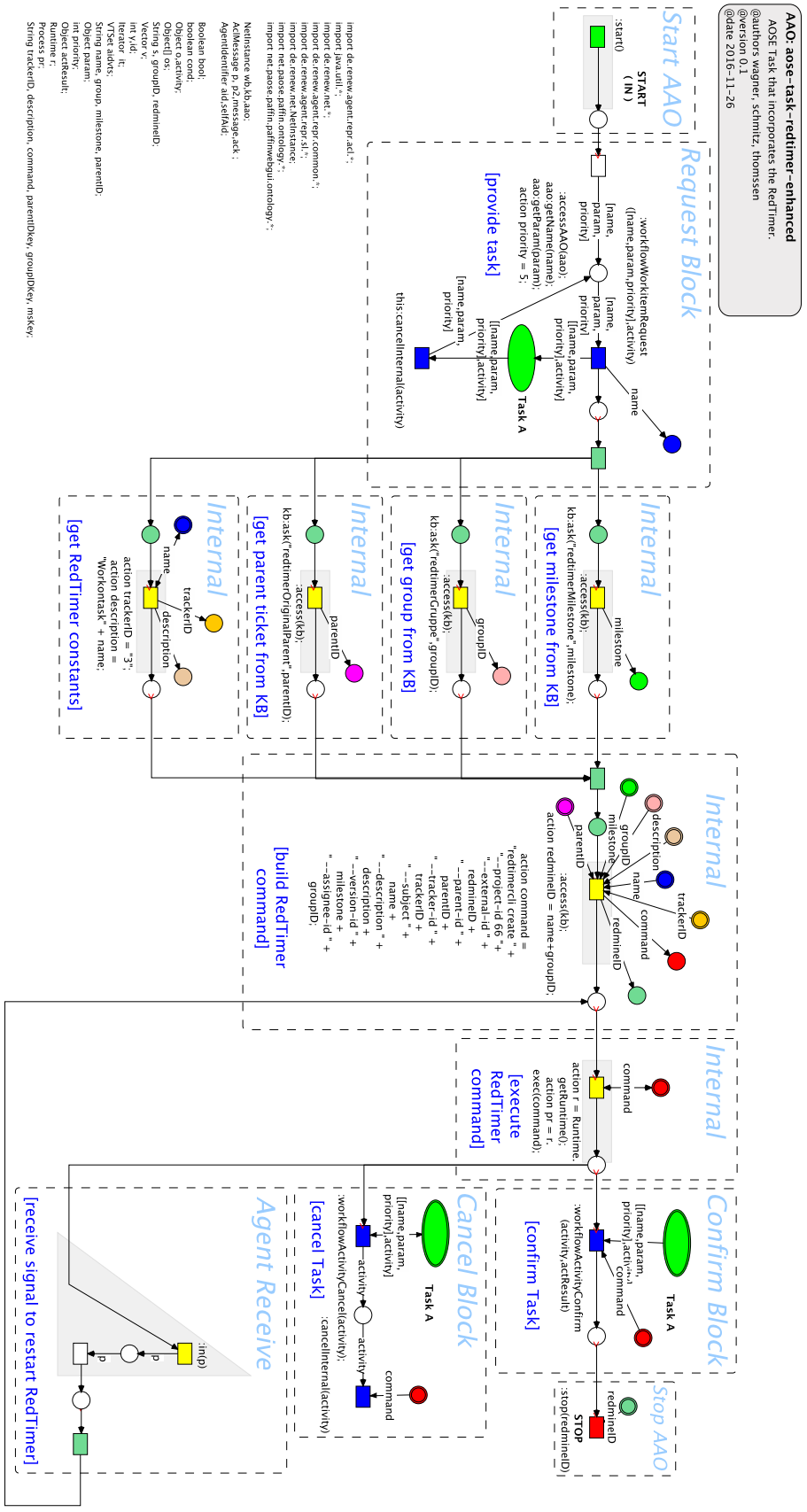

Figure 11.36: Worksheet 6, Ex. 1: AAO for the RedTimer-enhanced task

After the initial input tasks, the students begin work on the actual exercise. Here, all non-advice tasks are realised with the AAO *aose-task-redmine-enhanced*. The corresponding internal process can be seen in Figure 11.36. After the students request the workitem associated with the task, the Paffin executing this Agent-Activity executes a number of agent actions. First, it concurrently reads a number of knowledge base entries (the previously entered parameters) and automatically generates other data objects (data for the current task). Following that, the RedTimer command is created by concatenating the different Strings. The RedTimer syntax is irrelevant here, but in short, the command takes the constant RedMine project ID (for the Paose project), the redmine ID (created for management purposes), the parent ticket, the tracker ID (tracker type, e.g. ticket, feature), the subject (the workflow task name), an automatically generated description, the milestone and the student's group ID as parameters. Following the creation of the command as a String, the command is called in the command line once. Calling that command attempts to create a new ticket in the Redmine with the given parameters and then track time for that ticket. If a ticket with the given identifiers already exists, the existing ticket is used and time tracked here. The String command is not consumed when the command is called. Rather, it is stored for future use. That future use becomes necessary, whenever the students select another activity in the GUI. When that happens, the above mentioned mechanism sends a message to the newly selected activity's Agent-Activity. That message is received in the lower right component of the process in Figure 11.36. The content of the message is irrelevant. Only that a message arrives is important for the Agent-Activity. Receiving a message initialises a loop that causes the stored String command to be reexecuted, thus restarting time tracking for the current Redmine ticket. When the students confirm or cancel the activity, the next or newly requested task will then track the time. While this causes a slight inaccuracy in the time tracking, the students are after all not working on the ticket anymore after they have confirmed it and before they request another workitem, the extension to manually pause or stop tracking in the RedTimer was beyond the scope of this test case.

Overall, a screenshot of the web GUI with the RedTimer can be seen in Figure 11.37. Here, it can be seen that the Paffin-System has the same activity active, as is being tracked by the RedTimer. The two student support tools are aligned via their activities and tickets.

Test Execution The test execution was identical to the execution of the second test.

Test Evaluation This final test iteration supported the students almost completely in both the processes of the exercise and the time tracking in Redmine. After inputting the data in the first few tasks, they could devote all of their attention to the exercise, instead of having to split it between the exercise and prudent manual time tracking in the Redmine.

During the test, the Paffin-System worked great. With the minor issues and bugs discovered in the first two test cases having been fixed immediately, the system experienced no technical setbacks. Performance-wise the Paffin-System also performed suitably. While not a focus of the implementation and improvements, the performance was expected to be an issue considering the increased overhead due to the amount of management messages being exchanged between the different Paffins in each system. However, on the student's computers, no acute performance problems were noticeable. For future applications, though, improving and streamlining the management and administrative messages is a focus.

The RedTimer also worked quite well. Some command calls were not registered on first try, requiring the students to cancel and newly request the workitems. However, these

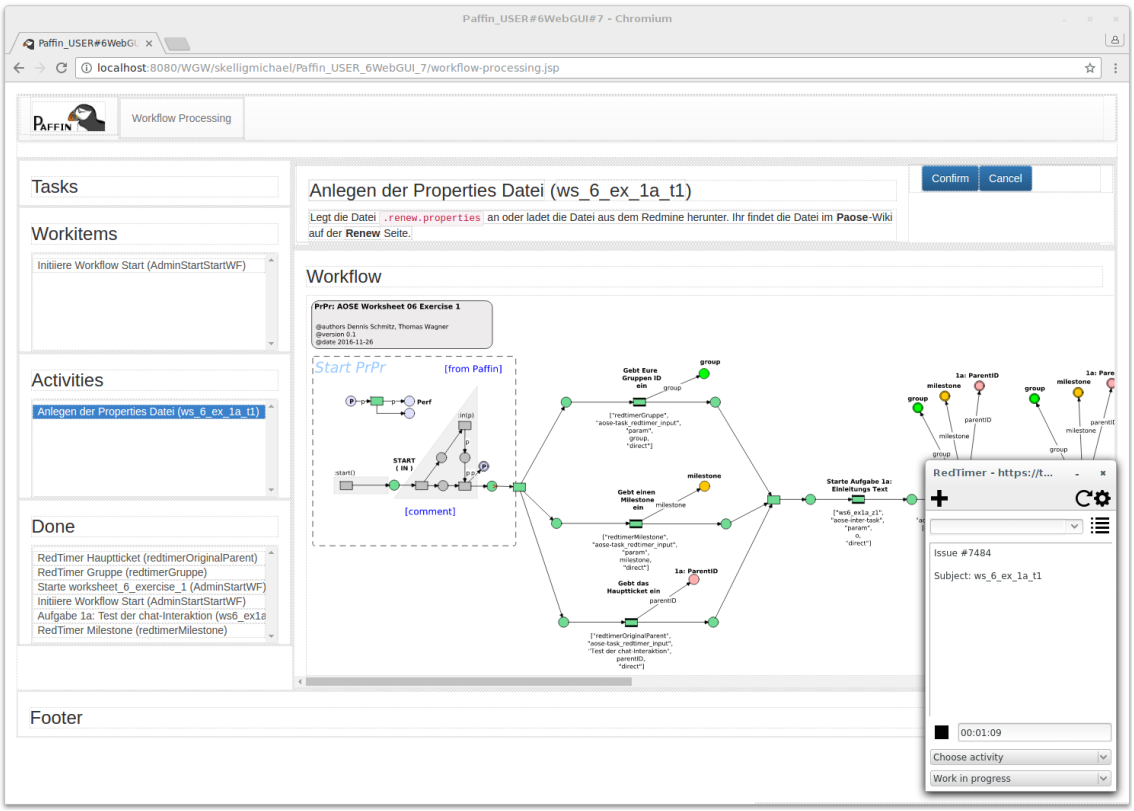

Figure 11.37: Worksheet 6, Ex. 1: Screenshot of the Paffin-System and RedTimer

issues could be quickly fixed by Frederick Thomssen, so that the students were supported throughout the exercise.

Test Conclusion This third test was, as the two before it, a success. The students appreciated the improved project management support and continued taking well to the process support provided by the Paffin-System. Of course, these test cases could still be continued from an improvement perspective. Further refining the RedTimer incorporation to more accurately track the time is just one of the possible future improvements. Another is to realise a supervisor support. If students are stuck on an exercise, their Paffin could, for example, automatically ask a supervisor for help. This would represent another extension to the Paose task Agent-Activity already implemented and seen in Figure 11.36. However, the sixth worksheet was the final one. Its completion marked the beginning of the second project phase, in which the students not just learn Paose, but actually use it in a practical software engineering project. General Paose support is a separate application area of the Paffin-System, which is discussed later. Any further improvements of the teaching support are being made in preparation for future teaching projects in the coming terms. These, however, are not covered in this current thesis.

From an integration perspective, this third and last test case finally features some more intricate agent/workflow integration mechanisms. Instead of just acting as an agent before starting the workflow, as was the case in the second test, the Paffin actually acts as both agent and workflow at the same time. After the workitem request for each RedTimer-enhanced task, the Paffin autonomously calls on the RedTimer and then waits, as an agent, for new information. In other words, while the Paffin is a workflow with a

task being worked on by the students, it is also a reactive agent controlling the RedTimer tracking. This is the kind of integration mechanism that was intended by the motivation to integrate agents and workflows. It is not an automatic framework mechanism that always tracks time, but rather a sophisticated, from a modelling perspective, integration of agent and workflow functionality. It is still a rather simple example, but it showcases what else is possible.

The example of possible future work regarding supervisor Paffins exemplifies this even more. If implemented as intended, the Paffin, while still being a workflow, would not only be a reactive, but also a proactive agent. It would, depending on whatever circumstances were modelled, decide autonomously to inform the supervisor of any problems. On the other side, the supervisor Paffin could also learn to evaluate the information from student Paffins. For example, it could learn from past experience and only react on the second inform if the students have previously shown to simply take their time.

In general, the example of the RedTimer-enhanced task illustrates well, what the Paffin-System is capable of from a functional perspective. It can, without cumbersome changes or adaptions to the framework, incorporate any agent or workflow functionality into the respective other one. Here, it incorporated agent behaviour into a workflow task. Incorporating workflow tasks into agent behaviour is just as possible though. This kind of functionality is similar to what is possible in agent-based workflow management systems or workflow-based agent management systems (see Chapter 3). However, usually in those cases the additional functionality not related to the main modelling construct needs to be implemented in the framework. In the Paffin-System, it is possible to provide that functionality on the application level, thus not forcing other parts or applications of the system to be encumbered by the additional functionality. This represents an increase of flexibility when compared to the rigidness of a framework based solution. Besides the conceptual, i.e. agent/workflow dynamic, perspective-related, i.e. the provision of behavioural and structural views, and modelling, i.e. modelling construct capabilities, aspects, this functional aspect also strongly contributes to the expressiveness and power of the Paffin-System.

Concluding Evaluation

Overall, the deployment and use of the Paffin-System in the Paose teaching project proved to be a success. As the test cases themselves featured extensive evaluation discussions, the following will only shortly summarise the results.

- It showed that the Paffin-System worked outside the more controlled toy example test environments. When testing the system before, it was mostly done on the author's or Mr. Schmitz's computers. It was also always used and operated by people who knew exactly what was going on in the background and what was expected from them as users. The students in the Paose project knew neither. They showed the development team issues within the system, which would have never occurred to them on their own. The fact that these issues were always rather minor was a pleasant validation of the work the development team had done up to that point. All in all, the Paffin-System endured this first practical test admirably. With ongoing work dedicated to improving reliability, robustness and performance, the Paffin-System showed, in this limited scenario, that it has the potential to be used in more complex and demanding scenarios.
- The incorporation of the RedTimer also showed how easy extensions of functionality within the Paffin-System are. As discussed, the external tool for this application

could be incorporated into the Paffin-System by incorporating it into the application code instead of the framework code.

- From an application perspective it showed that students could benefit very much from process support. After the initial lack of knowledge was overcome, the students took well to the system and appreciated what it offered them. By explicitly guiding them through the exercises in an automatic way the students could focus on the content, rather than on other things like other exercises on the paper. This was even more pronounced once the RedTimer and time tracking was also incorporated here. This area is also full of potential for future work. Supervisor Paffins as mentioned above or other mechanisms for supporting the student's learning experience are just some examples. The overall topic of the supporting the teaching, especially of Paose, is discussed in detail in [Schmitz, 2016, Schmitz et al., 2016].
- For the integration aspects of the Paffin-System this group of prototypes was slightly limited. Except for the control of the project management aspects and the outlook of future work, the test cases didn't include much potential for utilising the hybrid aspects within the Paffin-System. However, as mentioned at the start this was never the intention for these prototypes. They were intended to validate the Paffin-System beyond the controlled toy example test environments with users unfamiliar with the system. Utilising and showcasing advanced integration mechanisms was only a secondary goal.

11.4 Application Visions

The previous section featured application prototypes that were practically implemented during the creation of this thesis. For this section, the focus moves to *possible* applications and application scenarios, in which the integration efforts implemented in the Paffin-System can be used to an advantage. These scenarios are only sketched out and represent drafts of possible system. The goal of this section is to showcase the "larger" picture of the application possibilities that unfortunately could not be implemented and practically evaluated within the scope of this thesis. The scenarios were sketched out in discussions with different colleagues during the creation of this thesis.

Note that the following visions explicitly relate to the Paffin-System and Agent-Activities. Implementation options are especially regarded with the Paffin-System in mind. However, these application visions are also generally valid for other, potential integrations of agents and workflows. By abstracting from the concrete integration details, these visions provide a perspective of what is possible when agents and workflows are integrated.

11.4.1 Workflow Cloud Applications

This application vision relates to collaborations with Dr. Sofiane Bendoukha. In his dissertation thesis [Bendoukha, 2016] he examined workflow management in a cloud-based environment. Dr. Bendoukha and the author of this thesis collaborated on some topics over the years [Bendoukha and Wagner, 2012, Bendoukha et al., 2012, Bendoukha et al., 2013, Bendoukha and Wagner, 2015]. Unfortunately, due to technical limitations in the cloud functionality, the collaboration could not be implemented in a prototype with the Paffin-System. This current section therefore outlines how Dr. Bendoukha's results could contribute to the Paffin-System in practice and then sketches an application vision of a Paffin-based WFMS in the cloud.

In general, the concepts and mechanisms presented in [Bendoukha, 2016] could be utilised in any Paffin application. The cloud transition could be reconfigured to utilise the already existing synchronous channels for the three workflow operations. In that case, the activity, that is now being handled by the platform WFMS in the Paffin-System, would be handled through a specialised component realising the connection to the cloud. Agent-Activities and, on a more abstract level, process-protocols could also be used to realise the concept of InterCloud-Net (ICNets) and InterCloud-Workflows (ICWorkflows). Both of these concepts may be used to specify and manage the control flow between collaborating and interacting clouds with an emphasis on workflow management. Here, Paffins and the Agent-Activity may serve as modelling abstractions to encapsulate individual participants (e.g. users or cloud providers) that interact in some way. Further results from [Bendoukha, 2016], like the RenewGrass image processing functionality, could be incorporated into any Paffin application that required that functionality.

This current and concrete application visions focuses more on the InterCloud agent-based workflow management (IC-AgWFMS) system proposed in Chapter 12 of [Bendoukha, 2016]. The IC-AgWFMS basically describes an agent-based system that lets users model workflows, the tasks of which are then distributed and executed on different clouds before results are gathered from the distributed environment and sent to the user for inspection and further usage. Agents in the IC-AgWFMS provide the GUI for modelling workflows and then take care of distributing the tasks on different clouds before collecting and processing the distributed results. For the cloud distribution the agents analyse the requirements, check available cloud platforms and their properties, select the best suited platform and then instantiate the (sub-)workflows and/or tasks to the selected cloud platforms.

One important aspect of the IC-AgWFMS is that agents use workflows as artefacts that they themselves handle. Users model the workflows with GUI provided by the agents, after which the agents distribute the workflows to the cloud environments. Finally, the agents receive the results of the workflow executions in the cloud and return those to the users. Obviously, agents and workflows are quite separate in the IC-AgWFMS, with agents having priority of control. In other words, the IC-AgWFMS is an agent-based WFMS as discussed in Section 3.3.1 and evaluated against the Paffin-System in Section 13.3.1.

The idea of this current application vision is to replace the agents and workflows of the 13.3.1 with Paffin entities. A Paffin with a GUI would still realise the GUI, user representation and modelling within the system. But instead of receiving the modelled workflows, these workflows would be translated into the form of Agent-Activities and process-protocols. This way, instead of an agent receiving workflows as artefacts, the user Paffin would receive the workflows as Paffin behaviour. With these behaviours, the user Paffin would instantiate new Paffins that would exhibit that behaviour as their main content. In addition to this user-specific workflow behaviour to be executed in the cloud, the "workflow"-Paffins would feature a number of standardised mechanisms. These mechanisms would, for each individual Paffin, perform the functionality provided by the agents in the IC-AgWFMS, albeit in a slightly different way.

Analysis of the requirements and checking and selecting the best cloud environment would be analogous, albeit only concerned with the current Paffin workflow behaviour. Instantiation, however, would be handled differently. Instead of simply instantiating the workflows and executing them in the cloud environment, the Paffin may use extended mechanisms. If mobility is implemented in the Paffin-System and if it is supported by the cloud environment, the workflow-Paffin may migrate to the cloud in order to execute its workflow behaviour. After execution is complete it would remigrate to the user's system. Another option is for the Paffin to utilise an extended subordinate resource mechanism

to instantiate a new Paffin in the cloud environment and then delegate the workflow behaviour to that Paffin. Once the subordinate Paffin is finished with the workflow execution it would return all obtained results to the original workflow-Paffin. In both cases, the workflow in the cloud environment would be executed as a Paffin entity. After execution of the workflow behaviour the workflow-Paffin would process the results and return them to the user Paffin for display.

The above variation of the IC-AgWFMS maintains the basic operation mode. Workflows are modelled by users and then distributed for execution in different cloud environments. Utilising Paffins does not change that or add something on the basic operation level. However, using Paffins instead of workflows does, in fact, open up the possibilities of details, which can be used to greatly improve upon the inter-cloud workflow execution in ways the traditional workflow model can't.

In general, the workflows being executed in the cloud environment now have access to agent functionality, mechanisms and properties. They can act and react autonomously, which is probably the most significant features. While the exact form of agent functionality support is dependent on the concrete workflow application the following provides some examples of how agent functionality in Paffins may be used for inter-cloud workflow execution.

Partial workflow execution: Given an especially complex and time-consuming workflow it is feasible that modelling that workflow also takes a long time. In order to save time it would be possible to start a Paffin with only part of a workflow. That part could then begin execution while the user still modelled the rest of the workflow. Once the user is done modelling, the remaining workflow could be added to the running Paffin as an additional process-protocol or exchanged (i.e. more detailed) Agent-Activities.

Intermediate results: The workflow as a Paffin could return intermediate results during its execution. This could happen either at regular intervals (e.g. after every task is completed) or at predefined points in the execution. It is also feasible for the Paffin to wait at these points until a user can send further instructions. By doing so it would be possible to control the workflow execution with changed parameters, if the workflow execution appeared to be going into the wrong direction.

QoS violation reaction: Quality of service (QoS) in a cloud environment describes the circumstances the cloud environment agrees to maintain during the execution. This can include performance and accessibility metrics. Paffins executing workflows may be configured to react to QoS violations. If, for example, performance dropped below acceptable levels it could abort its execution and reinitialise in another cloud environment. It could also be possible for the Paffin to store its data if it detects that the cloud environment were about to become inaccessible or fail entirely.

Parameter adaption: During long-running workflow executions it may become apparent from intermediate results that a workflow may not yield the expected results. Similarly to the handling of intermediate results the Paffin itself may process these results instead of informing the user. Depending on the results the Paffin may then autonomously decide to alter certain parameters. This would require predefined assumptions about the workflow execution to be provided to the Paffin for comparisons. Take the following scenario as an example. A workflow is set up to calculate 1000 values and the modeller expects these calculations to take one hour. After 100 calculations the Paffin observes that the calculations are being processed a lot faster. Maintaining the current rate of calculations the calculations would be done in half an hour instead of the planned one hour. The Paffin now can choose multiple options of how to proceed.

It can increase the parameter for planned calculations to 2000 values. If the goal of the workflow is to approximate a result through the calculations the increased amount of calculations may yield a better result. The Paffin may also renegotiate with the cloud environment to half the available processing power and cut the costs of the workflow execution. That way the calculations would happen in the planned period of time and money might be saved. The Paffin may also initialise other, independent concurrent tasks and thus use the available system resources to maximum capacity and reduce the total execution time.

Inter-cloud coordination: Using agent communication mechanisms, the workflows distributed in the inter-cloud environment may also coordinate their execution. For example, if one Paffin is already finished with its workflows it may take over part of the workload of another Paffin. That way total execution time could be reduced. It is also feasible for Paffins to interchange intermediate results and utilise the above mentioned mechanisms for parameter adaption or reaction to QoS violations in a distributed way. Instead of only relying on the intermediate results or information from one source, the Paffins could make better informed decisions based on data from multiple sources and environments.

Cost optimisation: Another way to utilise the Paffins is to optimise, i.e. reduce, the costs associated with cloud executions. Cloud processing time usually costs money and choosing the most cost effective cloud environment is part of the selection process. However, the Paffins executing the workflows could be configured to keep monitoring the available cloud environments. The most cost effective cloud environment during initialisation may not remain so during a long running calculation. If a better option becomes available a Paffin may migrate to it (if available) or instantiate a new Paffin there and transfer all runtime data before terminating. This would ultimately save on costs because Paffins would always switch to the best alternative.

Additionally, the ideas of migration and problem solving discussed for the green cloud computing example in Section 11.4.2 can also be applied in the IC-AgWFMS context.

Another aspect of the realisation of the IC-AgWFMS with Paffins relates to the content of the workflows. [Bendoukha, 2016] is mostly concerned with scientific workflows [Ludäscher et al., 2009, Barker and van Hemert, 2008, Kok et al., 2010], e.g. workflows for processing large amounts of data and doing highly complex calculations. Traditional workflows only model the control flow as tasks to be executed. Any data processing and calculation functionality needs to be called by the workflow management in the framework. That way, maintenance, configuration and specialisation of the functionality can become quite cumbersome and difficult. In Paffins, the functionality can be incorporated directly into the workflow-like Paffin behaviour. Task tracking, for performance logging or monitoring reasons, may still be used, but the Agent-Activities as abstract tasks can directly implement the processing and calculation functionality. Configuration, adaption and maintenance happen within the Agent-Activities, which is an improvement over the framework solution.

All in all, using the Paffin-System to realise the concept of an IC-AgWFMS opens up a large number of possibilities. These possibilities may improve upon many aspects of the system, including flexibility, maintainability and execution. Furthermore, by enabling agent functionality in the workflows that are distributed in the inter-cloud environment, the expressiveness of the approach is increased as any specialised functionality can be included within the workflows, as long as there is an agent implementation of it.

11.4.2 Green Cloud Computing

Integrated (Paffin) entities can be used to model any general workflow. If that workflow observes or is informed of any issues of its execution, agent functionality can be activated to deal with the issues. In that case, the Paffin as a workflow becomes a Paffin as a hybrid of both agent and workflow.

Such behaviour is desirable in many scenarios. The following are only representative examples:

- A workflow is missing data at runtime. As a Paffin it can activate its agent functionality to attempt to retrieve that data from somewhere (e.g. another Paffin, a data storage, a human user, etc.)
- The user that executes tasks in the workflow (physically) moves elsewhere, where direct access to the workflow is no longer possible. The agent functionality can then be used to fix this issue by, e.g., enabling access to the user (by creating a secure VPN connection), by cancelling active task execution of the user and make the tasks available for other, still available users or by starting a new entity which has access and takes care of user interactions and sends results to the workflow.
- If data is input incorrectly the workflow can activate agent functionality to transform or fix the data for smaller issues (e.g. wrong text encoding, leading or trailing white space, falsely formatted data like phone numbers or zip codes)
- If users continuously input false or erroneous data the workflow can intelligently decide to either block the user (if it suspects malicious intentions) or guide and support the user by starting other workflows to support his or her work.
- The execution environment of a workflow encounters technical problems and has to be restarted or possibly shut off. As an integrated entity the workflow can either migrate to another execution environment (platform) using agent mobility or, if agent mobility is unavailable, instruct another (trusted) environment to start a new integrated entity and then transfer all administrative and runtime data and control to the new (Paffin) entity. That way the execution can continue transparently to the user.

The last example is the application scenario that is going to be focussed on here. This application scenario originates and was discussed at a meeting with the company windcloud GmbH[5]. Windcloud operates a computing centre at a wind park in the northern part of Germany. They utilise the cheap costs of electricity available directly at the wind park to provide highly affordable computing power. That electricity is gained completely from renewable (wind) energy, which makes the approach of windcloud especially green and worthwhile from an environmental perspective.

Currently, the computing centre uses electricity from the grid, i.e. potentially from traditional, non-renewable sources, as a last resort in case the wind turbines do not generate enough power. This can happen, for example, during maintenance of the turbines, when a number of turbines fail or when there is not enough wind and the stored capacity runs out. In such cases, the computing centre must remain continuously functioning and reachable to ensure that customers receive the computing service they are paying for. The solution to use electricity from the grid is technically feasible, although it clashes with the generally green and environmentally friendly approach.

This is where Paffins might be able to help in the future. This scenario has three assumptions. First, windcloud operates multiple computing centres physically distributed

[5] `https://windcloud.org/` (last accessed May 28th, 2017)

so that failures in one centre can't affect others. Second, windcloud provides customers with the means to execute highly complex, possibly long running workflows (e.g. scientific workflows [Ludäscher et al., 2009, Barker and van Hemert, 2008, Kok et al., 2010]). Third, the Paffin-System is extended with concepts and mechanisms providing agent mobility to the Paffin entities. Realising this functionality for the Paffin-System is out of scope for this thesis, but technically feasible as mobility functionality exists for the technical Capa basis (cf. Mapa [Cabac et al., 2009]). Alternatively, the option described above where a Paffin transfers runtime and control data to another (trusted) Paffin in a safe environment is also viable for this scenario and would already be technically possible.

Taking the assumptions into account the scenario unfolds as follows. Windcloud executes a set of workflows in their primary wind park. These workflows are implemented as Paffins. Paffins are accessible to users via some sort of standardised cloud interface. The technical details of that interface are irrelevant for these descriptions, except for that the interface must be able to access the Paffins anywhere in the cloud operated by windcloud. In case the wind park has issues with its power capacity it informs the Paffins about these issues. Issues can be scheduled or unscheduled maintenance or a decrease in wind power availability. When such issues arise, the Paffin can suspend its execution as a workflow. It would then, using agent actions, do everything it could to ensure that its workflow can continue the execution with as little delay as possible and with no data loss.

Depending on the concrete situation there are a lot of ways that this scenario might unfold. Minor issues might only require a suspension of execution or storing the runtime data persistently and then reinitialising after the issues are resolved. If the issues pertain to the security or privacy of data (e.g. an unauthorised data access) the Paffin could use encryption mechanisms to protect the data. Yet if the issues in the environment are critical and prohibit, for a prolonged and/or uncertain period of time, the execution of workflow, the Paffin would have to migrate to another environment. In the scenario the target environment would be one of the other computing centres not affected by issues at that time. Using this migration tactic there would be no noticeable difference for the execution of the workflow, except for the time it takes to completely migrate the Paffin entity. Basically, the procedure would be transparent to the customer. Assuming that the other computing centres are equivalent to the original one in terms of capacity and capabilities, the Paffin would not even necessarily have to migrate back but could complete its execution there. However, remigration due to load balancing or missing resources and capabilities is just as easily possible.

Figure 11.38 illustrates the scenario. Computing centre A notices an impending problem and informs the Paffins being executed on it about the problem. Paffin A3 executes a non-critical workflow and decides, given the data, to simply suspend its execution temporarily and restart once the problems have been resolved. Paffins A1 and A2, however, execute critical workflows that can't or shouldn't be suspended. Because of this, they autonomously decide to migrate to computing centre B. Paffin A1 can finish its execution in computing centre B and does not intend to remigrate back to A. Paffin A2 on the other hand requires special resources only available in computing centre A, so it intends to remigrate to that centre once the problems have been resolved.

In this scenario, the agent mechanisms available in the Paffin-System allow for workflows to become reactive, intelligent, autonomous and mobile. They can receive signals about problems and issues from their environment. Then, they can process those signals and decide on their own authority the most suitable action. That most suitable action may be one of the options discussed above or it may be a migration to another, safe environment. All this happens transparently to the user.

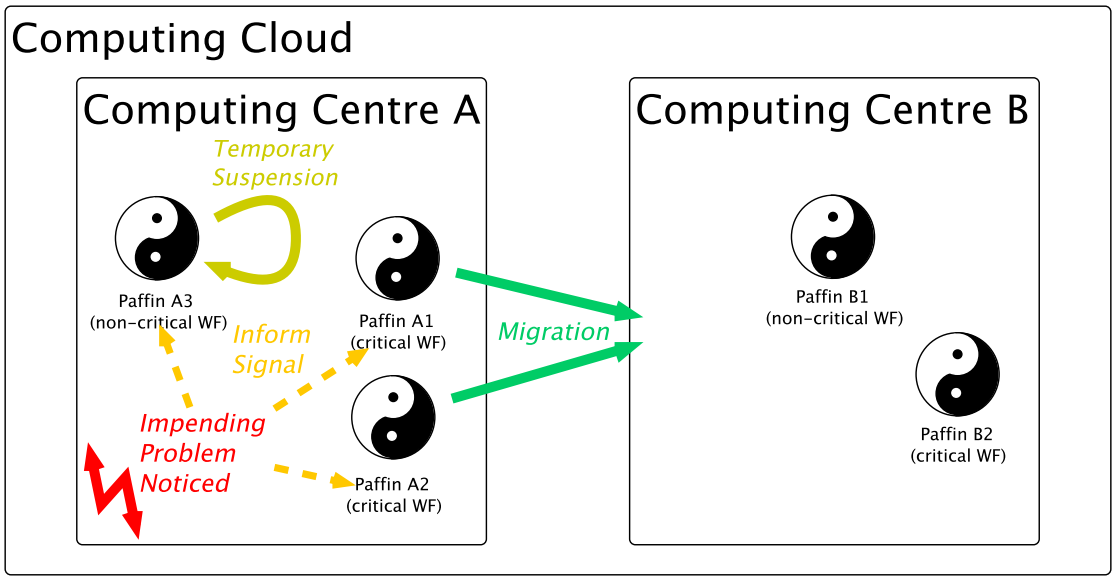

Figure 11.38: Green cloud computing vision illustration

While this scenario emphasises the workflows from the application and user perspectives, the application still strongly utilises the integration of agents and workflows overall. Workflows may be what are offered to the users, but since the Paffins realising those workflows must always be ready to act and continuously process their environment data they are constantly also agents. Throughout their execution they are both the workflow being executed and the agent responsible for making sure it itself can be executed.

One might argue that this kind of behaviour is just an extended way of enhancing workflows with agent properties as found in agent-based WFMS (see Section 3.3.1). However, such agent-based WFMS may utilise agent properties and functionality to some degree, but the Paffin-System goes beyond just making those properties and functions available. It enables the workflow to **be** an agent, which is required in this application scenario. Agent-based WFMS are not capable of doing that, as is discussed further in Chapter 13.

It is even possible to go a step further with this scenario. [Hosman and Baikie, 2013] presents the concept of solar-powered cloud computing, i.e. computing centres based on solar energy. The concept contains the idea of energy-aware cloud computing, meaning that computing power in computing centres would be regulated according to the available energy, i.e. the available sunshine and stored energy. The solution offered in [Hosman and Baikie, 2013] is to design the computing centres in such a way that during minimum energy times only critical services may be executed while others are deprioritised and must therefore wait for when more energy is available. Considering integrated entities and the Paffin-System other options become available. When energy is low, the Paffins may migrate to other computing centres being powered by other energy sources. But assume that there are multiple, solar-powered cloud computing centres around the world in different time zones. When night falls at a computing centre in Japan, it is already daytime in India. Nightfall in India means daylight in Germany, nightfall in Germany means daylight on the east coast of the USA. The solar energy available travels westward until it is morning again in Japan. Before one computing centre effectively shuts down for the night all active Paffins and their workflows may migrate to the next active computing centre. This would create a constant “wave” of computing power moving with the sunlight

across the globe. According to the windcloud executives, energy cost is the highest cost factor in operating a computing centre. Being able to use cheap and sustainable energy and furthermore temporarily shutting down centres whenever that energy is unavailable saves money. When it is also possible to transparently migrate all running systems, workflows and services without disadvantages to the customer it becomes a viable business model. Integrated entities, and possibly even future versions of the Paffin-System itself, provide one possible solution to tackling these and similar challenges.

All in all, the example of green cloud computing showcases how integrated entities in concept and Paffins in practice may be used to facilitate novel ideas of environmentally friendly computing. Paffins offer abilities and properties traditionally not available to workflows. By enabling the workflows to be agents as well, the Paffin-System also enables the realisation of novel ideas such as migrating computing power.

11.4.3 Workflow Administration

One of the features missing in the current Paffin-System is sophisticated workflow administration. Currently, administration is only possible through manipulation of the initial database. For the proof-of-concept purposes and application prototypes, this manual administration is sufficient. However, going forward beyond this thesis an administration interface including database access is a priority.

Due to the integration nature of the Paffin-System, however, the challenge of providing an administration interface provides a rich opportunity for a Paffin-based application. The purpose of this application vision is to showcase how agent and workflow mechanisms within the Paffin-System can be interconnected in order to bring workflow execution and workflow administration onto the same modelling level. By doing so workflow administration would no longer be an auxiliary add-on to a system but could be modelled, monitored, performed and maintained in the same way as workflow execution. In doing so the effort for these activities would be decreased as the introduced uniformity would reduce the number different model types.

Consider that a system administration is just an user setting a task to the system to change the configuration in some way. That task may involve, for example, adding a new user to the database, changing permissions of a user or task or setting up a new database etc. Seeing administration duties as tasks makes the overall administration a flexible workflow. Administrators start the workflow and then choose what they want to configure. After confirmation the system deploys the configuration so that it is, possibly following a restart, active. During that deployment, the agent and integration mechanisms of the Paffin-System come into play as discussed later.

Clearly, the basics of workflow principles can be applied in this way to administration in general. Here, the discussion will focus on workflow management, meaning, for the Paffin-System, creating, reading, updating and deleting (CRUD operations) database entries pertaining to workflow resources (credentials), tasks, roles, permissions and activity specific GUIs. While this focus persists, it is possible to apply the principles to a general administration scheme, although this is only shortly covered as an outlook at the end.

With the basic premise having been clarified, it is now possible to consider the workflow administration workflow (WAW) in more detail. There are a number of assumptions for the vision of the WAW. First, the general environment is assumed to be the current Paffin-System extended with the ability to (write) access the persistent database. Second, the assumption is that no special GUI area or other extended functionality should be added for workflow administration. All administration duties are to be handled using the existing task interface. This enforces the desired uniformity of execution and administration.

In general, the WAW works in the following way. The WAW would be either constantly active in the system or manually started when required. It would be executed on a special Paffin that is aware of the database content (in its knowledge base) and can initiate CRUD operations on the database. When the WAW is active, it would be waiting and ready for administration duties by providing corresponding workitems. Through the implemented RBAC system, only administrators would be provided with these workitems. Each administration function would be handled by a different task. When the administrator requests the tasks he or she is provided with a specific GUI. That GUI can contain readouts about the current database, GUI elements for building new database entries or modifying old ones etc. The administrator can enter all the data he or she wishes and then confirm the activity. Back in the WAW, the executing Paffin would turn into an agent. It would process the GUI results from the activity, transform and read the required administration duty and then perform it. The result of that performance would afterwards be processed. If the administration succeeded the WAW would be reset, waiting for new administration duties. If it failed it would also reset to the original admin task but would, through special parameters, initiate special error handling and support variants of the admin tasks. That way all administration decisions would be made by a workflow user (the administrator), while the administration duties were performed by a Paffin acting as an agent.

This is the general and basic way of realising the WAW: Model tasks and activity GUIs for administrators to inspect the database and enter commands, and then have the Paffins, as agents, take care of the actual execution of the commands. What an implementation of a WAW could look like as an Agent-Activity can be seen in Figure 11.39 in a mock-up form[6]. It mostly correlates to the description above. First, the administrator is offered a number of tasks with the database content as parameters, in which he or she can enter the administration commands. The result of such a task is processed and then a new process-protocol started that actually implements the administration duty. Decoupling the actual administration duty into a separate process-protocol allows for more flexibility and reusability in the WAW. When the administration process-protocol is done, it informs the WAW Agent-Activity via agent message, the result of which is then checked and processed. If the administration process-protocol succeeded, the WAW Agent-Activity is reset to the state before any workitem were offered to the administrators. If it failed, the WAW Agent-Activity would also be reset, albeit with predefined error parameters that would be handled differently in the GUI so that administrators could retry and fix the issues.

For the basic workflow administration of the Paffin-System, the integration is limited to the Paffin being a workflow while the administrator enters data and then turning into an agent to execute the commands. It is, however, feasible to imagine more complex administration duties that require a more complex interplay between agent and workflow states of a Paffin. For example, the administrator may require updated database information before he or she confirms the command. In that case, the Paffin would become active as an agent, querying the database, while still being a workflow. Another example would be if the administration should gather and execute multiple commands. Looping between error handling and command execution would diminish the clear separation between agent and workflow concerns in the WAW.

The basic WAW presented here is only a starting point. Basically, any general administration duties may be done in a WAW-like way. Consider distributed administration. Here,

[6]The mock-up from Figure 11.39 does not contain functioning code. The basic principle works, but without the additional ontology concepts, process-protocols and workflow data it could not be compiled and executed.

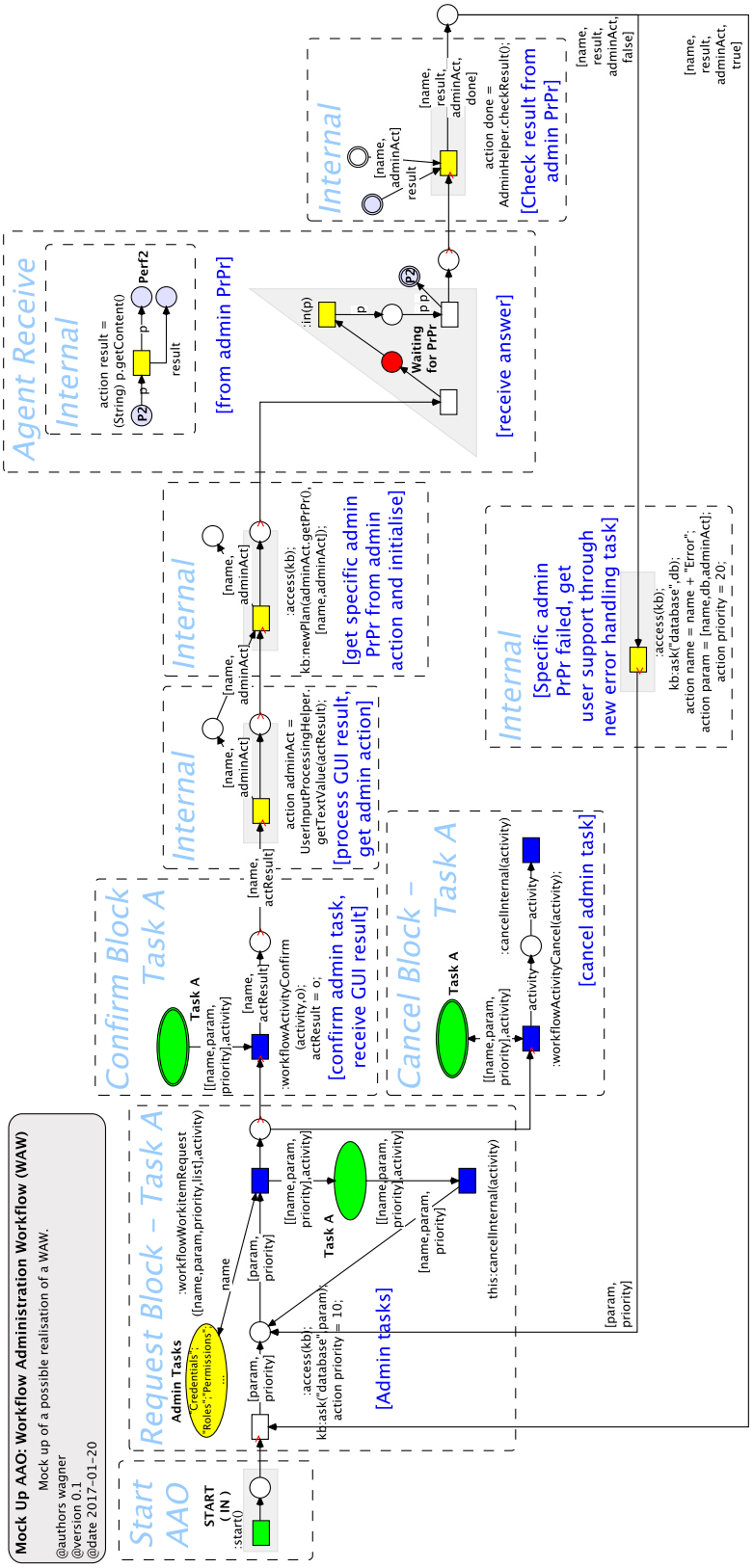

Figure 11.39: Mock-up AAO for a possible WAW

the Paffin executing the WAW may migrate to other platforms (when mobility is available) and perform the administration duties there as well. Another extended mechanism would be guided administration, similar to established (administration) wizard interfaces. Examining the workflow administration in the Paffin-System, certain administration duties require further steps to be useful. For example, when adding a new resource or user that user needs to be added to workflow roles in order to be able to actually do some work. The WAW could be extended to support administration in multiple, subsequent and related steps, according to predefined, reasonable administration patterns. The possibilities of the WAW principle are only limited by the scope of functionality supported by the Paffins.

In conclusion, the practical value this kind of solution would provide, besides the direct gain of providing the missing workflow administration, is uniformity. While the database of the Paffin-System is currently still rather simple, more complex applications may require more complex database schemes. Further configuration may also be necessary. System modellers ultimately also have to model any administration they require for their applications. By providing the uniform workflow administration in the form of a WAW, this administration does not need to be implemented in the framework or in a separate add-on plugin. By keeping the administration uniform with the rest of the system modellers may utilise specialisations that might not be generic enough to warrant inclusion into the framework. Furthermore, anything that is added to the framework would be used in any other application in that framework as well. This could negatively affect performance and would increase the already significant complexity of the Paffin-System even more.

All in all, what this application vision shows is how the integration of agents and workflows can be used to implement ordinary functionality like administration in a novel way. In practice, a middle ground has to be taken. Some administration functionality is so essential and ubiquitously required that an incorporation into the framework is more than justified. However, other, specialised functionality may always be added in a WAW-like way. The capabilities of the Paffin-System allow for this and other innovative concepts.

11.4.4 Paose Installation Support Wizard

This application wizard is strongly related to the Paose support environment prototype discussed in Section 11.3.3. Like that prototype, this vision was developed in collaboration with Dennis Schmitz.

The first step in the yearly Paose teaching project is to install and configure the full Paose development environment. This includes installing and configuring the Eclipse development suite[7], configuring access to the source repository and revision control system, initially retrieving Renew, Capa and specialised project sources, configuring the build environment and finally building and configuring the sources itself. The entire process is complex and error prone if done manually. Currently, an installation wizard already support students by automating as much of the process as possible. However, many automated steps involve actions the students need to learn for the project anyway. Consequently, an improved installation wizard is desirable that guides the students through the installation and automates some parts while still maintaining learning value for the students. The Paffin-System provides the opportunity to realise such an installation wizard.

The application vision of the installation wizard works in the following way. The overall installation process is implemented as a process-protocol being executed by a Paffin on a remote host. Students access the process-protocol via the Web GUI in a browser connecting to the remote host. Each step of the installation is implemented as an Agent-Activity.

[7] `https://eclipse.org/` (last accessed May 28th, 2017)

Each of these Agent-Activities contains one workflow task providing descriptions of what exactly the students have to do. Additionally, the Agent-Activities also contain the automated functionality steps as agent actions. Those agent actions are configured in such a way that they access the student's system via a secured connection.

For example, the eclipse installation file may be transferred via the secured connection from the remote host to the student's system. The workflow task can then display the exact file location where the student may find that installation file. This kind of dynamic parameter display in workflow tasks is already available in the Paffin-System (see Section 11.2.1). Furthermore the task would also display the instructions the student needs to follow.

Other tasks may require the student to enter manual commands, e.g. for retrieving sources from the repository or building the environment. Here, the student could be asked to enter the command in the workflow task in the Paffin-System. The GUI here would simulate the system console to enter the command. The Paffin on the remote host would check the command the student entered. If there were any errors it would report back to the student with hints about what was wrong with the command. Only when the student entered the command correctly would the system continue. Then the Paffin would use agent actions to actually execute the command on the student's system. This way the student would be taught the correct syntax of the commands without risking errors in the installation resulting from faulty commands.

Overall, the Paose installation wizard would utilise workflow tasks for instructing and teaching students while using agent actions for the parts of the installation that could and should be automated. Students would learn how to install and configure their environment correctly, while the Paffins ensured that the installation was correct and working. This scenario is only enabled by the strong coupling of agent and workflow parts in the Paffin-System. The Paffin supporting the student constantly switches between the roles of teacher (workflow tasks designed to instruct students) and installer (agent actions that actually perform installation duties).

11.4.5 Distributed Software Engineering

Another application vision for the Paffin-System relates to distributed software engineering, especially with Paose. Paose (see Section 2.2.3) applies the multi-agent system metaphor not only to the system being designed, but also to the development team. By doing so, the creation of a multi-agent system is described by a multi-agent system with agents (development teams) executing complex interactions (meetings, agreements, implementation, code-integrations) between one another and the system they are creating.

Currently, the interactions between development teams in Paose are relatively unstructured. In small, localised settings, e.g. the weekly sessions of the yearly Paose teaching project, the developments team can organise themselves easily in order to interact effectively. However, once the setting becomes larger and more distributed, the need for some kind of support for Paose development processes becomes clear. Problems even occur when students of the Paose teaching project collaborate for homework. Often, explicit interactions and agreements with other development teams become neglected or even disregarded completely with hopes of clearing up any miscommunications and issues later on in the next weekly sessions. While this usually works out in the smaller teaching contexts and systems, the issues already negatively affect effectiveness and performance in the development teams. Consequently, a structured support system for Paose is desirable.

Building such a support system as a workflow system would not be sufficient. Each development team has a workflow. That workflow fully describes what the development

team is doing at any time. Therefore, it is justified to equate the workflow with the development team, i.e. the development team is its own workflow. When development teams in Paose interact, their workflows also interact. Capturing interacting workflows without merging them, thus violating the autonomy of each development team, is quite difficult with traditional workflows. In order to do so, the executing WFMS would have to merge the workflows for the interaction and then somehow unmerge them afterwards without sacrificing any data in either one.

Building a Paose support system as an agent system would also not be sufficient in of itself. Capturing the development teams as agents in a multi-agent system aligns perfectly with the guiding metaphor of Paose. Interactions are fully and naturally supported. However, what is missing is the process the developers execute. Using agents to support Paose would require the development team processes to be in some way coded into behaviour of the agents. While this is possible to implement, capturing any form of process views beyond local contexts, i.e. restricted to one agent or one group of agents, would be difficult.

Workflows miss the capabilities to capture interactions, while agents miss a coherent and substantial process perspective. The frame of this problem suggests that a combination of agents and workflows would provide solutions. Paffins are therefore promising candidates for the implementation of a Paose support system. They can represent development teams as both agents and workflows, thus supporting all requirements. Furthermore, the alignment of structure and behaviour in Paffins can be utilised even more, as will be discussed later.

Please note that building an application to support Paose with Paffins is not directly related to extending Paose itself to support Paffins instead of traditional agents. That topic is picked up again in Section 12.6.3. Here, a support application for traditional Paose is discussed. However, it is probable that the support for developers would be compatible when Paose is based on Paffins instead of agents.

The general principle of a Paose support system utilising Paffins would be to have individual developers and development teams be represented by Paffin entities. Each Paffin would feature a set of process-protocols implementing the rough development cycles for each Paose artefact. Within these cycles, best standard procedures and best practices would be supported through Agent-Activities. These best practices would include, for example, making a proposal for a new ontology concept or arranging a new interface between components.

Figure 11.40 illustrates a mock-up of a Paose support process-protocol. The overall process-protocol is modelled around the different phases of development, e.g. design, implementation, integration, test. These are indicated with the larger area boxes around groups of Agent-Activities. Between each phase, there is an evaluation about the achieved results. Depending on the outcome of the implementation, either the next phase is started or the process-protocol moves back to a previous phase in an agile way. While each phase is active, a number of best practice Agent-Activities are activated. Using reactive triggers from the GUI, these Agent-Activities could be started by developers to initiate the individual best practices. Besides the above mentioned best practices, standard development tasks would also be included in the Agent-Activities of a phase. Note that the Agent-Activities of each phase differ, though some may be reused in different contexts as well, e.g. changing an interface may happen during implementation and during integration.

One of the main challenges in implementing these best practice Agent-Activities would be to capture and guide the interactions between development teams. Development

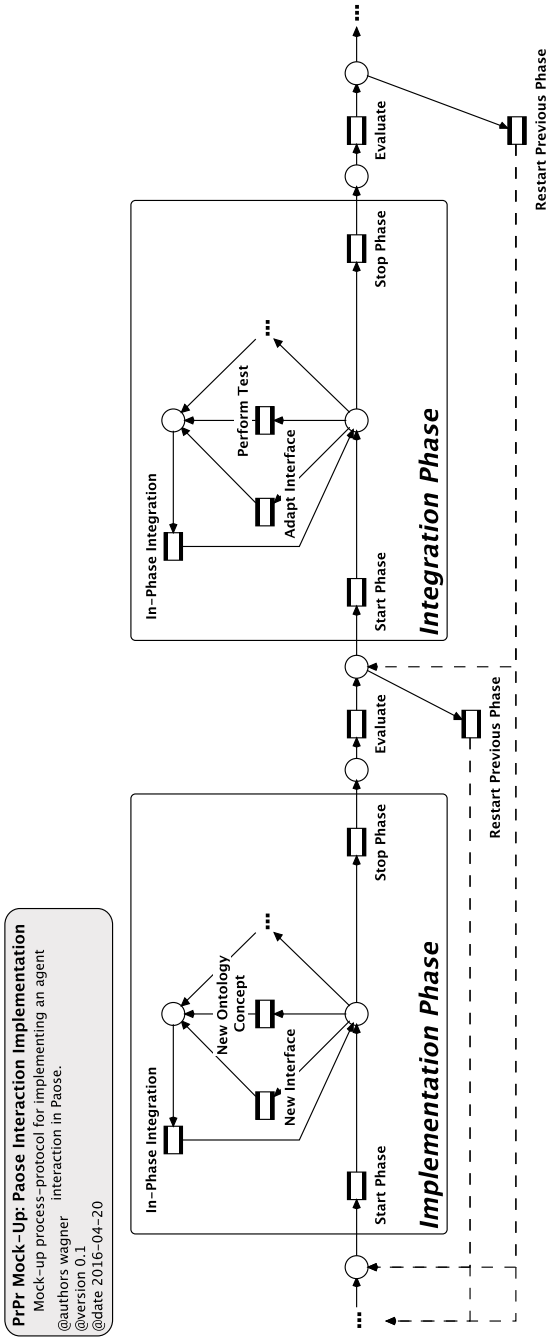

Figure 11.40: Mock-up process-protocol for Paose interaction development support

teams need to interact quite often in Paose, as the separation of structural (agent roles) and behavioural (agent interactions) development responsibilities is enforced through the Paose matrix (see Figure 2.12 in Section 2.2.3). In optimal Paose each interaction and role is implemented by a different development team. This means that whenever there is a handover of data or control between a structural and a behavioural element, the responsibilities switch to a different development team and a common interface has to be agreed upon.

The following describes a possible realisation of supporting such interactions. It uses the example of adding an interface between an interaction and a DC: The development team of interaction A discovers that an interface to DC B needs to be added. They reactively start the Agent-Activity for adding or changing an interface in their process-protocol. The Agent-Activity supports the following process. Agent actions and workflow operations are indicated in the following descriptions to illustrate how the executing Paffin changes between agent and workflow.

Agent: Coordinate with all other development team Paffins and gather current data about agent roles and DCs

Workflow: Provide the data about roles and DCs to the current development team. They select DC B and confirm the task.

Agent + Workflow: Contact the Paffin representing the development team for DC B. That Paffin starts a new process-protocol. In the Agent-Activities of that process-protocol the counterparts to the interaction Agent-Activity functionality are implemented. It starts with providing a workflow task that the developers can request when they are ready to deal with the interface. When the DC developers request the task, a new task is also automatically assigned to the interaction developers.

Agent + Workflow: Both the interaction and DC development Paffins start the workflow task incorporating a chat client as the specialised GUI. Here, the development teams communicate and agree on an interface. Another part of the GUI contains the functionality to specify and document the results. Both development teams can edit that specification. Only when both development teams agree to an interface specification can the task be confirmed. The result of that task is, for both teams, the interface specification.

Agent: The interaction development Paffin stores the interface specification internally and finishes the Agent-Activity.

After the best practices Agent-Activity is completed, an additional integration Agent-Activity reads the result interface specification, checks it again for compliance with, e.g., code documentation and formatting standards, and communicates again with the DC development Paffin to ensure that the versions in both Paffins match exactly. It then stores the result in a project repository accessible by all authorised development teams so that they can use it in their work.

By realising all the possible activities concerned with Paose in similar ways, the interactions between development teams can be structured and guided. This would fulfil the requirements of the Paose support system as described above. This system utilises agent and workflow mechanisms to large and concurrent degree. Traditional agent or workflow solutions would be hindered and restricted in many of the affected areas.

Overall, Paose development can actually be considered as an inter-organisational workflow (see Section 2.3.3). Every development team is an organisation and the different organisations interact in order to achieve a common goal. In general, inter-organisational

workflows are an application area in which the integration mechanisms of the Agent-Activity approach and the Paffin-System can be utilised quite well. This is discussed further in Section 11.5.

The paragraphs above discussed the basic support that Paffins could provide in Paose: Use workflow tasks for user interactions and agent actions for coordination between the development teams. However, the inherent duality of structure (through agent states) and behaviour (through workflow states) in the Agent-Activity concept and Paffin-System could be utilised even further. A similar duality exists in Paose, especially in the Paose matrix which is arranged according to structure and behaviour. Finding ways to utilise the highly aligned dualities in both Paose and the Agent-Activities could support Paose even more.

The idea here is to equalise the development teams and the multi-agent system they create to an even higher degree. Development processes and the processes being developed should, on a system level, be indistinguishable. The same holds for development teams and the agents they develop, as well as for all other development and modelling artefacts in the system being created. Each element of both the development level *and* the level of the system being developed would be represented as a Paffin. As it stands, any coordination in Paose happens between development teams and is about an element of the system. By equalising the development and the system being developed as Paffins, the nature of the coordination would change. Now the elements, as Paffins, are also part of the coordination directly. The content of the coordination would transcend beyond the element in question into a new meta-level.

Taking the previous example about adding an interface into account, the interaction previously only involved the two development teams for the interaction and the role. Now, the interaction would also take into account the Paffins representing the interaction itself and the DC. The general process would remain, namely that the development teams agreed on an interface between the interaction and the DC. However, the Paffins representing the interaction and the DC would also be involved. Their role would be limited, as development teams would lead the interactions, but they could raise issues regarding incompatibilities of proposed changes with other elements or parts of the system. In the end, the interaction would still produce a new version of the interface. Here, the equalisation would take full effect. When the interface would previously only be agreed upon by the development teams, it could now be directly implemented in the Paffins representing the interaction and the DC. Their role in the interaction is to ensure that the interface is compatible and realisable, so it is feasible to have it automatically implemented after all parties agree. In the end, instead of just guiding and supporting the development in Paose, the Paffins would actually implement the development as well. This would mean that the equalisation of the elements of the development and the system being developed would also equalise the interactions in the development and the implementation/realisation in it.

Note that the previously sketched equalisation is only a rough sketch of an idea. Providing a support system as illustrated before with more classical agent and workflow ideas is already an extensive undertaking. Going beyond that and changing the view on the development as fundamentally as proposed here is clearly beyond the scope of this current thesis.

11.5 Inter-organisational Contexts

Highly coupled processes and services are common nowadays. It is often not economically feasible for an organisation to operate on its own. In such cases organisations can collaborate

with other organisations in order to create products or provide services to the customer. The research area of inter-organisational workflows (see Section 2.3.3) examines such collaborations. Since the AGENT-ACTIVITY approach and PAFFIN-System utilise workflow principles, the following discussion uses the term *inter-organisational context*. While the equivalent term inter-organisational workflow is more common, the term inter-organisational context is more suitable here since it emphasises the extended nature of the AGENT-ACTIVITY approach.

Inter-organisational contexts can arise for a large number of reasons. Organisations work together to, e.g. increase output or expand operations with new products or in new areas. Often, the desired goal is economically unfeasible (e.g. associated with too high costs) to achieve alone. Other types of inter-organisational contexts can arise from outsourcing or when new organisations are created as corporate spin-offs. Corporate spin-offs may still closely interact and collaborate with the parent organisation, yet are often separate, independent entities which should be treated as such.

A common theme in both outsourcing and spin-offs is the extraction of structures and processes from the original organisation. The transition between the extracted structures and processes and the ones remaining within the original organisation now needs to cross the organisational border. Where issues and problems could be handled with relative ease before internally, they now involve different and separate organisations. Due to questions of, e.g., organisational autonomy, confidentiality of data or even just geographical differences (e.g. time-zones), the interactions between organisations need to be properly defined, executed and supported.

Modelling and supporting inter-organisational contexts faces a number of challenges due to the requirements introduced by the multi-organisational nature. In addition to the inter-organisational challenges, the challenges of traditional mono-organisation business process management need to be dealt with as well. However, the traditional challenges are often made more complex in the inter-organisational setting, e.g. considering confidential data access.

Applying the concept of an integrated entity can help to address the challenges and requirements of inter-organisational contexts. One of the particularities of inter-organisational contexts is that organisational entities in one organisation need to be able to act in different subcontexts. An organisational entity may, e.g. be involved in an internal process while at the same time interacting across the organisation border for the inter-organisational collaboration. The organisational entity may even be the element of the organisation that links the internal process with the inter-organisational collaboration. Conceptually, the organisational entity is, at that time, *both* (part of) a workflow (internal process) *and* an agent (inter-organisational collaboration). This duality is naturally supported by the AGENT-ACTIVITY approach and PAFFIN implementation. A PAFFIN entity can act as agent, workflow, both or a hybrid of both. It has the capabilities to capture the duality clearly present in inter-organisational contexts in terms of modelling, concepts and technical realisation.

The AGENT-ACTIVITY is, by design, a flexible tool for a wide number of application domains. Consequently, there are different variations of how to apply the concept for inter-organisational contexts. Figure 11.41 depicts three possible applications.

In Figure 11.41, each organisational entity is represented by a PAFFIN (symbolically indicated by two connected AGENT-ACTIVITIES in a grey box). The level of abstraction for the organisational entities is entirely dependent on the modellers and can range from the individual worker or technical resource over whole departments to entire (sub-)organisations. Each PAFFIN may actually represent entire PAFFIN (sub)systems. This can be realised either

via hierarchic workflows and workflow roles within subsystems or when the subordinate resources mechanism of the PAFFIN-System is extended. Communication between the PAFFIN entities in all three scenarios can generally be asynchronous and agent-like or use workflow-task-like interaction between PAFFIN entities according to the WORKBROKER principle (see Definition C.13). The choice here is up to the modeller, as PAFFINS support both agent and workflow-like interactions and it is possible to freely combine the two.

Figure 11.41 *(a)* describes an option in which the inter-organisational interface is modelled explicitly and is conceptually centralised in a PAFFIN. That inter-organisational PAFFIN controls the overall inter-organisational context (workflow) and oversees and manages the interaction and communication of the entire inter-organisational system. Communication internal to the organisation is also overseen and managed, although it is possible to exempt internal communication and interaction from the management to gain some flexibility and reduce overhead. The inter-organisational PAFFIN entity is not directly associated with one of the organisations and is executed (conceptually) between the organisations. Figure 11.41 *(b)* shows an option where the inter-organisational interface is also modelled explicitly, but is distributed in a number of PAFFINS. Each of these PAFFINS represents the inter-organisational interface for its own organisation. Similarly to option *(a)* these PAFFINS control the inter-organisational context (workflow) and oversee the interaction and communication between and within the organisations (with possible exemptions for internal aspects). In that way the PAFFIN entities act as intermediary brokers between the organisations. Lastly, Figure 11.41 *(c)* describes an option with an implicit inter-organisational interface. There are no PAFFIN entities directly responsible for representing and handling inter-organisational aspects. These aspects are included and implemented directly within the PAFFINS of each organisation, i.e. each PAFFIN is responsible for its own interaction and communication inside and outside of the organisation.

Examining the options in Figure 11.41, it is noticeable that they focus on different aspects of the AGENT-ACTIVITY integration approach. Option *(a)* is very similar to a more classical BPM and workflow approach. The inter-organisational interface (workflow) is explicitly modelled and executed by its PAFFIN as a workflow engine, with distributed resources depending on the process representation and execution within the PAFFIN entity. Option *(c)*, on the other hand, is similar to an agent-oriented approach. Each organisational entity (role) is represented by a PAFFIN entity as an actor/agent. These PAFFIN entities interact and communicate with one another directly and without outside control. Even though options *(a)* and *(c)* are conceptually close to the classical approaches of agents and workflows, they nonetheless profit from the integration provided by the AGENT-ACTIVITY and PAFFIN entities, as is discussed later in this section. Lastly, option *(b)* offers a middle ground. The inter-organisational interface is explicitly modelled, similar to a workflow approach, yet distributed among the autonomous organisational PAFFIN entities.

All in all, each of the options represents a viable approach to inter-organisational contexts. Depending on the type of collaboration and the specifics of the particular application domain the options may be more or less suitable. As discussed in Section 2.3.3, [van der Aalst, 1999a] identifies six types of interoperability between organisations in inter-organisational contexts: *(i) Capacity sharing* (central control of distributed tasks), *(ii) Chained execution* (sequential subprocesses), *(iii) Subcontracting* (distributed subprocesses), *(iv) Case transfer* (distribution of process instances), *(v) Extended case transfer* (distribution of process instances with specialised variants) and *(vi) Loosely coupled* (distributed, variable subprocesses with only interaction fixed). All of these types of interoperability can be captured by the options described in Figure 11.41, yet some options are especially well suited for certain types of collaboration. Option *(a)* is very well suited for types (i), (iv) and

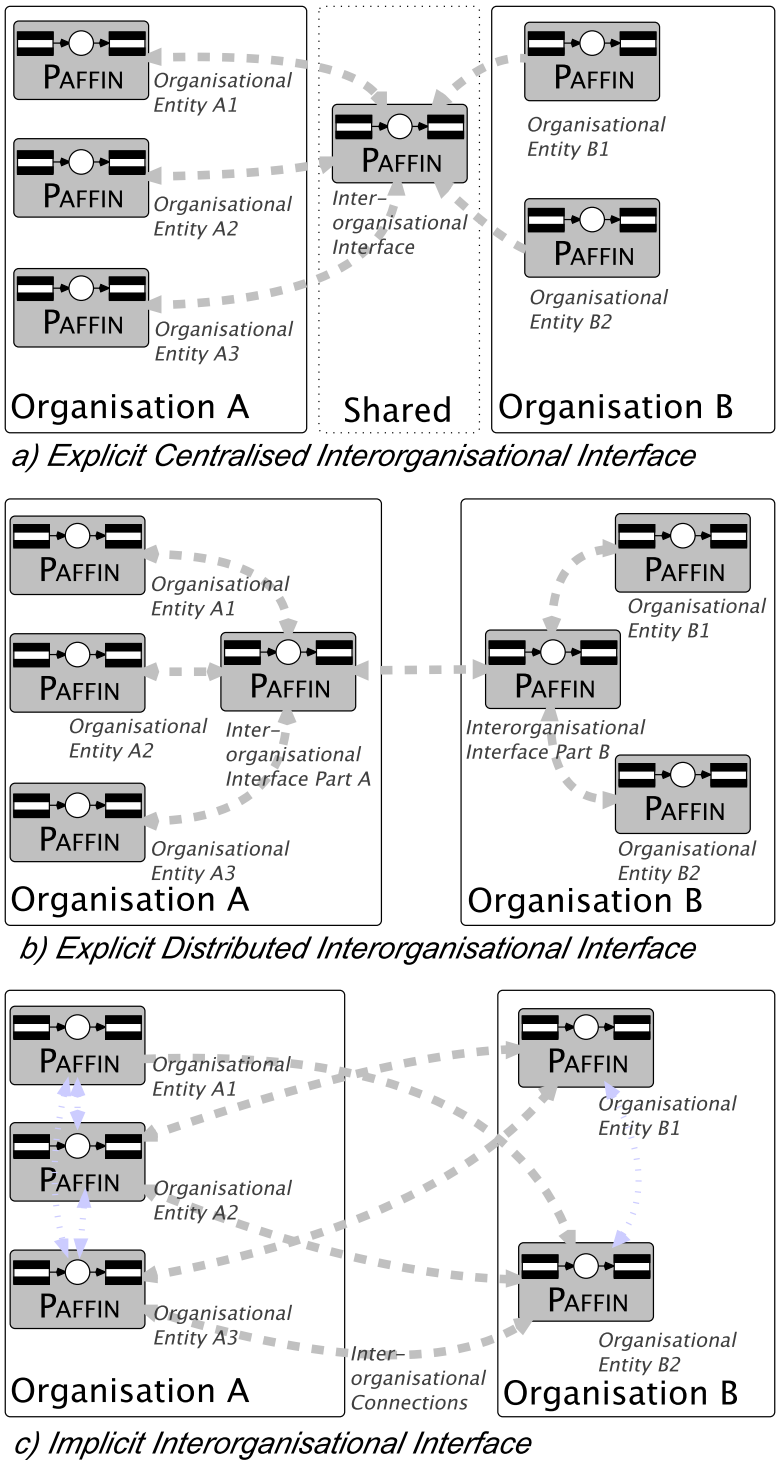

Figure 11.41: Inter-organisational application options for Paffins

(v) as its more centralised nature can manage the distribution of tasks and cases quite well. Option *(b)* is well usable for types (ii), (iii) and (vi) since control can be passed between organisations freely and flexibly. Option *(c)* is practical for types (iii), (iv), (v), and (vi) as its very loose coupling and flexibility can be useful in these types of collaborations.

Note that the ideas represented by the options may also be mixed. It is feasible that some parts of the inter-organisational context require the flexibility of options *(b)* or *(c)*, while some profit from the strong control aspects of option *(a)*. It is also possible to mix the options on the different levels of the organisations themselves. For example, the different departments of an organisation may be aligned according to option *(b)*, while the inter-organisational collaboration may be aligned according to option *(a)*. The Agent-Activity provides the flexibility for modelling not just the inter-organisational but also the **intra**organisational architecture of a system in a uniform way. As a concept, it is adaptive and versatile enough to support organisations on both the micro and the macro-level.

Overall, Agent-Activities and Paffins provide a novel way to consider and design complex systems. In inter-organisational contexts, though, the effects can be made clearly evident, since the challenges and requirements emphasise the duality of structure and behaviour very strongly. Paffins can act as both agents *and* workflows, if necessary at the same time. As described before, this can be necessary in inter-organisational contexts when an entity needs to act as a process internally and as an actor externally. Paffins provide a suitable tool for mentally and abstractly modelling this duality which can be found in the real-world domain. If this duality is not adequately captured, a model needs to find ways to circumscribe it. This can lead to a distortion of the mapping between (software) model and real-world domain. Even in general, non-inter-organisational contexts the duality is often present, which makes the ability to capture it relevant.

The concept of a Paffin entity is, in itself, organisation-oriented. Apart from the obvious representation of entire organisations (see option *(b)* in Figure 11.41) each organisation usually consists of different departments and levels of hierarchy. Using Paffins as modelling abstractions for the internal parts of an organisation creates a simplified overview with details being added later. The versatility of Paffins can then be utilised for these details, which are the elements *within* the internal parts of the organisation, as well. The individual modelling context of an element decides if the Paffin representing it is an agent, a workflow, both, something in between or even yet another full system representing a larger subcontext.

Another beneficial aspect is that even if a Paffin acts as an agent or workflow, it can still exhibit properties, mechanisms and capabilities of the other concept. This means it is possible to exploit properties of either concept to enhance the other one. Such partial integrations were already discussed as related work in Chapter 3 and will be more thoroughly examined and compared to the Paffin-System in Chapter 13. In short, partial integrations may improve agents with workflow management concepts and principles and workflows with agent management concepts and principles. The results and ideas behind both of these kinds of enhancements are available in the Paffin-System. Workflows can, e.g. feature agent intelligence or mobility, while agents can feature, e.g. task atomicity or resource management. Concrete examples of this kind of transference of properties were discussed in the application prototypes and visions of this chapter.

Of course, there are also drawbacks and disadvantages as well. While the approach includes agent and workflow aspects, which are well known and researched, it moves beyond the individual concepts. It basically features two modelling constructs, agent and workflow, on the same level of abstraction. Although this is a desired effect, which actually enables the entire approach in the first place, it is more complex and unfamiliar to handle for

modellers. Having to manage and coordinate two main modelling constructs of equal importance requires more careful organisation of modelling efforts. The complexity of the approach is discussed further in later chapters.

In general, the complexity and expressiveness of the approach, the inherent duality and the versatility are positive effects. However, to fully utilise these effects a system needs to be large enough. Small-scale systems, which are relatively "flat" (i.e. do not involve many partners or do not contain complex hierarchies or structures) or where the requirements simply are not diverse enough, cannot benefit to the same degree as large, complex systems can. As discussed in the context of the general integration specification and vision, an integration of agents and workflows, as realised by Paffins and Agent-Activities, is intended for large-scale systems. In such systems structure and behaviour may be individually very complex and may also feature intricate dependencies and interactions on different levels of abstraction. Inter-organisational contexts usually pose these kinds of challenges, which is why they are a prime application area for the Paffin-System. Small-scale systems still benefit from the integration, as showcased in the application prototypes presented in Section 11.3. However, in some cases it may not be worth the costs and effort associated with the introduction of such an elaborate and extensive approach.

From a technical standpoint there are also some issues. The complex management of two abstractions, as well as the handling of the flexible nature of Paffin entities requires an increased amount of communication overhead. Interactions between Paffins acting as agents but using workflow tasks to interact require more message and data exchange then classical agent interactions. Entity management within the execution platforms also requires more data exchange due to, e.g. workflow resource status tracking. These issues, too, are picked up again in later discussions.

The remaining discussions in this section are oriented around the classification of challenges of inter-organisational workflows given in [Legner and Wende, 2007] (see Section 2.3.3). That classification of six groups of challenges is well suited as a basis for a structured discussion.

Representation of inter-organisational business processes Business processes modelled within an inter-organisational context need to be clearly represented. This includes not only the processes themselves, but also the structure within and between the organisations. Suitable abstraction mechanisms are also needed.

Regarding the representation issues, the approach relies on principles grounded in established research. Business processes are represented within the process-protocols of Paffin entities. Depending on the implementation, the scope of each individual process-protocol may reach from only individual parts of the context (see option *(c)* in Figure 11.41) to overarching inter-organisational parts (see option *(a)* in Figure 11.41). Paffin behaviour defined in Agent-Activities and process-protocols is similar in nature to workflow nets. They are relatively easy to model, understand and monitor. It is also possible to apply the ideas of workflow verification [van der Aalst, 1997], although this is generally outside of the scope of this thesis and only shortly addressed as future work in the following chapter.

The structure of and between organisations is given by the Paffins and the relations between them. In this context the Paffin entities act as agents and represent the organisations as interacting agent systems. Inter-organisational contexts are usually distributed in some sense. Modelling sophisticated distributed systems is a strength of the agent concept, which is fully utilised in the agent-like representation employed here.

Regarding the mechanisms for abstraction, Paffins can be used to represent not just individual entities, but entire systems. Similarly, the Agent-Activity describes an

abstract task consisting of a number of individual actions and operations. Both can be used to abstract from details irrelevant for a particular situation, e.g. coarse modelling. AGENT-ACTIVITIES represent a behavioural abstraction, while PAFFIN entities represent a structural one.

Allocation of tasks to actors Responsibilities for subprocesses and tasks of an inter-organisational business process need to be clearly assigned and managed. Tasks need to be allocated to the correct organisational entity and executed by the correct worker or resource.

The granularity of responsibilities is given by the AGENT-ACTIVITIES. Each PAFFIN is directly responsible for the execution of AGENT-ACTIVITIES in its own behaviour. It can either execute it itself or it can delegate it, e.g. via WORKBROKER principles, to another PAFFIN entity, possibly as a subordinate resource or within a subsystem. The responsibility, though, remains with the original PAFFIN entity and that entity ensures that the correct worker or resource is active. Shared responsibilities are also supported, by extracting the overlapping parts of the inter-organisational context into a separate inter-organisational PAFFIN. That entity, which is shared by the involved organisations or departments, then implements the shared responsibility. Option *(a)* in Figure 11.41 implements the central PAFFIN with such a shared responsibility.

The PAFFIN entities clearly separate and allocate the responsibilities to organisational entities and the possible subsystems these entities represent. The entities partition the responsibility, while the AGENT-ACTIVITIES partition the processes. There is no uncertainty regarding responsibilities in the approach, as shared or overlapping responsibilities are explicitly modelled and supported.

Alignment of semantics Different organisations may have different understandings of processes and terms. For an inter-organisational collaboration to work, there has to be an understanding within the common vocabulary. PAFFINS communicate with one another using a common ontology, the creation of which is part of the modelling process. Translations between definitions of terms and objects in different organisations need to be made available to PAFFINS involved in inter-organisational interactions. Here, agent intelligence can be utilised in PAFFINS to ensure that the different understandings are correctly translated in the involved ontologies.

Decoupling of internal and external processes Internal processes of an organisation contained within an inter-organisational context need to be decoupled from the (external) interactions with other organisations. This decoupling has to be ensured in order to protect the autonomy of each organisation involved in the inter-organisational context.

The division of processes onto PAFFIN entities partially addresses this issue. If, like in options *(a)* and *(b)* from Figure 11.41, the communication and interaction with other organisations is handled by specialised PAFFIN entities the decoupling is implemented here. These PAFFINS can distinguish between internal and external communication and enforce the decoupling within their behaviour. This creates two separate, logical interfaces which these entities operate. However, these interfaces are only logical/virtual and still use the same technical interfaces (agent or workflow communication). Consequently, modellers still need to take care of the decoupling manually, although in a structured way.

The problem is more severe when PAFFIN entities from different organisations can communicate with one another directly. Modellers need to decouple internal and external processes (behaviour) for PAFFINS manually. However, this does not critically affect the

autonomy of an organisation. As described in the area of challenges concerning representation, a PAFFIN clearly belongs to an organisation. This means that the organisation has complete control over that entity. Even if internal and external processes are not completely decoupled within the PAFFIN, they are decoupled over the different PAFFIN entities involved in the interaction. Furthermore, another issue strongly relating to autonomy is that of confidentiality and security, which is discussed next.

Visibility of internal processes to external partners The processes of an individual organisation in an inter-organisational context contain both organisation-internal and external aspects. Internal aspects represent, for example, the actual work being done inside of the organisation, while the external aspects represent the interactions with other organisations. Internal aspects of the processes can include internal (confidential) data about the organisation that must not be available to other organisations. Consequently, it is very important to clearly and securely encapsulate the internal aspects of the inter-organisational context.

PAFFIN entities are autonomous system components, due to their agent aspects. Like agents, they clearly encapsulate their data and behaviour. By using encryption or other security mechanisms it is possible to limit the visibility of PAFFIN behaviour to only what is necessary for the collaboration (i.e. the external interface and expected communication exchanges). The autonomy and encapsulation of the PAFFIN entity can be transferred to the organisation they represent so that organisational autonomy and confidentiality are ensured.

Formal specification of process interfaces For an inter-organisational context to work, the interfaces between the organisations have to be clearly defined. The form, time and scope for each interaction with another organisation need to be modelled and represented clearly.

In the PAFFIN-System interactions of the inter-organisational context can be handled and represented in a number of ways. The most straightforward solution is to utilise the agent-like representation of the structure of the organisations. Agent interactions and their modelling are a well-established and suitable for distributed systems such as inter-organisational contexts. Another option is to utilise the workflow aspects of PAFFINS, especially the WORKBROKER principle. Using this option results in a task-like interaction between organisational entities (within and between organisations).

Consequently, modellers can choose whether to define the interface in established agent- or workflow-like fashion. This provides a flexibility in choosing the most suitable interface for a concrete interaction between organisations.

Support for alignment with multiple partners Inter-organisational context are not necessarily limited to two organisations. Since the PAFFIN-System has no restrictions regarding the number of involved PAFFIN entities, there are also no restrictions to the number of involved organisations. In fact, for the modellers in a single organisation there is little conceptual difference when modelling with one or more partners. Since the inherent mechanisms of the PAFFIN-System already take care of issues like encapsulation, autonomy and explicitness of interfaces the modellers can focus their attention and modelling efforts on the content of the interactions between organisations rather than on the circumstances.

In conclusion, the AGENT-ACTIVITY integration approach and PAFFIN-System implementation offer suitable solutions to the challenges of inter-organisational contexts. In some

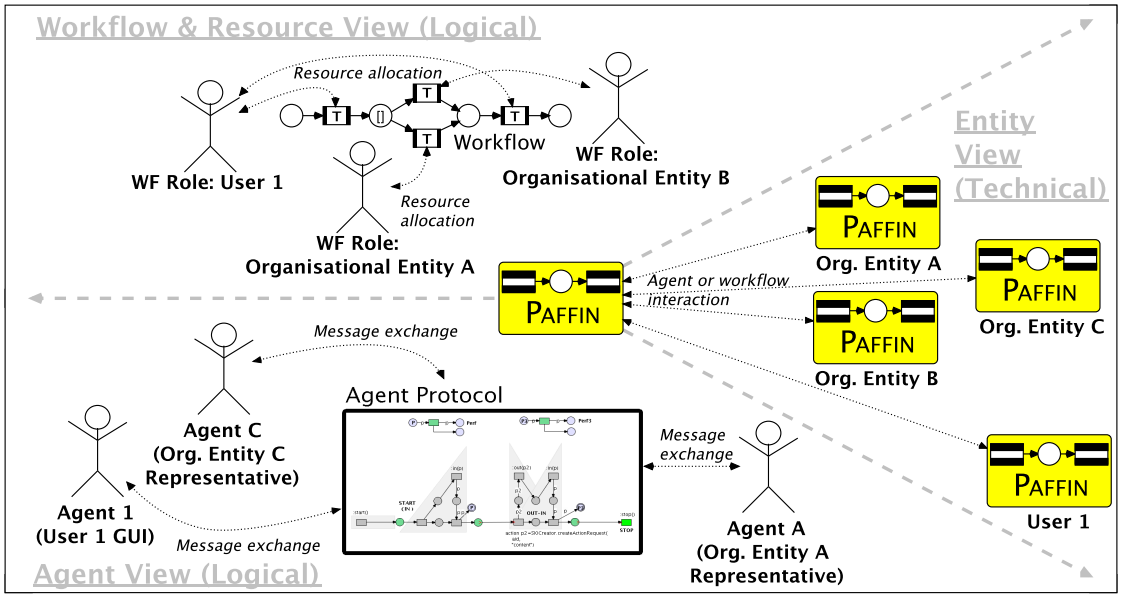

Figure 11.42: Duality and mental modelling

cases the solutions are taken directly from the individual concepts of agents and workflows. In these cases the added value of using the Agent-Activities and Paffins is that the solutions of *both* are available, which means that there are no compromises. For other cases though, the approach and implementation combine the capabilities of agents and workflows in ways that have previously not been available. The concurrent representation of behaviour through workflow concepts and structure through agent concepts, as well as the choice of having organisational entities interact in both agent and workflow fashions are examples of this.

Figure 11.42 illustrates this concurrent representation of agent and workflow in a Paffin entity. Note that Figure 11.42 represents a more concrete version of Figure 7.4 from Section 7.2.1. There, inter-organisational contexts were discussed for the general integration vision. Here, the inter-organisational contexts are considered from an Agent-Activity and Paffin perspective. Clearly, the views discussed previously align with what the Agent-Activities can realise. The general alignment of Agent-Activity integration approach and the integration vision is further discussed in Section 12.1.

The central Paffin in Figure 11.42 is active as a workflow (upper logical view). It is also active as an agent (lower logical view). In fact, it can be or represent each of the elements indicated in those views in Figure 11.42. It can even be or represent multiple of these elements. The central Paffin is therefore, logically, both an agent and a workflow in concurrent and/or possibly related contexts. Workflow and agent (protocol) may be related (e.g. agent protocol provides additional data to the workflow), but are not necessarily so (i.e. agent protocol is unrelated to the workflow). Technically, the Paffin uniformly communicates *only* with other Paffins using either agent or workflow interactions (right-hand technical view).

11.6 Application Conclusion

This chapter dealt with applications in the context of Agent-Activities and the Paffin-System. While the descriptions emphasised realisations in the Paffin-System, the results

are generally applicable to the conceptual Agent-Activity approach as well. Technical details and available features may change when considering another implementation of the Agent-Activity concept, but since the Paffin-System, as established in Section 10.4, fully realises the Agent-Activity concept and integration approach these difference are expected to be negligible. Each section of this chapter discussed a different aspect of applications in the Paffin-System:

- Section 11.1 discussed how to model applications in the Paffin-System. This substantiated the basis provided by the Paffin-System for any system development. It also separated the Paffin framework from those parts of the system that are application-specific.
- Section 11.2 described the maturity of the Paffin-System w.r.t. application modelling. It discussed the feature set of the Paffin-System beyond the basic ability to model and execute Agent-Activities. This section emphasised those features of the Paffin-System that contribute to the practical applicability. While some are essential realisations of implicit parts of the Agent-Activity approach, others are completely optional, yet improve the usability in terms of modelling, execution and monitoring drastically. This section also discussed the limitations, though. While the Paffin-System goes beyond the status of a simple proof-of-concept, it still has some technical limitations that should be addressed in future work in order to make the Paffin-System an even better modelling framework. In essence, this overall section described what is possible in the current state of the Paffin-System and what is not.
- Section 11.3 presented the main contribution to this thesis in the context of applications, the practically implemented application prototypes. Each of the three prototypes illustrates how the Paffin-System can be used to implement practical scenarios.

  The first prototype, the producer-store-consumer system, showcased the versatility of Paffins by providing three different implementation of the scenario. One emphasising agents, one workflows and one mixing the two where it best suited. These three prototypes showed that Paffins can in fact model both traditional multi-agent systems and WFMS, as well as freely combine elements from both. More so, each of the prototypes showcased that even if the Paffin-System is used to merely "simulate" a traditional system, that system can still benefit from the integration mechanisms and characteristics in the Paffin-System.

  The second prototype, the pizza service, then emphasised the flexibility of modelling constructs in the Paffin-System. In the scenario, each step of the original BPMN workflow was implemented by using either agent, workflow or integration mechanisms. This resulted in a well-fitting implementation of the simple inter-organisational scenario.

  Finally, the third prototype, the Paose teaching support system, substantiated the practical and operational readiness of the system. The individual test cases were deployed and tested outside of controlled test environments. Real-life, actual users that had never been affiliated with the Paffin-System showed that the system is indeed ready for practical use. In addition to that, the incorporation of the RedTimer also illustrated another detailed example of how to use the novel integration of agent and workflow functionality in a beneficial way.

  Overall, the practical application prototypes showed that it is indeed possible to model and implement actual systems with it and that those systems can indeed benefit from the new integration mechanism and properties.

– Section 11.4 presented applications that, due to time or scope limitations, could not be implemented for this thesis. Instead, these applications were sketched out as visions. These visions showcased larger and more complex practical scenarios, in which the new integration mechanisms and properties of the Paffin-System could be utilised to full effect. This facilitated the understanding of what the Paffin-System and Agent-Activity might be capable of implementing in the future. While most of the visions assumed some extent of additional functionality to be available in the Paffin-System, for example Paffin mobility, the base concepts and mechanisms are already fully functional in the current Paffin-System prototype.
– Section 11.5 then departed from individual application scenarios and prototypes. Instead, it discussed inter-organisational contexts as a general application area for the Paffin-System. It showed how the challenges of inter-organisational context can be addressed by Paffins and Agent-Activities. In doing so, it began to generalise the practical results obtained from working with the Paffin-System. Overall, the Paffin-System is well suited for inter-organisational contexts, a result which is picked up again when application areas of the Paffin-System are generally discussed in the next chapter.

In conclusion, this chapter has shown that the Paffin-System can in fact be used to create applications that utilise and exploit the integration of agents and workflows to their benefit. This, in turn, affirms the Agent-Activity integration approach specifically and the integration of agents and workflows in general as a favourable approach to modelling software systems. Based on the application discussions in this chapter, the following chapter can now provide a general discussion and evaluation of the usefulness and usability of Paffins and Agent-Activities.

12 Overarching Discussion Areas

Having described the concepts, prototypes and applications developed and researched in this thesis, it is possible to take a look at the overall picture and consider general, overarching discussion areas. This is done in this chapter.

This chapter contains seven sections. Section 12.1 discusses how the integration vision from Chapter 7 has been achieved in the Agent-Activity integration approach. Next, Section 12.2 examines how the results of this thesis are applicable to different and more general fields. Section 12.3 then discusses the different synergies of agents and workflows, i.e. concrete areas where an integration of agents and workflows yields benefits. Section 12.4 presents general strengths and open points of both the Agent-Activity integration approach and the Paffin-System. Afterwards, section 12.5 discusses general application areas. Section 12.6 considers the way forward, providing concrete outlines and discussions of future work. Finally, Section 12.7 concludes the general discussion with an overall evaluation of the discussions and, in general, the results of this thesis. This evaluation considers the goal of the thesis, the research questions and the requirements to the results.

12.1 Relating the Integration Vision

This section relates the results of this thesis, namely the conceptual Agent-Activity integration approach and the practical Paffin-System, to the vision of an integration as described in Section 7.1. Previously, Sections 9.3 and 10.4 established that the Agent-Activity integration approach and the Paffin-System both fulfilled the integration criteria derived from the integration vision. This current section now returns to the more abstract vision and specification of the integration. That vision described, informally, what a system featuring an integration of agents and workflows should look like. Now, it is possible to discuss, on a more abstract level than the concrete criteria, if and how the vision was realised through the results of this thesis.

The following discussions assume, as confirmed by Section 10.4, that the Paffin-System fully realises the Agent-Activity integration approach. Therefore, the discussions refer to the technical Paffin-System and its terminology (e.g. Paffin entity instead of generic integrated entity). In some explicit cases, the more general view of the Agent-Activity integration approach is assumed, when a technical limitation or missing feature of the prototype is concerned.

Relating Structure and Behaviour in an Integration One of the core goals of the integration is to eliminate the indirection between structure and behaviour caused by an emphasis on a modelling construct focussing on only one of the two. The vision of the integration eliminates the indirection by mutually incorporating agents and workflow into one another. Agents execute workflows as their behaviour and workflows are executed by agents as resources. That way neither the agent nor workflow concepts have to adapt from their perspective, yet an integration and equalisation of modelling levels is achieved.

The Agent-Activity integration approach and Paffin-System implementation realise all parts of this mutual incorporation. Integrated Paffin entities using Agent-Activities

can be considered as agents and workflows, a point previously established and discussed further later on.

As agents, Paffins execute process-protocols as their behaviour. Agent-Activities in process-protocols represent abstract tasks so that process-protocols consisting of Agent-Activities represent abstract workflows. Therefore, integrated Paffin entities as agents execute (abstract) workflows as their behaviour. The fact that the abstract tasks and workflows contain both agent functionality and concrete workflow tasks does not matter, as this concerns only the content internal to the abstract task. The basic form of Paffin behaviour is given by the process-protocols and Agent-Activities, regardless of what happens inside those elements. This realises the first part of the mutual incorporation.

As workflows, integrated Paffin entities model their processes within their process-protocols as well. On the abstract level of considering an Agent-Activity as an abstract task, the executing Paffin is always both the engine and the resource for that abstract task. As previously established, integrated Paffin entities are always in some way active as agents, namely as an active participant in the system. Therefore, the abstract workflow defined in the process-protocols is always executed by a Paffin as an agent. Here, however, the concrete level concerning the content is more relevant. Within the abstract tasks there may be concrete workflow tasks. These tasks are managed by the platform WFMS in the Paffin-System or some general management system in the platform and management level of the approach. The resources for these tasks are always other Paffin entities. These entities either represent human users or automated devices in the system or are automatic resources themselves. In either case these Paffins act as active participants of the system and are therefore agents. In case of automatic resources, the workflow execution between engine and resource follows the WorkBroker principles (see Definition C.13), which explicitly relate how agents can interact using workflow and task principles. For all of the above cases and levels the resources executing workflow tasks are either Paffins themselves or Paffins representing the resources external to the system, i.e. human users. This means that for all workflows in the Agent-Activity approach and Paffin-System, Paffin entities are the resources. Even if a Paffin represents external resources, the corresponding communication and interaction partner within the system is still only the Paffin. As active participants of the system, all of these resource Paffins are automatically considered agents. Therefore, the second part of the mutual incorporation is also realised.

Considering both sides of the mutual incorporation shows that the individual perspectives are unchanged. Agent behaviour may be implemented in a workflow-like form of process-protocols and Agent-Activities, but agent functionality still has the same status within the agent perspective and the structure as defined in agent terms in Definition B.3. Workflow structure may be given by integrated Paffin entities as agents, but they are still resources and also retain the same status within the workflow perspective and the behaviour as defined in workflow terms in Definition B.6.

Even the looped hierarchy can be discovered in the Agent-Activity approach, considering a Paffin as an agent, executing a process-protocol containing a workflow task for another Paffin as an agent, whose behaviour yet again contains workflow tasks for other Paffins and so on.

As prescribed in the integration vision the mutual incorporation of agents and workflows eliminates the difference and indirection of modelling levels. Integrated Paffin entities can be modelled as agents or as workflows, with the difference being only the content within their process-protocols.

Relating Modelling Constructs in an Integration Regarding modelling constructs, the integration vision prescribes that the main modelling construct should be able to dynamically model agents, workflows, both or something in between. A modelling construct with such capabilities is called an integrated entity and is defined in Definition B.7. Integrated PAFFIN entities fully satisfy that definition and the requirements set out for them:

PAFFIN *entities as agents:* Integrated entities in the AGENT-ACTIVITY approach and PAFFIN-System utilising only agent actions in their behaviour are effectively (CAPA) agents. Properties, mechanisms and structural role are identical.

PAFFIN *entities as workflows:* Integrated entities in the AGENT-ACTIVITY approach and PAFFIN-System utilising only workflow operations are effectively workflows. The operations model workflow tasks, which in combination model a workflow (net). Properties[1] and behavioural role are identical.

Here, though, there is a limitation. As mentioned before in Sections 9.3 and 10.4, a PAFFIN is always active as an agent as well. It can always (technically[2]) receive messages and is active as a structural component of the system, even if it does not do anything as an agent, i.e. not execute any agent actions in one of its AGENT-ACTIVITIES. However, as the AGENT-ACTIVITY approach prescribes that the content of the AGENT-ACTIVITIES defines if an entity is agent or workflow, this limitation can be neglected. If there are no agent actions to initiate actions or process messages, the passive "agentness" of a PAFFIN does not interfere with it being a pure workflow.

PAFFIN *entities as both agents and workflows:* A PAFFIN entity can be both an agent and a workflow in different ways. If a PAFFIN concurrently executes an agent AGENT-ACTIVITY and another workflow AGENT-ACTIVITY, it is, for that time, both an agent and a workflow. This is the trivial case.

However, there is a more complex case. The internal processes of AGENT-ACTIVITIES can model PAFFIN behaviour in such a way that the entity is also both an agent and a workflow at the same time. Take the RedTimer-enhanced workflow task (see Figure 11.37 in Section 11.3.3) from the PAOSE teaching support system as an example. The PAFFIN here is active as a workflow task representing part of an exercise for teaching PAOSE. However, at the same time, while execution of the task by a user is still ongoing, it gathers and manages the task data and autonomously and reactively calls the external RedTimer tool to track the time users spend in the project management. It also receives agent messages from the GUI to reinitialise the RedTimer command whenever necessary. This makes the PAFFIN an agent as well as a workflow.

In general, agent actions and workflow operations can be freely combined to create full hybrids of agents and workflows. Through the ability to freely create these hybrids, this current requirement of integrated entities and the integration vision is fulfilled.

PAFFIN *entities as something in between agents and workflows:* Partial hybrids are, like the previously discussed full hybrids, created by combining agent actions and workflow operations within internal processes of AGENT-ACTIVITIES. The distinction between full and partial hybrids in the PAFFIN-System lies in the scope of how agent actions and workflow operations are combined. In the RedTimer example the scope is quite large. Here, the PAFFIN models a workflow task that has autonomous behaviour and

[1] Including soundness if the requirements are fulfilled.

[2] The interface is active regardless of whether there is an agent action to receive a message somewhere in an AGENT-ACTIVITY. Technical reception is always considered outside of agent action related reception and processing of messages.

communicates with other system elements during its execution. The PAFFIN clearly is *both* agent and workflow for the execution of the AGENT-ACTIVITY and therefore a full hybrid.

Smaller scopes would be constituted by limited use of functionality between agent actions and workflow operations. A workflow simple sending out status messages via asynchronous messages or an agent utilising workflow tasks only for user interactions would be considered partial hybrids in the PAFFIN-System. A partial hybrid AGENT-ACTIVITY determines a PAFFIN as either an agent or a workflow, not both. It can utilise the respective other concept, yet its role in the system is clear. The important aspect for the distinction of partial hybrids therefore is the exclusive consideration as either agent or workflow.

Ultimately, the distinction between partial and full hybrids is relatively ambiguous, as it is often a question of the perspective of the observer. However, the distinction isn't really relevant. PAFFINS are never restricted and always have the potential to be full hybrids. They have access to the full functionality of both agents and workflows. They are restricted into something that might be considered as a partial hybrid not because of technical limitations, but rather by the requirements of the current application. Partial hybrids of the PAFFIN-System are, in a way, just full hybrids that are restricted in order to achieve some design goal.

From the relevant modelling perspective, partial and full hybrids therefore do not need to be distinguished in the PAFFIN-System, because modellers simply combine the actions and operations they require for their design goal. There is no need to switch between models, modes or tools.

Altogether, this requirement of the integration vision is satisfied.

The dynamic of states of integrated entities is also supported by the PAFFIN-System. The execution of AGENT-ACTIVITIES is fleeting, meaning that the state prescribed by an AGENT-ACTIVITY is only valid while that AGENT-ACTIVITY is being executed. A PAFFIN may execute an agent AGENT-ACTIVITY and therefore be an agent at one point and then execute a workflow or hybrid AGENT-ACTIVITY at the next point and be in that appropriate state. There are no restrictions as to how states of PAFFINS may or may not be changed from AGENT-ACTIVITY to AGENT-ACTIVITY.

Also within a hybrid AGENT-ACTIVITY there is a dynamic between executing as an agent and as a workflow. Take, as an example, the AGENT-ACTIVITY for paying for a pizza in the pizza collaboration application prototype from Figure 11.22 in Section 11.3.2. Here, the customer PAFFIN is a workflow for the customer to input the payment information, after which the PAFFIN turns into an agent to send out the payment and receive the receipt. Any sequence alternating or concurrently executing agent actions and workflow operations is possible in both the approach and the PAFFIN-System.

The ability to have integrated entities be able to be and represent agents, workflows, both or something in between is essential to the integration vision. It ensures that there are no separate modelling constructs necessary for modelling structure and behaviour. PAFFIN integrated entities fully comply with the definition of integrated entities and therefore fully comply with the vision of the integration.

Relating the Integrated Modelling Perspective The modelling perspective in the AGENT-ACTIVITY approach and the PAFFIN-System complies with the idea of *Everything is an Integrated (*PAFFIN*) Entity.* A software system in the approach and prototype consists of PAFFINS cooperating in order to achieve an overall design objective.

A particularity of the approach, however, is an additional specification of the modelling focus. While everything is a Paffin, a Paffin is just an empty shell without Agent-Activities defining what it does. Through the Agent-Activities a Paffin becomes an agent, workflow, both or something in between as discussed in the previous section. Therefore, it is more fitting to say that *Everything is an Integrated* Paffin *Entity Utilising* Agent-Activities is the core idea of the modelling perspective in the approach and prototype. Still, this is only a specialisation and concretisation of the abstract idea and therefore satisfies the vision of an integration.

Just as Paffins can be pure agents or workflows, so can the individual agent-oriented and workflow-based modelling perspectives be applied to such Paffins. If a Paffin only executes either agent actions or workflow operations at a time, the corresponding modelling perspective can be applied. The Agent-Activity and Paffin-System therefore fully support these individual modelling perspectives.

A more interesting case is when agent actions and workflow operations are mixed or if the scope of examination is extended to include both. Then, the amalgamation of the modelling perspectives comes into effect. For partial hybrids, i.e. Paffins that still emphasise either agent or workflow, the traditional modelling perspective can still be applied. The hybrids gain the capabilities of the respective other concept, yet from a modelling perspective they are still considered in traditional terms. For full hybrids, i.e. Paffins that are modelled in such a way that neither agent nor workflow aspects are emphasised, neither of the traditional perspectives offers the correct answers. The main problem here concerns the roles of full hybrids within the structure and behaviour.

A full hybrid is part of both structure and behaviour and therefore needs to be considered with that in mind. For the modelling perspective, this means that one perspective isn't enough. Modellers have to consider a full hybrid in the structure-centric agent-oriented modelling perspective *and* in the behaviour-centric workflow-based modelling perspective. This includes its relationships to other structural and behavioural elements, including other (full) hybrid Paffins. This makes the modelling with full hybrids more complicated, as it requires more effort and careful deliberation. However, the added capabilities of the integration make up for the added difficulty, as is discussed throughout this part of the thesis.

Relation Conclusion In conclusion, the Agent-Activity approach and Paffin-System implementation feature the mutual incorporation of agents and workflows, as well as satisfy the requirements of integrated entities as modelling constructs. With these two foundations of the integration provided, the modelling perspectives for integrated entities also fall into place.

Figure 12.1 illustrates how the integration vision is realised. Paffins as agents represent structural components of the system. Agent functionality is modelled in the agent behaviour given by process-protocols describing an agent workflow. Paffins as workflows are behavioural components by being equated to the process-protocols they execute. With the process-protocols being abstract workflows, the Agent-Activities are abstract tasks, for which the Paffin entities are the resources. This illustrates the mutual incorporation. Paffins as agents execute behaviour workflows, which are again executed by Paffins. Finally, Paffins as hybrids achieve the balance of agents and workflows (and the respective structure and behaviour) by incorporating agent actions for their structural role and workflow operations for their behavioural role.

Overall, the Agent-Activity approach and Paffin-System satisfy the vision of the integration as laid out for this thesis. Agents and workflows are integrated into the

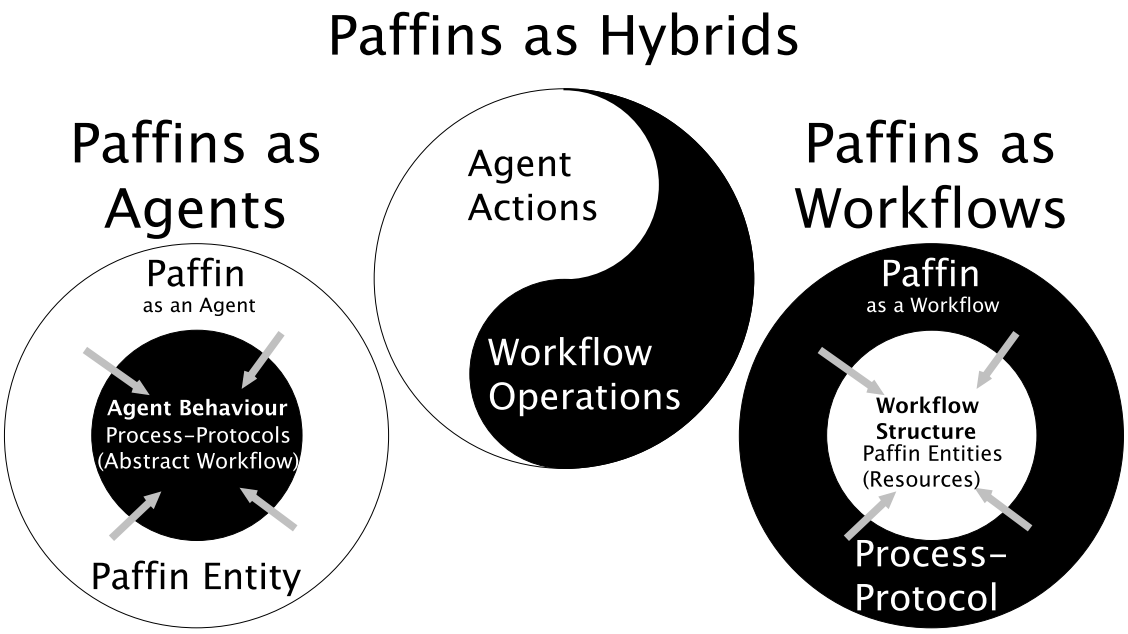

Figure 12.1: Overview of the integration vision in the Agent-Activity approach

Paffin entities, with the integration being based within the Agent-Activities that freely combine agent actions and workflow operations. Thus, the core motivation of the integration, namely the elimination of the disparity and indirection between structure and behaviour is realised.

12.2 Generality of the Approach

The results of this thesis are, in many places, oriented around Mulan/Capa on the agent side and workflow Petri nets on the workflow side. The Agent-Activity integration approach is formulated in a generic way, but its illustration is Petri nets based and its proof-of-concept implementation, the Paffin-System, is completely based in reference nets, Capa and workflow net principles. An open question that is now addressed in this section is how the results, especially the conceptual Agent-Activity integration approach, can be applied in and transferred to other contexts. The section is structured according to the different contexts, becoming more abstract throughout the section. The emphasis of the following discussions lies on the applicability and transferability of the approach to alternative agent and workflow contexts, since these are conceptually closest.

General Agents and Workflows The first thing to consider is how to apply the Agent-Activity approach and other results to general agent and workflow models. In essence, the Agent-Activity integration approach uses fundamental agent actions and basic workflow operations at its core. It combines sets of them into abstract tasks for integrated entities and also integrates them by providing them as a singular modelling construct.

In general, the approach is very generic. All agents have a set of fundamental agent actions and all workflows have a set of basic workflow operations. When these are combined and integrated into a modelling construct to be executed by integrated entities, the result is the Agent-Activity, regardless of the models taken as a basis.

The chosen sets of actions and operations are very generic. For agent actions, sending a message, receiving a message and performing an internal action were chosen as the set of fundamental agent actions. Asynchronous message communication is a standard of agent-orientation, meaning that sending and receiving a message should occur in some way in any agent model. The rest of the functionality is covered by the generic "internal action", which describes any internal operation on any piece of data. For workflows, accepting a workitem, confirming an activity, and cancelling an activity were chosen as the set of basic workflow operations. A workflow is defined in Definition A.2 as the facilitation of a process, which in turn is defined in Definition A.8 as consisting of tasks. Starting and stopping the execution of a task are very generic, yet fully encompassing, ways of describing what happens during the execution of a workflow.

Overall, all of the fundamental agent actions and basic workflow operations chosen for this thesis are general. They may have been influenced by the Mulan agent model and workflow nets, but it is reasonable to assume that they can be discovered in the majority of existing agent and workflow models in some way. Due to their specification, the two sets cover the entirety of agent and workflow behaviour, meaning that they would suffice for an Agent-Activity approach with other agent or workflow models. However, while the chosen actions and operations may suffice, they may not be the best or most efficient options.

When considering other agent models additional actions or different sets may be desirable. This is due to different emphases in different models.

For example, when considering a BDI agent model central concepts are reasoning and deliberation. Many BDI models contain a reasoning cycle. The cycle for Jason/Agentspeak BDI agents [Bordini et al., 2005], for example, consists of (1) perceiving events and receiving messages, (2) updating the intentions, (3) selecting an intention (and the associated plans), (4) executing the plan body and finally (5) creating events from that execution and updating the beliefs (which are all processed in the next iteration of the reasoning cycle). Both parts of step (1) can be covered by the message reception action, because perceiving an event can be regarded as a sensor creating and processing an internal message. Steps (2) and (3) are special internal actions. Step (4) can be any action, as it is part of the application-specific code. Here, the agent either processes data (internal action) or sends messages to other agents. Receiving additional messages outside of the reasoning cycle is also feasible here. Finally, step (5) can be captured by the agent sending messages to itself to process in the following iteration of the reasoning cycle.

While the above shows that it is feasible and possible to capture a BDI reasoning cycle with the three fundamental agent actions selected for this thesis, it also shows that it is not necessarily the best way. Distinguishing between perceiving/creating events and receiving/sending messages facilitates modelling by reducing ambiguity. This may be especially important if the events happen synchronously in the environment, making them distinctly different from asynchronous messages. To a lesser degree, distinguishing updating and selecting intentions from regular internal actions might yield similar increase in modelling clarity.

Still, adding and/or emphasising these actions does not fundamentally alter the Agent-Activity approach. A different selection of agent actions merely determines a different characterisation of the Agent-Activity construct. In the selection made for this thesis, other actions are simple subsumed into the generic set of three actions. Reasoning, decisions making, updating belief sets, perceiving events, etc. are not omitted or ignored in any way. This yields the (agent) characterisation of the Agent-Activity as described and discussed throughout the past chapters. Other characterisations are equally possible and

follow the same principles, while providing different/changed modelling tools. The results and discussions for the current Agent-Activity approach would also be valid.

The same applies for workflow operations. In other workflow models the distinction may be more implicit. For example cancelling may not be modelled explicitly, but as a special variant of completing the activity. This would reduce the basic operations to just requesting and confirming (either successfully or unsuccessfully). Workitem assignment, i.e. a "push" workitem mechanism, could also be an explicit focus. Yet other models may have a strong focus on flexibility of the workflow, indicating that changing parts of the workflow could be a basic operation. Again, these kinds of operations are not ignored, but only offered on a different abstraction level. Flexibility for example is realised through the flexibility in the AAO or by having the integrated entity do an internal agent action to change its behaviour. It is, in this case, not a basic operation, but a direct result of the integration and realisation of the Agent-Activity.

Concerning the generality of the Agent-Activity approach this means that the distinction of what exactly the fundamental agent actions and basic workflow operations are is conceptually mostly irrelevant. As long as the actions and operations cover the entirety of agent and workflow behaviour, the specific selection only defines a modelling focus. While conceptually irrelevant, this modelling focus is potentially very important from a practical perspective. The key is to find the right selection and consideration between too abstract, with a few generic actions and operations, and too concrete, with specific actions and operations for any mechanism. The selection must provide a manageable number of actions and operations covering the entirety of the technical possibilities and yet still emphasising the particularities and core concepts of the chosen agent and workflow models. For the Agent-Activity integration approach in this thesis, the selection of three fundamental agent actions and three basic workflow operations is the most fitting one.

Overall, the Agent-Activity approach with the current selection of actions and operations is generally applicable to other agent and workflow models. Yet particularities of the models may advocate different selections to better capture core concepts of the chosen models.

Changes to the selection might require some changes to the details of reference architecture. If, for example, reasoning would be incorporated, the technical backend would have to be extended to help manage plans, intentions, beliefs, etc. Platform management details could also be affected. Workflow flexibility, for example, might require the platform to track and manage some of the changes and versions of workflows. These kinds of changes, however, only affect the details of the components of the reference architecture. The basic function of each level remains unchanged. Therefore, the discussions relating to the reference architecture are applicable for other agent and workflow models as well.

Up until now, the conceptual aspect concerning the selection of agent actions and workflow operations was covered in this section. On the technical side, however, there is a major obstacle to applying the integration to other agent or workflow models. In this thesis, reference (Petri) nets serve as the basis of the integration. Both the basic agent and workflow models are realised through reference nets, which facilitates and actually technically enables the integration in the first place. The fundamental agent actions and basic workflow operations are all modelled as net components, encapsulated in a process net, which is managed by an AAO net, which is again executed and managed in a process-protocol net and the backend nets. The entire technical basis is net-based. This common basis simplified the development tremendously. If agent and workflow models had not been available as reference nets, a translation, transition or connection between the

different technical bases would have had to be developed before any work on a prototype could have begun. These translations would cascade throughout the management systems, which would have to deal with both bases at the same time.

Therefore, a technical integration of more general agent and workflow models requires either a translation or a common basis. A common basis might be the Java programming language. Many agent models and frameworks are Java-based, and there are implementations of many workflow models available as well. Through careful use of inheritance between classes, integration mechanisms could be realised. It would even be possible to return back to a reference net basis for the integration, as reference nets are Java constructs themselves and fully support the language. Therefore an integration of any Java-based code into a reference net basis is feasible.

Overall, technical challenges and issues can be overcome. Depending on the specific chosen agent and workflow models, realisation effort may vary but there are no fundamental problems that can't be solved. Therefore, the technical side of the Agent-Activity approach is also generally applicable to other models.

As a final note on the applicability and generality of the Agent-Activity approach regarding agents and workflows, the consideration of the Agent-Activity as an agent or workflow itself should be discussed. Up until now, the Agent-Activity has been considered as a modelling construct to define the behaviour of an integrated entity that can be agent or workflow. Now, the consideration changes to the Agent-Activity being an agent or workflow itself. This increases the abstraction and allows discussing the applicability on an equally abstract level. Note that this consideration is only conceptual.

If an Agent-Activity is seen as a full agent or workflow, the process-protocol becomes the integrating modelling construct. The process-protocol, or rather the integrated entity executing the process-protocol, would control the different Agent-Activity agents and workflows and govern their execution. While this would diminish the integration within the Agent-Activities it would increase it on the more abstract level of the entity executing the process-protocol. In a holonic view the entity would be all the agents and workflows it would be executing. Combination and integration aspects would be dealt with internally by controlling the data flow between the Agent-Activity agents and workflows.

For the generality of the Agent-Activity approach, this means that the combination and integration would be moved up to full agents and workflows, instead of combining and integrating agent actions and workflow operations. This way, the entity would be executing not an integration of agent and workflow behaviour, but an integration of agent and workflow systems. This consideration is potentially more helpful when attempting to apply the Agent-Activity approach to other agent or workflow models. By applying this abstract view, the interactions between agent and workflow mechanisms may become clearer. Through the understanding of the interactions on the abstract, conceptual level, the necessary and/or best fitting selection of agent actions and workflow operations for the more concrete conceptual and technical levels would become easier. Instead of breaking down the agent and workflow models into actions and operations beforehand, the interactions between agents and workflows are considered and the selection derived from that. Consequently, the abstract view could facilitate the development of Agent-Activity characterisations for other agent or workflow models.

Structure and Behaviour Agents and workflows are not the only way to model structure and behaviour. Structure may, for example, be represented by objects in object-orientation while behaviour may be represented by services and their orchestration. When attempting to apply the Agent-Activity approach to more general representations of structure

and behaviour, the specifics of those representations have to be taken into account. For behaviour, the differences can mostly be found in the details. Workflows in the current Agent-Activity approach are used to facilitate any general processes according to Definition A.2. Processes are a very generic way of considering behaviour. Therefore, they can be applied to or identified in most representations of behaviour. In fact, service orchestration is often considered to be a continuation or modern form of workflow modelling and management. For structure, the difference may be more severe. Agents distinguish themselves through specific properties and mechanisms. They communicate asynchronously and are autonomous. For the example of object-orientation, objects communicate synchronously through method calls and have little control about their own autonomy. This represents a marked difference to the autonomous and asynchronous agents. These differences may potentially strongly influence the specifics of how the principles of the Agent-Activity approach could be applied.

Another issue relates to the definitions of structure and behaviour themselves. Chapter 4 thoroughly discussed the terms in different contexts. The following chapters then refined them to best suit the goal of an integration of agents and workflows. However, different characterisations may require different interpretations of what structure and behaviour actually constitute. The exact characterisation of the underlying terms also strongly influences the application of Agent-Activity principles.

These principles prescribe identifying a level on which to integrate structure and behaviour where individual elements can be used as the concrete connections. For agents and workflows that level is the behaviour and the elements are the agent actions and workflow operations. For other representations of structure and behaviour the identification may be difficult. In object-orientation, for example, the possibilities could include method calls and object instantiation and termination. For the example of services, possibilities include service call and completion. When the required integration level and elements are selected, the conceptual principles of the Agent-Activity can be applied. The result would be a modelling construct counterpart to the Agent-Activity construct with a different basis.

For a technical realisation, though, the common technical basis is also required. Here, again, the versatile Java programming language may be used. By implementing services as objects and then incorporating more object functionality, e.g. application-specific methods, into them, the technical basis would be provided.

In conclusion, moving away from agents and workflows introduces an ambiguity to the applicability. The essence and principles of the Agent-Activity integration approach may still be applied, but a lot of the establishing work, like the definition of structure and behaviour, need to be redone. Furthermore, finding the counterpart to agent actions and workflow operations in other representations of structure and behaviour may be difficult. However, if an integration basis is found, the idea of an Agent-Activity can be applied.

As a final note, considering service-orientation and web services in the Agent-Activity approach and the Paffin-System is, in fact, quite easy. Through the incorporation of Capa legacy code, the full functionality of the WebGateway (see Section 2.2.3) is available in the Paffin-System. The WebGateway allows agents, now Paffins, to provide their functionality as web services and also access web services. Therefore, services can already be used within the Agent-Activity as special agent functionality. The prototypes presented in this thesis utilise the WebGateway functionality, for example, to realise the web GUI.

The incorporation of web services does not fundamentally affect the Agent-Activity approach and realisation. Currently, it merely extends its capabilities of accessing and providing functionality to a wider context environment. Questions about how the incorporation of web services can be extended to significantly increase the role of services within the Agent-Activity approach are outside of the scope of this thesis.

Beyond Structure and Behaviour Going beyond even structure and behaviour, applying the results of this thesis becomes difficult. Broken down onto the most abstract level, the Agent-Activity approach integrates two perspectives through a shared basis. If such a shared basis can be discovered for two perspectives on a software system, the general principle can also be applied. The Agent-Activity approach takes elements from the common basis, combines and integrates them into a novel modelling construct and then has an entity, which is an amalgamation construct from constructs from the original perspectives, execute it.

What this thesis has shown is how an integration can be achieved for structure and behaviour and how it was achieved for agents and workflows. Beyond that, only the abstract main principle described above can be applied for certain and given the right circumstances. By following the abstract main principle researchers and modellers may find a specific approach on how to integrate their chosen perspectives.

Applicability Conclusion In summary, the results of this thesis are overall applicable and transferable to other contexts. The further removed from the agent and workflow concepts these contexts are, though, the less directly the principles of the Agent-Activity approach can be applied. For other agent and workflow models, different selections in agent actions and workflow operations yield different characterisations of the Agent-Activity modelling construct. However, conceptually, the Agent-Activity approach remains unchanged and can be applied as is. Researchers and modellers do have to take care, though, to provide a suitable technical integration basis in addition to the conceptual basis. Agents and workflows either have to share a common basis or a translation between the two must be available. Moving away from agents and workflows, yet remaining with representations of structure and behaviour, the Agent-Activity principles can only be applied less directly. Both perspectives need to provide elements to combine and integrate and which are then executed by an integrated entity. Here, the main challenge is discovering the correct counterparts to agent actions and workflow operations, yet when this is done much of the Agent-Activity principles can be reused. Beyond structure and behaviour, though, only the core idea can be applied for certain. Here, researchers and modellers will have to reestablish core ideas and find similarities before Agent-Activity results can be used.

To conclude, this section answered the question how the results of this thesis can be applied beyond the context of Mulan/Capa/Paose/Workflow nets etc. The Agent-Activity integration approach may be oriented towards those contexts but it is still transferable to more general settings. Other researchers working with agents and workflows only need to adapt details and can use the results to integrate their models in similar ways. Even researchers working on more general contexts can apply certain principles and ideas to their work. In general, this thesis provided ideas, concepts and approaches towards integrating two different perspectives on software systems. By doing so, the strengths for each of the two perspectives could be combined and made available on equal levels of modelling.

Regarding perspectives, the following quote is relevant: "*Promote and encourage work that expands our knowledge of BPM beyond the control flow perspective. Research on BPM data is increasing, but also the resource perspective is promising. Also the temporal perspective could be more intensively studied in order to further integrate BPM with operations management research and statistics. Beyond that, other context perspectives have not yet been deeply analyzed, such as social [...] and location-based contexts [...]. Congruently, more research work should be considered that integrates these varied perspectives into comprehensive and*

encompassing theories and solutions. Finally, expanding the perspectives on BPM research may also entail broadening the definition of BPM and processes to encompass research in other fields on other types of processes, such as software process improvement [...], scientific workflows [...], organizational work routines [...] and others." (adapted) [Recker and Mendling, 2016, p. 65] The AGENT-ACTIVITY approach lays the groundwork for exactly this kind of integration of varied perspectives. Behaviour in this thesis strongly corresponds to the control flow this quote refers to. The structure and structural perspective that is integrated with the behaviour and control flow can be considered as a resource perspective. This quote showcases the potential relevance of the results of this thesis. By integrating different perspectives, they become more accessible and the capabilities of system modelling are increased.

While the previous quote is from the field of BPM, the other direction of agents is also quite relevant. Agents can and do benefit from a strong behavioural perspective. Consequently, the research presented in this thesis exhibits relevance in general contexts. Other researchers can use the results and lessons learnt here and apply them to their own work.

12.3 Synergies of Agents and Workflows

This thesis dealt with an integration of agents and workflows. The result, the AGENT-ACTIVITY integration approach, fully satisfies the specification and vision of an integration as defined in Chapter 7. One of the motivations behind pursuing an integration was to yield benefits and advantages from it. The synergies created by enabling the combination of agent and workflow mechanisms, properties and principles are numerous. Much of the previous discussions have dealt with these synergies and benefits in an implicit way. This section now explicitly discusses the most prominent synergies yielded by the integration. It serves as part of a summary of previous discussions, especially of the applications Chapter 11, part evaluation of the integration and part outlook at what is possible to achieve through the integration when used in the future. The discussions are structured around the AGENT-ACTIVITIES themselves and the four levels of the reference architecture (see Section 9.2).

Note that this section only discusses the most prominent ways in which the integration can be used advantageously. For each level of the reference architecture, these prominent synergies, which were discovered, observed and developed during the creation of the AGENT-ACTIVITY approach and the implementation of the PAFFIN-System, are presented as representative examples. From these characteristic examples, general conclusions about the synergies on each level are drawn.

Synergies within Agent-Activities Synergies modellers can exploit and utilise within AGENT-ACTIVITIES relate to specific patterns of agent actions and workflow operations used within the internal processes. Please note that all of the following synergies are fully supported in the PAFFIN-System and most have also been explicitly utilised in the application prototypes from Section 11.3.

Agent actions between workflow operations: Modelling workflow tasks with agent actions between the workitem request and activity confirm operations is a major synergetic pattern in the AGENT-ACTIVITY integration approach. Through this pattern it is possible to incorporate arbitrary functionality into the workflow instance/workflow

engine concurrently to the execution to a workflow task. The workflow therefore is both workflow (engine) and an agent at the same time.

Practical examples include calling external applications for management/monitoring purposes (see the RedTimer example from Section 11.3.3), receiving progress updates or preliminary/partial results from the resource or GUI, updating and providing dynamic parameters/data to the WFMS or resource, initiating exception handling behaviour when certain conditions are met (e.g. inform user or user supervisor when task execution experiences a timeout) and gathering/preparing data for future use (e.g. logging data based on resource parameters, comparative data for task result evaluation). Basically, the integrated entity can perform any kind of functionality while the task is being executed by the resource/user. This is especially useful, whenever the functionality being performed relates to the task or uses or processes data from/for the task.

Direct engine operations: Direct engine operations enable the entity as a workflow/workflow engine to autonomously decide to confirm or cancel a workflow activity. This synergetic effect results from the workflow being imbued with agent autonomy and intelligence. It is practical and beneficial in many settings, since it allows the workflow to decide when an activity ends. Through direct engine operations the workflow itself can, for example, evaluate the result quality and confirm a good result, cancel a bad result (and then retry the task), cancel an activity after a timeout or confirm an activity that is no longer needed. The PAOSE teaching support prototype from Section 11.3.3 utilised direct engine operations to support enduring activities.

Nesting workflow tasks: While subworkflows and nested tasks are a rather classical workflow topic, the synergy here lies in the fact that the entity as an agent, or rather the entity's backend, manages the nesting. Nesting tasks can be used to realise concurrent, subordinate behaviour within the scope of a task. The practical examples are similar to the ones given for agent actions between workflow operations, only that here the functionality is not necessarily executed by the same entity. The nested tasks describe units of work concurrent to the super-task that is being executed somewhere (else) in the system. In the pizza collaboration example from Section 11.3.2, for example, the entire subprocess of baking and delivering the pizza in the different entities of the pizza service organisation is initiated by a workflow task within the pizza customer.

Agent actions for task variables: Another prominent pattern is to use agent actions before a workflow task to provide the task parameters. Here, agent intelligence can be used to compute or determine dynamic parameters, instead of hard-coded ones. Agent actions can evaluate the dynamic, runtime properties and available AGENT-ACTIVITY parameters and change/apply the most fitting workflow parameters. Examples include gathering and evaluating system data to determine the priority of a workflow task, choosing specific exception handling tasks given the data about an error, formatting/transforming workflow parameters from incompatible data types or simply mapping given parameters to different tasks (e.g. payment type to specific payment task). The provision of dynamic permissions in the pizza collaboration from Section 11.3.2 is also an example of this pattern.

Agent actions for result processing: Post-task result processing represents another synergetic pattern. Here, the agent functionality is used to process the result of a workflow task. This can include storing the result persistently, evaluating the result, formatting/editing the result for further use etc. This pattern is useful whenever the raw result data from a task is not quite in the state that is needed for the next execution. Maybe just a field from the result object is required for a future branch or maybe the

result needs to be transformed into another object class. In any way, result processing is essential in some applications and captured in the integration on the same level as the workflow task itself. The application prototypes of Section 11.3 extensively used this pattern to process and extract results from the web GUI.

Decision component access at any time: Decision components are complex and completely internal behaviours of entities as agents. They represent the mechanism most suited for incorporating agent intelligence. Accessing them is considered as a special kind of internal action that can be used anywhere in an Agent-Activity. This enables access to the functionality realised in them for any workflow operation or other agent action. The PSC prototype from Section 11.3.1, for example, utilised DC access to simulate the storage component including maintaining the maximum storage capacity.

Workflow tasks for exception handling: Previous patterns have emphasised improving workflow tasks. However, workflow operations and the tasks they represent can be used to improve agent functionality just as well. One such pattern revolves around exception handling. When the (agent) functionality within an Agent-Activity encounters issues or problems during its execution, a workflow task may be added to realise a reaction to the present issues. The workflow tasks may be used to solve specific problems (e.g. data is not available so the tasks instructs another (entity) resource to provide that data), initiate automatic problem solving processes (e.g. the execution couldn't progress for a set amount of time so another (entity) resource is tasked with finding out why) or even inform human administrators (e.g. the execution has encountered a critical error that only the administrator can solve). This pattern could, for example, be used in a possible extension to the pizza collaboration from Section 11.3.2, in which the pizza clerk may inquire with the customer (user) if the chosen pizza type was unknown in the system. Incorporating exception handling in such a way provides an explicit, versatile and easily configurable tool for system modelling.

Workflow tasks for load balancing: Another synergetic pattern emphasising agents is to use workflow tasks for load balancing. If an entity acting as an agent recognises a performance bottleneck in its own execution it is possible to enable a workflow task, which starts the same functionality in another automatic resource. That way the original entity may outsource parts of its functionality to other entities which are not as used to capacity. This pattern represents a recursive nesting of functionality realised through the organisation into workflow tasks. While it favours agent functionality here, it is just as easily applicable to any functionality also containing workflow operations. As an example, the PSC scenario from Section 11.3.1 could be extended to have a store Paffin at full capacity utilise a store workflow task to store a product with another store that still has capacity. In fact, the system could be adapted to utilise the same store Agent-Activity the producer usually uses.

Workflow tasks for abstraction: The final concrete pattern to be considered here is to use workflow tasks as an abstraction for more complex (agent) functionality. Entities as automatic resources execute behaviour to perform tasks in the context of Agent-Activities. That behaviour can be any process-protocol containing any Agent-Activity. Therefore, the workflow tasks in an Agent-Activity can be easily used as an abstraction for more complex behaviour. That enables modelling a basic overview workflow and adding the details later, thus supporting the modelling process. Workflow tasks for agent behaviour are a core component of the WorkBroker principles (see Definition C.13). The pattern is used throughout the application prototypes. The pizza

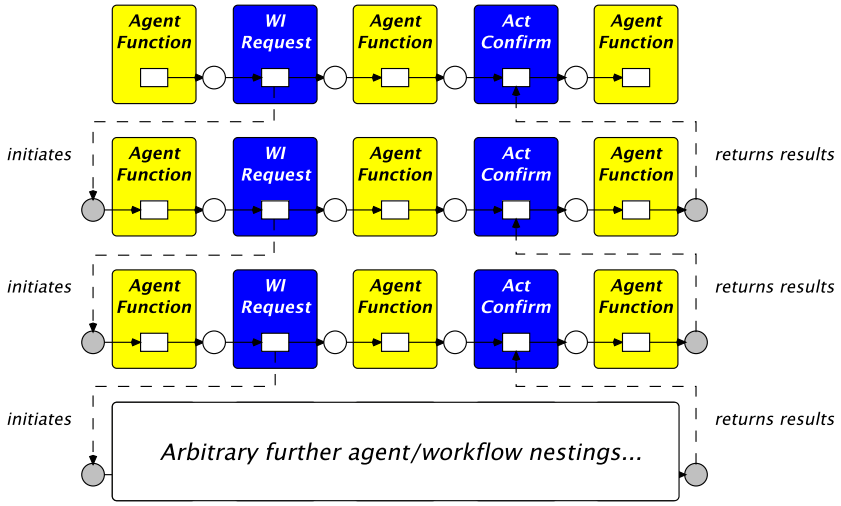

Figure 12.2: Illustration of synergies on the AGENT-ACTIVITY level

collaboration from Section 11.3.2, especially, heavily utilises the pattern to organise the work for the PAFFINS implementing the employees of the pizza service.

What can be seen in these examples of concrete synergies is that they all relate to the integrated entities ability to freely combine agent actions and workflow operations within an AGENT-ACTIVITY. Figure 12.2 illustrates this. Agent functions representing arbitrary, possibly empty, sets of agent actions can be put in front of, between and after workflow operations[3]. The workflow operations correspond to other behaviour that can also, as shown in the figure, contain more agent functions and further nested tasks.

Considered on an abstract level, the agent functionality provided through the agent actions provides an arbitrary software functionality. Sending and receiving messages realises the interaction with other components, while internal agent actions interact with internal components (e.g. knowledge base, process-protocol factory or decision components) or realise data processing. In the PAFFIN-System, an internal action corresponds to a reference net transition, which can be inscribed with *any* Java code. Workflow operations, on the other hand, provide a more organisational view on the functionality. They define that there is a piece of work, which has certain parameters, that has to be performed by some (other) resource. In other words, the agent actions describe what an integrated entity actually does, while the workflow operations describe organisational units of work for resources outside of the current AGENT-ACTIVITY. Combining this free-form functionality from agents and the organisational power of workflow tasks is the general core synergy of the integration on the level of AGENT-ACTIVITIES.

Synergies in process-protocols On the level of process-protocols the available synergies mostly relate to the execution order of AGENT-ACTIVITIES. Within process-protocols, AGENT-ACTIVITIES can be freely combined. In the PAFFIN-System AGENT-ACTIVITIES are implemented as special transitions and their possible combinations are defined by regular Petri net rules.

[3]For readability Figure 12.2 omits the cancel activity operation. Since it is not enforced the figure is still valid, it only contains less elements. A cancel operation can always be added at any point leading back to the place before the request workitem operation.

Individual Agent-Activities*:* The Agent-Activity is the core modelling construct within the integration. As an abstract task each individual Agent-Activity defines the state of the entity that is executing it for the duration of its execution. If an Agent-Activity contains only agent actions an entity executing it is an agent. If an Agent-Activity contains only workflow operations an entity executing it is a workflow. If an Agent-Activity contains both agent actions and workflow operations an entity executing it is a hybrid of agents and workflows. This is a fundamental synergy of agents and workflows within the integration. It enables most of the other synergies on other levels of the reference architecture. It is supported by the Paffin-System, with the limitation that a Paffin is always an agent to a small degree as it is always ready to receive agent messages even if these aren't processed in process-protocols.

Concurrent Agent-Activities*:* When Agent-Activities are being executed concurrently, this concurrency can define a hybrid state for the executing entity as well. If one Agent-Activity contains a workflow task, while another only agent actions, the concurrent execution creates a full hybrid entity that is both agent and workflow. Practically, this synergy can be exploited for use cases, in which the duality is useful. For example, one branch of a process-protocol might gather data from different source and then pass that data as parameters to workflow tasks in another branch. When both branches run continuously, a constant workflow being fed by dynamic parameters can be created. For long running applications this kind of dynamic can be useful. In general, the concurrent execution of Agent-Activities can be utilised whenever an entity could or needs to do different things at the same time. When these things relate agent and workflow behaviour, the current synergy arises. In the Paffin-System this synergy is fully supported.

Sequential Agent-Activities*:* Sequential Agent-Activities represent a rather minor synergy when considering viewing an entity as agent and workflow. A sequence of Agent-Activities allows for the entity switching between agent, workflow and hybrid states. This can be useful, for example, when an Agent-Activity containing a workflow task is followed by an Agent-Activity realising functionality working on the result of that task. The synergy here is the ability to dynamically switch between states. It is fully supported in the Paffin-System.

Overall, the synergies on the process-protocol level relate to the ability to incorporate Agent-Activities and therefore different entity states into an abstract workflow-like process. That process represents a common model for the behaviour of an integrated entity. Agent, workflow and hybrid aspects can be combined on an abstract level, with details being added within the individual Agent-Activities. The synergies on this level were showcased throughout the application prototypes in Section 11.3. The pizza collaboration in particular showcased Paffin entities switching between and concurrently utilising agent and workflow states.

Synergies in integrated Entities/Paffins On the level of integrated entities the synergies are concerned with what the entity can represent. It is clear that the integration basically allows an entity to be agent, workflow, both or something in between. How that state is realised was the topic of the synergies on the process-protocol level. Now, the focus lies on the amalgamation and conceptual utilisation of the state. In other words, the synergies here are related to what a single entity can be in an application.

Encapsulation of a process: An integrated entity can encapsulate a single process. When this is done, the capabilities of the entity apply to the process. The process can

incorporate agent functionality into its workflow tasks and utilise all of the synergies discussed on the AGENT-ACTIVITY and process-protocol levels. This means that the process can act autonomously and intelligently. It can react to outside events and messages, but also proactively initiate behaviour. If mobility is available it can move to another execution platform, if cognitive concepts are supported it can strive to achieve goals. All of this, and more, is possible through this synergy on this level of the reference architecture. In the PAFFIN-System, this synergy is supported to the degree of capabilities provided by the CAPA agent basis and workflow Petri nets.

Note that this synergy applies both to an entity encapsulating a process instance and all process instances of a process scheme. Conceptually, there is little difference regarding the quantity of instances. However, in some application the encapsulation of all instances of a schema into a single entity might allow for easier/improved monitoring.

A good example of this pattern can be found in the PAFFINS representing the worksheet exercises in the PAOSE support prototype from Section 11.3.3. Here, the PAFFINS clearly encapsulate the workflow modelled for the exercises. However, the prototypes also utilise agent functionality to call and control the external RedTimer tool and communicate with other system components like the GUI. This realises functionality additional to the pure workflow of the exercise.

An interesting effect of the encapsulation of a process is highlighted in the following. Usually, when modelling processes the process management and additional functionality is provided by the framework and execution engines. The process itself does not contain the functionality required to execute itself. An integrated entity, though, can do just that. It is feasible for an entity to provide the functionality (as an agent) that it itself requires for its tasks (as a process/workflow). For example, an entity might require a special encryption for the results in its tasks. Both the tasks and the encryption functionality could be easily incorporated into the integrated entity. Resources could then work on the tasks of the entity and subsequently encrypt the results with that entity before confirming the activity with valid, encrypted results. Through the encapsulation of both agent and workflow functionalities at the same time, the autonomy of the integrated entity as a process is significantly increased. It can provide necessary functions itself and does not have to depend on other (possibly remote or unsafe) entities.

Encapsulation of a Case: An integrated entity may also encapsulate a case. When a case only affects a single process there is no difference to the previous synergy. However, if the case affects multiple processes, a new synergy becomes available. An entity encapsulating all of the workflow and supplementary processes affected by a case represents a collection of a complex subject into a single modelling unit. Similarly to the individual process, this enables the case to exhibit the properties, mechanisms and concepts available to the entity. It also represents a modelling abstraction with which the modellers can easily work. Changes to the handling of a case are centralised and can be easily followed. The PAFFIN-System fully supports this synergy by endowing an entity with all of the connected process-protocols realising case handling.

Encapsulation of a (workflow) actor: Similarly to encapsulating a case it is also possible for an entity to encapsulate a workflow actor or workflow role. Note that this does not refer to an entity as being a workflow resource, which as a synergy is discussed next. Rather it means that all of the processes for a single actor or role are gathered into a single entity. This combines these processes not by relating them to a single case, but rather to a single actor or role. The synergetic effect here is the same as with the

case, though. Still, it provides modellers with the ability to better delimit a single actor within the larger scope of the application. Use cases for this synergy can relate to configuration issues (e.g. all processes for an individual actor should be presented in the preferred language of the actor) or to more thematic issues (e.g. gathering the critical processes for a supervisor actor). As an example, the PAFFINS in the pizza collaboration prototype from Section 11.3.2 each encapsulated one of the actors of the scenario. Overall, this synergy is fully supported by the PAFFIN-System.

Encapsulation of a resource: The previous synergy represented a gathering of different processes related to a single actor. However, it is also possible for an entity to encapsulate and represent an actual resource of a workflow. How this can be done has been extensively discussed in the description of the PAFFIN-System prototype, as the backend resource component (see Section 10.2.11) is responsible for exactly this functionality. The benefits are the implementation of entities as automatic resources and of user entities providing a GUI to human users. As an example, the pizza service employee PAFFINS from the pizza collaboration prototype in Section 11.3.2 all also encapsulate and implement the (automatic) resource they represent. The PAFFIN-System fully supports this synergy.

Encapsulation of an agent: An entity can also encapsulate an agent. The synergy here lies in the fact, that the agent encapsulated by the entity can also access the workflow functionality. It can act as a workflow engine for tasks it wishes to dispense and it can act as a resource for the tasks of other entities. The latter, especially, realises a very loose coupling of work organisation in tasks and actual agent functionality. In other words, the integration makes it possible for agents to be controlled through workflows. Workflows, as an established tool for managing, controlling and supporting the work of heterogeneous resources can now be fully utilised for agents. This basically means that the principles and research results of large parts of the field of BPM can conceptually be applied to agents now, which is an important synergy in of itself. Overall this synergy is fully supported in the PAFFIN-System, especially in the incorporation of the WORKBROKER principle (see Definition C.13). As an example, the producer and consumer PAFFINS from the hybrid PSC application prototype from Section 11.3.1 are considered as agents that possess store and retrieve tasks for the store PAFFINS.

Encapsulation of a multi-agent subsystem: Another option for encapsulation are full multi-agent system subsystems. This can be regarded in two different ways. The first option is for the entity to represent a proxy or interface to the system. In the PAFFIN-System this would correspond to the planned subordinate resources mechanisms. An entity would request workitems on behalf of its system and then distribute the activities internally. The second option is for the entity to incorporate the behaviours of multiple agents. This is akin to CAPA were multiple agents roles can be associated into a single agent. In either case, the synergy again lies in the ability to access workflow mechanisms and concepts. For the second option, the effects are identical to the singular agent. For the first option, those effects are also valid, but through the distribution of activities via the proxy entity the entire execution becomes more loosely coupled. Overall, encapsulating multiple agents within an entity provides an abstraction that modellers and monitors can use to support or simplify modelling by concentrating functionality into more compressed modelling constructs. For the PAFFIN-System the second option is fully supported, while the first one is envisioned and prepared in the limited subordinate resources mechanism.

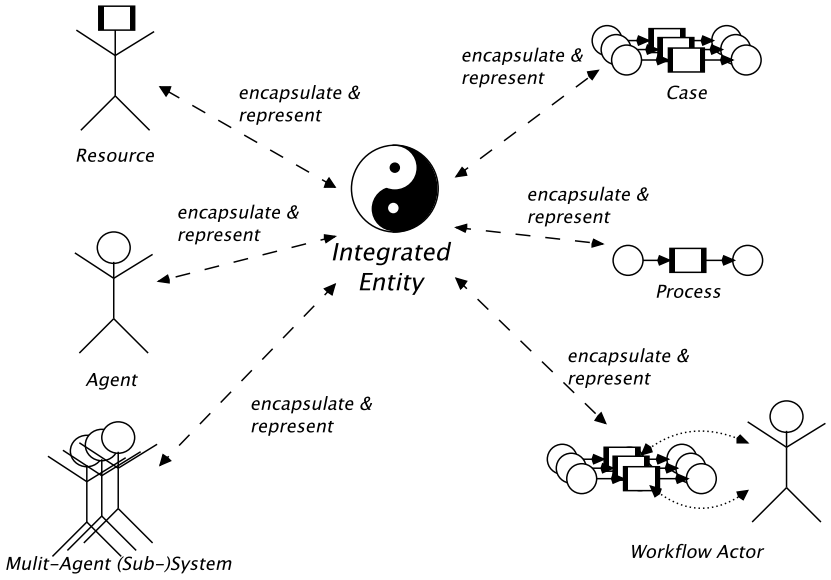

Figure 12.3: Illustration of synergies on the entity and Paffin level

From these points it is clear, that the synergy on this level concerns encapsulation. An integrated entity can encapsulate any concept, component and even sets of components of a workflow and agent system. Figure 12.3 illustrates this. Clearly, the encapsulation options of the integrated entities are very versatile. The encapsulation can be utilised in any number of ways for an application, but the most prominent use is to encapsulate based on a relation (e.g. process, actor or resource) and then utilise mechanisms and concepts from agents and workflow to a certain benefit. In other words, the encapsulation allows for a process or any component of a process to be an agent and for an agent or sets of agents to be processes.

It should be noted that not all encapsulation options are necessarily advantageous for all systems and applications. However, for any feasible encapsulation variant application scenarios can be constructed in which that encapsulation is useful. In the context of gaming applications, for example, many otherwise superfluous encapsulation options can be reasonably utilised. The unrestricted ability of integrated entities to generally encapsulate arbitrary concepts therefore remains a clearly beneficial synergy. Nonetheless, future work may provide more restricted best practices for encapsulation options in specific application areas.

Synergies in platform and management For the platform and management level of the reference architecture the synergies lie mostly in how entities in different states can interact with one another and in their unified management.

Processes interacting with other processes: A major synergy on this level of the reference architecture is that (entities as) processes can interact with other (entities as) processes. Interacting processes can be applied usefully in many ways. For example, a process may ask another process for missing data, a process may negotiate with another process to solve a performance bottleneck (e.g. the second process occupies too many resources),

a process may provide its own results as parameters to another process, a process may outsource a case to another process, etc. The proposed extension to the PAOSE teaching support prototype from Section 11.3.3, in which the exercise PAFFIN as a process would contact a supervisor or tutor, which would also be processes, for help, is another example of this synergy. These kinds of interactions in which one process communicates with another process to achieve some kind of objective are all available in the AGENT-ACTIVITY integration approach. Modellers simply need to incorporate the required interactions (as agent actions) in relation to the workflow tasks into the process-protocols (see synergies on the level of AGENT-ACTIVITIES). On the other side, it is also possible for processes to become resources in tasks of other processes. This opens up even more possibilities of nested processes. The fact that these incorporations are available in a single model and with the same modelling tools (in the case of the PAFFIN-System reference Petri nets) is a synergy in of itself. Yet, that processes are now able to turn into active components (agents) is the main synergy at this point and it is fully supported by the PAFFIN-System.

Processes interacting with actors: Another synergy lies in the ability of processes to interact directly with other actors/components of the system. This does not refer to actors/components executing tasks of the process (i.e. the regular workflow interaction), but rather that the process directly interacts with the actors/components. For example, a process may gather data for its tasks from a persistent data storage, a process may provide results to a system monitoring entity, a process may receive instructions/data influencing its future execution, a process may initiate functionality in a subsystem relating to results obtained by the process, etc. The customer process-protocol asking for the pizza with the pizza clerk actor in the pizza collaboration in Section 11.3.2 is another example of this synergy. In other words, the process is able to react to incoming data/events from the system and also to create events that cause further functionality not directly tied to the process. As with the previous synergy, a major strengths for modelling here is the unified model within the AGENT-ACTIVITIES. However, the main synergy remains that, through the incorporation of agent functionality, processes become active parts of a system that don't have to rely "only" on workflow tasks to affect system execution. The PAFFIN-System fully supports this synergy.

Actors interacting with other actors: This synergy relates to supporting the interaction of actors of a common workflow process. Usually, the resources of a workflow are only related via the control and data flow in that process. Here, though, it is possible to create a direct connection using agent actions as well. This can be useful especially in collaborative scenarios. For example, a user may ask another user for help in a task the second users has more experience in. Or an entity may retrieve results of a previous process instance from the entity that worked on it before. Generally, having workflow actors and resources interact directly, as opposed to the indirect interaction through the workflow process itself, is a helpful synergy whenever the indirection is insufficient in transferring data. It is also helpful when more complex behaviour is required between the actors, e.g. a handover in responsibilities. Additionally, the fact that an actor can simply define a task for another actor and essentially become a process is another aspect of this synergy. Here, the organisation of work mentioned for the synergies on the AGENT-ACTIVITY level is relevant again. If it is necessary or helpful an actor can freely turn into a process. Examples include outsourcing of (parts of the) workload, simple data/information inquiries or initiating complex behaviour to rectify issues in the environment. Overall, this synergy is fully supported by the PAFFIN-System.

Actors interacting with processes: The final synergy on this level relates to actors interacting directly with the processes. As with the reverse synergy, this synergy does not relate to regular workflow-task interaction. Again, it relates to actors and other system components being able to directly interact with the processes as a whole. Use case examples range from simple management issues (e.g. start/stop/suspend/resume process instances) over data exchange (e.g. provide parameters to processes and retrieve partial/intermediate/full results) to more complex interactions. Such complex interactions may, for example, include actors/components initiating changes in the processes (i.e. adaptive processes) or negotiating task priorities. Generally, the synergy of a unified way of interactions allows actors and components to communicate and influence processes in a way that is not possible without the integration. This synergy is supported by the Paffin-System, although the more complex examples of adaptive processes are considered as future work.

Overall, the synergies on this level of the reference architecture all originate from the unification of agents and workflows into the single integrated (Paffin) entity construct. Actors, components and processes all have access to a unified interaction mechanism in agent interactions. This is in addition to the interaction via workflow tasks, which is also unified into the modelling. Through these unifications it becomes possible to disregard the state of an entity. This, in turn, enables modellers to freely choose the best way of interactions. If a process needs data at runtime, simply use agent interactions to get that data. If a component needs to prioritise the tasks of a process, simply have it negotiate with the process. All of this is made possible through the unification and integration of agent and workflow interactions on this level relating to inter-entity interactions. For illustration, consider Figure 12.3. Each of the concepts that can be represented by an integrated entity can utilise the synergy on the platform and management level to interact with one another. This creates an extensive and complete web of communication relations between each possible pair of encapsulations.

Synergies in the overall system The final level of the reference architecture, the system level, inherently deals with global perspectives on the system. As it is an implicit generation from the lower levels, the abstract views originating from the lower levels are the main causes of synergies here.

Workflow state to describe global behaviour: Every entity in a system can be a workflow at any time of its execution. In general it always has some form of behaviour defined through its Agent-Activities. Through the organisation of entity behaviour into Agent-Activities and therefore into abstract workflows, it is possible to create a global view on the behaviour of a system. This global view represents a synergy that can be utilised to represent, illustrate and monitor the system. Each Agent-Activity represents a task in that global workflow, with complex relations to other Agent-Activities determined by the agent actions and workflow operations within the Agent-Activities. These tasks can be filtered to create global agent, workflow and hybrid views on the system. For the Paffin-System, the potential to generate the global workflow perspective is available. However, there is no implementation yet.

Agent state to describe global structure: Every entity, even those that only contain workflow tasks, is a component of the system. In the Paffin-System it is also a latent agent due to its capability to technically receive agent actions, yet for the conceptual approach each entity is an abstract agent and therefore an element of the structure.

Consequently, it is possible to create a global overview of the structure at any time. Each entity, regardless of what it represents (agent, actor, resource, database, process, hybrid, etc.) is part of this global perspective. As with the workflow behaviour view, the structural perspective can be used for representation, illustration and monitoring purposes. Filtering according to entity state then allows for helpful restricted views when examining specific aspects of the system. As with the global behaviour view, the groundwork is available in the Paffin-System, yet no perspective generator exists yet.

Overall, the synergy on this level lies in the globally unified view on structural and behavioural components of a system as integrated entities. Through their dynamic state these entities may emphasise either structure or behaviour at any moment. Yet still, they are always in some way part of both structure and behaviour, as it should be. Through the generation of these perspectives and the reasonable filtering for specific purposes, system modellers and monitors can gain greater insight. Two global unified perspective, inherently related by the fact that each entity is part of both structure and behaviour, are a powerful synergy and tool and should be a focus of future work.

Synergies Conclusion The previous paragraphs showed that each level of the reference architecture features beneficial synergies and synergetic patterns of some kind gained through the integration of agents and workflows via Agent-Activities. In relation, these synergies are often strongly connected over various levels of the architecture. In other words, the synergies on the lower levels often support and enable the more conceptual/abstract synergies on the higher levels. For example, when a process is encapsulated by an entity it gains access to the agent mechanisms. This happens on the entity and Paffin level. However, on the platform and management level the synergy relates to processes interacting with one another. This is only possible through the previously mentioned encapsulation.

Overall, the synergies described here are the concrete effects of having an integrated entity be able to be agent, workflow, both or something in between. All of these synergies are available at all times and can be freely combined. This may be considered as the core synergy above all of the other ones: Integrated entities are completely dynamic in their state and can therefore freely and actively choose, combine and integrate the most suitable properties, mechanisms and tools from agents, workflows and hybrids.

In conclusion, the integration via Agent-Activities allows integrated entities in this approach to be more than the sum of their parts. The synergies not only enable access to all mechanisms from the individual concepts, but also create beneficial and novel mechanisms not available without the integration.

12.4 Strengths and Open Points

This section contains a general discussion of the overall strengths of the Agent-Activity integration approach and Paffin-System prototype. These strengths have been the implicit topic of discussion in much of the previous sections and chapters. Afterwards, open points of the approach and implementation are discussed and evaluated regarding their severity. Consequently, this section serves on the one hand to summarise some of the previous detailed descriptions and on the other hand also to evaluate the results of this thesis in their entirety.

Strengths The following describes the most prominent and valuable strengths of the Agent-Activity integration approach and Paffin-System prototype. The order of the

strengths aligns to the later ones being more exclusively related to the PAFFIN-System, while the earlier ones apply to both the concept and the implementation.

Synergies: Probably the most significant and major strength of the AGENT-ACTIVITY integration approach is the realisation of the synergies discussed in Section 12.3. These synergies allow for a large number of beneficial effects and mechanisms, as was discussed extensively in that previous Section. The PAFFIN-System framework prototype already implements a large degree of these synergies, making them practically available for application modelling and implementation. As previously discussed, many of the synergies were utilised and showcased in the application prototypes presented in Chapter 11.

Agents as workflows: The encapsulation[4] of workflows (definitions, instances or other workflow components like resources or cases) in the agent-like hulls provided by entities is another major strength. It allows for agent properties, mechanisms and functionality to be transferred to and available in workflows. This enables the modelling of, for example, intelligent, communicating and/or autonomous workflows.

Workflows as agents[4]: The reverse of the previous strengths, the encapsulation of agents in workflows is another strength of the AGENT-ACTIVITY approach. It represents the ability to model an entire agent as the sum of its (workflow-)processes, thus providing a behaviour-oriented view of a structural component. In practice, this allows for the utilisation of the task concept to create atomic agent behaviour, organise (and decouple) the individual steps of the behaviour and generally focus more on what an agent does than on what it represents in the structure of the system. Additionally, it allows for the consideration and, in future work, validation and verification of the agent in its entirety using established workflow principles.

Agents for workflows[4]: Another major strength stemming from the synergies is that agents are also fully available to work with workflows. This goes beyond the encapsulation and towards the interaction between workflows and agents. Beneficial effects include, for example, agents as workflow resources or representing (human) resources, agents for management aspects, agents for data handling etc. In fact, anything that agents can do for workflows (see discussion in Section 3.3.1 based on [Delias et al., 2011]) is covered by this strength combined with the strength about agents as workflows. This is discussed in more detail in the related work discussion in Section 13.3.1.

Workflows for agents[4]: Workflows are fully available to agents. This is the final major strength related to the overall synergies. This is exemplified in the WORKBROKER principle (see Definition C.13). Agents can utilise workflows to organise their behaviour. Instead of hard-coding interaction patterns into the behaviour, these can (but do not have to) be handled as standardised task patterns. This decouples the work organisation (within the workflows) from the actually implemented functionality, allowing for different implementations as long as the generic task interface is satisfied. Other benefits of this strength include enabling task atomicity, possible soundness verification and incorporating user interactions in a unified way.

Expressiveness: The expressiveness of the AGENT-ACTIVITY approach is another major strength. Modelling with AGENT-ACTIVITIES encompasses both agent and workflow

[4] This strength is actually a specific synergy discussed in the Section 12.3. However, due to its especially high significance within the overall results of this thesis it is explicitly extracted and discussed on an abstract level again.

modelling. In addition to that it also enables hybrid modelling, which again relates to the synergies discussed before.

Versatility: The Agent-Activity approach is not only expressive, it is also quite versatile. As shown in the example of the simple producer-store-consumer scenario described in Section 11.3.1, the Agent-Activity approach (and Paffin-System) provides various ways of addressing the same problem. It is possible to approach a problem in an agent-oriented way, a workflow-oriented way or a novel hybrid way.

Suitability: The versatility of the Agent-Activity approach allows system modellers to dynamically choose the best available tool/mechanism from agent-orientation, workflow management or hybrids of the two for any problem. This ensures that the most suitable tool is always available for system modelling. This strength was highlighted in the pizza collaboration application prototype in Section 11.3.2, as well as the hybrid PSC prototype from Section 11.3.1. In these prototypes, each requirement of the scenario was implemented with the most fitting tool from agents, workflows or hybrids.

Handling of complex problems: Combinations and integrations of agents and workflows in applications can require highly complex solutions. The best-fitting solution to an application problem might require switching between agent and workflow perspectives and mechanisms multiple times, intricately interweaving aspects from both concepts. Such problems require powerful modelling mechanisms able to handle the involved complexity and intricacies. Integrated entities have shown, through the application examples presented in this thesis that they can indeed deal with these highly complex problems. The RedTimer-enhanced task from Figure 11.36 in Section 11.3.3, for example, dynamically switches between agent and workflow mechanisms to implement a workflow task for users, which also controls external software and asynchronously communicates with other components. This complex solution is available in a single and well-arranged model in the Paffin-System.

Simplicity of scope: Even though they are expressive and powerful, the abstractions provided by the integrated entities and the Agent-Activities realise a significant simplicity of scope. This means that an entity or an Agent-Activity is still a clearly distinguished component of the system, even though it can realise highly complex mechanisms incorporating arbitrary agent, workflow and hybrid concepts. Both the integrated entities and Agent-Activities encapsulate parts and aspects of a system clearly. They represent abstractions of complexities that can always be set into relation with the unified other components (entities and Agent-Activities), which also abstract from more complex implementation details. In other words, the concepts and constructs of the Agent-Activity integration approach can encapsulate very complex subject matters while maintaining a simple, abstract scope for relation with other components.

Alignment with Paose*:* The Agent-Activity approach is intricately aligned with the Paose approach. Both emphasise structure, behaviour and their orthogonality in similar ways. This alignment represents a strength for Agent-Activities, as Paose can be almost directly applied as a software development methodology. With some adaptions (discussed in Section 12.6.3) it can be used to even more potential.

However, this strength also constitutes an opportunity for the Paose approach. Paose has a traditional focus on agents. However, the acronym is often changed slightly including, among others P for processes or O for organisations. Its emphasis on the duality of structure and behaviour, i.e. what creates the alignment with the

Agent-Activity approach, is, in these cases, often not realised strongly enough with the traditional agent mechanisms. This can be alleviated by incorporating Agent-Activities and the Paffin-System.

Support of structural and behavioural duality: Mentioned in the previous strength description, the support offered by Agent-Activities to model entities with a strongly dual nature of structural and behavioural aspects is a strength in itself. Integrated entities using Agent-Activities can act as both actors (as agents) and processes (as workflows). When modelling with traditional concepts, only one of the two is directly possible. How this strength can be utilised was showcased in the pizza collaboration prototype from Section 11.3.2 and relates to general inter-organisational contexts (see Section 11.5) as well. Inter-organisational contexts in particular feature organisations that are required to be considered as both actors and processes. Through Agent-Activities and integrated entities it is now possible to also model this duality.

Scalability: Agent-Activities are intended to support large-scale and distributed systems. The concept of the Agent-Activity is in itself highly scalable, even if the technical performance and vertical scalability of the Paffin-System remain open points due to Renew issues (see below). Through, for example, the encapsulation options, the loose coupling of work organisation and the general autonomy of integrated entities the *horizontal* scalability of the Agent-Activity approach is excellent. Functionality can be easily distributed amongst other integrated entities on different platforms (and therefore different computation nodes), while the asynchronous communication between entities ensures a high degree of transparency.

Generality: This thesis heavily emphasised a Petri net basis. For practical purposes it also explicitly focussed on Mulan/Capa agents and workflow Petri nets. However, as discussed in Section 12.2, the Agent-Activity approach is applicable and transferable to general settings beyond what was emphasised here. Its establishment in this thesis required a fixed focus in order to clearly work out the approach, details and a proof-of-concept. Still, Agent-Activities were always intended to be more generally applicable. This has been achieved. Overall, this constitutes a distinct strength of the Agent-Activity approach as it can be used in various fields and settings and benefit different researchers.

Conceptual extendability: The Agent-Activity approach is also extendable on the conceptual level. While this thesis motivated and emphasised specific sets of fundamental agent actions and basic workflow operations as the core building blocks of an Agent-Activity, these sets can be easily altered. The sets of actions and operations can be extended or changed[5] depending on the specific needs of modellers. While it was argued, for example in Section 12.2, that cognitive concepts could also be captured by (internal) agent actions, their explicit incorporation as special agent actions represents a prime example of a possible extension of the Agent-Activity concept and approach.

Uniformity: The inherent uniformity of structural and behavioural elements of a system, provided by the integrated entities, represents another strength of the approach. All agent and workflow aspects, mechanisms and properties are available in a unified model and are accessible for modelling in the same toolset and technique. Generally, this allows for some of the synergies between agent and workflow aspects discussed before to be utilised. However, it also enables the generation of global views on structure and behaviour. While not emphasised in this current thesis, any generation of such global

[5] Changing the set would have to ensure that every scenario can still be captured by Agent-Activities.

perspectives in the future is only possible through the uniformity established by the Agent-Activity approach.

Perspective coverage: Agent-Activities and integrated entities provide means to create specialised structural and behavioural perspectives on a system. As discussed in the context of the PSC application prototypes from Section 11.3.1, these perspectives provide better views on behaviour than available for traditional agents and better views on structure than available for traditional workflows. Agent-Activities form an abstract workflow for each integrated entity and therefore a complete view of the behaviour of a system, while the entities themselves prescribe a clear and distinguishable organisation and architecture for the structure of a system. Both perspectives provide useful insights into the system and are both fully available for each element of the system, regardless of its considered role. This also strongly relates to the synergies on the overall system level as described in Section 12.3.

The integration of structural and behavioural components also enables Agent-Activities to explicitly model extended application perspectives. For example, the resource perspective for workflows [Russell et al., 2005, He et al., 2014, Klinik et al., 2017] can be explicitly modelled in the unified model provided by integrated entities as workflow resources. The same is true for perspectives emphasising scheduling issues, data flow or negotiation. All structural and behavioural aspects are available for modelling, supported by different agent, workflow or hybrid mechanisms that are continuously available.

Socially-oriented workflows: Human users can be easily included into the Agent-Activity approach. The equation of human users and the Paffin entities representing them is, in fact, a common mechanism in the application prototypes from Section 11.3. Through this kind of inclusion of human users into the workflow/agent concepts, social workflows are very well supported. Social workflows support human users in their communication, interaction, collaboration etc. The ability to directly map the actors/users (agents) to the workflows and capture the interactions directly within the workflows is valuable in this context.

Conceptual flexibility and variability: Agent-Activities provide an abstraction of the work of an integrated entity (an abstract task). Within the process-protocols, only the interface to that abstract task is defined, i.e. the incoming parameters and outgoing results. This allows for a variable exchange of the specific Agent-Activity at runtime. As long as the interface is satisfied, the internal process is irrelevant to the process-protocol. This allows for easy and clearly recognisable modelling of variants and options within the same basic process-protocol.

For example, a process-protocol could model a payment process for a customer. Instead of explicitly modelling each payment option as a branch of the payment process, a single Agent-Activity suffices that instantiates the correct payment Agent-Activity (e.g. credit card, cash) using dynamic variables at runtime. This kind of variable behaviour can also be applied on the level of workflow tasks within Agent-Activities. In general, the variability of Agent-Activity instantiation provides a convenient and effective tool for realising flexibility in a system and is therefore a clear strength.

Going even further, it is feasible to utilise the variables of Agent-Activities to create fully generic process-protocols. In this case, the results of one Agent-Activity are used as to determine the next Agent-Activity. The process-protocol containing these Agent-Activities would feature only variable inscriptions. This functionality is, in

concept, already fully supported in the Paffin-System, but not yet tested. Therefore, it is picked up again and elaborated on as a future work direction in Section 12.6.1.

Reusability: Strongly related to the previous strength regarding flexibility is reusability. Agent-Activities can be filled with any internal process satisfying the relatively simple interface defined for it. This interface is identical for any application built with the Paffin-System framework. Therefore, any internal process of an Agent-Activity can be easily deployed in any application. If a common functionality, e.g. a regular workflow task or a simple agent message exchange, is required in multiple applications it only needs to be implemented once in a generic way with parameters covering the specific details and requirements.

Petri Nets Basis: The Petri net basis of the Agent-Activity and especially the Paffin-System provide a number of advantages and can consequently be considered as a strength. While not yet exploited in that way, the formal basis can enable verification and validation in future work, which is discussed in Section 12.6.2. Additionally, the graphical nature of Petri net models for software implementation cannot be discounted. The graphical Petri nets feature clear arrangements of control and data flows and are easy to model and understand. Processes featuring complex loops or other concurrency can be easily created and recognised. This way of modelling is different than implementing with more textual languages, yet when the basic concepts have been understood, modelling usability offers distinct benefits.

Compatibility: This strength relates to the Paffin-System. The design decision to fully enable Capa legacy code (see discussion in Section 10.2.5) represents a clear practical strength of the prototype. It enables full access to the extensive library of Mulan and Capa plugins, with the WebGateway and the OgeeJS-based GUI framework being two prominent examples that have already been utilised to great effect in the current Paffin-System prototype.

Technical extendability: The Paffin-System prototype is highly extendable. Through the use of Java as the inscription language of reference nets, any Java code or application can be incorporated into the Paffin-System. Even arbitrary external applications can be incorporated into the Paffin-System if they can be accessed via a Java mechanism, as exemplified by the incorporation of the RedTimer tool as discussed in Section 11.3.3.

Modularity: The main technical components of the Paffin-System prototype framework are implemented as reference nets included in Renew plugins. Both the nets and the plugins feature clearly defined interfaces. As long as these interfaces are satisfied, the implementation can be easily exchanged for new versions or variants. For example, exchanging the current GUI of the Paffin-System provided by the PaffinWebGUI plugin is easy. The ontology objects exchanged with the GUI are clearly defined in a generic way in the Paffin-System. As long as the interface is satisfied any other specialised GUI can be embedded into the system.

Open Points Even though the initial research and work on the Agent-Activity approach and Paffin-System have been completed with this thesis, there are still some open points left. These points mostly relate to challenges that should be addressed in future work and successor theses, as well as technical issues that need to be engaged in the wider context of Renew and Paose.

Note that the development and discussion of the Agent-Activity integration approach and the Paffin-System are strongly influenced by the particularities of the chosen modelling

and implementation basis, reference Petri nets. This causes a view biased towards Petri net principles and issues, which is applied to the general integration achieved with Agent-Activities and the Paffin-System. However, any implementation basis or programming language features a bias toward its own principles and issues. Furthermore, Section 12.2 already discussed the applicability and transferability of the results in a wider context. Therefore, this does not represent an open point but still needed to be addressed here.

Modelling complexity in general: Agent and workflow modelling by themselves are inherently complex ways of modelling software systems. They are capable of creating extensive, intricate and multifaceted applications for a large variety of real-world settings. The Agent-Activity integration approach features both of these ways of modelling. However, the integration itself also features new ways of modelling systems with elaborate combinations and synergies of the individual agent and workflow concepts. This increases the amount of tools, mechanisms and patterns available for system modelling and also incorporates some mechanisms and patterns that are highly complex themself. Therefore, the inherent modelling complexity is increased in the integration when compared to the traditional agent and workflow concepts. Modellers need to be able to handle this kind of complexity in their applications. Utilising the mechanisms and patterns is also more difficult, as benefiting from them requires an understanding of each individual aspect and element (see next open point).

Still, this increased complexity also enables many of the synergies and strengths described before. The complex interconnection and relation between structural (agent) and behavioural (workflow) elements within an integrated system and application directly and easily yields benefits that are hard, cumbersome or even impossible to achieve with traditional modelling techniques. Consequently, the increased modelling complexity is difficult to reduce or eliminate. It is something for which suitable support mechanisms and tools need to be provided. Additional research increasing the understanding of the integration can also help here. For example, net components for beneficial and common integration patterns, as well as clear instructions for best practices, may be utilised.

All in all, the increased modelling complexity can be considered as the cost of the integration. Through the integration, modellers gain the strengths and synergies previously described. However, they have to deal with more complex and, in parts, more difficult to understand models in exchange.

Requirement of expert knowledge: As mentioned in the strengths, the Agent-Activity approach is expressive, powerful, versatile and complex. Unsurprisingly, this has the consequence of requiring detailed knowledge about how the concept and practical system work before a modeller can start creating applications. This strongly relates to the previous open point about increased modelling complexity, but emphasis distinct requirements to the modellers. The required knowledge encompasses, for example, an understanding of the relations between the different levels of the reference architecture, the relations between the roles entities (i.e. agent, workflow resource, workflow engine) can inhabit and In general, a clear understanding of the conceptual integration and approach, as well as the elements and relations within them, is essential for modellers. Without this expert knowledge it is exceedingly difficult to productively model any application.

This open point is already alleviated to some degree by the Petri net basis, which provides good modelling usability. Furthermore, the abstractions provided by the simplicity of scope of the Agent-Activity allow for complex scenarios to be represented

in a simpler, more abstract way. However, future work should emphasise support for modellers. Adapting PAOSE (see Section 12.6.3) for PAFFINS and providing a sophisticated support system for it (see related application vision in Section 11.4.5) are some of the concrete points of future work related to this current open point. Beyond PAOSE, supporting modelling with AGENT-ACTIVITIES can be approached in various ways. The possibilities and opportunities span from, for example, optimising net components, to automating certain modelling steps (e.g. automatically create skeleton database code), to providing automated modelling guidance through intelligent workflows implemented by PAFFINS.

Complexity of possibilities: A specialised, yet highly relevant and therefore explicitly considered, aspect of the increased modelling complexity relates to the amount of available possibilities to model a specific scenario in an integration provided by AGENT-ACTIVITIES. Note that the implementation possibilities provided by the AGENT-ACTIVITY and the PAFFIN-System are, in general, a strength due to their extensiveness and versatility. However, the question is, if there are various ways of modelling a specific scenario, which one is the best? In order to adequately answer this question, future work should deal with best practices regarding when to choose agent, workflow or hybrid mechanisms. This thesis already provides a starting point for these best practices in the beneficial synergies discussed in Section 12.3. However, more applications and implemented use cases with the PAFFIN-System are required to fully evaluate these synergies, and more, as best practices.

Increased overhead: An open point, situated between conceptual and practical considerations, is the increased overhead regarding management issues within any system implementing the AGENT-ACTIVITY approach. As both agent and workflow aspects need to be managed and coordinated, there is an increased need for integrated entities to communicate with each other and also with the global workflow management facilities. This can have a negative effect on the performance of the system, as well as the clarity of entity interactions. Future work could consider novel management mechanisms that decrease the amount, complexity and size of the exchanged messages. Feasible solutions include minimising the involvement of global mechanisms, especially concerning workflow management, or reducing the update cycles for workitem and activity lists. On the conceptual level, it might even be feasible to incorporate holonic ideas that could abstract from certain subsystems. This could impact the reference architecture somewhat, introducing another level orthogonal to the different existing levels.

Technical maturity: A significant open point is the technical maturity of the PAFFIN-System prototype. As discussed in Section 11.2, it already exceeds its intended status as a proof-of-concept, yet still exhibits a number of limitations. Addressing these limitations is one part of the current open point. Future work, though, plans to address the limitations with a high priority. It should, however, be stressed again that the limitations all represent "bonus" features that go beyond the prototype and proof-of-concept scope that was intended for this thesis. Also, the deployment and test of the PAOSE teaching support prototypes, described in Section 11.3.3, have shown that the system is already technically mature and robust enough to endure outside of the development environment and be used by users with no prior experience.

Another issue of the technical maturity relates to wider features that are not exclusive to the PAFFIN-System. These include, for example, the incorporation of mobility into CAPA agents, runtime reference net adaptivity for RENEW or verification tools for

reference nets. These features are highly desired for an implementation of the Agent-Activity approach. Their provision and implementation however is completely outside of the scope of this thesis and rather relates to the wider context of Mulan, Capa, Paose and Renew. These features are being researched by the TGI group and due to the extendable design of the Paffin-System it is possible to add them at any later time.

Modelling tool support: Related to the open point about requirement of expert knowledge, the Paffin-System also has a concrete open point about the limited modelling tool support. Currently, the tools available for the Paffin-System are mostly repurposed Capa tools, like the MulanViewer and MulanSniffer. Only the Paffin net components and the Paffin logging mechanisms are tools specifically implemented for the integration. As future work, the repurposed tools should be replaced with or extended into tools specialised for an integration of agents and workflows. Examples of such possible, future tools are discussed in Section 12.6, especially in the context of the overall adaption of Paose.

Performance: A currently open point for the Paffin-System relates to performance. However, the performance issues are mostly related to the Renew basis. Only the open point regarding the increased overhead can potentially be addressed by the Agent-Activity approach. For the rest, optimisations in Renew and, to a lesser degree, the net architecture of the Paffin-System[6] are necessary. Improving the Renew simulator, deactivating unnecessary plugins or optimising deployment parameters are examples of options for the former. Splitting the large Paffin backend nets into smaller nets with better performance or simplifying the design of the platform WFMS are examples of options for the latter. However, these performance options need to be thoroughly tested and developed in order to ensure compatibility and the same expressiveness/feature set. Overall, the current Paffin-System requires a lot of computing resources, yet in no situations experienced by the author during its creation has performance been a practical problem. For future use it is advisable to invest effort into improving the performance in order to increase the vertical scalability.

Issues of Renew: On a similar note, other issues of Renew present an open point for the Paffin-System. Deployability, configuration and access to Renew could be optimised to improve working with the Paffin-System. For example, currently Renew is usually started using an extensive command line script, after building the project with a similarly extensive build process. Changes and adaptions to both building, configuring and starting Renew and its plugins, like the Paffin-System, are therefore not as easy as would be desired. This open point, though, needs to be addressed in the overall Renew context. Improvements here can conceivably be directly applied to the Paffin-System without further changes.

Security: Security topics were explicitly excluded from this thesis. However, for practical use of the Paffin-System future work is required to deal with security issues. This includes, for example, security of communication between entities, securing user details and passwords in the database and generally protecting the running system from unauthorised or malicious access. Again, this is a topic addressed in the overall Renew context. The Herold project[7] [Adameit et al., 2010b, Adameit et al., 2010a], in which the author participated, was, for example, aimed at providing agent-based control

[6] This relates to the differently nested reference nets and not the Agent-Activity reference architecture.

[7] The Herold project was funded and supported by the German Federal Ministry of Education and Research (Grant No. 01BS0901).

of network security components. That way a cell-based approach as opposed to a perimeter-based approach was realised. Developed concepts might be incorporated into the Paffin-System. It is feasible to have specialised security Paffins protecting (via interfaces and net synchronisations) subsystems and cooperating to provide security encompassing the entire system. A recent bachelor thesis [Schmidt, 2016] aimed at incorporating the Apache ActiveMQ message server[8] into the Capa framework to secure agent interactions. Utilising these research results from the general context of TGI for the Paffin-System should be the starting point of future work in the context of security for the Paffin-System.

Robustness: Another open point of the Paffin-System is its robustness. Again, this is linked to general issues of Renew. The high degree of concurrency in reference net systems is the cause of these issues. For example, transitions editing Java objects that are referenced in multiple net instances can cause corruptions in these objects. Avoiding these situations requires careful modelling efforts. Often, these situations evolve naturally when a net system is further developed, therefore avoiding them or automatically handling them is difficult. If such errors stemming from concurrency issues occur, the entire system may deadlock as critical data can no longer be accessed. For the current version of the Paffin-System, extensive tests have led to the elimination of these kinds of errors in the framework. However, for future applications this still remains an open point that needs to be addressed in the general Renew context.

Concluding Discussion This section discussed the strengths and open points of the Agent-Activity integration approach and Paffin-System implementation. Overall, the strengths cover many important areas. First and foremost, enabling the synergies between agents and workflows, previously presented in Section 12.3, may be the most prominent and important strengths of Agent-Activities. Many of the strengths listed in this section relate to these synergies, both implicitly and explicitly, highlighting the core significance of the synergies.

However, there are still a number of open points. Many of them relate to issues that were either explicitly excluded and out of scope for this thesis or that are subsumed in the general context of the TGI research group and Renew. Nonetheless these open points are worth considering and addressing in future work. Many relate to technical issues of the framework and environment that can be fixed going forward in the general context. For the others, feasible solutions have been presented. Overall, none of these open points can be considered critical or diminishing the general quality of the results presented in this thesis.

Some of the open points, though, relate to the increased modelling complexity and the requirements that follow from it. As discussed, they can be considered as the cost of the integration, in exchange for which the strengths and synergies are made available. The increased modelling complexity is a direct result of the integration, but can be addressed, as discussed, by providing adequate support tools and mechanisms.

In conclusion, comparing the strengths to the open points yields a clear view in favour of the strengths. The benefits of increased capabilities and especially the synergies outweigh the costs of the open points. Furthermore, all open points were presented with possibilities on how to address them as future work. This means that their effects can be reduced even further.

[8] `http://activemq.apache.org/` (last accessed May 28th, 2017)

Continued work on and with both the approach and prototype is, in the personal assessment of the author of this thesis, desirable. Successor theses are already planned and in some cases already in work. While future work will further investigate the strengths and address the open points, that future work is also required to progress the understanding of the approach and system. Increasing that understanding will elaborate on and possibly even discover new strengths, thus making the Agent-Activity integration approach and its Paffin-System implementation more usable, powerful and valuable.

12.5 General Application Areas

At its core, the Agent-Activity integration approach describes a way to create software systems that utilise an integration of agents and workflows. An important point of discussion are the general application areas in which the approach can most suitably be applied. Chapter 11 already discussed concrete possible applications, with especially Section 11.5 highlighting inter-organisational contexts as a prime application area. The discussions in this current section move away from concrete application areas, like inter-organisational contexts or distributed software engineering, and consider the application areas more abstractly and generally. Here, the most prominent abstract qualities that the integration via Agent-Activities can exploit are shortly discussed.

First of all, the integration via Agent-Activities is useful in any setting in which either agents or workflows are considered useful. For agents such settings include any area which requires large degrees of distribution, autonomy, intelligent components, reactivity etc. Basically, wherever the properties of intelligent agents (see Section 2.2.1) can be usefully applied. For workflows, the intended settings require strong (business) process focus, organisational questions and user/resource support. In any of these kinds of application areas and settings, the Agent-Activity approach is viable, because it fully supports the traditional agent and workflow views, and useful, because it additionally enables the strengths and synergies (see Sections 12.3 and 12.4) of an integration to be used as well.

Generally, an integration of agents and workflows is intended more for large-scale systems. The synergies and strengths are also beneficial in small-scale systems, but the costs of the integration, i.e. the increased management and modelling complexity (see discussion in Section 12.4), may still be too high. In large-scale systems the synergies and strengths can be more easily used to their full potential.

Another two very general areas relate to an overall structural or behavioural focus of a system[9]. Such a system may, in the structural case, emphasise the individual components, their distribution, their organisation/relation to one another or their internal architecture. In the behavioural case, a system may emphasise the process or task as the main concept. In both cases the integration can be used to fully support the intended structural or behavioural focus, while also maintaining strong capabilities for the respective other perspective. This way, as opposed to traditional agent and workflow approaches, both structure and behaviour can be modelled and handled equally without sacrificing the strengths for either of them.

A further general application area concerns agent systems where system modellers and developers discover a need for enhanced process- and/or task-orientation, as well as workflow systems in which a need for agent properties is detected. This largely corresponds

[9] Of course, many systems for which agents are intended and suitable feature such a structural focus. Analogously with workflows. However, the structural and behavioural focus should be highlighted and discussed here explicitly as well, due to the significance of structure and behaviour in the context of this thesis overall.

to the agent-based workflow management systems from Section 3.3.1 and the workflow-based agent management systems from Section 3.3.2. Processes can become mobile, interacting or intelligent, while agents can utilise workflow and task principles to organise the work/behaviour amongst each other and interact with human users. Here, the Agent-Activity approach can fully supply the advantages these partial integration efforts provide (discussed in detail in Section 13.3), as well as the other, previously discussed synergies.

Finally, there are some application areas and settings in which particular integration and hybrid characteristics of the Agent-Activity approach can be utilised to large effect. Some examples are given in the following.

Versatility: Whenever an application area requires a wide range of structural and behavioural functions and mechanisms, the Agent-Activity approach is quite suitable. Modellers are free to choose and combine the best suiting mechanisms from agents or workflows to solve any particular issue within the application. This versatility was showcased, for example, in Section 11.3.1 where three different solutions to the same application were presented, each with a different focus on agents, workflows or hybrids.

Flexibility: As discussed within the discussion about strengths, the variability of Agent-Activities at runtime, i.e. the ability to determine at runtime which specific Agent-Activity is instantiated in a process as long as it satisfies the interface, provides an effective and convenient flexibility mechanism. Many applications can benefit from this flexibility. Whenever there are many predefined variants or options within a larger behaviour (e.g. payment variants, assistance levels, problem solving algorithms) the Agent-Activity construct can be used to model the abstract behaviour in a clearly arranged and simple way, while the details are added within the different internal processes.

Collaboration: In settings in which a high degree of collaboration between users or components is required, the inherent duality of integrated entities and Agent-Activities can be fully exploited. In a collaboration the process describing and supporting the interactions, as well as the involved actors are highly important. Here, the ability to switch from actor (agent) to process (workflow) states can be used to model complex collaborations within a relatively simple model.

Encapsulation: As described in detail on the entity level discussion in the synergies section, integrated entities excel at encapsulating elements of a system based on almost any relation. They can encapsulate cases, actors, resources, etc. This ability of encapsulation based on relation can be exploited in many application settings. For example, encapsulating a single business object like a customer order in an enterprise resource planning system allows for all of the related order, shipping, accounting, invoice etc. documents to be gathered into a single element of the system. Modelling, analysing and monitoring the system can be improved this way. Furthermore, the encapsulation allows for separation of concerns as is needed in, for example, inter-organisational contexts. The entity can be made to appear to the outside as a black box, thus protecting and securing data and functionality from unauthorised access.

Interoperability: Applications settings in which openness of the systems mandates a large degree of interoperability are another area in which the Agent-Activity approach can be nicely utilised. Apart from relying on the FIPA standards of agent communication (see Section 2.2.2) to ensure that the entity/agent components are able to communicate, the general agent functionality can be utilised to improve interoperability. Using agent actions to transform different (process or otherwise) models so that they can be used

across the system is feasible. Further using the abstraction provided through workflow tasks in agent behaviour to decouple different entity as agent functionalities also improves interoperability. In the latter case the entities as agents only need to be able to communicate with a platform WFMS that can provide them with the task information and parameters.

Processing: A major synergy on the level of Agent-Activities is the ability to intersperse agent functions before, after or concurrent to workflow tasks. This can be utilised in any system that requires a large amount of result/data processing between actual work steps. These work steps can be organised in workflow tasks using Agent-Activities.

User assistance: Workflow systems often aim to support users in the work they perform. However, the workflow systems are also often rather static in the assistance they are able to provide. Agent intelligence, available in the integration via Agent-Activity, can be used to improve upon this. Take, for example, the Paose teaching support system from Section 11.3.3. An envisioned extension allows for the entity representing the worksheet workflow to monitor the progress of the students. If the students have obvious issues with the exercise or if they take far longer than the rest of the class, the entity may react and offer assistance. This assistance can be provided in the form of hints or partial solutions to exercises, but it can also go as far as to inform a tutor or a class supervisor and request help on behalf of the students. All of this is possible through the integration and provision of agent functionality within the workflow tasks. Generally, whenever user support and assistance are important in an application setting, the extended capabilities of the Agent-Activity as opposed to traditional agents or workflows can be utilised to great effect.

Of course, overall there are no restrictions of where the Agent-Activity approach can be used. In fact, the discussions in this chapter concerning synergies and strengths make it conceivable that individual benefits could be useful in almost any application. The Agent-Activity approach is powerful and versatile, yet with that power comes increased complexity and overhead. Ultimately, it is up to the modellers to decide on the right modelling tool for the application they desire. In smaller and simpler settings using the Agent-Activity approach may be excessive. The benefits might not outweigh the required effort. However, as discussed in the context of the general integration vision in Chapter 7, an integration such as the Agent-Activity approach is intended to support large-scale, complex and distributed systems. Here, the integration aspects really excel at providing benefits that clearly outweigh the increased complexity.

12.6 Going Forward

This section considers the Agent-Activity approach and the Paffin-System going forward beyond this thesis. Some of these aspects have already been hinted at or discussed, especially regarding applications (see the application visions described in Section 11.4). This current section now emphasises more fundamental aspects of future work, including the incorporation of flexibility into the nets of the Agent-Activity approach and the adaption of Paose to the new concepts.

Please note that this is explicitly *not* the general outlook of this thesis. The content of the following subsections is more concrete and elaborated than abstract ideas for an outlook. It presents concrete directions for future work.

This section considers four areas. Section 12.6.1 examines how flexibility might be added to the approach and prototype. Section 12.6.2, then, discusses validation and verification,

focussing especially on workflow soundness. Next, Section 12.6.3 considers how PAOSE can be applied or adapted to the needs of AGENT-ACTIVITIES. Finally, Section 12.6.4 discusses future work in the context of the PAFFIN-System prototype.

12.6.1 Net Flexibility

Flexibility is a highly relevant aspect in modern software systems. This is especially true when dealing with business processes, since "*flexibility is required to accommodate the need for evolving business processes*" [Reichert and Weber, 2012, p. 43]. Flexibility, in this specific context, refers to the ability to change a process at runtime. Changes may include adding tasks, removing tasks, adding new alternative branches, etc. Since the AGENT-ACTIVITY approach emphasises inter-organisational contexts as an application area, flexibility should be addressed in future work.

For this discussion, a distinction must be made between the conceptual AGENT-ACTIVITY approach and the PAFFIN-System implementation. On the abstract level the concept of the AGENT-ACTIVITY offers a large degree of flexibility. It is defined as an abstract task combining a number of agent actions and workflow operations. It is feasible to assume that the selection and order of actions and operations can be dynamically chosen whenever an AGENT-ACTIVITY is instantiated. This means that at runtime, an AGENT-ACTIVITY can be dynamically and fully adapted to any and all occurring needs. Similar flexibility can be applied to the concept of the process-protocol, resulting in not only the AGENT-ACTIVITIES themselves being internally fully flexible, but also their composition as well. This flexibility then cascades upwards through the system. Entities executing such flexible process-protocols are then fully flexible themselves, on flexible platforms forming fully flexible entity systems.

Of course, such full flexibility is only a hypothetical construct applicable to the general concept. In order to actually realise flexibility, it needs to be applied or related to a practical implementation like the PAFFIN-System.

Currently, flexibility in the PAFFIN-System is limited to variable execution options. AGENT-ACTIVITY-TRANSITIONS can be inscribed with variables determining their AAO at runtime. That way it is possible to have multiple options for one position in the overall control flow of the process-protocol. In other words, the AGENT-ACTIVITY-TRANSITION currently supports dynamic, variable, predefined subroutines. Similarly, workflow task identifiers for workflow operations within AGENT-ACTIVITIES are variable. Here, too, the dynamic situation at runtime can be taken into account and specific predefined variants chosen for execution.

Using variables in the inscriptions of AGENT-ACTIVITIES can, potentially, be used to create generic process-protocols, though. These generic process-protocols not would feature true net flexibility, but would simulate it by constructing procedures from predefined and modular AGENT-ACTIVITIES. The core idea revolves around a loop, in which the first AGENT-ACTIVITY is used to select and determine the following AGENT-ACTIVITIES. All iterations of that loop would then create a virtual, generic process-protocol.

Figure 12.4 illustrates this mechanism to realise generic process-protocols[10]. The loop begins on the left-hand side. Here, initial and external input parameters can arrive from outside of the process-protocol. The first AGENT-ACTIVITY, named *Select*, is used to select the following functionality. This AGENT-ACTIVITY could be implemented as a human user GUI task or as an automatic processing of the input. The result of the selection

[10] Figure 12.4 exclusively uses direct triggers of AGENT-ACTIVITIES. This choice was made to simplify the descriptions. Reactive and proactive triggers can also be supported in the approach. These would have to be added at the preprocessing step as an additional value within the used tuples.

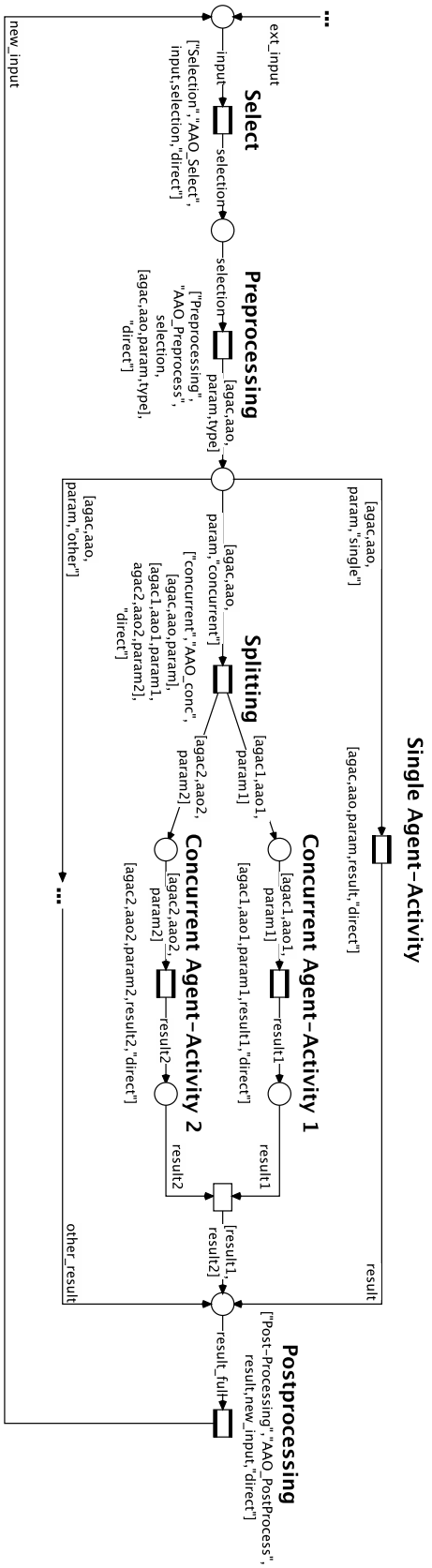

Figure 12.4: Mock-up of a generic process-protocol

contains the Agent-Activities to execute as well as the pattern they are to be executed in. Figure 12.4 only features the patterns for executing a single Agent-Activity and for executing two Agent-Activities concurrently. Many other options are possible and would need to be implemented to enable the generic process-protocol to cover as much functionality as possible. Options for these patterns can be taken, for example, from net transformation [Murata, 1989], workflow patterns [van der Aalst et al., 2003] or the ADEPT metamodel for flexible workflows [Dadam et al., 2009].

Next, the result of the selection is processed in the *Preprocessing* Agent-Activity. In this Agent-Activity the selection is transformed into the concrete values used for identifying the Agent-Activity that should be executed. The results contain the name of the Agent-Activity, the identifies of the AAO, the parameters (possibly processed from the original input and selection) and the pattern. The pattern is used to determine which of the next net branches to use.

The upper branch in Figure 12.4 implements the single Agent-Activity. Here, the results of the preprocessing can be directly used in the inscription of the Agent-Activity. The generic result of that single Agent-Activity can also be put directly onto the precondition place for postprocessing. The middle branch in Figure 12.4 implements the execution of two concurrent Agent-Activities. Here, the results from the preprocessing can't be directly used as the individual execution parameters need to be distinguished. This is implemented in the *Splitting* Agent-Activity, which merely unpacks the composite result of preprocessing into individual results for its postconditions. When the two concurrent Agent-Activities are finished their results also need to be composited, indicated here by creating a tuple of the individual results. The lower branch Figure 12.4 indicates where additional execution patterns could be added.

Finally, the *Postprocessing* Agent-Activity reads and processes the previously obtained results. Its own result is then used to provide new input data for the selection Agent-Activity and thereby reinitialise the loop.

With this mechanisms generic process-protocols can be modelled that can simulate net flexibility. The Agent-Activities have to be predefined, which restricts the flexibility to a degree. This can be alleviated by considered that the Agent-Activities can themselves initiate new generic process-protocols, creating a flexible behaviour hierarchy. However, at some point predefined components have to be used.

The technical basis for this generic process-protocol mechanism is already fully available in the current Paffin-System prototype. However, the implementation details, including modelling support and how to actually pre- and postprocess the results, as well as which patterns to implement and how to suitably support them, still need to be fully formulated.

The PPB prototype (see Section 10.3.3) also exhibited the potential to realise some degree of flexibility. It used a text based syntax to describe the processes within Agent-Activities. The net of that process is then generated at runtime. Here it is feasible to edit the textual description to generate flexible nets. Adding new tasks is possible through adding and editing different lines of the textual description. However, due to the discussed usability issues the PPB approach was abandoned. Still, some of the implemented mechanisms to generate nets may be reused in the future.

Further flexibility for reference nets is an active research effort examined by multiple recent master theses [Eberling, 2015, Schmolke, 2015]. However, there is currently no usable, technical solution available for editing and adapting reference net instances at runtime.

In the context of flexibility and more generally nets-within-nets the **H**igher **Or**der **Nets** (Hornets) [Köhler-Bußmeier, 2009a, Köhler-Bußmeier and Heitmann, 2013] approach is also

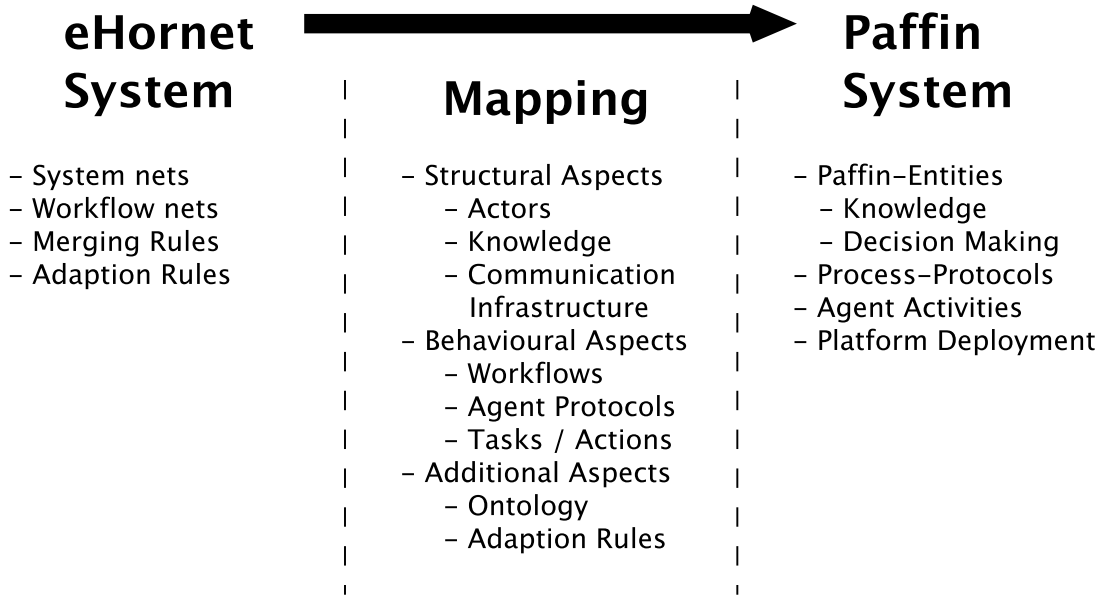

Figure 12.5: Mapping between eHornets and the Paffin-System (from [Wagner et al., 2016a, p. 13])

relevant. Hornets are a net formalism featuring algebraic operations that allow combining and dividing nets at runtime. A mapping between elementary Hornets (eHornets), a simpler version of Hornets, and the Agent-Activity approach in the Paffin-System was presented in [Wagner et al., 2016a]

An eHornet system can be used to describe a system of cooperating actors. Interaction within the eHornet system is realised by merging the nets representing the different actors. This interaction is flexibly handled through algebraic operations on the different nets. Other scenarios include merging (actor) nets with additional branches describing new functionality. This could be utilised in the Paffin-System to provide more flexible process-protocols or Agent-Activities that can be extended at runtime if necessary. The first step of making eHornet operations available for use in the Paffin-System was to provide a mapping between the elements of an eHornet system and the components of the Paffin-System.

Figure 12.5 illustrates that mapping. It identifies structural aspects, behavioural aspects and additional aspects from the eHornet system and translates them into the components of the Paffin-System. For example, actors, part of the structural aspects, are mapped onto Paffin entities while the processes of those actors are mapped onto process-protocols. (Groups of) tasks executed by actors within their processes are mapped onto Agent-Activities.

Currently, the mapping can only be done manually. It serves as a basic reference for translation between the two kinds of models. However, in future work that mapping could be at least semi-automated. Some issues, e.g. the distinction between actors, will still need to be solved by manually, yet the generation of at least skeleton code for the Paffin-System is feasible. A full, round-trip engineering solution can be considered as a distant target of this research.

In summary, while net flexibility can be added to any level of the Agent-Activity reference architecture, the current implementation is still limited due to technical restrictions

of reference nets. The PAFFIN-System already technically supports models that may simulate net flexibility, but no concrete implementation for these models exists as of yet.

12.6.2 Verification and Validation

Topics like system verification and validation are outside of the scope of this thesis. However, given the formal Petri net background, these topics should be considered going forward. Possible, concrete research directions utilising the Petri nets basis of both the AGENT-ACTIVITY approach and PAFFIN-System are therefore envisioned in this section.

Reference nets themselves are too expressive and powerful to be verified. Earlier work [Hewelt et al., 2011], in which the author of this thesis was involved, provided a RENEW plugin with which to apply verification checks of the Lola tool (*A Low Level Petri Net Analyzer*, [Schmidt, 2000b, Wolf, 2016]). This plugin might be used to check the nets of the PAFFIN-System prototype for simple control flow issues, like liveness or deadlock freedom. Advanced checks involving the reference semantics, Java inscriptions and especially nets-within-nets issues are not supported. Another verification approach was examined in [Mascheroni et al., 2010], in which a restricted nets-within-nets formalism called Hypernets was utilised.

There are many other tools concerning Petri net and workflow net analysis and verification, such as Prom[11] [van Dongen et al., 2005] or Woflan[12] [Verbeek and van der Aalst, 2000]. These could feasibly be incorporated into RENEW and used in the context of the PAFFIN-System.

Ongoing and recent research in the context of RENEW and MULAN is concerned with verification, validation and analysis issues. [Stiefelmann, 2016] examined providing model checking facilities in RENEW, [Lehmann and Moldt, 2004] analysed agent protocols. [Azzalini, 2017] examined and prototypically implemented different verification and analysis mechanisms including generating workflow nets from AIP diagrams. [Denz, 2012] analysed multi-agent simulations with the help of process mining mechanisms. [Ortmann, 2010] examined modelling possibilities and verification for business processes and [Haustermann, 2010] analysed reference net based workflows. Finally, the overall SONAR model [Köhler-Bußmeier, 2009b] (see discussion in Section 3.3.3) also heavily emphasises formal aspects.

Going forward, Michael Haustermann currently creates his dissertation thesis [Haustermann, 2017] with exactly the topic of reference net agent verification. Results from that work can conceptually be applied to the AGENT-ACTIVITY and PAFFIN contexts as well. Currently ongoing work of Dennis Schmitz and Martin Wincierz [Wincierz, 2017] is concerned with performing tests in the PAFFIN-System. Based on the results of Mr. Wincierz's bachelor thesis [Wincierz, 2016], they create JUnit-like tests for different components of the system. The tests can take into account individual nets like DCs and the backend nets, but may also consider overarching interactions or entire PAFFIN entities. The goal of these tests is to enable the validation of the behaviour of the considered components in the system.

Overall, the current (as of May 2017) status of the technical facilities for verification and validation in the context of RENEW, reference nets, MULAN and CAPA are not yet *practically* readily available to be used for the PAFFIN-System. Still, the nature of the PAFFIN-System still provides some opportunities, where, once implemented, verification mechanisms may be utilised favourably.

[11] `http://www.promtools.org` and `http://www.processmining.org/` (both last accessed May 28th, 2017)
[12] `http://www.win.tue.nl/woflan/doku.php` (last accessed May 28th, 2017)

Many aspects of the Agent-Activity approach and the Paffin-System relate to workflows and workflow nets. For workflow nets the soundness criterion (see Definition 2.20) is elemental to verification questions. Assuming that a practical implementation to check workflow soundness in reference nets is available as a plugin or an external tool, the soundness criterion can be applied to a number of components within the Paffin-System.

At the moment, there are no restrictions on how to structure application-specific code in Paffin applications. This is especially true for process-protocols. Modellers are free to model Agent-Activities, regular transitions and places, different arc types, etc. in any way they choose (as long as basic Petri net principles are not violated). While this does allow modellers freedom in what they can achieve in a process-protocol, it introduces issues when the size and complexity of the process-protocols increases. It may be possible to understand the behaviour of a small, sequential process-protocol, but given interconnected loops, concurrent executions, etc. it quickly becomes difficult to make accurate assumptions about the net. Here, restricting process-protocols to the principles of workflow soundness can help.

Process-protocols already contain part of the basic structure of a workflow net. They are started by the Paffin at some point, execute a number of abstract tasks, the Agent-Activities, and are then terminated. The idea is to encourage modellers to model the process-protocols fully as sound workflow nets. This means that, for starters, they restrict themselves to unique start and end components and have all Agent-Activities on paths between them. Additionally, it has to be ensured that the end component can be reached from anywhere within the net, that there are no more (control) tokens in the net once the end component is reached and that there are no dead transitions. All this should be monitored by a workflow soundness tool at design time. Whenever the modeller adds, edits or removes an element the tool would check the net and inform the modeller if soundness was violated. Alternatively, the check could be called manually, although this might defeat the purpose if the modeller intends to avoid the effort.

An open question is if the soundness should be enforced or only encouraged. If it is enforced the Paffin-System would only function with sound workflow process-protocols. This could limit its capabilities, prohibiting none-workflow-like behaviour in applications. However, if it is enforced it would verify a certain level of robustness for an application. The author's suggestion is to only encourage it, though. Enforced soundness might hinder modellers in testing out new ideas easily and a thorough modeller could still check all process-protocols and ensure the desired level of robustness.

Analogously, workflow soundness can also be applied to the internal process of the Agent-Activities. Here, the individual components, i.e. actions and operations, might prove too varied to effectively ensure properties during runtime, but basic structural soundness can still be observed. As with the process-protocols, encouraging a sound modelling is the preferred choice here.

The previous passages discussed how workflow soundness might be used during the design time to ensure Paffin applications themselves feature sound Paffin behaviour. However, soundness can also be used in the applications itself. Paffins may serve as automatic resources to workflow tasks of other Paffins. Given an open system the Paffins might not have the same origin or quality of implementation. Consequently, the engines may wish to ensure that a resource Paffin is indeed capable of executing the work associated with their tasks. A possible approach here is for the engine Paffin to request the process-protocol associated with the automatic task execution as a net file from the resource and check its workflow soundness before the engine agrees to a workitem request from the resource Paffin. This could increase the confidence of an engine into the resource and confirm its

ability to execute the work. Such mechanism would further utilise agent intelligence and decision making within the workflow (engine) execution.

Ultimately, the questions of verification and also validation could be highly useful in the Agent-Activity and Paffin contexts. The examples and basic ideas presented above are just rough possibilities of how to utilise the formal Petri net basis. Many more options exist. Unfortunately, the technical availability and maturity of current tools is insufficient at the moment. Given successful future work in those technical areas, an application of the results in the Paffin-System is clearly desirable.

12.6.3 Paose for Paffins

The Paose methodology has strongly influenced the results of this thesis. Its guiding metaphor of considering everything as an agent (see Section 5.1.1), the Paose matrix distinguishing structural roles and behavioural interactions as main dimensions of a system and the iterative approach to prototyping are just some of the examples where the Paose aspects can be discovered in this thesis. Furthermore, the entire development of the Paffin-System heavily utilised tools, methods and mechanisms from the context of Paose (e.g. the WebGateway).

Overall, Paose has established itself as a mature and practical way of realising multi-agent systems. It has been in use at the TGI group of the University of Hamburg for a number of years and has evolved to keep up with the latest developments of Renew, Mulan and Capa. Considering the Agent-Activity integration approach and the Paffin-System framework as another evolution in this context encourages the adoption of Paose for integrated entity systems.

In general, Paose could conceivably be used as is for the creation of Agent-Activities and Paffin applications. All the mechanisms from Capa are available in the Paffin-System, as are counterparts to all modelling artefacts. Agents correspond to Paffins, Agent protocols to process-protocols, etc. However, many of the newly introduced features are not covered by Paose and would have to be manually handled by modellers. Therefore, a number of extensions are reasonable, some of which are discussed in the following paragraphs.

Adapting the Paose Matrix

The Paose matrix (see Section 2.2.3 and Figure 2.12) is one of the central metaphors and artefacts in Paose. In short, it is created as an overview of which agent roles are involved in which agent interactions. Since Agent-Activities no longer consider components that are just agents or interactions, the Paose matrix should be adapted to incorporate the integration concepts, especially the integrated entities that can be agents, workflows or both. Please note that the following is just a preliminary sketch of one possible adaption of the Paose matrix.

While the original Paose matrix captures roles and interactions, that distinction is more difficult for Agent-Activities. Integrated (Paffin) entities can serve as both actors (having roles) and processes (including interactions). Figure 12.6 therefore proposes different dimensions for the matrix. Instead of having distinct sets of roles and interactions the new matrix has a shared set of entities with the dimensions being the entity considered as an actor and the entity considered as a process.

With these adapted dimensions it is possible for the matrix to capture the novel ways in which entities can interact, thanks to the Agent-Activity concept. For example, in Figure 12.6, entity one is involved in the processes of entities two and four. The process of

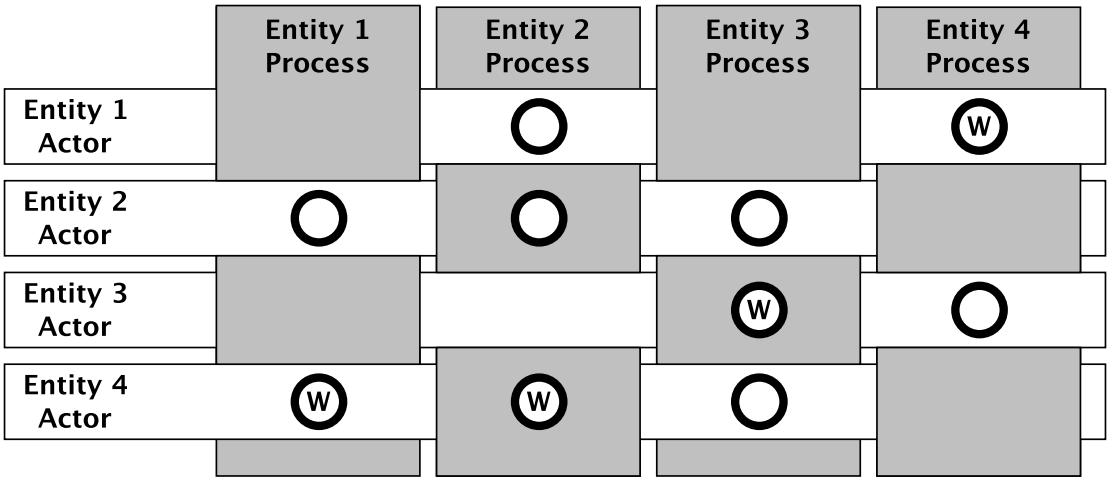

Figure 12.6: Adapted Paose matrix for the Agent-Activity approach

entity four involves only entities one and three as actors. This would represent a regular interaction between the two actors, with an additional marking of **W** to indicate that entity one's part of that process is realised as a workflow task. The process of entity two then features another particularity of the integration. Not only are entities one and four involved, entity two is itself involved in its process as an actor. This means that at some point of the process, entity two becomes an actor in addition to being a process, indicating, for example, that the workflow task for entity four in that interaction requires additional agent functionality to be executed by the process/workflow itself. As a final example of what is possible to represent with the new matrix, consider the entity three. Entity three is involved as actor in entity four's process, as well as in its own process, but as a workflow task. Here, the marking does represents that entity three as an actor becomes a workflow for its own process. This could, for example, happen if entity represents a human user. The marking would then indicate that there are workflow tasks in entity three for (the human user represented by) entity three.

These are just some examples of integration aspects that cannot be conveyed in the original Paose matrix. As this new matrix is still an early sketch, it is unclear if the new dimensions are capable of sufficiently covering the integration for the purposes within the wider Paose approach. This has to be examined and tested in practical settings, for example in the yearly Paose teaching project. Other approaches may consider allowing different types of entity aspects in the same dimensions, more connector variants or adding an additional dimension. However, even the approach to a new matrix presented here already highlights that some changes are necessary in order to enable the full capabilities of the Agent-Activity approach to be properly utilised in the methodology.

Paose with Paffin Modelling Artefacts

The modelling artefacts in Paose can be roughly classified into two groups. The first one are those artefacts that are used to provide conceptual models that are not directly, technically executed. These are the coarse design diagram (CDD), the Paose matrix, the agent role model (ARM), the ontology diagram and the agent interaction protocols (AIP). From these artefacts basic, skeleton code for the technical artefacts is generated. Agent

protocols, decision components, knowledge bases, the ontology classes and additional Java classes constitute those technical artefacts.

How the conceptual models can be adapted to the Agent-Activity approach has been shown above for the example of the Paose matrix. The other diagrams should also be adapted to capture integration mechanisms and concepts. For example, the CDD strongly distinguishes between actors and interactions. That distinction isn't as clear anymore when considering that integrated entities can be both actors and processes. Similarly, the ARM needs to capture workflow roles and permissions when describing the basic knowledge of an integrated entity. How the AIP can be extended is discussed further below when tool support is examined. The ontology largely remains unchanged, because its main role in Paose, namely connecting structure (roles) and behaviour (interactions) and ensuring that different components of a system understand each other, can be directly adopted for Agent-Activities.

On the technical side, the artefacts remain mostly unchanged, though some additional ones are required. Note that the technical artefacts relate to the implementation in the Paffin-System, not the conceptual Agent-Activity approach. A different implementation of the approach could possibly yield different technical artefacts. Decision components remain unchanged, as do the knowledge base contents, ontology and supplementary Java classes. Necessary changes to incorporate the integration have been adapted into the conceptual models from which these technical artefacts are generated. Agent protocols are replaced by process-protocols. This does not greatly change their role in the execution though. However, a number of technical artefacts are added.

First and foremost, the process-protocols contain Agent-Activities. These represent a new modelling level below the rough entity behaviour given by the process-protocols. Nets for the AAOs are therefore a new technical artefact. Furthermore, the administrative workflow data needs to be provided in a persistent storage, presumably a database. Entries must be provided for workflow roles, resources, rules, permissions, tasks and GUIs. Further database entries may become necessary in future evolutions of the Paffin-System.

One of the requirements of the adaption of Paose to the Agent-Activity approach is to provide a methodological approach to incorporating all of these changed artefacts into a coherent modelling process. Just providing the technically and conceptually adapted artefacts is not sufficient. Only through changes in the methodology can the integration be efficiently utilised. Any future work dealing with Paose and Agent-Activities needs to provide a vision capturing such changes.

Tool Support

Tool support is the final issue examined in this consideration of a future adaption of Paose for Paffins and Agent-Activities. Currently, there are a number of specialised tools available in the Renew environment. Tools for generating folder structures from the CDD, for monitoring the system, etc. Such tools are also valuable for Agent-Activities and Paffins and are necessary to properly support modellers.

Here, a vision of a modelling tool for process-protocols is discussed as an example of possible future tool support. The functionality of this tool is oriented around the Agent Interaction Protocol (AIP) diagram tool for Capa agents in Paose. AIP diagrams enable designing the interactions between Capa agents in an abstract way. The tool allows for the generation of skeleton code for the protocols described in the AIP interaction. Skeleton code provides the basis for interaction implementation. The generated skeleton protocols

already contain the basic order and structure of agent actions[13] and only need to be filled with additional details and data flow.

For the process-protocols in the PAFFIN-System a similar tool is desirable. However, directly utilising the AIP diagram tool is not possible, since it does not support workflow operations and workflow concepts in the design of interactions. The desired tool should follow the same design principles, since it is well established and a similar tool would make it easier for CAPA modellers to utilise it.

The basic purpose of the tool is to design interactions between different PAFFINS. Interactions consist of process-protocols for each involved PAFFIN. Here, the basic swimlanes from the AIP diagram tool can be reused. Process-protocols consist of AGENT-ACTIVITIES and are connected/related through the agent actions and workflow operations defined in the different AGENT-ACTIVITIES. Additionally, AGENT-ACTIVITIES may also describe workflow tasks that need to be executed by human users. These tasks do not correspond to another process-protocol involved in the interaction, yet this kind of behaviour also needs to be captured in the tool.

The tool requires to model two things in particular. The process-protocols of the involved PAFFIN entities and the AGENT-ACTIVITIES they consist of. An advantageous realisation would enable the two levels to be modelled in one tool. Modellers could then create the rough structure of AGENT-ACTIVITIES for a process-protocol, and then expand or zoom into the individual AGENT-ACTIVITIES to model the individual actions and operations. This would provide two views on each AGENT-ACTIVITY of the model. A concise view that would collapse/zoom out of the AGENT-ACTIVITY and a detailed view that would expand/zoom into the AGENT-ACTIVITIES. Modelling in the same tool would also allow easily connecting an action or operation from within an AGENT-ACTIVITY to another AGENT-ACTIVITY in possibly another process-protocol. Note that the usual requirements of modern modelling tools also apply including the ability to undo errors, drag and drop functionality with connectors on elements etc.

The internal process of each AGENT-ACTIVITY consists of an ordered set of fundamental agent actions and basic workflow operations. These actions and operations are arranged in a specific execution order. That order has to be modelled for each AGENT-ACTIVITY. However, most of the actions and operations also describe some form of interaction with external (w.r.t. the current AGENT-ACTIVITY) elements. These connections also need to be modelled.

The following describes how connecting the action/operation externally affects the model. The internal connection, i.e. the arrangement of actions and operations into execution orders, is simple Petri net modelling and not discussed further.

One of the key concepts for the following is the *corresponding receiving action*[14]. The corresponding receiving action for an agent action or workflow operation is the required counterpart to that action or operation. For example, when a message is sent somewhere the corresponding receiving action is the reception of that message somewhere else in the system. These corresponding receiving actions are defined for each action and operation in the following.

In general, when connecting an outgoing agent action or a workflow operation to its corresponding receiving action, the target is always another lane describing another process-protocol or a human user (or other resource not represented by a process-protocol). To

[13] The actions that can be designed in the AIP diagram tool are, in fact, the fundamental agent actions described in Section 5.2, even if they are not known or named as such in CAPA.

[14] Action here does not necessarily refer to agent action, but rather to an abstract action of the target entity. This can be an agent action, a workflow operation or an administrative action.

simplify these descriptions the target lane is considered to be already existent. However, there are still three variants to consider when connecting an outgoing agent action or workflow operation. If the target lane does not yet include the target Agent-Activity a new, blank Agent-Activity should be created in that lane, a corresponding receiving action added with default parameters and the outgoing action connected to it. If the target lane already includes the target Agent-Activity and that target Agent-Activity already contains an otherwise unconnected corresponding receiving action, the outgoing action should be connected to that corresponding receiving action. Lastly, if the target lane already includes the target Agent-Activity, but that Agent-Activity doesn't contain an otherwise unconnected corresponding receiving action, a new corresponding receiving action should be created and the outgoing action connected to that. Of course, the new corresponding receiving action needs to be manually arranged afterwards to fit into the Agent-Activity model.

- **Send Message:** Sending a message indicates that some data is to be transmitted from one Paffin entity to another or a set of others. The corresponding receiving action is a receive message action in the process-protocol of the other Paffin entity/entities.

  Sending a message can only be connected to another process-protocol in another Paffin entity, since the agent nature prohibits human users or other resource that are not represented by Paffin entities from receiving messages. If, however, the human user (or other resource) is represented by a Paffin a message can be sent to that proxy. In that case, though, a process-protocol exists for the representing Paffin entity.

  In the detailed view of both involved Agent-Activities, a direct arrow should be drawn coming from the send message action and going into the receive message action. In the concise view the direct arrow would originate in the Agent-Activity containing the send action and end in the Agent-Activity containing the receive action.
- **Receive Message:** Receive message functions analogously to its corresponding receiving action of send message.
- **Internal Action:** Internal actions do not connect externally and consequently do not need to be considered w.r.t. external model connections and corresponding receiving actions.
- **Request Workitem:** Requesting a workitem indicates that the entity executing this workflow operation provides a task (workitem) and that an external resource is set to begin work on that workitem. The main issue with modelling this operation is that there are two kinds of resources that have to be handled, connected and displayed differently.

  The first option indicates that the workitem is to be executed by another Paffin entity explicitly modelled in this interaction. In that case the beginning of work on that workitem would initiate a new process-protocol in that Paffin entity. Consequently, the corresponding receiving action is the administrative action of starting the new process-protocol and transmitting the parameters from the workitem to that process-protocol. Alternatively, it is also feasible that the start of work corresponds to a delayed, i.e. reactively or proactively triggered, action inside of an existing process-protocol. In that case the corresponding receiving action is an input of data (possibly as an agent message) into the existing process-protocol. Represented as a different kind of arrow, the connection would start at either the request workitem operation (in the detailed view) or the Agent-Activity (in the concise view) and end at the start of the new process-protocol or at the start of work in an existing process-protocol.

The second option is to connect the operation to a resource that is not directly modelled in the current interaction. This includes human users and other resources that are not represented directly by PAFFIN entities. But, it can also include PAFFIN entities that are simply not modelled in the current interaction. Such PAFFIN entities possess a process-protocol which is initiated by this workitem, but it is outside of the scope of the current interaction diagram and considered as a black box. Situations like this can happen if, for example, the target PAFFIN entity is under the responsibility of another development team. In that case the interface is clearly defined through the workflow task exchange so that further details are unnecessary. The corresponding receiving action should be a diagram element that illustrates the abstract task and black box nature. One possible option is to use the task-transition from RENEW workflow nets (see Section 2.3.2), as it is an established representation of a workflow task. Represented with the same arrow as the first option, the connection would also start at either the request workitem operation (in the detailed view) or the AGENT-ACTIVITY (in the concise view) and end at the element used to represent the task.

- **Confirm Activity:** Confirming an activity indicates that a workflow task (activity) has been successfully completed and that work in the PAFFIN entity controlling that task can continue beyond it. As described before, a workflow task can correspond to two things. It can be an abstract task being executed by a human user, a resource not represented by a PAFFIN entity or by a PAFFIN entity outside of the scope of the current diagram. Alternatively, it can correspond to a process-protocol being actively modelled in the current diagram. For both cases the corresponding receiving action is the administrative end of the workflow task under examination. In the first case, there is no explicit end to the task because it is represented by a single element. In the second case, however, the end of the task is the end of the process-protocol or the end of the work in the process-protocol that corresponds to that task. The model connection would start at the representative element in the first case and at the end of the process-protocol or task-ending part of the process-protocol in the second case. The connection should be represented by the same kind of arrow the request workitem connection is represented in. The different directions are sufficient for distinction.
- **Cancel Activity:** Cancelling an activity indicates that a workflow task (activity) can't be completed successfully and must be aborted with a failure. It is completely analogous to the confirm activity operation, with the exception that, if the task corresponds to another process-protocol, it can be called from potentially anywhere in that process-protocol. Determining where, in another process-protocol, an activity can be cancelled is one of the responsibilities of the system modellers.

Different symbols should be used for each kind of element in the model. For AGENT-ACTIVITIES the AGENT-ACTIVITY-TRANSITION should be used as a symbol. For abstract tasks the original workflow task-transition should be used. For the remaining elements (actions and operations in AGENT-ACTIVITIES, starting and stopping an AGENT-ACTIVITY, starting and stopping a process-protocol), clearly distinguishable symbols should be used.

Each element and symbol should be inscribed. This way, parameters and details could be added to the model that could be directly used for the later generation of skeleton code. For each element and symbol, default values need to be defined that are used for code generation if no specific values are provided by system modellers. While most of the elements can be inscribed with the direct inscriptions used later in nets, workflow operations differ from that. For these elements the task, task parameters (e.g. input data, task description, result, utilised user interface, utilised resource task action) and

administrative parameters (e.g. roles, rules, rights) also need to be defined. For that purpose special inscriptions need to be defined. While it is possible to simply inscribe the full specification of the ontology object representing the task, this option is not very user friendly for system modellers.

Figure 12.7 shows a mock-up that illustrates how a diagram following the previously laid out vision for an AIP-like model for the PAFFIN-System might look like. Regular transitions with descriptive names currently represent most elements. An implemented tool should provide more memorable and intuitive symbols. Collapsed AGENT-ACTIVITIES are represented as AGENT-ACTIVITY-TRANSITIONS with a + symbol. Expanded AGENT-ACTIVITIES are larger areas similar in shape to the AGENT-ACTIVITY-TRANSITIONS with a large - in the upper thick bar and containing a simplified representation of the internal process. The + and - symbols should be interactive buttons for switching between the collapsed and expanded views. The diagram in Figure 12.7 incorporates all agent actions and workflow operations, including usage variants. Starting point is the lane of *Paffin Entity A PrPr 1*. Note that there is no explicit application case for the interaction in Figure 12.7, although it would be possible to create one Such an application case is irrelevant for these current discussion.

The diagram shown in Figure 12.7 provides a large amount of information about the interaction being designed. Which integrated entities are directly involved, which external actors (human users and external PAFFINS) are involved, which AGENT-ACTIVITIES are being executed by which PAFFINS, what agent actions and workflow operations are executed in the AGENT-ACTIVITIES, how the AGENT-ACTIVITIES are connected, etc. As previously implied, the tool should also be able to generate skeleton code from those diagrams. This tremendously supports the modelling effort as the automatically generated code only needs to be enriched with certain details and data flow to make it directly executable. The following three types of nets/objects should be generated from the AIP-like diagrams:

Process-protocols: The process-protocol nets correspond directly to lanes of the concise model view (i.e. with all AGENT-ACTIVITIES collapsed). The process-protocols would only contain the AGENT-ACTIVITIES as AGENT-ACTIVITY-TRANSITIONS, the basic control flow between the AGENT-ACTIVITIES and the provided or default inscriptions for the AGENT-ACTIVITIES. Data processing, flow and exchange between AGENT-ACTIVITIES would still need to be added to these nets.

Internal processes: For each AGENT-ACTIVITY the tool should generate the internal process as a net. The generated net would include the actions and operations (using the PAFFIN net components from Section 10.2.3), the control flow between them and the provided or default inscriptions for the actions and operations. Additional data flow, detailed inscriptions and additional data processing still have to be added.

Database entries: Workflow tasks in the PAFFIN-System are supported by a database running on each platform and management. This database contains all data necessary for executing and managing the workflow tasks (see Section 10.2.14). For code generation in this tool only data about tasks is relevant. Task data needs to be provided in the form of ontology objects of the class `PaffinDBTask`. Fields of that class include task name, task title, task description, resource task action, gui identifier, required permission set and the assignable status. The generation should create these objects with either data from inscriptions in the model or default/generated values.

Using an AIP-like tool in the PAFFIN-System would support system modellers by separating design into two stages. First, they would take the global (w.r.t. one interaction) perspective on the AGENT-ACTIVITIES and how they are interconnected by modelling the

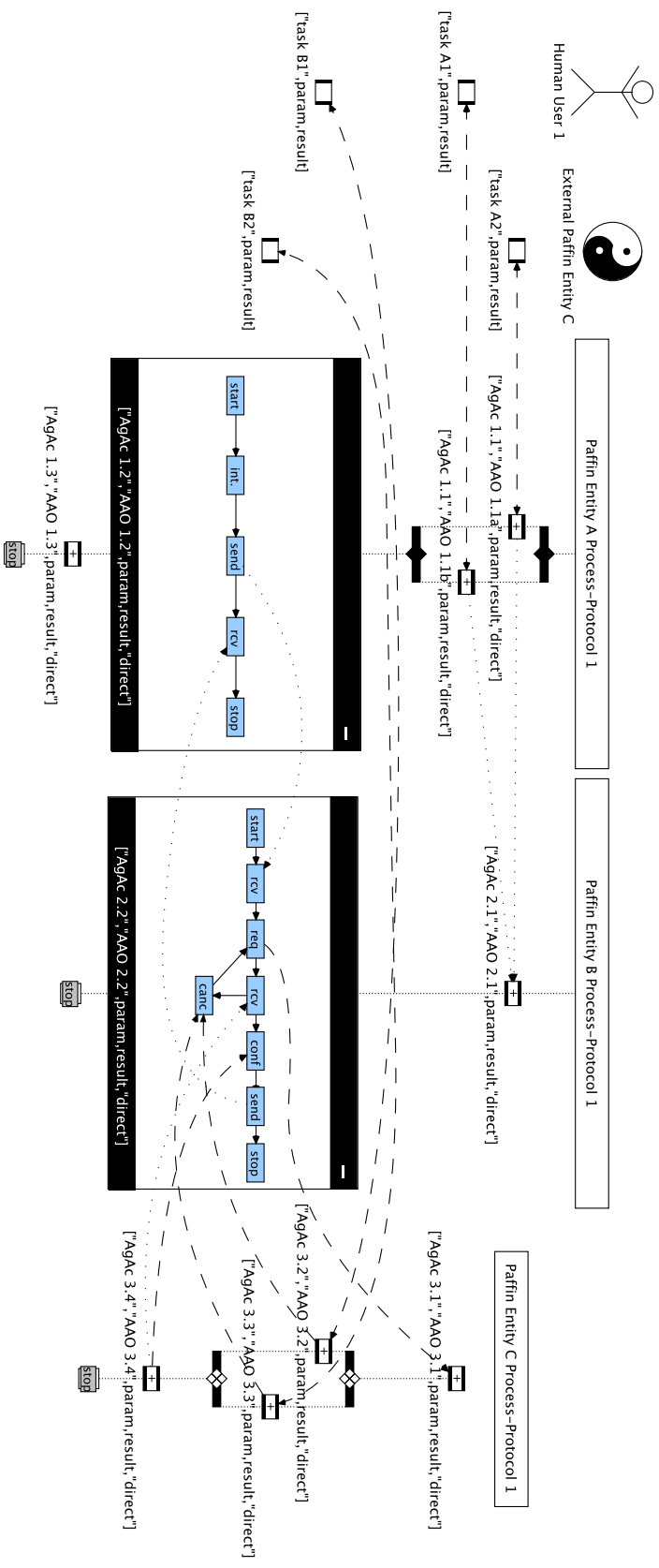

Figure 12.7: Mock-up of AIP-like diagram for PAFFINS

AIP-like model. This would help keep overarching issues in mind and focus modellers onto the actual functionality by abstracting from implementation details. Second, they would take the local (w.r.t. one specific net at a time), detailed perspective. Here, the overarching issues can be mostly disregarded (because they are taken care of on the first level) and attention can be paid to the details of the implementation to solve issues of efficiency and data flow. Overall, this tool would focus system modellers on distinct issues at any time and reduce the effort of implementation. Consequently, it is a very desirable tool for the Agent-Activity approach and the Paffin-System.

Paose Extension Conclusion

The previously described extensions to Paose enable it to better utilise the mechanisms introduced by the Agent-Activity integration approach. More extensions are conceivable on both the methodological and the practical support level. These extension range from new tools that generate global structural and behavioural perspectives, to adding a new dimension to the matrix to capture user interactions. Each of these kinds of extensions opens up the capabilities of an integration ever more to system modellers. In fact, it is even feasible to fundamentally adapt the approach to not consider everything as just an agent, but also as a workflow and an integrated entity. Besides such fundamental questions, the current and future alignment of Agent-Activities and Paose can be exploited to provide a comprehensive modelling support environment featuring a process-, organisation- and agent-oriented guidance for system modellers.

12.6.4 Paffin-System Extensions

This section discusses some possible future extensions of the Paffin-System prototype. Note that the following are concrete features of future work. General notions, such as security or performance improvements, are not considered here.

Address the current limitations: Section 11.2.2 discussed the current limitations of the Paffin-System prototype. Some of the issues presented there are signify important advances for the system when they are implemented. Therefore, future work on the Paffin-System should first address these limitations.

First and foremost, the persistent data storage at runtime should be addressed. Here, the existing functionality from the PersistentOntology plugin can be utilised. Currently, only the platform WFMS Paffin has read access to the database. The first addition should be write access in combination with administration functionality in the GUI. That way new roles, users, etc. can be added and edited at runtime. The second addition could involve the application Paffins with the database. Here, runtime application data could be stored.

Beyond the persistent data storage, the Agent-Activity update mechanism should be enhanced. Currently, only resetting the AAO is implemented. Changing/replacing the AAO or editing the AAO are the obvious extensions here. More advanced features could combine different AAO by merging them. Concepts from eHornets could also potentially be applied (see Section 12.6.1).

The remaining limitations discussed before are mostly concerned with improving usability and modelling. Providing automatic task connections for workflow operations while modelling and the mutually exclusive task choice fall into this category.

Dynamic progress representation in GUI: Currently, the only indication of the progress a human user has are the lists of available workitems, current activities and done activities. A dynamic, visual representation of the progress within a process is a highly desirable feature.

For the existing PAOSE teaching support system from Section 11.3.3 a static, hardcoded overview of the process-protocol for each exercise was displayed. The envisioned dynamic representation works similarly. Since the Petri net of a process-protocol or the internal process of an AGENT-ACTIVITY is already a reasonable and clearly structured graphical representation of the process the net can be used as a basis. Displaying the net context for a workitem or activity is not a large implementation issue. Task information could be extended with a file location for the net, which could then be read and displayed by the GUI. More difficult to implement is how to indicate the progress dynamically. For that, a simple image representation of the net is insufficient. Rather the net structure has to be generated as individual elements in the GUI so that the already finished transitions (AGENT-ACTIVITY-TRANSITIONS when the net is a process-protocol) are marked somehow. The generation could utilise the existing list of done activities by matching them to the transitions.

Beyond that interactivity for that net representation is also planned. Each element of the net should then be inspectable. This could be used as a logging and monitoring tool, simply displaying information about the parameters and results involved in the selected elements. However, it could also be used as a novel control mechanism. Instead of using classical worklists the users could select and request their available workitems from the net representation. Instead of lists of current activities the users would see what was assigned to/requested by them and inspect it to reveal the task functionality. Such a net representation of the control could be more intuitive and potentially guide users by providing more insight and overview over the extended context of the workitems and activities they are working on.

Monitoring tools: Current monitoring tools in RENEW only capture agent aspects. Especially the MULANVIEWER is an essential tool for monitoring a multi-agent system. It displays all active platforms and agents on the local machine and allows inspection of all net schemas and instances. Through the nets-within-nets principles, the system is displayed in a hierarchical way. It is, however, specifically tailored to MULAN and CAPA, meaning that anything that is not part of that framework is not displayed and can't be easily accessed. The elements that should be added to the overview of the new VIEWER are listed in the following:

- The technical backend should be displayed as a standard component on the same hierarchy level as the knowledge base and process-protocol factory. The entry should expendable to grant direct access to the three subordinate backend nets for workflow engine, resource and agent functionalities.
- The AAO nets or internal processes should be used to represent the AGENT-ACTIVITIES as subordinate hierarchy entries to the process-protocols.
- For each PAFFIN as an engine the lists of workitems and activities should be accessible on the same level as the process-protocols to better capture the workflow aspects.
- Workflow resources registered by a PAFFIN entity should be displayed. This could be done in the same entry as the knowledge base or in a separate entry on the same level.

These extensions would enable the new Viewer to capture all of the technical artefacts used during the execution of a Paffin-System application.

Another tool that should be extended is the MulanSniffer, which logs messages exchanged between Capa agents. Here, the changes are more conceptual in nature. Currently, only agent messages are logged. For the Paffin-System, the workflow operations should also be captured. Each task is provided by an engine and executed by a resource, both are Paffin entities. The information about the workflow operations is already available in the agent messages exchanged between the entities and the platform WFMS. However, multiple messages correspond to a single workflow operations. Therefore, it is desirable to filter these messages and display them as a single task between engine and resource entities. Open questions here relate to the optimal common representation of agent messages and workflow tasks. The MulanSniffer creates a diagram of messages similar to an AIP. It has to be examined if that diagram structure is still reasonable when including workflow tasks. Rough sketches of what an AIP with workflow elements might look like have been discussed before in the context of an extended AIP tool for Paffins in Section 12.6.3.

New tools for the Paffin-System could also be created. Feasible are tools to capture exclusively the workflow data exchanged with human users or tools to better monitor performance in the system.

Collaboration extensions: A focus in many of the applications and application visions discussed in Chapter 11 is the collaboration between the human workflow users. In order to better support the collaboration a number of extensions to the framework are feasible:

- A tool for remote monitoring of workflows would enable users to get an overview over how far other users had progressed in their workflows. This tool would be especially useful in a setting where supervisors would want to or need to check on other users. Such a setting, for example, is the Paose teaching environment. Here, the teacher could check on the progress of the students. If all students were stuck on a particular exercise the teacher could initiate a plenum session where he or she explained the context and possible solutions. If only individual students had problems, the teacher could approach them and provide help directly. Of course, the system would have to be protected in some way to ensure that only authorised users can access the workflows of other users. If privacy is a concern the tool could be implemented to provide only general, anonymous statistics, e.g. indicating the percentage of users that have progressed to a specific task.
- A chat interface could be implemented to allow all registered users to chat and exchange tips and hints. That interface could be added to the standard web GUI as an additional area.
- A knowledge repository could be provided on a Paffin platform. That repository could contain hints, frequently asked questions, a wiki and even provide access to system monitors, logs and statistics. Similarly to the chat interface, a knowledge repository could be used to provide answers quickly and directly. Optimally, the repository should grow naturally over time. In the teaching environment example a knowledge repository could contain hints to the current exercises, sample solutions to previous exercises, general questions and how-tos for the environment and framework, etc.

These kinds of extensions would, when implemented, emphasise collaboration aspects, which would benefit many of the targeted application domains.

Persistent state storing: The ability to suspend (parts of) a system, store their state and then, at a later time, resume the execution is a highly desirable feature. In the Paffin-System, the principle could be applied to different levels. A single Paffin entity could store the state of its process-protocols or even just its Agent-Activities. That way, longer, ongoing processes could be better supported. It is also feasible for the Paffin to suspend the execution, store the state and then have another Paffin resume the execution. This could be helpful in situations where local resources become unavailable and the behaviour needs to be moved to where resources are available (see Section 11.4.2). Beyond single process-protocols, entire Paffin entities or platforms could be suspended and resumed later or elsewhere.

For an implementation a number of things would need to be provided. First and foremost, a persistent data storage would need to be made available to the entirety of the system. Here, the PersistentOntology functionality could be further extended, which was already discussed above.

Second, an encoding of a net instance state would have to be developed. That encoding needs to be generated from an arbitrary reference net, have a form that can be stored persistently and finally be read and instantiated in a new net instance. Neither of these points is easy to implement.

A reference net can have arbitrary Java objects as references. These need to be encoded and stored as well. Reference can also include other reference nets, making the encoding recursive. Potentially, the state of a net encoding can be highly complex and extensive. Therefore, an efficient encoding must be developed to store it. Finally, the net and all of the objects need to be instantiated in the new net. All of the relations between the instantiated objects need to be intact, even those beyond the relations introduced by the reference net structure.

Third, the functionality must be incorporated into the Paffin-System framework. Here, design decisions must be made about which nets to allow suspension and resumption of. Open questions involve where in the system the suspend/resume functionality is controlled. Most reasonably, the platform, which is responsible for the entity life-cycle should control it. This means that the functionality to resume previous entities should be implemented here. However, entities, presumably as part of the backend functionality, need to be able to initiate or request that functionality.

All in all, suspending, storing and resuming the state of a net instance is a difficult and extensive endeavour. In fact, it is desirable as a general Renew and reference net feature and not just limited to the Paffin-System. It has often been approached, but no viable technical solution is available yet.

12.7 Concluding Evaluation Concerning the Requirements

To conclude this chapter, this section considers the overall goal, research questions and hypothesis posed in the introduction. This evaluates the results achieved in this thesis in a concluding matter.

Section 7.4 defined a number of partial results that were required of this thesis. The following listing discusses how these results have been achieved.

A specification of the integration: The specification of an integration was presented as the vision of an integration, as well as the integration criteria derived from that vision, in Sections 7.1 and 7.3. Both the vision and the criteria were used throughout the thesis to evaluate and compare intermediate results with the required specification. Most recently, Section 12.1 related the integration vision to the overall and finalised practical results.

A conceptual integration approach: The Agent-Activity integration approach presented in Chapter 9 constitutes the required result for the conceptual model and approach for the integration and combination of structure and behaviour through agents and workflows. It was selected from the abstract integration approaches presented in Chapter 8 based on the integration criteria. Then, it was further refined into a concrete, conceptual integration approach and again evaluated against the integration criteria in order to ensure its compliance with the integration vision.

A technical proof-of-concept for the conceptual approach: The Paffin-System prototype framework, described in Section 10.2, constitutes the working proof-of-concept for the Agent-Activity integration approach and therefore the proof-of-concept for the conceptual model and approach as prescribed by this required result. It was evaluated against both the Agent-Activity integration approach and the general integration criteria in Section 10.4, in order to ensure compliance with both.

A set of application prototypes using the integration: Section 11.3 presented three application prototypes implemented in the Paffin-System framework. Each of these prototypes is fully functional, thus proving the usability of the Paffin-System, and also showcases important and relevant integration mechanisms. Additionally, Section 11.4 presented a number of application visions that, while not yet implemented, are possible in the Paffin-System framework and further demonstrate its capabilities.

Clearly, the required partial results have all been provided. The conditions set out for these results in Section 7.4 have all been fulfilled and in some cases even exceeded. The Paffin-System, for example, has implemented features that go beyond the basic status required for a proof-of-concept.

Next, Section 7.4 provided a list of supplementary requirements to the results. These are aimed at clarifying points and aspects towards the goal of the thesis and the related research questions. The following listing addresses the requirements. Note that these requirements overlap in part with the previously established results.

Alignment with the integration vision: The alignment between the achieved concepts/systems and the integration vision was evaluated and discussed on multiple occasions. Section 8.1.4 examined the alignment with the abstract integration approach using agent actions and workflow operations. The positive result of that evaluation ultimately led to the selection of that integration approach for further refinement and development. Section 9.3 then examined the alignment of the refined Agent-Activity integration approach, substantiating that the concrete approach evaluated even slightly better than the more abstract version. Section 10.4 examined the alignment of the Paffin-System with both the integration criteria and the Agent-Activity integration approach. Again, the result was positive by demonstrating a good alignment. Finally, Section 12.1 considered the overall results, including Agent-Activities and the Paffin-System, in a concluding matter and evaluated them against the vision of an integration on a more abstract and general level. That evaluation also yielded a positive result. Overall, this requirement is fully satisfied, as all stages of development of an integration, i.e. abstract

approach, concrete approach and proof-of-concept prototype, all align well with the integration vision and the integration criteria.

Technical feasibility of the developed concepts: The technical feasibility of the developed concepts, the Agent-Activity integration approach, was established through its proof-of-concept, the Paffin-System. The Paffin-System was presented in Section 10.2, while its alignment with and proof-of-concept status were examined in Section 10.4.1. Descriptions in the latter section have shown that the Paffin-System is indeed a proof-of-concept of the Agent-Activity approach, thereby satisfying this current requirement of the thesis.

Capability to build applications utilising the integration: The creation of applications with the Paffin-System was the main topic of Chapter 11. A number of sections from that chapter stand out for this current requirement. Section 11.1 discussed how modelling is done in the Paffin-System, while Section 11.2 examined its features and limitations for application modelling. Section 11.3 presented the three application prototypes that were practically implemented in the Paffin-System. These confirm that this current requirement is indeed satisfied as the applications were implemented in the Paffin-System and are fully functional.

Transferability: Section 12.2 examined the generality of the results and their applicability beyond the current Petri net emphasis. In summary, while the results focus on the Petri net basis, the general concepts and ideas can be transferred to wider contexts. However, the farther removed from the current focus an area is, the more difficult it is to directly transfer the results. In such cases adaptions become necessary. Still, the discussions have shown that this requirement of the thesis is satisfied.

Synergies: The extensive amount of synergies has been discussed in Section 12.3. Overall, the provision and enabling of beneficial synergies between agents and workflows is just one of the strengths of the Agent-Activity integration approach, as discussed in Section 12.4. Synergies are present on all system levels and yield various benefits for system modelling and management. Consequently, this requirement of the thesis is satisfied.

Improvement and conceptual Maturity: The strengths of the Agent-Activity approach have been discussed in Section 12.4. That section showed the extensive strengths that an integration of agents and workflows offers on the conceptual level. One of the most prominent strengths related to enabling the synergies between agents and workflows, as discussed in the previous requirement and Section 12.3. Some open points also remain, though most refer to the context of the practical Paffin-System, thus not affecting this current requirement. Only the increased modelling complexity stands out as a more critical open point. The increased complexity, however, can be considered as the cost required to enable the strengths and synergies. There are options that address and support the modelling complexity in future work.

Altogether, though, the results have showcased a satisfactory and solid conceptual maturity. That maturity is picked up again in Chapter 13, where the results of this thesis are related to the work presented as part of the state-of-the-art in Chapter 3. Chapter 13 shows that, conceptually, the Agent-Activity approach represents an extensive and valuable improvement in many areas. Especially the fact that the entirety of agent and workflow mechanisms and properties is available for system modelling stands out positively. However, the *technical* maturity of the Paffin-System prototype often prevents a direct comparison to other more established and especially commercial

tools. Still, this thesis emphasised the concept over the practical implementation. Therefore, this requirement is completely satisfied.

Inter-organisational contexts and other application areas: Application areas of the integration were generally discussed in Section 7.2, which also touched on inter-organisational contexts. More details and the connection to the Agent-Activity approach and the Paffin-System were provided in Section 11.5, which explicitly examined the application of Agent-Activities to inter-organisational contexts. Moreover, Section 12.5 examined the properties of general application areas and settings which could beneficently utilise the integration provided by Agent-Activities. Strongly related is also Section 11.4, which provided application visions that are not yet implemented with the Paffin-System framework. Altogether, these sections show that the Agent-Activity integration approach and its implementation in the Paffin-System are indeed quite valuable for many application areas. Benefits, strengths and synergies can be utilised in various scenarios. Inter-organisational contexts, in particular, possess requirements that the Agent-Activity with its duality of structure and behaviour can satisfy very well. Therefore, this requirement of the thesis is also satisfied.

The previous listing shows that all of the requirements set out for this thesis have been satisfied. Before concluding this section, the following presents all of the specific achieved results. These mostly overlap with the required partial results discussed before, but some represent intermediate results that supported the explicitly required ones or their evaluation. In order to provide a complete overview of what this thesis achieved, all relevant results of this thesis should be recognised here:

- The *agent-oriented modelling perspective* from Chapter 5 defined the aspect of structure of a software system for this thesis and introduced the key concept of fundamental agent actions.
- The *workflow-based modelling perspective* from Chapter 6 defined the aspect of behaviour of a software system for this thesis and introduced the key concept of basic workflow operations.
- The *vision of an integration* from Chapter 7 provided the specification and concrete criteria for the integration desired for this thesis.
- The *abstract integration approaches* from Chapter 8 presented different ways of how to generally achieve the desired integration.
- The Agent-Activity *integration approach*, the conceptual main result, from Chapter 9 described how to concretely realise an integration of agents and workflows in this thesis. It represents the conceptual model and approach desired in the goal of this thesis.
- The Paffin-*System prototype* from Chapter 10 is the technical main result of this thesis and provides a working proof-of-concept for the Agent-Activity integration approach. Furthermore it serves as the framework in which all application prototypes for this thesis are built.
- The *additional prototypes* also presented in Chapter 10 represent intermediate practical results that influenced the creation of the Paffin-System.
- The *application prototypes* presented in Chapter 11 represent fully functional applications developed in the Paffin-System framework in order to showcase its own functionality and provide a concrete basis for application discussions.

– The *evaluation of the* Agent-Activity *approach and* Paffin-*System* is the final result of this thesis. It encompasses the discussions and observations from Chapters 11, 12 and 13.

To conclude this overall evaluation the original research questions from the introduction are concretely answered before the goal is discussed. The recapitulate the research questions:

1. "How can the concepts "agent" and "workflow" be combined and integrated in a reasonable, conducive and beneficial way?"
2. "Which beneficial effects can be achieved through a combination and integration of agents and workflows, potentially and substantiated through a technical proof-of-concept, and in which scenarios are they best applicable?"

As discussed in the introduction, the first question required a number of consecutive steps to answer. First, the exact extent and definition of the terms structure through agents and behaviour through workflows needed to be developed. This was done in Chapters 4, 5 and 6. Second, the specification of the integration was required, which was provided as the integration vision and integration criteria in Chapter 7. Third, different abstract integration approaches were developed, presented and evaluated in Chapter 8. From those, the Agent-Activity integration approach was selected and finally refined in Chapter 9 into the concrete model and approach for an integration of agents and workflows, which answers the first research question.

To summarise the answer to the first research question:

– An integration must feature a mutual incorporation of agents and workflows, as well as integrated entities (see Definition B.7) that can act as and be interacted with as agents, workflows, both or something in between.

– Based on the defined integration criteria, the concepts agents and workflows can best be combined and integrated using an approach based on fundamental agent actions (see Definition B.1) and basic workflow operations (see Definition B.4).

– In order to realise not only a combination of agents and workflows, but also an integration with form and function, an additional abstraction of agent actions and workflow operations is required. That abstraction is provided by the Agent-Activity concept (see Definition C.2) in the Agent-Activity integration approach (see Definition C.1).

– To support the Agent-Activity integration approach a system must implement the Agent-Activity reference architecture[15] (see Definition C.7) including the full management functionality.

The second question was subsequently answered, with the basis being provided by the implementation of the Agent-Activity integration approach in the Paffin-System in Chapter 10 and by the application prototypes and visions in Chapter 11. On this basis and the conceptual Agent-Activity approach, discussions, evaluations and observations were performed in Chapters 11 and 12. Especially Sections 12.3 and 12.4 elaborated upon the benefits, strengths and synergies created and/or enabled by the results of this thesis. The scenarios in which best to apply these results were discussed especially in Sections 11.5 and 12.5.

[15] Note that the term reference architecture does not refer to reference Petri nets. The implementation of the Paffin-System using reference Petri nets is just one possible implementation of the reference architecture.

To summarise the answer to the second research question:

- There are a number of main beneficial synergies available through the integration of agents and workflows. It is possible to freely combine agent and workflow functionality. Integrated entities can also encapsulate any agent or workflow concepts. All feasible interactions between integrated entities in any state are also supported. Furthermore, integrated entities provide a unified global model that can be utilised to generate specific structural or behavioural perspectives. These main synergies are the basis for many of the general strengths the integration can provide.
- The synergies were substantiated by the Paffin-System (see Definition C.10) and the implemented application prototypes.
- Application scenarios for the Agent-Activity integration approach are diverse. The approach can address various requirements of large and complex systems featuring either a strong emphasis on structure, behaviour or the duality of the two. Inter-organisational contexts are a prime example of application scenarios in which the integration can be applied well.

Having answered the research questions it is now possible to consider the overall goal of this thesis. Repeating the goal from the introduction: "In this thesis the two concepts agents and workflows are combined and integrated in order to create the hybrid system. The research, development and provision of this combination and integration are the overarching *goal* of the thesis. On a conceptual level this goal is constituted by a conceptual model and approach for the motivated combination and integration. This model and approach are supplemented by a set of prototypes constituting a working proof-of-concept realisation and implementation. This proof-of-concept is the goal of this thesis on the practical level."

Overall, it is clear that the goal has been fully achieved. The conceptual model and approach is constituted by the Agent-Activity integration approach. The working proof-of-concept of that model and approach is provided by the Paffin-System prototype framework. Both have been extensively examined, discussed and evaluated. The combination and integration of agents and workflows in a hybrid system realised by the Agent-Activity approach and Paffin-System was therefore successfully researched, developed and provided in detail.

Throughout the entirety of this thesis, reference Petri nets were used to represent and model the different concepts and systems. The Agent-Activity integration approach therefore also provides a concrete and executable Petri net model for a full integration and hybrid of agents and workflows through its implementation in the Paffin-System. Such a Petri net model was not available prior to this thesis and represents a clear contribution in the context of Petri nets that is in line with the proposals of Petri net models for objects [Moldt, 1996], agents [Moldt and Wienberg, 1997, Rölke, 2004], and workflows [van der Aalst et al., 1994].

Finally, it is possible to consider the original hypothesis proposed in the introduction: "The hypothesis of the thesis is that two individual modelling techniques and their related perspectives can be united to provide the strengths of both, as well as additional beneficial synergies, by integrating and combining their main modelling constructs. More specifically, the two orthogonal aspects of structure and behaviour of a software system are combined via agents and workflows in order to create an enhanced modelling perspective."

This hypothesis has been confirmed through the results of the thesis. The two orthogonal modelling techniques agents and workflows, as well as their related perspectives, have been united. Abstractly, this was done overall in Part B, with especially Chapter 7 describing

the enhanced and extended integrated modelling perspective. Part C then described how to concretely realise the integration and combination of the two modelling constructs agent and workflow via the Agent-Activity integration approach and the Paffin-System prototype. Finally, Part D evaluated and discussed the integration. As was discussed in detail, integrated entities featuring Agent-Activities can act as and be interacted with as agents, workflows, both or something in between. Therefore, they can fully utilise the mechanisms, properties and consequently strengths of both agents and workflows at the same time. Furthermore, as discussed especially in Section 12.3, there are numerous beneficial synergies created by the integration that are all now available in addition to the traditional agent and workflow strengths.

In conclusion, all requirements defined for this thesis have been met. The result requirements have been fulfilled, the research questions answered, the goal achieved and the hypothesis confirmed. This thesis intended to research and examine an integration of structure and behaviour via agents and workflows, which has been brought to a successful conclusion through the development, implementation and evaluation of the Agent-Activity integration approach.

13 Related Work

Chapter 3 presented the state-of-the-art from related research contexts. Now, this related work is picked up again and used as a basis to comparatively evaluate both the conceptual Agent-Activity approach, as well as the Paffin-System. The result of this chapter is an evaluation that puts the capabilities and properties of the achieved models and systems into context with what other researchers have achieved.

This chapter's structure is similar to the one from Chapter 3. Sections 13.1 and 13.2 compare the results of this thesis to traditional agent-orientation and workflow management respectively. Section 13.3 then compares previous integration efforts. Finally, Section 13.4 concludes this chapter with a short, overarching discussion.

13.1 Comparison to Traditional Agent-Orientation

The Agent-Activity integration approach represents a novel way of managing integrated entities that can be, among other things, agents. Cognitive concepts, such as beliefs, desired and intentions, are not featured in the approach, but, as discussed in Section 12.2, it is nonetheless generally applicable and transferable to more than just the Mulan agent model. To shortly summarise, the concepts are overall applicable, yet different characterisations of the fundamental agent actions might be beneficial. An open question also revolves around the integration basis, given in this thesis by the reference Petri nets. Without a shared basis to build both agent and workflow aspects, the realisation of an integration is impeded.

Cognitive concepts are not supported by the Paffin-System. There are conceptual, unpublished prototypes for Capa attempting to include external BDI mechanisms, yet these are not fully functional or mature. Therefore, a comparison of the practical Paffin-System prototype to BDI architectures is difficult. An incorporation of cognitive concepts is technically feasible, yet outside of the scope of this thesis.

When compared to other agent management systems, the Paffin-System prototype, while promising, also lacks technical maturity. As described in Section 11.2, the prototype offers a good feature set and goes beyond its intention as a proof-of-concept. Yet, it has still only been in development for a relatively short time. Many agent frameworks and platforms, like Jadex, MaDKit, Jason, etc., have been in development for over a decade. The agent mechanisms of the Paffin-System may benefit from the maturity of the Capa basis, which has a similar maturity compared to other established agent frameworks. Still, the new mechanisms, above all the technical backend, are prototypes. They fulfil their function and provide a good basis for future extensions, but, as discussed in Section 12.4, they still lack in aspects such as performance or security. However, research and development of the Paffin-System has only just begun, with follow-up theses already planned. The issues troubling the system when compared to established and/or commercial agent platforms can be addressed. Performance can be optimised, security features added, and so on. There is nothing that fundamentally keeps the Paffin-System from being used as a robust and efficient agent platform for large-scale applications. The formal Petri net basis even enables the potential to provide the means for verification and validation, as discussed in Section 12.6.2.

Beyond that, the benefits and extended capabilities of the workflow and integration mechanisms are, of course, also available. These extended capabilities are what the Agent-Activity integration approach and Paffin-System were designed and developed for. Being able to imbue agents with workflow concepts (and vice versa) is a clear advantage over traditional agent-oriented modelling. This point is discussed in more detail throughout this chapter.

13.2 Comparison to Traditional Workflows

Chapter 3 distinguished research in workflow modelling and workflow management. This distinction is retained here.

Workflow Modelling The Agent-Activity integration approach is considered with reference Petri nets in mind. While Section 12.2 examined the generality of the approach, the main results of this thesis have been presented before that background. Therefore, the workflow modelling expressiveness and power of the Agent-Activity concept also need to be considered under those terms, especially since any practical examples provided in the Paffin-System have, in fact, been modelled with reference nets.

For workflow modelling there are two relevant areas: the process-protocols and the internal processes of the Agent-Activities. On these two levels it is possible to fully exploit the expressiveness of reference nets. The process-protocol can describe the abstract overall process, while the Agent-Activities provide the detailed descriptions of the abstract tasks of the overall process.

When comparing the modelling with process-protocols to workflow nets, it is clear that the Agent-Activities can be used to represent any workflow net processes. Workflow nets consist of tasks connected by Petri net elements. Process-protocols consist of Agent-Activities connected by Petri net elements. Modelling each Agent-Activity to contain exactly one workflow task (i.e. one each of the three fundamental workflow operations) essentially equalises a process-protocol with a workflow net containing the same structure of tasks. It would also be possible to have any workflow net modelled within the internal process of an Agent-Activity. Each task would correlate to a triplet of fundamental workflow operations. Overall, this means that workflow nets can be captured in the Agent-Activity approach on both the process-protocol and Agent-Activity level. The same is valid for the Paffin-System, with the exception of the technical and administrative start and end components of the process-protocol and AAO, which are added to the basic workflow net content.

While it is possible to model any workflow net in the Agent-Activity integration approach, there are restrictions to workflow nets which are not enforced. As described in Definition 2.19, a workflow net has a specific structure with a source place, a sink place and strong connectivity when these are connected with a transition. No such restrictions are in any way enforced in the Agent-Activity. In the Paffin-System, the administrative start and stop components of process-protocols and Agent-Activities are similar to source and sink places, but they are also not expressly prescribed. Continuous behaviour modelled as process-protocols or Agent-Activities would, for example, not feature an end component and therefore no sink place equivalent.

On the one hand, this missing restriction means that it is possible to model processes in the Agent-Activity approach that would not be considered as workflow nets. Processes with multiple starting- and end-points are possible, for example. Any process that can be captured in a reference net can be captured in a process-protocol or an Agent-Activity.

This expressiveness is enhanced even more when considering that Agent-Activities can contain more than just triplets workflow operations. It is possible to nest workflow tasks by nesting the operations and prescribing specific finalisation orders of the nested tasks. While the idea of subworkflows being represented as tasks in workflow nets is similar, here it is more directly available. In addition to structuring, combining and nesting workflow operations, process-protocols can also add agent actions, which realise functionality beyond the scope of workflow modelling. This however, is picked up again later in these discussions.

On the other hand, without the restrictions imposed on workflow nets it is also impossible to apply any results concerning workflow soundness and verification for process-protocols and Agent-Activities. Currently, no application of verification mechanisms is possible, but it is feasible to restrict process-protocol and Agent-Activity modelling to the structure of workflow nets (see discussion in Section 12.6.2). By doing so it would conceptually equalise the modelling capabilities of workflow nets and the Agent-Activity approach.

However, workflow nets do not support all workflow patterns. This is why, for example, the YAWL modelling language started on a basis of Petri nets and built upon that to capture advanced patterns, such as non-local cancellation and synchronisation. In the Agent-Activity integration approach, support for such patterns can come from the agent functionality. A workflow modelled as an integrated (Paffin) entity can exhibit complex functionality and agent-based intelligence to manage synchronisations between different process-protocols or Agent-Activities, possibly even in different entities. Non-local cancellations are currently not possible in the Paffin-System, as discussed in Section 11.3.3 in the context of enduring activities, but using agent functionality and the Paffin knowledge base to solve that challenge is feasible. For example, a Paffin may record the state of a process-protocol in its knowledge base and reset to it when necessary. On the Agent-Activity level an update and replacement of the AAO is already drafted.

Regarding other modelling formalisms, like BPMN, the Agent-Activity and Paffin-System yield similar comparisons. Due to the expressiveness of the reference (Petri) net basis, most workflow patterns are directly supported, as well as freely formable processes not adhering to workflow structures. Advanced patterns that can't be directly mapped onto Petri nets can be supported by agent functionality and intelligence. The workflow modelling capabilities of the Agent-Activity approach are therefore in no way restricted and are comparable to the standard and state-of-the-art in workflow and process modelling.

Workflow Management The Agent-Activity integration approach represents a novel kind of management system. Instead of just managing workflows, it manages integrated entities which can be agents, workflows or both. Workflow management is therefore a subset of the management of an Agent-Activity reference architecture system. Workflows in the Agent-Activity approach can exhibit agent and hybrid properties, mechanisms and functionalities, in addition to the traditional workflow ones.

For this thesis and the Agent-Activity integration approach, workflows are defined in Definition A.2 for general processes. Business processes, as a specialisation, can of course be modelled, but workflows in Agent-Activities are not restricted to this application context.

This generalisation regarding processes is similar to the definition of a process-aware information system (PAIS). PAIS manage and execute operational processes [Dumas et al., 2005, p. 7]. While (traditional) workflow and business process management systems are a subset of PAIS, PAIS go beyond those. A system following the Agent-Activity approach does the same.

PAIS distinguish themselves by emphasising the process within the system. While the Agent-Activity approach emphasises the integrated entity, that entity is equivalent to the process it executes. This means that the Agent-Activity approach, too, emphasises the process. The fact that it also emphasises the structure is part of, and actually key to, the underlying integration of agents (structure) and workflows (behaviour). When considering examples of PAIS, such as collaboration tools, project management or issue tracking, these kinds of systems can be nicely captured by Agent-Activities. The Paose support environment vision described in Section 11.4.5 is a good example of all three of those kinds of systems. One of the key goals of that environment is to support the collaboration and interaction between the different development teams. Project management and issue tracking is part of the support of that collaboration. In fact, the Paose teaching support prototype from Section 11.3.3 already supports some project management functionality with the incorporation of the RedTimer tool into the teaching workflows.

All in all, considering a system following the Agent-Activity integration approach as a PAIS is justified. Processes are emphasised and workflow management is fully encompassed by Agent-Activities, with additional capabilities go beyond it.

An important thing to note here is that the Agent-Activity approach is not *just* a PAIS. Agent-Activities feature the additional perspectives provided by agents and the integration. PAIS manage and execute operational processes and that is one of the things that Agent-Activities can do. Agent-Activities, however, can also represent and be active components that utilise process and workflow parts, but are only secondarily concerned with the operational processes. The different application prototypes and visions of Chapter 11 have illustrated how the structural and behavioural elements can be utilised advantageously by combining and integrating them in different ways. Structural elements are secondary in PAIS and an integration, even with the different abstraction and emphases levels, is not contemplated. The capabilities of the Agent-Activity approach therefore go beyond those of PAIS.

These extended capabilities of the Agent-Activity approach represent the main added research value. As described in Section 3.2.2, current research in PAIS focuses on, for example, flexibility, security and compliance. The structural aspects of the Agent-Activity approach can be utilised to provide solutions in these areas. Using Agent-Activities, agent intelligence in integrated entities may be used to manage flexibility and enforce compliance. Agent properties like autonomy and the increased encapsulation of the processes (see the discussion about inter-organisational context applications in Section 11.5) can be used for security research. On the other hand, concepts researched for PAIS may be used in Agent-Activities. Employing change patterns to schemas and instances as proposed in [Weber et al., 2013], for example, may be incorporated into the workflow aspects of Agent-Activities. Overall, both PAIS and the Agent-Activity concept can benefit from research conducted in the respective other field.

Note that the same holds true for research in classical BPM. Results from research on flexibility, resource allocation and analysis can be used to extend the approach and the implementation in the Paffin-System in many ways. Moving the execution to the cloud or utilising cloud resources within the Agent-Activities or the Paffin-System are also feasible. Simulation of processes is also supported, due to the Petri net basis of the approach and implementation.

The extended nature of the Agent-Activity approach also enables another perspective in the context of workflow management. An integrated entity can also be seen as an agent. That agent can represent an actor participating in the workflows of the system. However, it is just as feasible to consider the agent as an active and autonomous representation of other

elements of workflows. One possibility is to have the integrated entity represent one or more cases of a workflow. The integrated entity could hold all data, (intermediate) results, basic processing etc. in its knowledge and interact with other integrated entities as workflows and agents to fully realise and complete its own execution. This would shift the perspective and therefore the emphasis of the execution towards the case concept, supporting the consideration of the Paffin-System as an adaptive case management (ACM) system. Of course, ACM usually strongly features flexibility and adaptivity, which are not explicitly part of the Agent-Activity approach, but are also not excluded. Flexibility features fall into the responsibilities of the implementation. The Paffin-System currently only features flexibility in the form of an AAO reset and variable Agent-Activity execution parameters (see discussion in Section 11.2.1). Adding flexibility in some other way is technically feasible, making the Paffin-System, with the case emphasis as described above, a viable ACM system.

Other related work considered in Section 3.2.2 was concerned with more practical issues. The only, currently available implementation of the Agent-Activity integration approach is the Paffin-System. As discussed in Section 11.2.1, the Paffin-System is a promising prototype and proof-of-concept, but is still limited in some of its practical features. Therefore, it can't compare to commercially available or other established workflow and business process management systems.

However, achieving a commercial level of maturity was never the intention of the Paffin-System. The intention of the Paffin-System in this thesis was to prove that an integration of agents and workflows, as proposed in the Agent-Activity integration approach, could actually be implemented and work as intended. It was to showcase a practically implemented integration and present ways of how to utilise such an integration in the future. This intention was completely fulfilled.

While the *practical* implementation currently can't compare to commercial systems, the *conceptual* integration provides many potential benefits (see the discussion in Section 12.4). Commercial and established systems may have more complete feature sets, but the conceptual strengths introduced by the Agent-Activity approach and validated by the Paffin-System can enrich the field of BPM in general.

13.3 Comparison to other Integration Research

While the previous two subsections related Agent-Activities and the Paffin-System to traditional agents and workflows, this current section now considers research featuring an integration of the two. As Section 3.3 showed, there has already been a lot of work trying to couple agents and workflows. In fact, agent-based workflow management can even be considered an established and extensive research field.

This section is structured as follows. Sections 13.3.1 and 13.3.2 consider agent-based workflow management and workflow-based agent management respectively. Section 13.3.3, then, compares the Agent-Activity approach to advanced integration efforts.

13.3.1 Agent-based Workflow Management

In agent-based workflow management the agent concept and related mechanisms are exploited to enhance or support workflow management with various benefits. To the system modellers and users the system created in agent-based workflow management is still only a workflow system, albeit with the extended capabilities of agents often directly influencing the execution. The following mirrors the order of interfaces and functions

according to the survey in [Delias et al., 2012] that was also used in Section 3.3.1. For each interface and function realisation examples and options in the Agent-Activity integration approach and Paffin-System are discussed. This shows that all functions, which agents can have for workflow management as presented in [Delias et al., 2012], can be realised with Agent-Activities.

Process Definition Tools The process definitions tools interface handles all functionalities relating directly to the workflow definitions/schemas. For Agent-Activities and the Paffin-System the workflow definitions are the process-protocol nets including the Agent-Activities and their internal processes. Arbitrary agent functionality (actions) can be incorporated into these nets.

Analyse, model, compose, describe and document a BP: First and foremost, the ability of integrated (Paffin) entities to encapsulate their workflows is important for the modelling aspect of this function. As stated before, the entity *is* the workflow it executes, both conceptually and technically. Through that encapsulation and representation it becomes possible to translate any agent or hybrid property to the workflows, which is something often focussed on in agent-based workflow management. This can, for example, generally refer to agent intelligence (e.g. the workflow deliberating over which available resources are best for its execution), agent communication (e.g. workflows being executed on distributed notes) or agent autonomy (e.g. refusing activity confirmations until results have reached a threshold quality).

In a similar way, composition can be realised in the Agent-Activity approach. The agent functionality inside integrated entities can be used to compose partial workflows into a larger orchestration. Controlling and managing the instantiation of the different involved nets is a simple matter for, e.g., an agent decision component.

Considering the other capacities referred to in the title of this function, analysis, description and documentation of a process definition are not quite as directly coverable. However, it is feasible to use integrated entities in a sort of simulation mode. Each entity is both workflow engine and resource, meaning that conceptually the platform WFMS can be ignored for certain purposes. In that case the entity would serve as workflow engine and automatically assign/request and confirm its own tasks as a resource. That way it could be used to analyse and test a workflow and describe or document it and the results.

Process definition write/edit: For each Agent-Activity the AAO is prescribed through the inscription. That part of the inscription can be realised with a variable that can be used to exchange internal behaviour at runtime. This means that an integrated entity executing Agent-Activities always has this form of control over its internal process definitions by controlling the variables determining the AAOs during execution. Beyond that, assuming some form process-protocol repository, the integrated entities could be designed to retrieve any process-protocol necessary to their current execution. Integrated entities building/adapting their process-protocols is also feasible, requiring a way to formalise process requirements and generating/changing a net structure accordingly.

Definition Retrieval: Currently, the definitions for the process-protocols are stored as Renew net files in the file system and are accessed via the net path environment variable of Renew. This can, however, be changed to have the net definitions stored in a persistent database and have an integrated entity, as an agent, retrieve the

definitions and then instantiate that definition at runtime. In fact, this has already been done in a previous WFMS of Capa [Wagner, 2009b], where a special agent had access to the workflow definition database and distributed the definitions to the workflow engines. Since then, the Capa persistence has been further developed with the `PersistentOntology` plugin [Mosteller, 2016], which is already used in the Paffin-System prototype. Realising entity access to persistent data storage, including process definitions, is therefore generally possible to implement.

Client Applications The client applications represent those parts of the system that, directly or indirectly, interact with the human users. For the Agent-Activity integration approach no predefined assumptions about the user interface exist, only that it is somehow technically supplied. For the Paffin-System the user interface is provided by the PaffinWebGui plugin (see Section 10.2.12). To simplify these discussion they assume the already existing and implemented user interface as a discussion basis.

Worklist Handling: Worklists in the Paffin-System are already handled by Paffin entities representing users. They register, on behalf of their users, with the platform WFMS, receive information about workitems and activities and forward that information to the GUI. Here, again, agent intelligence can be used to further improve functionality. For example, the Paffin may filter or organise certain workitems based on history or user preferences before displaying them. That way the system could influence or possibly even improve user efficiency by, e.g., encouraging users to work on workitems they are experienced in.

Process Control: The integrated entities already serve as the workflow engines for the workflows they encapsulate. This means that they already start, stop, suspend, resume and generally control the process executions. In the Paffin-System this is also already implemented. Extending this functionality may yield multiple advantages. The user interface can be extended to provide administrative access for authorised users to initiate operations on processes. It is also feasible to have the (Paffin) entities analyse the dynamic runtime circumstances and react with certain operations on processes. For example, before an imminent (scheduled) shutdown of the system the entities might suspend and save their processes and afterwards automatically resume them.

Data Handling: Integrated entities as agents automatically manage data handling aspects. As workflow engines they supply the parameters, knowledge and other data to the execution of their processes (which are process-protocols). They receive data from the outside, pass it to the process-protocols for further use and also send out data on behalf of their process-protocols. Furthermore, just like with the workflow definition data, entity access to a persistent database which stores application data can be implemented using the `PersistentOntology` plugin. Then, (possibly specialised) Paffin entities may provide, manage and store application and runtime data.

User Interface: The user interface is already implemented in a special GUI Paffin entity. It is feasible to make these GUI Paffins more complex to better support users, but for model clarity this kind of functionality is better provided in the resource Paffin, i.e. the Paffin representing the human user for the rest of the system. The possibilities, like worklist filtering or ordering, are analogous here. Apart from that it is always possible to extend the web interface controlled by the GUI Paffin entity. Ideas for that have already been discussed in Section 12.6.4.

Invoked Applications The invoked applications interface connects workflows and tasks with other software applications. For Agent-Activities and the Paffin-System this is especially prominent, as the resources for workflow tasks can always be automatic entity resources. Consequently, the focus now lies on how automatic entity resources can be used to improve workflow management. External applications can be incorporated into these automatic entities by providing a command interface and having the entity control the external applications. This was demonstrated by the Paose support prototype from Section 11.3.3, where the external RedTimer tool was called and controlled from the Paffin-System.

Worklist Handling: The worklist for automatic resources is already being handled within the resource Paffin entity. Currently, a minimum of intelligence is used to manage that worklist, i.e. the highest priority workitem is always selected. This can be extended to consider history, workload or other runtime variables.

Process Control: Process control for invoked applications, i.e. automatic resources, is fully handled by the elements of the reference architecture and already fully implemented in the Paffin-System. It is a collaboration of the integrated entity as a workflow engine, the platform WFMS and the entity as a resource. At any point, the mechanisms can be extended as necessary to improve process control and management according to the current needs. Apart from that, the general observations from the analogous point from the client applications interface can be applied here as well.

Data Handling: This function is analogous to the function of the same name in the client application interface. Integrated entities and the Paffin-System do not need to distinguish between data handling for users and automatic resources.

Service Discovery: Imbuing entities with the functionality to interact with service directories is possible. In fact, the WebGateway functionality already available in Capa and the Paffin-System can be directly utilised to facilitate this interaction. Therefore, the integrated entities (of the Paffin-System) can be directly configured and built to discover and orchestrate different services. The arguments are analogous to general process control.

Other Workflow Enactment Services The interface to other workflow enactment services realises interoperability for a WFMS. Capa agents, and therefore Paffin entities, support the FIPA standards (see Section 2.2.2), ensuring that they can indeed interact and communicate with other systems following those standards. Assuming some form of interoperability through following established standards is a prerequisite for this interface. Therefore this assumption is made for the Agent-Activity approach in general.

Common Interpretation of Process Definition: If different workflow formats are to be integrated into an entity system, a translation would need to be provided from whatever modelling technique was used for process-protocols. Conceptually, this translation would be relatively simple, translating a task in the workflow representation into an Agent-Activity with just the three workflow operations. The control flow relation could then be more or less directly deployed to the process-protocol. For the entity system (or Paffin-System) to work with other systems, a similar translation into the target modelling technique would also be required. Here, the translation could filter out (or transform into special tasks) the agent actions and return only the basic workflow. In either case the translation could be implemented into a special platform

entity serving as a gateway. That gateway would provide a central, w.r.t. the platform, interface which both the external systems and internal entities could utilise.

Workflow Data Interchange: Here, too, a gateway entity could be utilised that translates incoming and outgoing data into specific standardised formats. Otherwise this functionality is similar to analogous to the data handling of the previous two interfaces.

Administration and Monitoring Tools The administration and monitoring tools interface represents the auxiliary management functionality of a WFMS. Regarding the Agent-Activity approach, most of this functionality is provided on the platform and management level of the reference architecture. Monitoring facilities have already been implemented in the Paffin-System, yet basic administration is part of future work as discussed in Sections 11.4.3 and 12.6.4.

User/Role Management: Paffin entities, and by generalisation any integrated entities, already represent users within the system. They register with the platform WFMS, receive workitem and activity data and provide the GUI Paffins with that data for display. In that way, Paffins already are the virtual avatars described in this functionality. Expanding that representation to roles/groups of users is also already envisioned with the concept of subordinate resources. Here, one Paffin would represent a group of users or resources to the rest of the system. It would request workitems and then distribute them to its subordinate resources as necessary. Having an integrated entity act as a user/role administration entity is also feasible, especially in combination with data handling/database access. This was already described as part of the vision of the administration workflow in Section 11.4.3.

Audit Management: In concept, logging mechanisms can be added to any part of the reference architecture for the Agent-Activity integration approach. In the Paffin-System multiple logging mechanisms are already implemented, including ones on the platform and management level and the Paffin level. Currently these logs are not further utilised within the system. However, they could be easily incorporated into runtime mechanisms, including the creation of worklists according to historical workload or the choice of workitems by automatic resources based on historical execution times. Further analysis of logs from previous runs could also be used by a system to create an optimal deployment of entities on different platforms. This could be achieved by having an entity analyse the log and then initialise the rest of the system according to the results of that analysis.

Resource Control: Analysing and influencing the execution of workflow tasks by human (or automatic) resources has already been covered. Basically, data processing as agent functionality can be used to filter, sort, order or emphasise worklists for resources, which in turn partially manages the execution. Analysis is also an option, similar to how it is covered in the audit management function. An interesting idea is to have entities not just analyse, monitor and sort worklists, but also edit them in certain ways. For example, considering the application example of Paose teaching support, task descriptions may include hints that are only gradually displayed to the students working on the tasks.

Process Monitoring: Logging mechanisms have already been discussed in the audit management function. Having a special (platform) monitoring entity is feasible, yet the logging functions within the rest of the system have an advantage in that they can log the operations that are happening in their direct field of observation and influence,

i.e. within themselves. A special entity would centralise the results and possible obtain a more global view, yet the added overhead would be a drawback. It is also questionable if that entity could gain a better overview than the platform WFMS, which, at least for workflow operations, is globally involved.

Workflow Enactment Service The workflow enactment service provides the central workflow management facilities and serves as a runtime environment for the workflow engines. As the workflow engines are the different integrated entities that execute workflow behaviour, the entire system needs to be considered as the workflow enactment service. However, many global (w.r.t. a platform) management functionalities that are part of the workflow enactment service can also be found in the platform WFMS.

Runtime Control Environment: Integrated entities are the workflow engines and therefore the runtime control environment for workflow instances in the form of process-protocols. This automatically fully encapsulates the workflow instance and workflow engine functionality, so that it can be easily deployed in a distributed environment. Furthermore, many of the previously discussed functions are made possible through this principle. Another interesting aspect in this function is the ability to intersperse agent actions into the workflow execution. This is only possible because the integrated entities serving as workflow engines can also act as agents. Here, the more advanced integration functions become possible, e.g. agent interactions as part of workflow tasks, result processing and quality assurance before activity confirmation or simply the ability of engines to autonomously and directly confirm or cancel activities.

Definition Interpretation: It is feasible to have the integrated entities reason about the process-protocols they execute. Even when BDI concepts are not supported, entities analysing their own behaviour as a special, concurrent behaviour for influencing the original behaviour is possible. For example, an integrated entity may monitor its own reactively started process-protocols and then proactively request the platform to initialise another entity with its roles to balance the workload of recurring workflows. While not implemented in the current prototypes, this kind of functionality can easily be added to the Paffin-System.

Execution of Tasks: Execution of tasks is seen as entities serving as automatic resources in workflow tasks from other integrated entities. For the Agent-Activity approach this execution of tasks function is analogous to the process control function from the invoked applications interface.

Scheduling: Scheduling can be approached on the level of the individual integrated entity as a workflow engine and on the platform WFMS level. In either case, agent intelligence can be added to the integrated entities to filter, sort or prioritise workitems and activities for resources in order to optimise workflow execution. Interaction could also be utilised here. If, for example, an entity's workflow contains some critically prioritised tasks, it could communicate with other entities and the platform WFMS to ensure that its tasks get the highest priority for execution.

Data Functions: Data functions in this group are analogous to data handling functions in other interfaces. Integrated (Paffin) entities can have access to persistent data storage and provide data to and from their workflow instances.

Task Assignment: Task assignment can be a function of integrated entities as engines and as resources, as well as of the platform WFMS. In fact, the Paffin-System already implements automatically assignable tasks, as well as a mode where resource entities

automatically assign all available workitems to their resource. In any case, agent intelligence and data functions, possibly in combination with analysis of historical logs, can be used to extend this task assignment to factor in any runtime variable.

Resource Allocation: Similarly to task assignment, larger-scale resource allocation can be realised in the individual workflow engines and the platform WFMS. Here, agent intelligence in the form of processing can be added to dynamically alter task execution rules and permissions in order to balance the workloads across the entire system. The basis for this kind of function is already implemented in the Paffin-System. Dynamic permissions were used in the pizza application example from Section 11.3.2 to ensure that related tasks were always assigned to entities representing the same pizza delivery service. The dynamic permissions mechanism can be used in a similar way to allocate resources to specific sets of workflows.

As shown in the previous descriptions, the Agent-Activity integration approach, and also already in large parts the Paffin-System prototype, provide solutions for all twenty-four functions agents can have in workflow management. In the conceptual approach there are no restrictions to the agent principles, concepts and mechanisms that can be introduced. In the practical prototype the restrictions are given by the specific Capa agent model that, for example, doesn't feature cognitive concepts. Altogether, the Agent-Activity approach is capable of providing *full* access to the agent paradigm. The reasoning for that is simple. Through Agent-Activities the workflows can dynamically and concurrently:

- Be agents
- Turn into agents (and back)
- Exchange data with agents
- Be controlled by agents
- Control agents

In fact, any relation between agents and workflows can conceivably be realised through Agent-Activities, since the agent functionality, not just through agent actions interspersed with workflow operations but also through the interaction of different entities serving different roles, can be freely utilised. This conceptual versatility is the main benefit that the Agent-Activity approach has over agent-based workflow management. When compared to some of the research efforts, the Paffin-System is often not as technically mature, due to those research efforts being in development longer and having been worked on by more personnel. However, the conceptual possibilities of the Paffin-System and the Agent-Activity integration approach envelop all functions that agents have provided for workflow management in the past. Therefore, the entirety of agent-based workflow management can be considered as a true subset of the full integration provided by Agent-Activities.

13.3.2 Workflow-based Agent Management

In workflow-based agent management, workflows are used to build or enhance agent management. What is important for these kinds of research efforts is that agents are the focus of modelling. In these ways workflow-based agent management is the opposite of agent-based workflow management.

Contrary to agent-based workflow management there are relatively few research efforts that fall into this category. Reasons for this have shortly been addressed at the end of

Section 3.3.2. This also means that there is no clear categorisation of functions for workflows in agent-management available. Most research efforts focus on workflows describing or enhancing the behaviour of agents. Still, for this section the individual research efforts presented before are examined and related to the results of this thesis.

In [Korhonen et al., 2002] workflows are used to describe flexible agent behaviour. Due to the limited amount of contributions it is unclear how flexible the workflows actually are and how exactly they are realised. Therefore, a concrete comparison is impossible. However, as far as the general idea is concerned, the WorkBroker principle (see Definition C.13) supported in the Agent-Activity approach and implemented in the Paffin-System does exactly the same thing. Instead of hard-coding the interactions of entities (as agents) into the process-protocol, the WorkBroker principle allows for entities to define tasks that other entities execute for them. The interaction is standardised via the workflow management functionality of the platform and management level. Flexible workflows are possible in the conceptual approach of Agent-Activities, but not implemented for Renew. However, some flexibility is already available and implemented in the Paffin-System by allowing Agent-Activities to feature variable AAO identifiers and workflow operations to use variables for task identifiers. The result is that the basic execution structure is fixed for any net, but what is actually executed in that structure is variable. For Agent-Activities this can tremendously change the execution, as different AAO may feature any conceivable functionality, even starting other process-protocols. For workflow tasks this effect is more restricted, though considering automatic resources a different task identifier may also trigger completely different behaviour.

The work of [Kotb, 2011, Kotb et al., 2012] compares goals and capabilities of agents, based upon which collaboration workflows are proposed and executed. For integrated entities in the Agent-Activity integration approach it is feasible to realise a similar mechanism. Even without the cognitive concept of goals, integrated entities can interact and exchange information based on their roles, historic logs or some other description within the model made specifically for this purpose. If enough similarities could be found they could form a cooperation. The contemplated mechanism of subordinate resources from the Paffin-System could be used here. The entities sharing similar goals could become subordinate resources for a newly instantiated entity. That entity could then execute a collaboration workflow for its subordinate resources or simply act on their behalf in the overall system and balance the workload internally. [Kotb, 2011, Kotb et al., 2012] also utilise soundness for their workflows, which was already discussed for Agent-Activities in Section 12.6.2.

Mobility as proposed in [Mislevics and Grundspenkis, 2012] is supported in the conceptual Agent-Activity approach, yet is not available in the Paffin-System yet. Still, [Mislevics and Grundspenkis, 2012] uses workflows to model and control agent behaviour. That, again, is handled by the WorkBroker principle, which serves analogously here as it did for the previously discussed related work in [Korhonen et al., 2002]. Applying the principles in settings were agent mobility is available to integrated entities is not difficult. Integrated entities as automatic resources migrating in the context of the workflow tasks they execute are possible.

[Küster et al., 2015] uses BPMN models to specify multi-agent systems. The exact nature and model of the workflow aspects used in the Agent-Activity approach is not fixed on

Petri nets. This thesis emphasised the net perspective to gain a generic basis on which to build the integration and open the way for future work utilising the formal aspects. However, as discussed in Section 12.2 the results of this thesis are generally applicable to other models for agents and workflows. Exchanging the BPMN models with workflow Petri nets yields a similar approach to the WorkBroker principle, with workflows serving as the basic specification of behaviour between different integrated entities acting as agents.

[Nouzri and Fazziki, 2015] also uses BPMN models to specify multi-agent systems. Analogously to the previous point, any workflow model may be used with the conceptual Agent-Activity approach. Therefore, the general ideas from [Nouzri and Fazziki, 2015] are also applicable. Explicitly considering business goals may be incorporated into the entity knowledge as a parameter guiding the execution.

[Gomes et al., 2016] uses YAWL workflow models to specify patterns for the functionality of agents. These patterns are later combined into more complex agent behaviour. This, again, represents a specification of agent behaviour and interaction based on workflows. With YAWL being closer to Petri nets than BPMN, the application of the Agent-Activity approach and therein the WorkBroker principle is even more directly possible. The arguments are analogous to the ones previously made in this section regarding the WorkBroker.

From the previous descriptions it is clear that the WorkBroker principle from Definition C.13 captures the functions of workflows in agent management. It enables the specification of agent behaviour through workflows, which is the most focussed research area in this context of related work. Apart from the specification of agent behaviour, workflows in the Agent-Activity approach may also be used to influence the system, as described in the vision of an administrative workflow from Section 11.4.3. The overarching control or management of agent systems is not considered in the available related work. All of the contributions focussed on the individual agents or their interactions on the direct application level. The Agent-Activity approach is capable of going beyond that, enabling the workflow control not only for the application but also for the implicit, background and supplemental/management/administrative processes of the entire system. Like with agent-based workflow management, this shows that workflow-based agent management is fully contained in the Agent-Activity approach and that there are integration efforts that push beyond even that. These points are picked up again later in Section 13.4.

13.3.3 Advanced Integration Efforts

The previous two subsections considered research efforts that, even though they utilised both agents and workflows, still only provided or emphasised one of the two for modelling purposes. This section now considers the more advanced integration efforts. Advanced integration efforts begin to obscure the clear distinction between modelling with agents and modelling with workflows. They may still feature an emphasis or focus, but that modelling focus must at least exhibit a strong duality of agent and workflow aspects. The order of the following paragraphs mirrors the one in Section 3.3.3.

Agile, Goal-oriented BPM The agile, goal-oriented BPM presented in [Burmeister et al., 2006, Burmeister et al., 2008] endowed processes with cognitive concepts like goals to achieve flexible process execution. The Agent-Activity and Paffin-System do not

feature BDI-like cognitive concepts, which is why there can be no direct comparison. Section 12.2 examined how other agent models including BDI could be incorporated into or employ the Agent-Activity concept, but such applications are outside of the scope of this thesis.

However, some comparisons can still be drawn. Agile, goal-oriented BPM defines goals and contexts for processes, which are used to adapt process execution to runtime circumstances. Plans describing behaviour are chosen depending on runtime goals and context. Something similar is possible in the Agent-Activity integration approach and the Paffin-System. The inscription of the Agent-Activity-Transition contains variables that can be filled by tokens provided via the incoming arcs. The tokens represent dynamic runtime data. If the content of the plans from agile, goal-oriented BPM is considered as the internal processes for Agent-Activities, a similar flexibility can be achieved. The runtime data token determines which AAO to execute. It can be edited by the Paffin at runtime to reflect dynamic changes to the process. These changes need to be predefined, though, but so are the plans in agile, goal-oriented BPM.

Overall, agile, goal-oriented BPM does not feature as full an integration as the Agent-Activity approach or the Paffin-System. Processes are still emphasised for modelling. They are heavily endowed with (cognitive) agent concepts and mechanisms, but, at their core, they are still business processes. The entire goal of the research efforts is to improve BPM, while that is only one direction of what the Agent-Activity and Paffin-System can do, as previously discussed.

Living Systems Process Suite LSPS strongly emphasises agent concepts in the processes it supports. It appears[1] as if the processes in LSPS are, for all intents and purposes, BDI agents. Their execution is oriented around goals, they are autonomous, interact, etc. However, LSPS still only emphasises processes/workflows for modelling. Therefore it limits itself in the same way as the partial integrations discussed before. Systems following the Agent-Activity approach do not have that singular emphasis, but rather have agents and workflows on the same dualistic level. This means that in LSPS only workflow modelling can benefit from the agent concept. With Agent-Activities agent modelling can also benefit from the workflow concept and both can be used to build fully hybrid systems, neither of which appears possible with LSPS.

WADE WADE heavily features workflows to describe the behaviour of both agents and human users. How far these two groups can be dynamically interchanged is not known from the published contributions on WADE. It is however feasible that human tasks and agent tasks can be modelled in combination in some way. WADE therefore represents a rather practical integration of agents and workflows. Workflows benefit from agent properties such as autonomy and encapsulation, while agents benefit from workflow-like behaviour specification.

However, on a technical and conceptual level agents and workflows are still separated. It is always assumed that agents possess a set of workflows and attempt to execute them as best as possible in the dynamic runtime environment. This means that workflows are still subordinate to agents in WADE. This is in contrast to the integration desired for this thesis and realised by the Agent-Activity approach and Paffin-System. Workflows in WADE may be used to specify agent behaviour, but due to the separate nature of agents

[1] Due to the missing scientific publications the exact characteristics of the concepts used in LSPS can only be assumed. The technical documentation on `https://docs.whitestein.com/lsps/3_1/index.html` (last accessed May 28th, 2017) also isn't clear enough on these topics.

and workflows in WADE the two concepts never truly mix. In fact, the WADE approach most closely resembles the WorkBroker principle (see Definition C.13), where workflows are used to specify agent interactions. WADE adds to that approach the possibility of defining workflows for human users, but that still doesn't achieve the possibilities of a full integration with Agent-Activities. Agent-Activities allow for integrated entities to *be* agents, workflows or even both. Practically this means that, if agent and workflow functionality is to be mixed, WADE agents will have to alternate between the workflow specification and the agent specification instead of accessing it from a shared state. As a result, freely combining agent and workflow mechanisms on the same level is not possible, which is one of the main integration and modelling features of the Agent-Activities.

Sonar and Organ Organisations in Sonar and Organ serve as a consolidation for agent and workflow aspects of collective actors. The collective actors in this case are the organisations. Each organisational unit represents all of the individual elements it consists of and these elements are roles/actors/agents and processes/interactions/workflows.

This also shows the main difference to the work presented in this thesis. Organisations, as collective actors, may feature agent and workflow aspects. However, "zooming" into these organisations reveals that these aspects are, in fact, separate entities of agents and workflows. In this way, organisations are rather combinations of agents and workflows that are conceptually integrated through abstraction. Through their internal elements organisations can be considered as having both agent and workflow aspects, but their individual parts can only be one of the two. This means that the integration is only available on the organisation level, not on the individual element level. In contrast, integrated entities as proposed by this thesis can always be both.

While the integration aspects may not be comparable between organisations in Sonar and Organ and integrated entities with Agent-Activities, there is a latent synergy here that can be exploited in future work. The view of organisations as collective actors fits well with inter-organisational contexts. In fact, throughout this thesis, and especially in Section 11.5, Agent-Activities have been motivated for inter-organisational contexts as well. It was even proposed to have an integrated (Paffin) entity represent an organisation in such contexts. Incorporating and emphasising the role of the organisation concept as proposed by Sonar and Organ into the Agent-Activity approach may yield valuable improvements to its application to the field of inter-organisational contexts. The reference architecture proposed by Organ could be used to provide a more structured organisation of groups of integrated (Paffin) entities relating to one overarching organisation. This could, potentially, also benefit the research in Sonar and Organ. The dual agent and workflow nature of integrated (Paffin) entities could be incorporated into these models as well in order to better capture the inherent behaviour/structure duality of organisations. The incorporation of organisations into the Agent-Activity approach is outside of the scope of this thesis.

Process-Infrastructure for Agent Application (PIA) The idea of an integration of agents and workflows as pursued by this thesis was inspired by the fifth tier of the integration architecture in [Reese, 2010]. However, while there is a common basis for this thesis and [Reese, 2010] the results are very different.

First and foremost, the integration approaches differ greatly. The principles of the fifth tier of the integration architecture of [Reese, 2010] were presented in Section 8.1.1 as integration via management. Integration via management evaluated poorly, with the main issue being that the approach didn't provide direct modelling access to integrated

entities. Due to this issue, the capabilities of integrated modelling were markedly restricted in the approach. For example, with integration via management it is impossible to model agent behaviour directly into a workflow task. At most, it is possible to approximate that behaviour by crafting agent behaviour concurrently to a workflow task. This approximation is inferior due to the lack of a directly modelled connection between agent and workflow in the integration via management. It is less intuitive, because modellers have to track two separate models for agents and workflows. It requires more implementation effort, because modellers have to implement additional behaviour to connect the agent behaviour to the workflow task. Finally, it also is generally less efficient both from a modelling and an execution perspective. Modelling is made more difficult and the additional functionality of tracking and connecting the two models can only decrease performance. The Agent-Activity integration approach and its implementation in the Paffin-System allow for the direct incorporation of agent behaviour into workflow tasks by allowing Agent-Activities to add agent actions between workitem request and activity confirm workflow operations.

More examples of behaviour and features impossible or only approximately realisable in the fifth tier/integration via management include the reverse situation, i.e. incorporating workflow tasks in between agent functionality, the ability to model an entity as part of both structure and behaviour at the same time, the possibility to model an arbitrary and complex nesting of agent behaviour in workflow tasks in agent behaviour and so on and the ability to have an entity as an agent interact with an entity as a workflow (and vice versa) without changing perspectives or modes. On the other hand, the capabilities of the fifth tier/integration via management are entirely included in the Agent-Activity approach. For example, workflow tasks can be used to model the entire (internal and external) behaviour of entities as agents, workflows can be executed by entities as agents as both engines and resources and agent or workflow perspectives can be achieved by abstracting from Agent-Activities that contain workflow operations and agent actions respectively. Overall, the fifth tier/integration via management is not as powerful an integration modelling approach as the Agent-Activity is.

The gap between what [Reese, 2010] considers as a full integration and what this thesis considers as the vision of an integration (see Chapter 7) can be traced back to the different emphases of both theses. In general, the research in [Reese, 2010] focuses on providing agent and workflow perspectives on a system. This current thesis, on the other hand, focuses on modelling and executing an integration of agents and workflows. In other words, [Reese, 2010] aimed to provide a system in which every element can be considered as an agent or a workflow, while this thesis aimed to provide a way to model an element as agent, workflow or a hybrid of both. A prerequisite for this thesis' kind of hybrid modelling of the two concepts is the ability to perceive an element in the associated individual perspectives. Consequently, the ambition of the fifth tier of [Reese, 2010] is part of what was achieved in this thesis.

Similarly to the conceptual basis, the practical results of the two theses also diverge. The main practical focus of [Reese, 2010], the process-infrastructure for agent applications (Pia), can be completely realised in the Paffin-System. The core idea behind Pia is to support and execute agent interactions through workflows. Paffins and Agent-Activities can completely realise this core idea. In fact, it is possible to realise *all* the different variants for Pia presented in [Reese, 2010, p. 106] through Paffins and Agent-Activities. As an example, agent interactions as workflows correspond to Agent-Activities containing tasks that are intended for other Paffins as resources (WorkBroker principle, see Definition C.13). These resource Paffins request or are assigned to a task, which initiates a new process-protocol in them. That process-protocol can contain further workflow

aspects and control further interactions but once it is finished it confirms the activity in the original workflow (along with transmitting any results). This functionality is only part of functionality of the Paffin-System prototype.

Likewise, the prototypes presented in [Wagner, 2009c] implementing the two variants of the fourth tier of the integration architecture of [Reese, 2010] are fully realisable in the Paffin-System. The Structure-Agentworkflow can be implemented by a process-protocol that contains Agent-Activities that initiate subworkflows in other Paffins. The Protocol-Dispatcher can be realised by a process-protocol that contains Agent-Activities, which initiate agent protocols in other Paffins. As with the general Pia, the Protocol-Dispatcher is similar in nature to the WorkBroker principle. However, the Protocol-Dispatcher can't intersperse regular agent behaviour within its workflow, disconnects the workflow from the (agent-)system and doesn't emphasise the management between engines and resources. This makes it far less versatile and powerful when compared to the WorkBroker principle.

Note that the examples and scenarios presented in [Reese, 2010] and [Wagner, 2009c] are conceptual and were, for the most part, not implemented. Consequently, this means that while it is possible to realise them in the Paffin-System, they were not available for a practical comparison and evaluation. This, in part, led to the implementation of other scenarios for the practical application prototypes in Chapter 11.

In conclusion, while the fifth tier of the integration architecture of [Reese, 2010] envisioned an integration of agents and workflows, that vision emphasised providing agent and workflow perspectives. The modelling emphasis of this current thesis contains this ambition as part of the vision of providing a way to model and execute an integration of agents and workflows. Consequently, the Agent-Activity approach presented in this thesis is more powerful and expressive than the fifth tier of the integration architecture. The practical realisation of the Agent-Activity in the Paffin-System is, in turn, also more powerful than the prototypes presented in [Reese, 2010] and [Wagner, 2009c], whose complete functionality is realised as part of the Paffin-System.

Hera and Potato The Potato system is a specialised integration focussed on one particular application area, namely distributed software engineering. It strives to utilise a full integration of agents and workflows to support its intended application area. However, due to the circumstances at the time of development, the technical requirements for that full integration were not yet available. For its structural view, the Hera helper and tool agents already sufficiently covered the requirements. However, for the behavioural dimension the author utilised the Process-Infrastructure for Agent Applications (Pia). As defined in [Reese, 2010] Pia is not a full integration, because while it integrates agents and workflows in the background, it still emphasises agents as the main modelling construct. In the terms of [Reese, 2010] it is part of the fourth tier of the overall integration architecture, instead of the fifth tier which features the full integration of agents and workflows. Pia simply doesn't support entities as *both* agents and workflows. Potato, though, envisioned and desired such a full integration: "*The complete flexibility, which an integration of the fifth tier would provide, would, of course, be desirable. Yet there are too many open questions, as the system would consist of something that would represent a completely new kind of object. Every participant of the system can be considered as both process and structural element.*" (translated from German) [Markwardt, 2012, p. 135] The Agent-Activity approach and the Paffin-System now provide an integration that satisfies the requirements of the fifth tier of the overall integration architecture from [Reese, 2010]. Integrated entities can be considered as both process and structural element and can be used to completely fulfil

the extended vision of Potato. While a (re-)implementation of Potato is outside of the scope of this thesis, Dennis Schmitz is examining providing support for distributed software development in his recently started doctoral thesis. This work aligns well with the principles and concepts from Potato and is intended to utilise Agent-Activities and Paffins.

Jadex Active Components Jadex active components are a technically mature, versatile and highly extendable framework that, as described in Section 3.3.3, can realise an integration of agents and workflows through its different internal architectures and kernels. In comparison to the research presented in this thesis, the focus of Jadex active components is more general as it features agents and workflows, as well as components, services and objects. While the Agent-Activity approach is more restricted to "only" agents and workflows, it also means that it is more clearly focussed on these concepts.

Concerning the integration approaches presented as options in Chapter 8, Jadex active components most closely resembles a realisation of the intermediate model approach (see Section 8.1.2). Here, the basic active component is the intermediate model, with the kernels determining the modelling construct. The criteria limitations discussed for the intermediate model approach do not directly apply to active components though. Through the use of hybrid kernels, such as GPMN, the facet to the outside can be both agent and workflow. Furthermore, by using the classical agent or workflow kernels it is also possible to build regular agent and workflow systems without any drawbacks.

Still, considering the basic approach of active components and Agent-Activities, there are some similarities. Active components are, basically, containers, with a standardised agent and service interface, into which an internal architecture is embedded. That internal architecture can be a BDI agent, a BPMN workflow, a GPMN workflow, etc. This basically means that if an active component has a BDI or microagent kernel, it is an agent. If it has a BPMN kernel, it is a workflow. If it has a GPMN kernel, it is a hybrid of agent and workflow. Thinking of an active component as a container is quite similar to how Agent-Activities are considered in this thesis. If the Agent-Activity contains only agent actions, the entity executing it is an agent, if it only features workflow operations it is a workflow and so on.

However, at this point the main difference between the integration performed by active components and Agent-Activities becomes clear. Active components approach the integration on the level of agents and workflows. They provide a container which is filled with one internal architecture, which determines what can be done by the component. Agent-Activities approach the integration on the level of agent actions and workflow operations. Integrated entities provide the container for Agent-Activities and provide the mechanisms of both agent and workflow functionality. They are always able to do what both agents and workflows can do.

Figure 13.1 illustrates this difference in the approach. Agent-Activities feature the integration on the behaviour level of what an integrated entity actually does and implements. Using integration terms, active components feature the integration on the level of the integrated entities (here active components). The internal architecture and kernel determine the integration in active components, while the functionality is modelled subordinate to that.

Overall, this difference in the approaches leads to agents and workflows in active components being less conceptually clearly integrated. The integration of agents and workflows in active components can be characterised in the following ways:

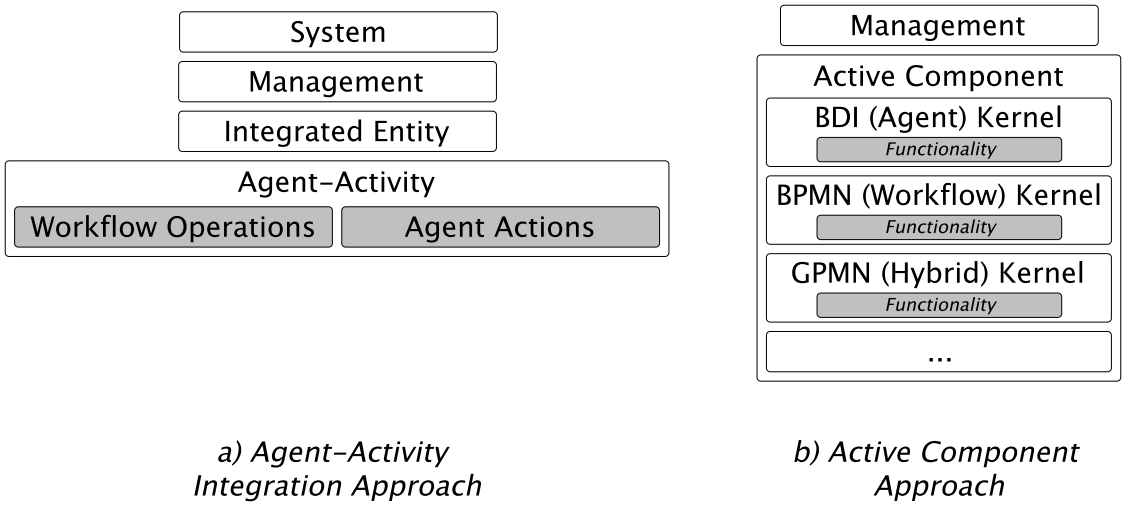

Figure 13.1: Integration in Agent-Activities and in Jadex active components

- Agents and workflows can be executed "next" to one another on the same level. This is realised by the standardised interface between active components that exhibit different internal architecture kernels.
- Agents and workflows can be combined as separate sub-components to a superordinate active component. That way, although they are practically separate, an active component can simulate a hybrid of agent and workflow to the rest of the system.
- Specialised kernels, such as GPMN, can be used that implement an integration of agents and workflows directly.

All of these three ways of integrating agents and workflows are also captured by Agent-Activities and the Paffin-System. Integrated (Paffin) entities can freely interact with one another, regardless of whether or not they are acting as agents or workflows at the time. An integrated (Paffin) entity can also have multiple process-protocols active concurrently. These process-protocols can realise separate agent and workflow behaviour that is unified to the rest of the system via the entity interfaces. Finally, actions and operations can be freely combined in Agent-Activities to create hybrids.

The key differences between active components and Agent-Activities in these integration points lie in the determination of what is possible. An active component must predefine that it wants to act as a workflow or an agent when selecting the used kernel. When it wants to be both it must either use a hybrid kernel or provide the suitable sub-components. In either case, an active component is always locked into the internal architecture that is chosen. An integrated (Paffin) entity can always dynamically switch between agent, workflow and hybrid. It always has the option to be what is needed, which only depends on the Agent-Activities it executes.

This predetermination limits the integration aspects in active components somewhat. Instead of allowing modellers to freely choose the behaviour of their modelling constructs at any time, modellers need to predetermine the role of their construct by choosing a specific kernel or sub-component setup. The behaviour is modelled afterwards. This does not mean that active components can do less then Agent-Activities. In fact, considering their technical maturity, extendibility and the additional concepts from services and objects,

they can do more. However, for the integration of agents and workflows they are more limited. Integrated entities using Agent-Activities can always dynamically act as agents, workflows, both or something in between. For active components this choice needs to be predetermined.

Furthermore, modelling the integration is less clear in active components. When considering the first two points of integration described above, agent and workflow functionalities are always separated in different (sub-)components. They are not modelled in a uniform and combined way. That is only possible when using hybrid kernels. Currently, the only published hybrid kernel is GPMN. Here, workflows are modelled together with cognitive agent concepts. This most closely resembles modelling in Agent-Activities, where agent and workflow elements, i.e. actions and operations, can be freely combined in a consistent Petri net model.

However, while GPMN clearly integrates workflows with BDI agents, it still emphasises workflows. GPMN models workflows in an agent-oriented way, but agents do not explicitly benefit from it. Therefore, GPMN, by itself, must be considered more in line with the partial integrations discussed before. Embedded with the other integration points provided by the active components, this restriction is clearly lifted. Yet, in order to fully support hybrids of agents and workflows, new and more balanced hybrid kernels need to be provided. Only in that way can the mismatch between agent and workflow emphasis motivated and targeted in this current thesis be completely overcome. Integrated entities using Agent-Activities do not feature this mismatch, making them, from a purely integration point of view, more advanced. Implementing a fully hybrid kernel would clearly alleviate the integration drawback discussed here. It may even be feasible, due to a common Java basis in Jadex and Renew, to consider implementing an Agent-Activity kernel for active components. This is, however, completely outside of the scope of this current discussion and thesis.

In addition to the integration aspects discussed above, the Petri net basis for Agent-Activities and the Paffin-System needs to be taken into account. First of all, concurrency is not an emphasis in active components: "*It is assumed that each component is executed sequentially, i.e. true concurrency exists only between active components. Within an active component a kernel may offer quasi-parallel execution by interleaving the execution of active behaviors.*" [Braubach et al., 2013, p. 4] Concurrency modelling and execution is a basic feature of (reference) Petri nets, meaning that it is also a basic feature for Agent-Activities and the Paffin-System. In (distributed) use cases where concurrency issues are prevalent and critical, the Petri net-based Agent-Activities therefore do have an advantage. Secondly, even though it is outside of the scope of this thesis, the formal basis of Petri nets provided to Agent-Activities and the Paffin-System constitutes another advantage. Future work (see Section 12.6.2) is able to exploit that formal basis to provide verification and validation results only possible through the Petri net basis.

In conclusion, Jadex active components are clearly more mature, versatile and extendable in the practical and technical aspects. However, considering the integration of agents and workflows pursued in this thesis and also considering modelling that integration, Agent-Activities feature some clear advantages. Agent-Activities allow for more continuous and dynamic switching between agent, workflow and hybrid states in both modelling and execution. Modelling of both agent and workflow elements is done in uniform, combined and executable Petri net models. Altogether it can be observed that Jadex active components emphasise and excel at the technical implementation, while Agent-Activities emphasise the modelling approach.

13.4 Related Work Conclusion

Throughout this thesis the terms partial and full integration have been referenced. In fact, the Paffin-System has a reference to a full integration directly in its full name **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration. Now that the comparison and evaluation of other integration research efforts has been established in this section it is finally possible to properly discuss the terms[2].

The traditional agent and workflow related work discussed in Sections 13.1 and 13.2 do not feature an integration at all. Here, only the possibilities of the utilised concept are available. Mature and established frameworks and models have been extensively researched and examined by many different research groups. However, w.r.t. the integration these efforts may only provide the components on which to build the desired integration. By choosing different frameworks as starting points or as inspiration different mechanisms can be made available, but the core of the research in this thesis is more concerned with conceptually providing a solution rather than focussing on concrete, particular mechanisms.

A *partial integration* refers to integration efforts that, while using both agents and workflows in the background, still only emphasises one of the modelling concepts for actual system modelling. This refers to agent-based workflow management, as discussed in Sections 3.3.1 and 13.3.1, and workflow-based agent modelling, as discussed in Sections 3.3.2 and 13.3.2. As examined and related in those sections, there are a large number of advantages and mutual benefits that both agents and workflows can gain from each other in that way. However, the limitation to modelling either workflow systems in agent-based workflow management or agent systems in workflow-based agent management still holds those research efforts back. Workflows may feature agent intelligence, but in a partial integration through agent-based workflow management they can't change their role in the system to an active participant that contributes to the system in any other way than being a process. Likewise, agents may have their behaviour be defined in workflows and partitioned into tasks, but in workflow-based agent management they can't assume the role of an overarching workflow influencing or even replacing system execution. This does *not* mean that agent-based workflow management or workflow-based agent management are in any way unproductive or useless. Most application scenarios can already gain excellent benefits from the two approaches. Yet there is a limit to what can be achieved, which is why this thesis coins the term partial integration for them.

A *full integration* on the other hand does not present such a limitation. It contains all the possibilities of both variants of the partial integration. Agents can use workflows and their principles, while workflows can do the same with agents. This also relates to the strengths of the Agent-Activity integration approach discussed in Section 12.4. The relationship between agents and workflows is completely unrestrained. Each of the two can use, be, turn into, interact with, etc. the respective other one. They are on the same level of modelling and abstraction, as proposed in the integration vision from Section 7.1 that underlies the results of this thesis. This opens up the possibilities touched upon throughout this thesis, especially in Chapters 11 and 12.

One aspect worth explicitly pointing out for a full integration is that the benefits of the partial integrations are not just both available, they are cumulative. That means that all the effects and advantages described for agent-based workflow management are available for the workflows applied to agents and their behaviour. Likewise the effects of workflow-based agents can be utilised for the agents that are used to support or improve

[2] An early discussion on the topic of full and partial integrations can be found in [Wagner and Cabac, 2013].

workflow aspects. In other words, a full integration does not just provide the possibilities of the partial integrations directly, but also applies those benefits to the respective other partial integration in order to advance and improve the possibilities even more.

The Agent-Activity approach and already to a large degree the prototypical implementation in the Paffin-System feature a full integration. Agents and workflows are on the same modelling level, they can be related in any way and the mutual benefits of the partial integrations can be fully utilised. As discussed in Section 12.1 the original vision of an integration is fully realised here.

The advanced integration efforts discussed in Sections 3.3.3 and 13.3.3 go beyond the partial integrations of agent-based workflow management and workflow-based agent management. In these efforts the clear distinction between agent and workflow begins to diminish, so that, in many cases, it is unclear and effectively irrelevant if an agent or a workflow is modelled. However, none of the discussed research efforts features a completely full integration. Some, like Pia, LSPS or agile, goal-oriented BPM, still focus on either agents or workflows. Others, like WADE and Jadex active components, come close to a full integration as envisioned in this thesis. In fact, Jadex active components provide all of the prerequisites and only miss a fully hybrid kernel to capture an integrated entity as both agent and workflow.

Something that the examinations of the advanced integration efforts have shown is the relevance of cognitive concepts, especially goals. Goals are heavily featured in many of the research efforts, but, along with other cognitive concepts, have been outside of the scope of this current thesis. Future work should examine options of incorporating them or similar concepts for the prototypes. This is shortly elaborated on in the outlook.

In conclusion, a full integration goes beyond what the established partial integrations can accomplish. It not only combines the two directions of partial integrations, but also applies them to itself and opens up completely new concepts. This also represents the conceptual contribution and improvement of the results of this thesis to the state-of-the-art. When compared to traditional agent and workflow modelling, the additional value lies in the hybrid capabilities and concepts, which can utilise any agent and workflow properties. When compared to agent-based workflow management and workflow-based agent management, the improvement is constituted by the enabling of bidirectional and cumulative enhancement. From a general software-engineering perspective, the results of this thesis enable extended and improved modelling options. The Agent-Activity integration approach and its implementation in the Paffin-System are currently the only completely full integration efforts available. Future work will be capable of addressing the remaining open points in order to supplement the already mature conceptual solution with a more sophisticated and refined technical solution.

Part E

Conclusion

Part E concludes this thesis. Chapter 14 summarises the thesis and its chapters to provide an overview of what has been achieved. Chapter 15 then presents an overall conclusion and outlook for future work beyond this thesis.

The purpose of this part is to finish and complete this thesis. By summarising the results, a retrospective is created on which the conclusion and outlook can be based. The outlook also outlines the basic directions of possible future research.

14 Summary

This thesis dealt with the integration of structure and behaviour of a software system. The chosen representation for structure was given by agents, while the representation of behaviour was given by workflows. Concretely, agents followed the Mulan agent architecture and Capa implementation, while workflows followed the workflow Petri net principles. All practical models and systems used reference nets, a high-level Petri net formalism emphasising the nets-within-nets principle.

The desired integration of structure/agents and behaviour/workflows required concrete definitions of the involved terms. Based on these definitions an integration vision was developed, from which integration criteria were derived. These criteria were then applied to a number of abstract integration approaches developed for this thesis. The evaluation of these approaches through the criteria yielded the integration via agent actions and workflow operations as the most promising approach to achieve the desired integration.

From that abstract approach, the concrete Agent-Activity integration approach was refined. An Agent-Activity is a modelling construct describing the behaviour of an integrated entity. Within an Agent-Activity an internal process is constructed, which freely combines agent actions and workflow operations. The Agent-Activities an integrated entity executes define its state. If the executed Agent-Activity contains only agent actions, the integrated entity is an agent. If it only contains workflow operations, the entity is a workflow. If it mixes agent actions and workflow operations, the entity is a hybrid of agent and workflow. Concurrently executing Agent-Activities of different types completes the state possibilities, causing the entity to be both agent and workflow. Through the concept of the Agent-Activity it therefore became possible to define integrated entities that could be agent, workflow, both or something in between. A system implementing the Agent-Activity would fulfil the integration criteria to a high degree and therefore realise the integration desired in this thesis.

To substantiate the Agent-Activity approach as the desired integration approach and prove that it was also technically feasible, a proof-of-concept prototype was developed. The Paffin-System (**P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System) was developed as an extension of the Capa agent framework, incorporating new and additional workflow and hybrid management functionality. It was shown that the Paffin-System implemented the Agent-Activity integration approach and also realised the integration vision outlined earlier. In its current state, the Paffin-System represents a full-fledged framework for the creation of hybrid agent/workflow applications.

To demonstrate the capabilities of the prototype, the conceptual approach and the integration in general, application prototypes were developed and discussed. These prototypes showcased numerous hybrid mechanisms in practical use cases. Furthermore, the general application options and application modelling were discussed thoroughly. Afterwards, a general and overall and overarching evaluation of the results was performed. This highlighted and discussed the generality, synergies, strengths, remaining open points and direct future work related to the results. Finally, the results were compared to related research work.

The following lists the main results of this thesis.

Agent-oriented modelling perspective and definition of structure: Chapter 5 discussed the agent-oriented modelling perspective, introducing the concept of fundamental agent actions (see Definition B.1) and providing the basis for the definition of structure based on agents used throughout this thesis. The latter was developed in an abstract (Definition B.2) and a concrete (Definition B.3) variant.

Workflow-based modelling perspective and definition of behaviour: The behaviour counterpart to the previous result, the workflow-based modelling perspective, was discussed in Chapter 6. Here the concept of basic workflow operations (see Definition B.4) was introduced, as well as an abstract (Definition B.5) and a concrete (Definition B.6) definition of behaviour of a software system developed.

Vision of an integration: Chapter 7 outlined the vision and specification of an integration. It emphasised concepts such as the mutual incorporation and introduced another core concept of the thesis, the integrated entity in Definition B.7. From the vision of an integration, a number of integration criteria was derived. These represented the basis on which integration approaches and prototypes were evaluated in the rest of the thesis.

Integration approaches: Chapter 8 presented, discussed and evaluated six abstract integration approaches developed for this thesis. The approaches were "Integration via Management", "Intermediate Model", "Combined Model", "Integration via Actions and Operations", "Nesting" and "Integration via Synchronous Channels". Ultimately, the approach using agent actions and workflow operations was selected as the most promising and best fitting, based on the evaluation of the different integration criteria.

Agent-Activity *integration approach:* A concrete refinement of the abstract approach of integration via agent actions and workflow operations, the Agent-Activity integration approach (see Definition C.1) is the conceptual main result of this thesis. In fact, it represents the conceptual model and approach for the combination and integration of agents and workflows as desired in the goal of this thesis. The main components of the Agent-Activity approach are the Agent-Activity modelling construct (see Definition C.3) and the reference architecture (see Definition C.7). Both were described in Chapter 9. The Agent-Activity modelling construct combines sets of fundamental agent actions and basic workflow operations to represent an abstract task for an integrated entity that can have that entity act as an agent, a workflow or a hybrid of both. For the Agent-Activity modelling construct a specific system architecture is required, which was defined in the reference architecture of the Agent-Activity approach.

Paffin-*System prototype:* The technical main result of this thesis is the Paffin-System prototype (**P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System, see Definition C.10). It serves as a proof-of-concept of the Agent-Activity integration approach, thereby fulfilling the second, related part of the goal of this thesis. The Paffin-System was presented in Chapter 10, especially Section 10.2. On the technical level it can be considered as an extension of the Capa agent framework with additional workflow and hybrid management facilities. It achieved the status as a full-fledged framework for the creation of applications utilising an integration of agents and workflows.

Additional prototypes: In addition to the Paffin-System prototype, Chapter 10 also presented a number of additional prototypes that did not directly contribute to the Paffin-

System, yet realised and tested important ideas for the integration, the most prominent being the WorkBroker prototype.

Application prototypes: Chapter 11 dealt with application modelling and application prototypes developed in the Paffin-System. Both the discussions and prototypes represent results of this thesis. The prototypes showcased and substantiated the capabilities and mechanisms of the Paffin-System, while the discussions provided insight into how application modelling was achieved in an integration and where it could be utilised. For this chapter three prototypes were practically implemented. These prototypes are the producer-store-consumer prototype described in Section 11.3.1, the pizza collaboration prototype described in Section 11.3.2 and the Paose teaching support system described in Section 11.3.3.

Evaluation of Agent-Activity *approach and* Paffin-*System:* The final result is the overall evaluation of the Agent-Activity integration approach and the Paffin-System implementation. This evaluation is provided as the sum of the discussions in Chapters 11, 12 and 13. These discussions dealt with the relation of the results to the original integration vision, the generality of the approach, the synergies enabled by it, its strengths and open points and the general application areas and settings. A consideration of future work going forward from this thesis was also included.

To conclude this summary, the following chapter by chapter summary provides additional details.

Chapter 1 - Introduction introduced and motivated this thesis and the integration of agents and workflows pursued in it. It defined the hypothesis and goal of this thesis, as well as the main research questions.

Chapter 2 - Basics provided information about the conceptual and technical background of the thesis. It discussed basic Petri nets, reference nets and Renew, agent-orientation, the FIPA standards, Mulan, Capa, Paose, workflows and workflow management, workflow nets, inter-organisational workflows and the term "process".

Chapter 3 - State-of-the-Art presented an overview of the state-of-the-art for the research fields related to the integration of agents and workflows. These fields were agent-orientation, workflow modelling and research integrating agents and workflows. In the latter, an additional distinction was made for agent-based workflow management, workflow-based agent management and advanced integration efforts.

Chapter 4 - The Aspects of Structure and Behaviour considered the composition of software systems, especially w.r.t. the term architecture. From these considerations the basic notions of the aspects of structure and behaviour of a software system, which were central to this thesis, was developed with an emphasis on a Petri net-based view.

Chapter 5 - Structure Based on Agents concentrated on the structure of a software system, taking software agents as the main modelling construct into account. It defined the agent-oriented modelling perspective, which describes how to approach modelling a software system with agents as the core concept. Based on that modelling perspective the concept of the three fundamental agent actions were introduced. All agent behaviour can be considered, abstractly, as either sending a message, receiving a message or performing some internal action. These fundamental agent actions represented a core concept for the remainder of this thesis. Finally, the chapter also refined the basic notion of the structure of a software system into an improved definition that takes agents into accounts. That definition was provided in two versions, an abstract one and a concrete one.

Chapter 6 - Behaviour Based on Workflows concentrated on the behaviour of a software system, taking workflows and workflows nets as the main modelling construct

into account. Here, analogously to the previous chapter, the workflow-based modelling perspective was defined, which describes how to approach modelling a software system emphasising workflows as the core concept. That perspective also introduced the concept of the three basic workflow operations. Workflows consist of tasks and each task is requested and subsequently confirmed or cancelled. These three operations are valid for all tasks and are the behaviour/workflow counterpart to the fundamental agent actions. The chapter also refined the basic notion of the behaviour of a software system by providing an improved definition taking workflows/workflow nets into account. Again, both an abstract and a concrete version of the definition were presented.

Chapter 7 - Specifying an Integration directly built upon the previous chapters. Taking the results pertaining to structure and behaviour into account, it introduced the vision and specification of what a system featuring an integration of agents and workflows should look like. That integration vision was described in detail, with suitable applications areas also being discussed. Based on the integration vision a set of integration criteria were then developed. Each criterion described a property or mechanisms that a system should possess in order to satisfy the integration vision. The purpose of these criteria was to provide a more concrete basis on which to compare and evaluate possible integration approaches discussed in the following chapter. Finally, this chapter also discussed the goal and research questions of this thesis, originally defined in the introduction, more concretely. A set of requirements for the overall results was also defined.

Chapter 8 - Possible Integration Approaches presented and evaluated six possible integration approaches. "Integration via Management" allowed for agents and workflows to be managed concurrently by a unified management system. The "Intermediate Model" approach provided an internal model that was constantly translated into its agent and workflow aspects. The "Combined Model" approach provided a model which was both agent and workflow at the same time. "Integration via Actions and Operations" integrated agents and workflows on the lowest level, namely their actions and operations. Here, the execution of actions and operations defined if an entity was an agent, a workflow or both. "Nesting" integration utilised nets-within-nets principles by nesting agents in workflows and workflows in agents, thus realising the core concept of mutual incorporation. Lastly, "Integration via Synchronous Channels" utilised net synchronisation between separate agent and workflow models to realise the integration. In the end, the integration via agent actions and workflow operations evaluated best against the integration criteria and was selected for further refinement and implementation.

Chapter 9 - Integration via Agent-Activities presented the conceptual main result of this thesis, the Agent-Activity integration approach. Refined from the previous selection of integration via agent actions and workflow operations, the Agent-Activity integration approach realised the integration vision set out in earlier chapters. The core of the approach is the Agent-Activity modelling construct. Within the construct, agent actions and workflow operations can be freely combined. Agent-Activities are executed by integrated entities, the state of which is defined by the actions and operations executed within the Agent-Activities. In order to support the execution of Agent-Activities, the approach also provided a reference architecture. That reference architecture described how a system must be structured and what kind of management facilities it needs to provide. The chapter concluded with an evaluation of the approach, ensuring that the integration criteria were still met in the concrete and more elaborate approach.

Chapter 10 - Prototypes presented the practically implemented prototypes developed for this thesis. The main focus of this chapter was the **Processes and Agents For a Full INtegration**-System (Paffin-System) prototype, a proof-of-concept for the

Agent-Activity integration approach. Building on the base Capa framework, the Paffin-System extended the management capabilities to include workflow and hybrid management. To that end, many reference nets were added or edited from their original Capa versions. A number of additional prototypes were also shortly discussed in this chapter. The WorkBroker prototype realised workflow task principles for agent interactions. The PPB prototype transformed textual net representations into actual net instances. The integration net prototype was a first, limited attempt of realising the Agent-Activity approach. Finally, this chapter evaluated the Paffin-System against the conceptual Agent-Activity approach to substantiate that it was indeed a proof-of-concept.

Chapter 11 - Application Discussion discussed applications in the context of the Agent-Activity integration approach and the Paffin-System prototype. It discussed how to model applications in the Paffin-System and examined the technical maturity of the prototype. Then, it presented three application prototypes built with the Paffin-System framework. These prototypes realised different scenarios showcasing integration mechanisms and the usability of the Paffin-System. A number of application visions were also presented in this chapter. These visions have not yet been realised in the Paffin-System, but present concrete, possible, future applications. Finally, the chapter discussed inter-organisational contexts and the general application of the Agent-Activity approach and Paffin-System in this area.

Chapter 12 - Overarching Discussion Areas provided the general and overarching discussion of the results of this thesis. It related the achieved results to the integration vision, examined the generality of the Agent-Activity integration approach beyond the Mulan and workflow net bases, discussed the synergies that have been achieved by both the approach and the implementation, examined the strengths and remaining open points and also presented the general application areas in which an integration could be especially useful. The chapter also presented some possible future work going forward in the context of Agent-Activities and the Paffin-System, as well as related the results to the hypothesis, goal, research questions and requirements of this thesis.

Chapter 13 - Related Work compared and evaluated the results obtained in this thesis to the research contributions presented in the earlier state-of-the-art chapter. Its structure mirrored the previous chapter, with the evaluation dealing with the traditional agent and workflow fields, as well as different degrees of integrations between agents and workflows.

Chapter 14 - Summary summarised this thesis and its results.

Chapter 15 - Conclusion and Outlook concluded this thesis and provided a general outlook on possible future work.

15 Conclusion and Outlook

This thesis examined an integration of structure defined through agents and behaviour defined through workflows. As evaluated in Section 12.7 it successfully confirmed its hypothesis, achieved its goal and answered the posed research questions. Its main results are the Agent-Activity integration approach and its prototypical implementation in the Paffin-System (**P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System).

Some concrete directions for future work have already been discussed in Section 12.6. These points related to concrete aspects that, in part, are already actively being developed. As a general outlook, a number of research directions can be identified and envisioned:

Continued development of the Paffin-*System:* Probably the most obvious direction of future work is the continued development and refinement of the Paffin-System framework prototype. Section 12.6.4 already discussed some of the options. The goal of this future work should generally be to improve the prototype and provide more extensive options and support for application modelling and execution.

Global perspectives: The topmost level of the Agent-Activity reference architecture is the implicit system layer. It is not explicitly modelled when a system is created, but is rather generated through the elements it contains in the lower levels, i.e. the platform and management, Paffins and Agent-Activities. The system level provides a global view on the elements of the system. It contains all structural and behavioural aspects of each of those elements. Generating perspectives based on the system level is a promising course for future work. These perspectives may capture the dynamic relationships between the different elements of the system and their states. Integrated entities can dynamically switch between agent, workflow and hybrid. Therefore, their relationships to other integrated entities can become quite intricate and interesting. For, at the very least, monitoring and analysis reasons, capturing the global perspectives on these relationships is a very relevant and useful ability. Starting points for the generation of these global perspectives could be provided by the already existing monitoring tools for Capa agents. Also feasible is the incorporation of special platform entities that continuously gather and aggregate system data to provide and display for authorised users.

Cognitive concepts: Throughout the discussions of related work it is clear that other research efforts value cognitive concepts, as found in BDI agents, highly when integrating agents and workflows. Consequently, future work in the Agent-Activity context could also examine introducing cognitive concepts. Especially the goal concept is considered useful for system modelling. This kind of future research could be undertaken in the general Capa agent context as well. Those results could then be incorporated into the Paffin-System. However, conceptually researching the incorporation for the Agent-Activity provides the opportunity to fundamentally incorporate cognitive concepts instead of just technically. As discussed in Section 12.2, reasoning can generally be captured by the Agent-Activity approach through internal actions. However, if integrated entities were imbued with cognitive concepts some relations between the levels of the reference architecture may change. Goal delegation and

distribution could become management emphases that would have to be partitioned onto entities and platforms. Furthermore, these concepts may not necessarily require changes to the fundamental agent actions and therefore the Agent-Activity concept, yet modelling may still be improved if they were available *and* explicitly recognised in the Agent-Activities. In any case, cognitive concepts represent a relevant research direction for Agent-Activities that should be examined in future work.

Organisation concepts: Organisation concepts are also a relevant topic for system modelling, especially for inter-organisational contexts. Explicitly excluded from this thesis, they still represent an interesting addition. An incorporation of the organisation concepts as an extra, possibly virtual, level of the reference architecture is feasible. Technically, in the Paffin-System, the subordinate resources mechanisms could be used as a starting point for tests. Future work would have to examine the conceptual consequences and effects the incorporation of organisations would have.

Integration of web services: Web services are already incorporated into the Paffin-System using the Capa WebGateway. On the conceptual level, however, they are only a tool to be used by agent actions. A possible area of future work could examine their full incorporation into the Agent-Activity concept, though. Web services orchestration and workflow management are related fields so that an extension of the workflow aspects within the Agent-Activity or possibly even a replacement of workflow tasks for web service calls is feasible.

Modelling support, Paose *for* Paffins *and best practices:* Parts of a possible adaption of Paose for Paffins were already sketched out in Section 12.6.3. In general, modelling support for the conceptual and technical Agent-Activities should be a priority in future work. Adapting Paose and providing new and improved modelling tools are just a part of that. Best practices should be defined that guide modellers to better utilise the available synergies. For example, direct engine operations should be used whenever the workflow engine has data-, and therefore result-, autonomy. Automation of certain modelling aspects, like the connection of workflow task places, is another part of possible modelling support. Overall, modellers should be supported as much as possible, because, due to their expressiveness and versatility, the Agent-Activity approach and Paffin-System are also quite extensive and complex.

Holonic Paffins*:* Holonic systems, i.e. systems in systems, have been considered before in this thesis on the conceptual level regarding the agent-oriented modelling perspective. The topic was also touched on regarding the consideration of decision components as internal agents or internal entities. Overall, this represents an interesting research direction. Integrated entities could represent and model full (sub-)systems and still appear as a single individual entity to the rest of the system. This would constitute a powerful abstraction mechanism. In the Paffin-System, the sketched out subordinate resource mechanism, which enables a Paffin to be a supervisor and controller of other Paffins, can be considered as a starting point for this future work direction. Relating it to the previously mentioned incorporation of organisation concepts might also yield useful and beneficial results.

Large-scale applications: This thesis emphasised the creation and development of concepts and frameworks. Practical applications were also considered, albeit to a lesser degree and smaller scale. Future work therefore should implement applications built with the Paffin-System framework on a larger scale. Such applications would further substantiate the integration and support the discovery of hereto unknown properties and effects. The application visions presented in Section 11.4 are a promising starting

Figure 15.1: The achieved integration on different levels

point. Here, the PAOSE support system should be highlighted as it also relates to the previously discussed future work direction of modelling support and PAOSE for PAFFINS.

Validation, verification and testing: Validation, verification and testing are important possibilities for future work. The theoretical Petri net basis of the results presented in this thesis should be utilised to enable confirming assumptions about the created systems. This applies to both the framework *and* the applications built within it. Section 12.6.2 already sketched out some concrete points for this future work.

Net adaptivity, flexibility: The AGENT-ACTIVITY approach already features some built-in flexibility through abstracting from the internal process. This is also implemented in the PAFFIN-System. However, truly adaptive and flexible nets are still a desirable feature and should therefore be pursued in future work. Section 12.6.1 already presented an approach based on eHornets. In general, the ability to change nets at runtime would help in dealing with unforeseen circumstances. AGENT-ACTIVITIES or actions/operations could be added on the fly to react to certain issues. Providing mechanisms for net adaptivity is a research topic from the general RENEW and reference net context. However, suitably incorporating them into application systems of the PAFFIN-System requires additional concepts.

Altogether, while the research presented in this thesis is a success and progresses the general research area of integrating agents and workflows, it also opened up numerous new research opportunities. The AGENT-ACTIVITY integration approach and the PAFFIN-System provide an extensive, versatile and solid basis for these research opportunities, as well as for general future work related to the context of an integration.

Going back to the beginning of the thesis, one of the main motivations was to create a hybrid system in which neither structure nor behaviour governed the respective other aspect. This motivation was illustrated with the symbol of the philosophical Yin and Yang concept, which is now revisited. Figure 15.1 illustrates what has been achieved in this thesis (from right to left):

- Neither agent actions nor workflow operations are emphasised in the behaviour of integrated entities. Rather, both are available at any time, on the same modelling level and with the same priority. Here, neither the structural side (through agent actions) nor the behavioural side (through workflow operations) is given advantage so that neither one governs the other.
- An agent state is defined through AGENT-ACTIVITIES that contain only agent actions. A workflow state is defined through AGENT-ACTIVITIES containing only workflow

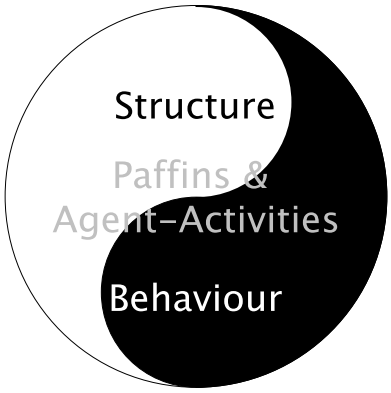

Figure 15.2: The achieved integration in total

operations. Lastly, a hybrid state is defined through Agent-Activities that contain both agent actions and workflow operations. This hybrid state also represents a new addition to the otherwise pure agent or workflow views. It realises the elements that are not available to the traditional views. Overall, all three options, i.e. agent, workflow and hybrid, can be used equally, concurrently and freely. It is easily possible to exchange an agent Agent-Activity with a hybrid or workflow Agent-Activity as long as the input and output are satisfied. The same holds true for any other variation of exchanging Agent-Activities. Therefore, there is also no governance of one type of Agent-Activity over the others.

– Paffin entities are agents, workflows, both or something in between, depending on the Agent-Activities they execute. A Paffin is an agent, if it executes agent Agent-Activities. It is a workflow, if it executes workflow Agent-Activities. It is a hybrid, if it executes hybrid Agent-Activities. It is both, if it concurrently executes any combination of the basic states. No matter the state of an integrated Paffin entity, it can always change its state with the next execution of an Agent-Activity, it can always interact with any other Paffin in any other state and it is always available in the same way in its execution environment. Here, too, there is no governance of any kind that stems from a preference or bias towards either structure/agents or behaviour/workflows.

Overall, the imbalances and prepossessions towards either structure or behaviour, which are found in traditional agent and workflow modelling, have been eliminated in the Agent-Activity integration approach and its implementation in the Paffin-System. Neither the individual actions and operations, nor the state-defining Agent-Activities, nor the entities themselves favour or emphasise structure or behaviour over the other one. This is ultimately illustrated in Figure 15.2. With the results achieved in this thesis it has become possible to create a hybrid system in which structure and behaviour are modelled in a unified and equal way. The core components of this hybrid system are the integrated Paffin entities and the Agent-Activities they execute.

In conclusion, the integration of structure and behaviour through agents and workflows is a relevant research topic. As was extensively discussed throughout this thesis, there are many benefits and synergies to utilise for modelling systems. This thesis did achieve what it set out to achieve. It introduced a suitable, concrete model and approach for the integration of agents and workflows and established and substantiated the technical feasibility and beneficial properties.

Part F

Appendices

Part F contains the thesis appendices. Appendix I presents an excerpt from the overall bibliography containing only the publications of the author of this thesis. Next, Appendix II contains the key term glossary. Finally, Appendix III lists the different acronyms used throughout this document.

The purpose of this part is to provide auxiliary and supplemental information about the thesis.

I Author's Publications

This appendix lists all publications of the author of this thesis. It represents an excerpt from the overall bibliography. The list is ordered by year of publication and within each year by the bibliography key as used in this thesis.

2009

[**Markwardt et al., 2009b**] Markwardt, K.; Moldt, D.; and Wagner, T. (2009b). "Net Agents for Activity Handling in a WFMS". In Freytag, T. and Eckleder, A., editors, *16th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2009, Karlsruhe, Germany, September 25, 2009, Proceedings*, CEUR Workshop Proceedings.

[**Wagner, 2009a**] Wagner, T. (2009a). "A Centralized Petri Net- and Agent-based Workflow Management System". In Duvigneau, M. and Moldt, D., editors (2009). *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, number FBI-HH-B-290/09 in Bericht, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Informatics, pages 29–44.

[**Wagner, 2009b**] Wagner, T. (2009b). "Modeling of a Centralized Petri Net- and Agent-based Workflow Management System". Bachelor's thesis (equiv.), University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[**Wagner, 2009c**] Wagner, T. (2009c). "Prototypische Realisierung einer Integration von Agenten und Workflows". Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

2010

[**Adameit et al., 2010a**] Adameit, S.; Betz, T.; Cabac, L.; Hars, F.; Hewelt, M.; Köhler-Bußmeier, M.; Moldt, D.; Popov, D.; Quenum, J.; Theilmann, A.; Wagner, T.; Warns, T.; and Wüstenberg, L. (2010a). "Herold - Agent-oriented, Policy-based Network Security Management". In *Future Security 2010, 5th Security Research Conference, Berlin.*

[**Adameit et al., 2010b**] Adameit, S.; Betz, T.; Cabac, L.; Hars, F.; Hewelt, M.; Köhler-Bußmeier, M.; Moldt, D.; Popov, D.; Quenum, J.; Theilmann, A.; Wagner, T.; Warns, T.; and Wüstenberg, L. (2010b). "Modelling Distributed Network Security in a Petri Net and Agent-based Approach". In Dix, J. and Witteveen, C., editors, *Multiagent System Technologies. 8th German Conference, MATES 2010, Leipzig, Germany, September 27-28, 2010. Proceedings*, volume 6251 of *Lecture Notes in Artificial Intelligence*, pages 209–220, Berlin Heidelberg New York. Springer-Verlag.

[**Mascheroni et al., 2010**] Mascheroni, M.; Wagner, T.; and Wüstenberg, L. (2010). "Verifying Reference Nets By Means of Hypernets: A Plugin for Renew". In *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, pages 39–54.

[**Moldt et al., 2010**] Moldt, D.; Quenum, J.; Reese, C.; and Wagner, T. (2010). "Improving a Workflow Management System with an Agent Flavour". In Duvigneau, M. and Moldt, D., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, number FBI-HH-B-294/10 in Bericht, pages 55–70, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Informatics.

[**Wagner, 2010**] Wagner, T. (2010). "Prototypische Realisierung einer Integration von Agenten und Workflows". In *Informatiktage 2010 - Fachwissenschaftlicher Informatik-Kongress 19. und 20. März 2010, B-IT Bonn-Aachen International Center for Information Technology in Bonn*, volume S-9 of *LNI*, pages 101–104. GI.

2011

[**Hewelt et al., 2011**] Hewelt, M.; Wagner, T.; and Cabac, L. (2011). "Integrating Verification into the PAOSE Approach". In Duvigneau, M.; Moldt, D.; and Hiraishi, K., editors, *Petri Nets and Software Engineering. International Workshop PNSE'11, Newcastle upon Tyne, UK, June 2011. Proceedings*, volume 723 of *CEUR Workshop Proceedings*, pages 124–135. CEUR-WS.org.

[**Wagner and Moldt, 2011**] Wagner, T. and Moldt, D. (2011). "Approaching the Integration of Agents and Workflows". In Bergenthum, R. and Desel, J., editors, *Algorithmen und Werkzeuge für Petrinetze. 18. Workshop AWPN 2011, Hagen, September 2011. Tagungsband*, pages 55–61.

2012

[**Bendoukha and Wagner, 2012**] Bendoukha, S. and Wagner, T. (2012). "Cloud Transition: Integrating Cloud Calls into Workflow Petri Nets". In Cabac, L.; Duvigneau, M.; and Moldt, D., editors (2012). *Petri Nets and Software Engineering. International Workshop PNSE'12, Hamburg, Germany, June 2012. Proceedings*, volume 851 of *CEUR Workshop Proceedings*. CEUR-WS.org, pages 215–216.

[**Bendoukha et al., 2012**] Bendoukha, S.; Moldt, D.; and Wagner, T. (2012). "Boxing Business Interaction Diagrams Within Classical Workflow Transitions Preserving Autonomy of Actors". In *AWPN 2012: Algorithmen und Werkzeuge für Petrinetze*.

[**Wagner, 2012**] Wagner, T. (2012). "Agentworkflows for Flexible Workflow Execution". In Cabac, L.; Duvigneau, M.; and Moldt, D., editors (2012). *Petri Nets and Software Engineering. International Workshop PNSE'12, Hamburg, Germany, June 2012. Proceedings*, volume 851 of *CEUR Workshop Proceedings*. CEUR-WS.org, pages 199–214.

[**Wagner et al., 2012**] Wagner, T.; Quenum, J.; Moldt, D.; and Reese, C. (2012). "Providing an Agent Flavored Integration for Workflow Management". In Jensen, K.; Donatelli, S.; and Kleijn, J., editors, *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900 of *Lecture Notes in Computer Science*, pages 243–264. Springer-Verlag, Berlin Heidelberg New York.

2013

[**Bendoukha et al., 2013**] Bendoukha, S.; Moldt, D.; and Wagner, T. (2013). "Enabling Cooperation in an Inter-Cloud Environment: An Agent-based Approach". In Morvan, F.; Tjoa, A. M.; and Wagner, R., editors, *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on*, pages 217–221. IEEE Computer Society.

[**Betz et al., 2013**] Betz, T.; Cabac, L.; Duvigneau, M.; Wagner, T.; and Wester-Ebbinghaus, M. (2013). "Integrating Web Services in Petri Net-based Agent Applications". In Moldt, D. and Rölke, H., editors, *Petri Nets and Software Engineering. International Workshop PNSE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 97–116. CEUR-WS.org.

[**Wagner and Cabac, 2013**] Wagner, T. and Cabac, L. (2013). "Advantages of a Full Integration between Agents and Workflows". In Moldt, D., editor, *Modeling and Business Environments MODBE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 353–354. CEUR-WS.org.

2014

[**Betz et al., 2014**] Betz, T.; Cabac, L.; Duvigneau, M.; Wagner, T.; and Wester-Ebbinghaus, M. (2014). "Software Engineering with Petri Nets: A Web Service and Agent Perspective". *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, pages 41–61.

2015

[**Bendoukha and Wagner, 2015**] Bendoukha, S. and Wagner, T. (2015). "Improving Performance of Complex Workflows: Investigating Moving Net Execution to the Cloud". In Moldt, D.; Rölke, H.; and Störrle, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings*, volume 1372 of *CEUR Workshop Proceedings*, pages 171–189. CEUR-WS.org.

[**Wagner and Moldt, 2015a**] Wagner, T. and Moldt, D. (2015a). "Integrating Agent Actions and Workflow Operations". In Müller, J. P.; Ketter, W.; Kaminka, G.; Wagner, G.; and Bulling, N., editors, *Multiagent System Technologies - 13th German Conference, MATES 2015, Cottbus, Germany, September 28-30, 2015, Revised Selected Papers*, volume 9433 of *Lecture Notes in Computer Science*, pages 61–78. Springer.

[**Wagner and Moldt, 2015b**] Wagner, T. and Moldt, D. (2015b). "Workflow Management Principles for Interactions Between Petri Net-Based Agents". In Devillers, R. and

Valmari, A., editors, *Application and Theory of Petri Nets and Concurrency - 36th International Conference, Petri Nets 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, volume 9115 of *Lecture Notes in Computer Science*, pages 329–349. Springer-Verlag.

2016

[**Wagner and Moldt, 2016**] Wagner, T. and Moldt, D. (2016). "Revisiting Workflow Interactions for Petri Net-Based Agents (Extended Abstract)". In Mendling, J. and Rinderle-Ma, S., editors, *Proceedings of the 7th International Workshop on Enterprise Modeling and Information Systems Architectures, EMISA 2016: Fachgruppentreffen der GI-Fachgruppe Entwicklungsmethoden für Informationssysteme und deren Anwendung, Vienna, Austria, October 3-4, 2016.*, volume 1701 of *CEUR Workshop Proceedings*, pages 9–12. CEUR-WS.org.

[**Wagner et al., 2016a**] Wagner, T.; Moldt, D.; and Köhler-Bußmeier, M. (2016a). "From eHornets to Hybrid Agent and Workflow Systems". In Cabac, L.; Kristensen, L. M.; and Rölke, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'16, Toruń, Poland, June 20-21, 2016. Proceedings*, volume 1591 of *CEUR Workshop Proceedings*, pages 307–326. CEUR-WS.org.

[**Wagner et al., 2016b**] Wagner, T.; Schmitz, D.; and Moldt, D. (2016b). "Paffin: Implementing an Integration of Agents and Workflows". In *Proceedings of the 14th European Conference on Multi-Agent Systems (2016)*, volume 14, page tba. Springer. Postproceedings to be published in 2017.

II Key Terms Glossary

This glossary lists the key terms used in this thesis. The key terms are heavily used throughout the different parts and require special attention. Therefore, they are repeated here. They are arranged according to coarse topics and within those topics ordered according to their order of appearance in the thesis.

The topics into which the key terms are categorised are:

- Agents and Structure
- Workflows and Behaviour
- General Integration
- The Agent-Activity Integration Approach
- The Paffin-System

Agents and Structure

Key Term Definition A.1 (Agent)**.** "*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.*" [Wooldridge, 2009, p. 21]

(from Section 2.2.1, Page 17)

Key Term Definition B.1 (Fundamental Agent Actions)**.** The three fundamental agent actions are: receiving a message, sending a message and performing an internal action. These three agent actions can fully describe the behaviour of an agent.

(from Section 5.2, Page 102)

Key Term Definition B.2 (Abstract Definition of Structure)**.** *The structure of a software system is constituted by (abstract) agents and the relations between them.*

(from Section 5.3, Page 103)

Key Term Definition B.3 (Concrete Definition of Structure)**.** *The structure of a multi-agent system is constituted by agents and platforms and the internal and external relations between them.*

(from Section 5.3, Page 104)

Workflows and Behaviour

Key Term Definition A.2 (Workflow)**.** The automation and facilitation of a process, in whole or part, during which data and tasks are passed from one participant to another for action, according to a set of procedural rules.

(from Section 2.3.1, Page 32)

Key Term Definition A.3 (Workflow Participant/Resource)**.** "*A resource which performs partially, or in full, the work represented by a workflow activity instance.*" [Hollingsworth, 1995, p. 54]

(from Section 2.3.1, Page 33)

Key Term Definition A.4 (Task)**.** "*A task is a logical unit of work. It is indivisible and is thus always carried out in full. If anything goes wrong during the performance of a task, then we must return to the beginning of the entire task.*" [van der Aalst and van Hee, 2002, p. 32]

(from Section 2.3.1, Page 33)

Key Term Definition A.5 (Workitem)**.** "*A workitem is the combination of a case and a task which is just about to be carried out. A workitem is created as soon as the state of a case allows it. We can thus regard a workitem as an actual piece of work which may be carried out.*" (adapted) [van der Aalst and van Hee, 2002, p. 33]

(from Section 2.3.1, Page 34)

Key Term Definition A.6 (Activity)**.** "*The term activity refers to the actual performance of a workitem. As soon as work begins upon the workitem, it becomes an activity.*" (adapted) [van der Aalst and van Hee, 2002, p. 33]

(from Section 2.3.1, Page 34)

Key Term Definition A.7 (Workflow Engine)**.** "*A software service or "engine" that provides the run time execution environment for a workflow instance.*" [Hollingsworth, 1995, p. 22]

(from Section 2.3.1, Page 35)

Key Term Definition A.8 (Process)**.** "*A **process** consists of **tasks** that have to be executed. These tasks can be in some order (**sequentially**), stating that one task can only be executed after the execution of another task is finished. If two tasks are not ordered, then they can be executed **concurrently**. Tasks can also be alternative, that is, if one task is executed, then the other task is not executed and vice versa. Tasks can be executed more than once in general. A process can be in different states. A process starts with an initial state (which is not necessarily unique) and might end with a final state (which is also not necessarily unique). Usually, it passes through several intermediate states.*" [Desel, 2005, p. 157]

(from Section 2.4, Page 41)

Key Term Definition B.4 (Basic Workflow Operations)**.** The three basic workflow operations are: Requesting a workitem, confirming an activity and cancelling an activity. Through these basic workflow operations a workflow can be described in its entirety.

(from Section 6.2, Page 118)

Key Term Definition B.5 (Abstract Definition of Behaviour)**.** *The behaviour of a software system is constituted by its processes and the relations between them.*

(from Section 6.3, Page 119)

Key Term Definition B.6 (Concrete Definition of Behaviour)**.** *The behaviour of a workflow system is constituted by a set of workflow nets. These workflow nets are related to one another and internally constructed in such a way that they capture and cover the distinct tasks of the system in the correct order.*

(from Section 6.3, Page 119)

General Integration

Key Term Definition B.7 (Integrated Entity)**.** An integrated entity (or entity for short) is the modelling construct within an integration of agents and workflows. It is capable of representing the different concepts that are required in such an integration. This means it can dynamically act as and be interacted with as an agent, a workflow, a hybrid of both or something in between.

(from Section 7.1.2, Page 129)

While not defined as key terms, repeating the integration criteria from Section 7.3 is reasonable in this current context as well:

Criterion MC1: Integrated Entities

The main modelling constructs are integrated entities,
which can be agent, workflow, both or something in between.

(from Section 7.3, Page 137)

Criterion MC2: Entity Dynamic

Entities are dynamic in their states and can freely switch between them
in order to represent different concepts.

(from Section 7.3, Page 138)

Criterion MC3: Logical Entities

All elements of the system can be considered as logical entities.

(from Section 7.3, Page 139)

Criterion SB1: Structure of the System

Structure is constituted by entities considered as agents and platforms.

(from Section 7.3, Page 140)

Criterion SB2: Behaviour of the System

Behaviour is constituted by entities considered as workflows.

(from Section 7.3, Page 140)

Criterion SB3: Mutual Incorporation I

Entities as agents interact via workflows.

(from Section 7.3, Page 141)

Criterion SB4: Mutual Incorporation II

Entities as workflows are executed by agents.

(from Section 7.3, Page 141)

Criterion ToP1: Properties and Mechanisms

Entities can exhibit agent and workflow properties and mechanisms.

(from Section 7.3, Page 142)

Criterion ToP2: Agent Actions and Workflow Operations

Entities perform fundamental agent actions and basic workflow operations.

(from Section 7.3, Page 143)

Criterion ToP3: Regular Multi-Agent Systems

Entities as agents can build regular multi-agent systems.

(from Section 7.3, Page 144)

Criterion ToP4: Regular Workflow Systems

Entities as workflows can build regular workflow systems.

(from Section 7.3, Page 144)

Criterion Mgt1: Functionality

Functionality is distributed among entities that can have different states.

(from Section 7.3, Page 145)

Criterion Mgt2: Hierarchies

Entities in different states can form logical hierarchies.

(from Section 7.3, Page 146)

Criterion Pra: Practicability

The integration approach is practically and technically feasible.

(from Section 8.1, Page 156)

The Agent-Activity Integration Approach

Key Term Definition C.1 (Agent-Activity Integration Approach)**.** The Agent-Activity integration approach is a concrete approach to integrate agents and workflows. It consists of two main components, the Agent-Activity concept (see Definition C.2)and the Agent-Activity reference architecture (see Definition C.7).

(from Section 9.1, Page 184)

Key Term Definition C.2 (Agent-Activity Concept)**.** The Agent-Activity concept combines agent actions and workflow operations into an abstract task that is executed by an integrated entity. Depending on the actions and operations of the Agent-Activities it executes, an integrated entity can be agent, workflow, both or something in between.

(from Section 9.1, Page 184)

Key Term Definition C.3 (Agent-Activity Construct)**.** The Agent-Activity construct (or Agent-Activity modelling construct) is a concrete model and realisation of the Agent-Activity concept.

(from Section 9.1, Page 184)

Key Term Definition C.4 (Agent-Activity)**.** The term Agent-Activity refers to an instance of the Agent-Activity concept or construct.

(from Section 9.1, Page 184)

Key Term Definition C.5 (Internal Process of an Agent-Activity)**.** The internal process of an Agent-Activity is the ordered set of actions and operations the Agent-Activity combines and is able to execute. It represents a process in the sense of Definition A.8.

(from Section 9.1.1, Page 188)

Key Term Definition C.6 (Agent-Activity-Object (AAO))**.** The Agent-Activity-Object (AAO) completely encapsulates the substance of an Agent-Activity instance. It keeps the internal process, runtime state, parameters and any other data pertaining to an Agent-Activity instance during that instance's life-cycle.

(from Section 9.1.2, Page 192)

Key Term Definition C.7 (Agent-Activity Reference Architecture)**.** The Agent-Activity reference architecture describes how a system needs to be constructed that features Agent-Activities to realise the integration of agents and workflows. It consists of four nested levels, from the bottom: Process-protocols, integrated entities, platform and management, system.

(from Section 9.2, Page 195)

Key Term Definition C.8 (Process-protocol)**.** Process-protocols are larger behaviours of integrated entities that combine Agent-Activities into ordered processes.

(from Section 9.2, Page 196)

Key Term Definition C.9 (Backend)**.** The (technical) backend of an integrated entity provides the internal management facilities for each integrated entity in the Agent-Activity reference architecture. It contains management functionality to support the execution and triggering of Agent-Activities, as well as of all agent actions and workflow operations (in both engine and resource variants).

(from Section 9.2, Page 198)

The Paffin-System

Key Term Definition C.10 (Paffin-System)**.** The **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration-System (Paffin-System) is a modelling framework and technical implementation of the Agent-Activity integration approach.

(from Section 10.2, Page 207)

Key Term Definition C.11 (Paffin Entity)**.** A Paffin entity (or integrated Paffin entity or Paffin for short) is the implementation of an integrated entity in the Paffin-System.

(from Section 10.2, Page 207)

Key Term Definition C.12 (Agent-Activity-Transition)**.** The Agent-Activity-Transition implements the Agent-Activity concept and Agent-Activity construct as a single transition in a process-protocol of the Paffin-System. The Agent-Activity-Transition is translated for execution into the full and complex Agent-Activity net structure of the Paffin-System, although it maintains the singular transition representation in the GUI.

(from Section 10.2.4, Page 225)

Key Term Definition C.13 (WorkBroker principle)**.** The WorkBroker principle allows agents to interact using workflow and task principles. Agents as engines execute behaviour containing workflow tasks that are intended to be executed by other agents as resources. The connection between agents as engines and agents as resources is realised through a special, platform-wide subsystem called the intermediate system.

(from Section 10.3.2, Page 286)

III List of Acronyms

This list gathers the acronyms used throughout this thesis. Note that, in order to keep this list concise and thereby more usable, acronyms that are used only in singular and very limited passages of the thesis have been omitted. This relates mostly to abbreviations from the basic background and from research in the state-of-the-art.

AAO . . . . . Agent-Activity-Object
ACL . . . . . (FIPA) Agent communication language
AgAc . . . . . Agent-Activity
AgWFMS . . . Agent-based workflow management systems
AIP . . . . . Agent interaction protocol
AMS . . . . . Agent management system
AOSE . . . . . Agent-oriented software engineering
ARM . . . . . Agent Role Model
BDI . . . . . Beliefs-Desires-Intentions
BP . . . . . Business process
BPM . . . . . Business process management
BPMN . . . . Business Process Model and Notation
Capa . . . . . Concurrent Agent Platform Architecture
CDD . . . . . Coarse design diagram
CRUD . . . . . Create, read, update and delete
DC . . . . . Decision component
DF . . . . . Directory facilitator
FIPA . . . . . Foundation for Intelligent Physical Agents
GPMN . . . . Goal-oriented business process notation
GUI . . . . . Graphical user interface
IEEE . . . . . Institute of Electrical and Electronics Engineers
KB . . . . . Knowledge base
MC . . . . . Modelling construct (used for integration criteria)
Mgt . . . . . Management (used for integration criteria)
Mulan . . . . . Multi-Agent Nets
Paffin . . . . . **P**rocesses and **A**gents **F**or a **F**ull **IN**tegration
Paose . . . . . Petri Net-Based Agent- and Organisation-oriented Software Engineering
PPB . . . . . Paose Process for Billboard

PrPr . . . . . Process-protocol
RBAC . . . . . Role-based access control
Renew . . . . . Reference Net Workshop
SB . . . . . Structure and behaviour (used for integration criteria)
SL . . . . . (FIPA) Semantic Language
ToP . . . . . Transfer of properties (used for integration criteria)
WAW . . . . . Workflow administration workflow
WFES . . . . . Workflow Enactment Service
WfMC . . . . . Workflow Management Coalition
WFMS . . . . Workflow management system

Bibliography

[Accorsi and Lehmann, 2012] Accorsi, R. and Lehmann, A. (2012). "Automatic Information Flow Analysis of Business Process Models". In Barros, A. P.; Gal, A.; and Kindler, E., editors, *Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, 2012. Proceedings*, volume 7481 of *Lecture Notes in Computer Science*, pages 172–187. Springer.

[Adam et al., 2016] Adam, C.; Danet, G.; Thangarajah, J.; and Dugdale, J. (2016). "BDI Modelling and Simulation of Human Behaviours in Bushfires". In Díaz, P.; Saoud, N. B. B.; Dugdale, J.; and Hanachi, C., editors, *Information Systems for Crisis Response and Management in Mediterranean Countries - Third International Conference, ISCRAM-med 2016, Madrid, Spain, October 26-28, 2016, Proceedings*, volume 265 of *Lecture Notes in Business Information Processing*, pages 47–61.

[Adameit et al., 2010a] Adameit, S.; Betz, T.; Cabac, L.; Hars, F.; Hewelt, M.; Köhler-Bußmeier, M.; Moldt, D.; Popov, D.; Quenum, J.; Theilmann, A.; Wagner, T.; Warns, T.; and Wüstenberg, L. (2010a). "Herold - Agent-oriented, Policy-based Network Security Management". In *Future Security 2010, 5th Security Research Conference, Berlin.*

[Adameit et al., 2010b] Adameit, S.; Betz, T.; Cabac, L.; Hars, F.; Hewelt, M.; Köhler-Bußmeier, M.; Moldt, D.; Popov, D.; Quenum, J.; Theilmann, A.; Wagner, T.; Warns, T.; and Wüstenberg, L. (2010b). "Modelling Distributed Network Security in a Petri Net and Agent-based Approach". In Dix, J. and Witteveen, C., editors, *Multiagent System Technologies. 8th German Conference, MATES 2010, Leipzig, Germany, September 27-28, 2010. Proceedings*, volume 6251 of *Lecture Notes in Artificial Intelligence*, pages 209–220, Berlin Heidelberg New York. Springer-Verlag.

[Albreshne et al., 2009] Albreshne, A.; Fuhrer, P.; and Pasquier, J. (2009). "Web Services Orchestration and Composition: Case Study of Web services Composition". Department of Informatics Internal Working Paper 09-03, University of Fribourg, Switzerland.

[Alfonso-Cendón et al., 2016] Alfonso-Cendón, J.; de Alba, J. M. F.; Fuentes-Fernández, R.; and Pavón, J. (2016). "Implementation of context-aware workflows with multi-agent systems". *Neurocomputing*, 176, pages 91–97.

[Anseeuw et al., 2016] Anseeuw, J.; van Seghbroeck, G.; Volckaert, B.; and Turck, F. D. (2016). "Design Time Validation for the Correct Execution of BPMN Collaborations". In Cardoso, J. S.; Ferguson, D.; Muñoz, V. M.; and Helfert, M., editors, *CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Volume 1, Rome, Italy, April 23-25, 2016.*, pages 49–58. SciTePress.

[Arevalo et al., 2016] Arevalo, C.; Cuaresma, M. J. E.; Ramos, I. M.; and Domínguez-Muñoz, M. (2016). "A metamodel to integrate business processes time perspective in BPMN 2.0". *Information & Software Technology*, 77, pages 17–33.

[Arias et al., 2016] Arias, M.; Rojas, E.; Lee, J.; Munoz-Gama, J.; and Sepúlveda, M. (2016). "ResRec: A Multi-criteria Tool for Resource Recommendation". In Azevedo, L. and Cabanillas, C., editors, *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.*, volume 1789 of *CEUR Workshop Proceedings*, pages 17–22. CEUR-WS.org.

[Arunagiri and Ramachandran, 2015] Arunagiri, A. and Ramachandran, P. (2015). "Data perspectives of workflow schema evolution". *Business Proc. Manag. Journal*, 21(1), pages 172–189.

[Austin, 1962] Austin, J. (1962). *How To Do Things With Words.* Oxford University Press, Oxford.

[Ayala et al., 2016] Ayala, I.; Horcas, J. M.; Amor, M.; and Fuentes, L. (2016). "Using Models at Runtime to Adapt Self-managed Agents for the IoT". In Klusch, M.; Unland, R.; Shehory, O.; Pokahr, A.; and Ahrndt, S., editors, *Multiagent System Technologies - 14th German Conference, MATES 2016, Klagenfurt, Österreich, September 27-30, 2016. Proceedings*, volume 9872 of *Lecture Notes in Computer Science*, pages 155–173. Springer.

[Aye and Tun, 2005] Aye, T. and Tun, K. M. L. (2005). "A Collaborative Mobile Agent-based Workflow System". In *6th Asia-Pacific Symposium on Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings.*, pages 59–65.

[Azzalini, 2017] Azzalini, L. (2017). "Sichtung, Diskussion und Nutzung allgemeiner Verifikationsverfahren für Petrinetze und prototypische Umsetzungen ausgewählter Verfahren im Werkzeug Renew". Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Badham, 2015] Badham, J. (2015). "An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NETLogo *by Uri Wilensky and William Rand*". *J. Artificial Societies and Social Simulation*, 18(4).

[Badica et al., 2016] Badica, A.; Badica, C.; Brezovan, M.; and Ivanovic, M. (2016). "Simulation of Dynamic Systems Using BDI Agents: Initial Steps". In Badica, C.; Fallah-Seghrouchni, A. E.; Beynier, A.; Camacho, D.; Herpson, C.; Hindriks, K. V.; and Novais, P., editors, *Intelligent Distributed Computing X - Proceedings of the 10th International Symposium on Intelligent Distributed Computing - IDC 2016, Paris, France, October 10-12 2016*, volume 678 of *Studies in Computational Intelligence*, pages 23–33.

[Baeyens, 2013] Baeyens, T. (2013). "BPM in the Cloud". In Daniel, F.; Wang, J.; and Weber, B., editors, *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, pages 10–16. Springer.

[Barker and Mann, 2006] Barker, A. and Mann, R. G. (2006). "Flexible Service Composition". In Klusch, M.; Rovatsos, M.; and Payne, T. R., editors, *Cooperative Information Agents X, 10th International Workshop, CIA 2006, Edinburgh, UK, September 11-13, 2006, Proceedings*, volume 4149 of *Lecture Notes in Computer Science*, pages 446–460. Springer.

[Barker and van Hemert, 2008] Barker, A. and van Hemert, J. (2008). "Scientific Workflow: A Survey and Research Directions". In Wyrzykowski, R.; Dongarra, J.; Karczewski, K.; and Wasniewski, J., editors, *Parallel Processing and Applied Mathematics: 7th International Conference, PPAM 2007, Gdansk, Poland, September 9-12, 2007 Revised Selected Papers*, pages 746–753. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Barreto et al., 2007] Barreto, C.; Bullard, V.; Erl, T.; Evdemon, J.; Jordan, D.; Kand, K.; König, D.; Moser, S.; Stout, R.; Ten-Hove, R.; Trickovic, I.; van der Rijn, D.; and Yiu, A. (2007). *Web Services Business Process Execution Language Version 2.0.* OASIS.

[Bass et al., 2012] Bass, L.; Clements, P.; and Kazman, R. (2012). *Software Architecture in Practice.* Addison-Wesley Professional, 3rd edition.

[Beck, 1999] Beck, K. (1999). "Embracing change with extreme programming". *Computer*, 32(10), pages 70–77.

[Becker and Moldt, 1993] Becker, U. and Moldt, D. (1993). "Object-Oriented Concepts for Coloured Petri Nets". In IEEE, editor, *Conference Proceedings, IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 279–286, Le Touquet, France. IEEE.

[Bekey, 2005] Bekey, G. A. (2005). *Autonomous Robots: From Biological Inspiration to Implementation and Control (Intelligent Robotics and Autonomous Agents).* The MIT Press.

[Bellifemine et al., 1999] Bellifemine, F.; Poggi, A.; and Rimassa, G. (1999). "JADE - A FIPA-compliant agent framework". In *Proceedings of the Practical Applications of Intelligent Agents.*

[Bellifemine et al., 2000] Bellifemine, F.; Poggi, A.; and Rimassa, G. (2000). "Developing Multi-agent Systems with JADE". In Castelfranchi, C. and Lespérance, Y., editors, *Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, ATAL 2000, Boston, MA, USA, July 7-9, 2000, Proceedings*, volume 1986 of *Lecture Notes in Computer Science*, pages 89–103. Springer.

[Bendoukha, 2016] Bendoukha, S. (2016). *Multi-Agent Approach for Managing Workflows in an Inter-Cloud Environment*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg. Available at `http://ediss.sub.uni-hamburg.de/volltexte/2017/8241/` (last accessed May 28th, 2017).

[Bendoukha et al., 2012] Bendoukha, S.; Moldt, D.; and Wagner, T. (2012). "Boxing Business Interaction Diagrams Within Classical Workflow Transitions Preserving Autonomy of Actors". In *AWPN 2012: Algorithmen und Werkzeuge für Petrinetze*.

[Bendoukha et al., 2013] Bendoukha, S.; Moldt, D.; and Wagner, T. (2013). "Enabling Cooperation in an Inter-Cloud Environment: An Agent-based Approach". In Morvan, F.; Tjoa, A. M.; and Wagner, R., editors, *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on*, pages 217–221. IEEE Computer Society.

[Bendoukha and Wagner, 2012] Bendoukha, S. and Wagner, T. (2012). "Cloud Transition: Integrating Cloud Calls into Workflow Petri Nets". In [Cabac et al., 2012], pages 215–216.

[Bendoukha and Wagner, 2015] Bendoukha, S. and Wagner, T. (2015). "Improving Performance of Complex Workflows: Investigating Moving Net Execution to the Cloud". In Moldt, D.; Rölke, H.; and Störrle, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings*, volume 1372 of *CEUR Workshop Proceedings*, pages 171–189. CEUR-WS.org.

[Benmerzoug, 2015] Benmerzoug, D. (2015). "Towards AiP as a Service: An Agent Based Approach for Outsourcing Business Processes to Cloud Computing Services". *IJISSS*, 7(2), pages 1–17.

[Benner-Wickner et al., 2015] Benner-Wickner, M.; Koop, W.; Book, M.; and Gruhn, V. (2015). "Supporting Adaptive Case Management Through Semantic Web Technologies". In Reichert, M. and Reijers, H. A., editors, *Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers*, volume 256 of *Lecture Notes in Business Information Processing*, pages 65–77. Springer.

[Bergenthum et al., 2015] Bergenthum, R.; Irgang, T.; and Meis, B. (2015). "Folding Example Runs to a Workflow Net". In van der Aalst, W. M. P.; Bergenthum, R.; and Carmona, J., editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015.*, volume 1371 of *CEUR Workshop Proceedings*, pages 62–77. CEUR-WS.org.

[Bergenti et al., 2012a] Bergenti, F.; Caire, G.; and Gotta, D. (2012a). "Interactive Workflows with WADE". In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'12), IEEE 21st International Workshop on*. IEEE.

[Bergenti et al., 2012b] Bergenti, F.; Caire, G.; and Gotta, D. (2012b). "Supporting user-centric business processes with WADE". In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., editors, *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1435–1436. IFAAMAS.

[Bergenti et al., 2013] Bergenti, F.; Caire, G.; and Gotta, D. (2013). "Mission-Critical Business Process Management with WADE". In Giordano, L.; Montani, S.; and Dupré, D. T., editors, *Proceedings of the Workshop AI Meets Business Processes 2013 co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 6, 2013.*, volume 1101 of *CEUR Workshop Proceedings*, pages 51–60. CEUR-WS.org.

[Bergenti et al., 2014] Bergenti, F.; Caire, G.; and Gotta, D. (2014). "Agents on the Move: JADE for Android Devices". In Santoro, C. and Bergenti, F., editors, *Proceedings of the XV Workshop "Dagli Oggetti agli Agenti", Catania, Italy, Sept. 25-26, 2014.*, volume 1260 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Best and Devillers, 1987] Best, E. and Devillers, R. R. (1987). "Sequential and Concurrent Behaviour in Petri Net Theory". *Theoretical Computer Science*, 55(1), pages 87–136.

[Betz, 2011] Betz, T. (2011). "Entwicklung eines Rahmenwerks für webbasierte Agentendienste – Modellierung auf Basis von Petrinetzen". Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Betz et al., 2013] Betz, T.; Cabac, L.; Duvigneau, M.; Wagner, T.; and Wester-Ebbinghaus, M. (2013). "Integrating Web Services in Petri Net-based Agent Applications". In Moldt, D. and Rölke, H., editors, *Petri Nets and Software Engineering. International Workshop PNSE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 97–116. CEUR-WS.org.

[Betz et al., 2014] Betz, T.; Cabac, L.; Duvigneau, M.; Wagner, T.; and Wester-Ebbinghaus, M. (2014). "Software Engineering with Petri Nets: A Web Service and Agent Perspective". *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, pages 41–61.

[Biegus and Branki, 2004] Biegus, L. and Branki, C. (2004). "InDiA: a framework for workflow interoperability support by means of multi-agent systems". *Engineering Applications of Artificial Intelligence*, 17(7), pages 825 – 839. Autonomic Computing Systems.

[Bisogno et al., 2016] Bisogno, S.; Calabrese, A.; Gastaldi, M.; and Ghiron, N. L. (2016). "Combining modelling and simulation approaches: How to measure performance of business processes". *Business Proc. Manag. Journal*, 22(1), pages 56–74.

[Blake and Gomaa, 2005] Blake, M. B. and Gomaa, H. (2005). "Agent-oriented compositional approaches to services-based cross-organizational workflow". *Decision Support Systems*, 40(1), pages 31–50.

[Bocciarelli et al., 2016] Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; and Paglia, E. (2016). "A BPMN Extension to Enable the Explicit Modeling of Task Resources". In Brusa, E.; D'Ambrogio, A.; Garro, A.; Tirone, L.; and Tundis, A., editors, *Proceedings of the 2nd INCOSE Italia Conference on Systems Engineering, Turin, Italy, November 14-16, 2016.*, volume 1728 of *CEUR Workshop Proceedings*, pages 40–47. CEUR-WS.org.

[Boissier et al., 2013] Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; and Santi, A. (2013). "Multi-agent oriented programming with JaCaMo". *Sci. Comput. Program.*, 78(6), pages 747–761.

[Booch et al., 2004] Booch, G.; Maksimchuk, R. A.; Engle, M. W.; Young, B. J.; Conallen, J.; and Houston, K. A. (2004). *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[Booch et al., 2005] Booch, G.; Rumbaugh, J.; and Jacobson, I. (2005). *Unified Modeling Language User Guide*. Addison-Wesley Object Technology Series. Addison-Wesley Professional, 2nd edition.

[Booth et al., 2004] Booth, D.; Haas, H.; McCabe, F.; Newcomer, E.; Champion, M.; Ferris, C.; and Orchard, D. (2004). *Web Services Architecture*. W3C. Available at `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/` (last accessed May 28th, 2017).

[Bordini et al., 2005] Bordini, R. H.; Hübner, J. F.; and Vieira, R. (2005). "Jason and the Golden Fleece of Agent-Oriented Programming". In Bordini, R. H.; Dastani, M.; Dix, J.; and Fallah-Seghrouchni, A. E., editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 3–37. Springer.

[Bordini et al., 2007] Bordini, R. H.; Hübner, J. F.; and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.

[Borrego et al., 2015] Borrego, D.; Gasca, R. M.; and López, M. T. G. (2015). "Automating correctness verification of artifact-centric business process models". *Information & Software Technology*, 62, pages 187–197.

[Both et al., 2012] Both, F.; Hoogendoorn, M.; van der Mee, A.; Treur, J.; and de Vos, M. (2012). "An intelligent agent model with awareness of workflow progress". *Applied Intelligence*, 36(2), pages 498–510.

[Boubaker et al., 2016] Boubaker, S.; Mammar, A.; Graiet, M.; and Gaaloul, W. (2016). "Formal Verification of Cloud Resource Allocation in Business Processes Using Event-B". In Barolli, L.; Takizawa, M.; Enokido, T.; Jara, A. J.; and Bocchi, Y., editors, *30th IEEE International Conference on Advanced Information Networking and Applications, AINA 2016, Crans-Montana, Switzerland, 23-25 March, 2016*, pages 746–753. IEEE Computer Society.

[Boucheneb and Barkaoui, 2015] Boucheneb, H. and Barkaoui, K. (2015). "Strongly Generalized Soundness of Time Workflow Nets". In *2015 15th International Conference on Application of Concurrency to System Design*, pages 130–139.

[Bouchet et al., 2016] Bouchet, F.; Harley, J. M.; and Azevedo, R. (2016). "Can Adaptive Pedagogical Agents' Prompting Strategies Improve Students' Learning and Self-Regulation?". In Micarelli, A.; Stamper, J. C.; and Panourgia, K., editors, *Intelligent Tutoring Systems - 13th International Conference, ITS 2016, Zagreb, Croatia, June 7-10, 2016. Proceedings*, volume 9684 of *Lecture Notes in Computer Science*, pages 368–374. Springer.

[Bouzguenda and Abdelkafi, 2015] Bouzguenda, L. and Abdelkafi, M. (2015). "An agent-based approach for organizational structures and interaction protocols mining in workflow". *Social Netw. Analys. Mining*, 5(1), pages 10:1–10:22.

[BPMN10, 2010] BPMN10 (2010). *BPMN 2.0 by Example*. Object Management Group – OMG. Available at `http://www.omg.org/cgi-bin/doc?dtc/10-06-02` (last accessed May 28th, 2017).

[BPMN10a, 2010] BPMN10a (2010). *Business Process Model and Notation (BPMN)*. OMG. Available at `http://www.omg.org/cgi-bin/doc?dtc/10-06-02` (last accessed May 28th, 2017).

[Braubach, 2007] Braubach, L. (2007). *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.

[Braubach et al., 2010] Braubach, L.; Pokahr, A.; Jander, K.; Lamersdorf, W.; and Burmeister, B. (2010). "Go4Flex: Goal-Oriented Process Modelling". In Essaaidi, M.; Malgeri, M.; and Badica, C., editors, *Intelligent Distributed Computing IV - Proceedings of the 4th International Symposium on Intelligent Distributed Computing - IDC 2010, Tangier, Morocco, September 2010*, volume 315, pages 77–87.

[Braubach et al., 2013] Braubach, L.; Pokahr, A.; and Lamersdorf, W. (2013). "Jadex Active Components: A Unified Execution Infrastructure for Agents and Workflows". In *Advanced Computational Technologies*, Bucharest, Romania. Romanian Academy Publishing House.

[Bresciani et al., 2004] Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; and Mylopoulos, J. (2004). "Tropos: An Agent-Oriented Software Development Methodology". *Autonomous Agents and Multi-Agent Systems*, 8(3), pages 203–236.

[Bride et al., 2017] Bride, H.; Kouchnarenko, O.; and Peureux, F. (2017). "Reduction of Workflow Nets for Generalised Soundness Verification". In Bouajjani, A. and Monniaux, D., editors, *Verification, Model Checking, and Abstract Interpretation: 18th International Conference, VMCAI 2017, Paris, France, January 15–17, 2017, Proceedings*, pages 91–111. Springer International Publishing.

[Brooks, 1986] Brooks, R. (1986). "A robust layered control system for a mobile robot". *IEEE Journal on Robotics and Automation*, 2(1), pages 14–23.

[Budimac et al., 1999] Budimac, Z.; Ivanovic, M.; and Popovic, A. (1999). "Workflow Management System Using Mobile Agents". In Eder, J.; Rozman, I.; and Welzer, T., editors, *Advances in Databases and Information Systems, Third East European Conference, ADBIS'99, Maribor, Slovenia, September 13-16, 1999, Proceedings*, volume 1691 of *Lecture Notes in Computer Science*, pages 168–178. Springer.

[Budimac et al., 2006] Budimac, Z.; Pesovic, D.; Ivanovic, M.; and Ibrajter, N. (2006). "Lessons learned from the implementation of a workflow management system using mobile agents". *Novi Sad J Math*, 36(2), pages 65–79.

[Buhler and Vidal, 2005] Buhler, P. A. and Vidal, J. M. (2005). "Towards Adaptive Workflow Enactment Using Multiagent Systems". *Information Technology and Management*, 6(1), pages 61–87.

[Burmeister et al., 2008] Burmeister, B.; Arnold, M.; Copaciu, F.; and Rimassa, G. (2008). "BDI-agents for Agile Goal-oriented Business Processes". In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, AAMAS '08, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Burmeister et al., 2006] Burmeister, B.; Steiert, H.; Bauer, T.; and Baumgärtel, H. (2006). "Agile Processes Through Goal- and Context-Oriented Business Process Modeling". In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings*, volume 4103 of *Lecture Notes in Computer Science*, pages 217–228. Springer.

[Cabac, 2002] Cabac, L. (2002). "Entwicklung von geometrisch unterscheidbaren Komponenten zur Vereinheitlichung von Mulan-Protokollen". Bachelor's thesis (equiv.), University of Hamburg, Department of Computer Science.

[Cabac, 2007] Cabac, L. (2007). "Multi-Agent System: A Guiding Metaphor for the Organization of Software Development Projects". In Paolo, editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 1–12, Leipzig, Germany. Springer-Verlag.

[Cabac, 2009] Cabac, L. (2009). "Net Components: Concepts, Tool, Praxis". In Moldt, D., editor, *Petri Nets and Software Engineering, International Workshop, PNSE'09. Proceedings*, Technical Reports Université Paris 13, pages 17–33, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse. Université Paris 13.

[Cabac, 2010] Cabac, L. (2010). *Modeling Petri Net-Based Multi-Agent Applications*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Cabac and Dörges, 2007] Cabac, L. and Dörges, T. (2007). "Tools for Testing, Debugging and Monitoring Multi-agent Applications". In Moldt, D.; Kordon, F.; van Hee, K.; Colom, J.-M.; and Bastide, R., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07)*, pages 209–213, Siedlce, Poland. Akademia Podlaska.

[Cabac et al., 2008] Cabac, L.; Dörges, T.; Duvigneau, M.; Moldt, D.; Reese, C.; and Wester-Ebbinghaus, M. (2008). "Agent Models for Concurrent Software Systems". In Bergmann, R. and Lindemann, G., editors, *Proceedings of the Sixth German Conference on Multiagent System Technologies, MATES'08*, volume 5244 of *Lecture Notes in Artificial Intelligence*, pages 37–48, Berlin Heidelberg New York. Springer-Verlag.

[Cabac et al., 2012] Cabac, L.; Duvigneau, M.; and Moldt, D., editors (2012). *Petri Nets and Software Engineering. International Workshop PNSE'12, Hamburg, Germany, June 2012. Proceedings*, volume 851 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Cabac et al., 2009] Cabac, L.; Moldt, D.; Wester-Ebbinghaus, M.; and Müller, E. (2009). "Visual Representation of Mobile Agents – Modeling Mobility within the Prototype MAPA". In [Duvigneau and Moldt, 2009], pages 7–28.

[Cabanillas, 2016] Cabanillas, C. (2016). "Process-and Resource-Aware Information Systems". In Matthes, F.; Mendling, J.; and Rinderle-Ma, S., editors, *20th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2016, Vienna, Austria, September 5-9, 2016*, pages 1–10. IEEE Computer Society.

[Cai et al., 1996] Cai, T.; Gloor, P. A.; and Nog, S. (1996). "DartFlow: A Workflow Management System on the Web Using Transportable Agents". Technical report, Dartmouth College, Hanover, NH, USA.

[Caire et al., 2008a] Caire, G.; Gotta, D.; and Banzi, M. (2008a). "WADE: A Software Platform to Develop Mission Critical Applications Exploiting Agents and Workflows". In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, AAMAS '08, pages 29–36, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Caire et al., 2008b] Caire, G.; Porta, M.; Quarantotto, E.; and Sacchi, G. (2008b). "Wolf - An Eclipse Plug-In for WADE". In *17th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2008, Rome, Italy, June 23-25, 2008, Proceedings*, pages 26–32. IEEE Computer Society.

[Cao et al., 2004] Cao, J.; Wang, J.; Zhang, S.; and Li, M. (2004). "A Dynamically Reconfigurable System Based on Workflow and Service Agents". *Eng. Appl. Artif. Intell.*, 17(7), pages 771–782.

[Cernuzzi et al., 2014] Cernuzzi, L.; Molesini, A.; and Omicini, A. (2014). "The Gaia Methodology Process". In Cossentino, M.; Hilaire, V.; Molesini, A.; and Seidita, V., editors, *Handbook on Agent-Oriented Design Processes*, pages 141–172. Springer.

[Červenka and Trenčanský, 2007] Červenka, R. and Trenčanský, I. (2007). *AML - The Agent Modeling Language: A Comprehensive Approach to Modeling Multi-Agent Systems*. Birkhäuser Verlag - Basel, Boston, Berlin.

[Chang and Scott, 1996] Chang, J. W. and Scott, C. T. (1996). "Agent-Based Workflow: TRP Support Environment (TSE)". *Computer Networks*, 28(7-11), pages 1501–1512.

[Chella et al., 2006] Chella, A.; Cossentino, M.; Sabatucci, L.; and Seidita, V. (2006). "Agile PASSI: An agile process for designing agents". *Comput. Syst. Sci. Eng.*, 21(2).

[Chen et al., 2016a] Chen, H.; Meng, J.; Zhu, J.; and Wang, J. (2016a). "ABS: Agent-Based Scheduling for Data-Intensive Workflow in Software-as-a-Service Environments". In *International Conference on Advanced Cloud and Big Data, CBD 2016, Chengdu, China, August 13-16, 2016*, pages 19–24. IEEE Computer Society.

[Chen, 1976] Chen, P. P.-S. (1976). "The Entity-relationship Model&Mdash;Toward a Unified View of Data". *ACM Transactions on Database Systems (TODS)*, 1(1), pages 9–36.

[Chen et al., 2000] Chen, Q.; Dayal, U.; Hsu, M.; and Griss, M. L. (2000). "Dynamic-Agents, Workflow and XML for E-Commerce Automation". In Bauknecht, K.; Madria, S. K.; and Pernul, G., editors, *Electronic Commerce and Web Technologies, First International Conference, EC-Web 2000, London, UK, September 4-6, 2000, Proceedings*, volume 1875 of *Lecture Notes in Computer Science*, pages 314–323. Springer.

[Chen et al., 2016b] Chen, Q.; Su, K.; Sattar, A.; Luo, X.; and Chen, A. (2016b). "A first-order coalition logic for BDI-agents". *Frontiers of Computer Science*, 10(2), pages 233–245.

[Christensen and Hansen, 1994] Christensen, S. and Hansen, N. D. (1994). "Coloured Petri Nets Extended with Channels for Synchronous Communication". In *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, pages 159–178, Berlin Heidelberg New York. Springer-Verlag.

[Cimino and Vaglini, 2014] Cimino, M. G. C. A. and Vaglini, G. (2014). "An Interval-Valued Approach to Business Process Simulation Based on Genetic Algorithms and the BPMN". *Information*, 5(2), pages 319–356.

[Clempner, 2017] Clempner, J. B. (2017). "Classical workflow nets and workflow nets with reset arcs: using Lyapunov stability for soundness verification". *J. Exp. Theor. Artif. Intell.*, 29(1), pages 43–57.

[CMMN16, 2016] CMMN16 (2016). *Case Management Model and Notation (CMMN)*. OMG. Available at `http://www.omg.org/spec/CMMN/1.1/` (last accessed May 28th, 2017).

[Coelho et al., 2016] Coelho, C. G. C.; Abreu, C. G.; Ramos, R. M.; Mendes, A. H. D.; Teodoro, G.; and Ralha, C. G. (2016). "MASE-BDI: agent-based simulator for environmental land change with efficient and parallel auto-tuning". *Appl. Intell.*, 45(3), pages 904–922.

[Coleman and Renaat, 1998] Coleman, G. and Renaat (1998). "A Quality Software Process for Rapid Application Development". *Software Quality Journal*, 7(2), pages 107–122.

[Conforti et al., 2016] Conforti, R.; Dumas, M.; García-Bañuelos, L.; and Rosa, M. L. (2016). "BPMN Miner: Automated discovery of BPMN process models with hierarchical structure". *Inf. Syst.*, 56, pages 284–303.

[Coria et al., 2014] Coria, J. A. G.; Castellanos-Garzón, J. A.; and Corchado, J. M. (2014). "Intelligent business processes composition based on multi-agent systems". *Expert Syst. Appl.*, 41(4), pages 1189–1205.

[Corradini et al., 2015] Corradini, F.; Polini, A.; and Re, B. (2015). "Inter-organizational business process verification in public administration". *Business Proc. Manag. Journal*, 21(5), pages 1040–1065.

[Cossentino and Potts, 2002] Cossentino, M. and Potts, C. (2002). "PASSI: a Process for Specifying and Implementing Multi-Agent Systems Using UML".

[Coulouris et al., 2001] Coulouris, G.; Dollimore, J.; and Kindberg, T. (2001). *Distributed Systems - Concepts and Design*. Pearson Education Ltd., 3 edition.

[Couvreur et al., 2011] Couvreur, J.-M.; Poitrenaud, D.; and Pascal (2011). "Branching Processes of General Petri Nets". In Kristensen, L. M. and Laure, editors, *Applications and Theory of Petri Nets: 32nd International Conference, PETRI NETS 2011, Newcastle, UK, June 20-24, 2011. Proceedings*, pages 129–148. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Cranshaw et al., 2017] Cranshaw, J.; Elwany, E.; Newman, T.; Kocielnik, R.; Yu, B.; Soni, S.; Teevan, J.; and Monroy-Hernández, A. (2017). "Calendar.help: Designing a Workflow-Based Scheduling Agent with Humans in the Loop". *CoRR*, abs/1703.08428.

[Croatti and Ricci, 2016] Croatti, A. and Ricci, A. (2016). "An extension of AgentSpeak(L) and Jason tailored to programming and software development". In Clebsch, S.; Desell, T.; Haller, P.; and Ricci, A., editors, *Proceedings of the 6th International Workshop on Programming Based on Actors, Agents, and Decentralized Control, AGERE 2016, Amsterdam, The Netherlands, October 30, 2016*, pages 1–10. ACM.

[Cunha et al., 2015] Cunha, F. J. P.; da Costa, A. D.; Viana, M. L.; and de Lucena, C. J. P. (2015). "JAT4BDI: An Aspect-Based Approach for Testing BDI Agents". In [WIAT2015, 2015], pages 186–189.

[Czepa et al., 2016] Czepa, C.; Tran, H.; Zdun, U.; Kim, T. T. T.; Weiss, E.; and Ruhsam, C. (2016). "Towards a Compliance Support Framework for Adaptive Case Management". In *20th IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2016, Vienna, Austria, September 5-9, 2016*, pages 1–8.

[Dadam et al., 2009] Dadam, P.; Reichert, M.; Rinderle-Ma, S.; Göser, K.; Kreher, U.; and Jurisch, M. (2009). "Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: "'Correctness by Construction"' und flexible, robuste Ausführung von Unternehmensprozessen". *EMISA Forum*, 29(1), pages 9–28.

[Dam et al., 2015] Dam, H. K.; Ghose, A.; and Qasim, M. (2015). "An Agent-Mediated Platform for Business Processes". *IJITWE*, 10(2), pages 43–61.

[Dastani et al., 2005] Dastani, M.; van Riemsdijk, M. B.; and Meyer, J. C. (2005). "Programming Multi-Agent Systems in 3APL". In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 39–67. Springer-Verlag.

[de Boer et al., 2007] de Boer, F.; Hindriks, K.; van der Hoek, W.; and Meyer, J.-J. (2007). "A verification framework for agent programming with declarative goals". *Journal of Applied Logic*, 5(2), pages 277 – 302. Logic-Based Agent Verification.

[De Masellis et al., 2017] De Masellis, R.; Francescomarino, C. D.; Ghidini, C.; Montali, M.; and Tessaris, S. (2017). "Add Data into Business Process Verification: Bridging the Gap between Theory and Practice". In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1091–1099.

[Debenham and Simoff, 2006] Debenham, J. and Simoff, S. (2006). "Intelligent Agents that Span the Process Management Spectrum". In *2006 3rd International IEEE Conference Intelligent Systems*, pages 386–389.

[Deelman et al., 2016] Deelman, E.; Vahi, K.; Rynge, M.; Juve, G.; Mayani, R.; and da Silva, R. F. (2016). "Pegasus in the Cloud: Science Automation through Workflow Technologies". *IEEE Internet Computing*, 20(1), pages 70–76.

[Delias et al., 2011] Delias, P.; Doulamis, A.; and Nikolaos (2011). "What agents can do in workflow management systems". *Artificial Intelligence Review*, 35(2), pages 155–189.

[Delias et al., 2012] Delias, P.; Tsafarakis, S.; and Doulamis, A. (2012). "Manual Intervention and Statefulness in Agent-Involved Workflow Management Systems". In Casillas, J.; Martínez-López, F. J.; and Corchado Rodríguez, J. M., editors, *Management Intelligent Systems: First International Symposium*, pages 239–249. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Denz, 2012] Denz, N. (2012). *Process-Oriented Analysis and Validation of Multi-Agent-Based Simulations : Concepts and Case Studies.* Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Desel, 2005] Desel, J. (2005). "Process-Aware Information Systems: Bridging People and Software through Process Technology". In Dumas, M.; van der Aalst, W. M. P.; and ter Hofstede, A. H. M., editors, *Process-Aware Information Systems*, chapter Process Modeling using Petri Nets, pages 147–177. John Wiley & Sons, Inc.

[Desel, 2008] Desel, J. (2008). "Controlling Petri Net Process Models". In Dumas, M. and Heckel, R., editors, *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 17–30. Springer Berlin Heidelberg.

[Desel and Reisig, 1998] Desel, J. and Reisig, W. (1998). "Place/transition Petri Nets". In Reisig, W. and Rozenberg, G., editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer Berlin Heidelberg.

[Dominguez et al., 2016] Dominguez, M.; Hernández-Vega, J.; Palomares-Gorham, D.; Hernández-Santos, C.; and Cuevas, J. L. S. (2016). "A BDI Agent System for the Collaboration of the Unmanned Aerial Vehicle". *Research in Computing Science*, 121, pages 113–124.

[Dumas et al., 2013] Dumas, M.; Rosa, M. L.; Mendling, J.; and A., H. (2013). *Fundamentals of Business Process Management.* Springer-Verlag.

[Dumas et al., 2005] Dumas, M.; van der Aalst, W. M. P.; and ter Hofstede, A. H. M. (2005). *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley.

[Duvigneau, 2002] Duvigneau, M. (2002). "Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten". Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Duvigneau, 2010] Duvigneau, M. (2010). *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen*, volume 4 of *Agent Technology – Theory and Applications.* Logos Verlag, Berlin.

[Duvigneau and Moldt, 2009] Duvigneau, M. and Moldt, D., editors (2009). *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, number FBI-HH-B-290/09 in Bericht, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Informatics.

[Duvigneau et al., 2002] Duvigneau, M.; Moldt, D.; and Rölke, H. (2002). "Concurrent Architecture for a Multi-agent Platform". In Giunchiglia, F.; Odell, J.; and Weiß, G., editors, *Agent-Oriented Software Engineering. 3rd International Workshop, AOSE 2002, Bologna. Proceedings*, pages 147–159. ACM Press.

[Duvigneau et al., 2003] Duvigneau, M.; Moldt, D.; and Rölke, H. (2003). "Concurrent Architecture for a Multi-agent Platform". In Giunchiglia, F.; Odell, J.; and Weiß, G., editors, *Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 59–72, Berlin Heidelberg New York. Springer-Verlag.

[Dzikowski and Hood, 2015] Dzikowski, J. and Hood, C. S. (2015). "Modeling cognitive radio networks in NetLogo". In Mittal, S., editor, *Proceedings of the Conference on Summer Computer Simulation, SummerSim 2015, Chicago, IL, USA, July 26-29, 2015*, pages 15:1–15:11. ACM.

[Eberling, 2015] Eberling, A. (2015). "Bereitstellung eines adaptiven, Petrinetz-basierten Werkzeugs zur Modellierung von PAOSE-Software-Entwicklungsprozessen". Master's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Eeles and Cripps, 2010] Eeles, P. and Cripps, P. (2010). *The Process of Software Architecting*. Addison-Wesley.

[Ehrler et al., 2006] Ehrler, L.; Fleurke, M. K.; Purvis, M.; and Savarimuthu, B. T. R. (2006). "Agent-based workflow management systems (WfMSs)". *Inf. Syst. E-Business Management*, 4(1), pages 5–23.

[Engel et al., 2016] Engel, R.; Krathu, W.; Zapletal, M.; Pichler, C.; Bose, R. P. J. C.; Aalst, W. M. P. V. D.; Werthner, H.; and Huemer, C. (2016). "Analyzing inter-organizational business processes". *Information Systems and e-Business Management*, 14(3), pages 577–612.

[Engelfriet, 1991] Engelfriet, J. (1991). "Branching processes of Petri nets". *Acta Informatica*, 28(6), pages 575–591.

[Esparza and Heljanko, 2008] Esparza, J. and Heljanko, D. K. (2008). *Unfoldings: A Partial-Order Approach to Model Checking*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Esparza and Hoffmann, 2016] Esparza, J. and Hoffmann, P. (2016). "Reduction Rules for Colored Workflow Nets". In Stevens, P. and Wasowski, A., editors, *Fundamental Approaches to Software Engineering: 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, pages 342–358. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Esparza et al., 2016] Esparza, J.; Hoffmann, P.; and Saha, R. (2016). "Polynomial Analysis Algorithms for Free Choice Probabilistic Workflow Nets". In Agha, G. and Van Houdt, B., editors, *Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings*, pages 89–104. Springer International Publishing, Cham.

[Esteves and Veiga, 2016] Esteves, S. and Veiga, L. (2016). "WaaS: Workflow-as-a-Service for the Cloud with Scheduling of Continuous and Data-Intensive Workflows". *Comput. J.*, 59(3), pages 371–383.

[Evans et al., 2001] Evans, R.; Kearney, P.; Caire, G.; Garijo, F.; Sanz, J. G.; Pavon, J.; Leal, F.; Chainho, P.; and Massonet, P. (2001). "MESSAGE: Methodology for engineering systems of software agents". *EURESCOM, EDIN*, pages 0223–0907.

[Fakas and Karakostas, 1999] Fakas, G. and Karakostas, B. (1999). "A workflow management system based on intelligent collaborative objects". *Information & Software Technology*, 41(13), pages 907–915.

[Fang and Yin, 2010] Fang, Z. and Yin, C. (2010). "BPM Architecture Design Based on Cloud Computing". *Intelligent Information Management*, 2(5), pages 329–333.

[Feng et al., 2015] Feng, N.; Yu, X.; Dou, R.; and Pan, B. (2015). "Managing risk for business processes: A fuzzy based multi-agent system". *Journal of Intelligent and Fuzzy Systems*, 29(6), pages 2717–2726.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). "A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems". In Demazeau, Y., editor, *Proceedings of the Third International Conference on Multiagent Systems, ICMAS 1998, Paris, France, July 3-7, 1998*, pages 128–135. IEEE Computer Society.

[Ferber and Gutknecht, 1999] Ferber, J. and Gutknecht, O. (1999). "Operational Semantics of Multi-agent Organizations". In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL), 6th International Workshop, ATAL '99, Orlando, Florida, USA, July 15-17, 1999, Proceedings*, volume 1757 of *Lecture Notes in Computer Science*, pages 205–217. Springer.

[Ferber et al., 2003] Ferber, J.; Gutknecht, O.; and Michel, F. (2003). "From Agents to Organizations: An Organizational View of Multi-agent Systems". In Giorgini, P.; Müller, J. P.; and Odell, J., editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer.

[Ferraiolo and Kuhn, 1992] Ferraiolo, D. and Kuhn, R. (1992). "Role-Based Access Control". In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563.

[FIPA01L, 2002] FIPA01L (2002). *FIPA Abstract Architecture Specification - SC00001L*. Foundation for Intelligent Physical Agents.

[FIPA08I, 2002] FIPA08I (2002). *FIPA SL Content Language Specification - SC00008I*. Foundation for Intelligent Physical Agents.

[FIPA23K, 2004] FIPA23K (2004). *FIPA Agent Management Specification - SC00023K*. Foundation for Intelligent Physical Agents.

[FIPA37H, 2002] FIPA37H (2002). *FIPA Communicative Act Library Specification - SC00037J*. Foundation for Intelligent Physical Agents.

[FIPA61G, 2002] FIPA61G (2002). *FIPA ACL Message Structure Specification - SC00061G*. Foundation for Intelligent Physical Agents.

[Fischer et al., 1995] Fischer, K.; Müller, J. P.; and Pischel, M. (1995). "A Pragmatic BDI Architecture". In Wooldridge, M.; Müller, J. P.; and Tambe, M., editors, *Intelligent Agents II, Agent Theories, Architectures, and Languages, IJCAI '95, Workshop (ATAL), Montreal, Canada, August 19-20, 1995, Proceedings*, volume 1037 of *Lecture Notes in Computer Science*, pages 203–218. Springer.

[Fischer et al., 2003] Fischer, K.; Schillo, M.; and Siekmann, J. H. (2003). "Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems". In Maríık, V.; McFarlane, D. C.; and Valckenaers, P., editors, *Holonic and Multi-Agent Systems for Manufacturing, First International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2003, Prague, Czech Republic, September 1-3, 2003, Proceedings*, volume 2744 of *Lecture Notes in Computer Science*, pages 71–80. Springer-Verlag.

[Fix, 2012] Fix, J. (2012). *Emotionale Agenten: Darstellung der emotionstheoretischen Grundlagen und Entwicklung eines Referenzmodells auf Basis einer petrinetz-basierten Modellierungstechnik*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Gacek et al., 1995] Gacek, C.; Adb-Allah, A.; Clark, B.; and Boehm, B. (1995). "On The Definition of Software System Architecture". In *Proceedings of the First International Workshop on Architectures for Software Systems*.

[Gascuena and Fernandez-Caballero, 2011] Gascuena, J. M. and Fernandez-Caballero, A. (2011). "Agent-oriented modeling and development of a person-following mobile robot". *Expert Systems with Applications*, 38(4), pages 4280 – 4290.

[Girault and Valk, 2003] Girault, C. and Valk, R. (2003). *Petri Nets for Systems Engineering – A Guide to Modeling, Verification, and Applications*. Springer-Verlag, Berlin Heidelberg New York.

[Giret and Botti, 2003] Giret, A. and Botti, V. J. (2003). "Towards a Recursive Agent Oriented Methodology for Large-Scale MAS". In Giorgini, P.; Müller, J. P.; and Odell, J., editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 25–35. Springer-Verlag.

[Glanzer et al., 2001] Glanzer, K.; Hämmerle, A.; and Geurts, R. (2001). "The Appliance of ZEUS Agents in Manufacturing Systems". In *12th International Workshop on Database and Expert Systems Applications (DEXA 2001), 3-7 September 2001, Munich, Germany*, pages 628–632.

[Gloger, 2010] Gloger, B. (2010). "Scrum". *Informatik-Spektrum*, 33(2), pages 195–200.

[Goesmann et al., 2015] Goesmann, T.; Schröder, S.; and Unger, M. (2015). "Adaptive Case Management im Vergleich mit BPM". *Wirtschaftsinformatik & Management*, 7(3), pages 43–51.

[Gomes et al., 2016] Gomes, N.; Oliveira, B.; and Belo, O. (2016). "Modeling Agents Working on ETL Processes". In *Advances in Practical Applications of Scalable Multi-agent Systems. The PAAMS Collection - 14th International Conference, PAAMS 2016, Sevilla, Spain, June 1-3, 2016, Proceedings*, pages 265–268.

[Gou and Yamaguchi, 2017] Gou, Z. and Yamaguchi, S. (2017). "Structural and Behavioral Properties of Well-Structured Workflow Nets". *IEICE Transactions*, 100-A(2), pages 421–426.

[Greenwood, 2008] Greenwood, D. (2008). "Goal-Oriented Autonomic Business Process Modeling and Execution: Engineering Change Management Demonstration". In Dumas, M.; Reichert, M.; and Shan, M.-C., editors, *Business Process Management: 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, pages 390–393. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Greenwood and Rimassa, 2007] Greenwood, D. and Rimassa, G. (2007). "Autonomic Goal-Oriented Business Process Management". In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pages 43–43.

[Grefen, 2009] Grefen, P. (2009). "Systems for Interorganizational Business Process Management". In *Handbook of Research on Business Process Modeling*, pages 403–425. Information Science Reference, Hershey, PA.

[Gregori et al., 2006] Gregori, M. E.; Cámara, J. P.; and Bada, G. A. (2006). "A jabber-based multi-agent system platform". In Nakashima, H.; Wellman, M. P.; Weiss, G.; and Stone, P., editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 1282–1284. ACM.

[Gutknecht and Ferber, 2000] Gutknecht, O. and Ferber, J. (2000). "MadKit: a generic multi-agent platform". In *Agents*, pages 78–79.

[Hammer and Champy, 1993] Hammer, M. and Champy, J. (2004 (reprint, originally 1993)). *Reengineering the Corporation - A Manifesto for Business Revolution*. HarperCollins.

[Harland et al., 2017] Harland, J.; Morley, D. N.; Thangarajah, J.; and Yorke-Smith, N. (2017). "Aborting, suspending, and resuming goals and plans in BDI agents". *Autonomous Agents and Multi-Agent Systems*, 31(2), pages 288–331.

[Harmon and Wolf, 2016] Harmon, P. and Wolf, C. (2016). "State of Business Process Management – 2016". Technical report, BPTrends.

[Hauder et al., 2014a] Hauder, M.; Münch, D.; Michel, F.; Utz, A.; and Matthes, F. (2014a). "Examining Adaptive Case Management to Support Processes for Enterprise Architecture Management". In *18th IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, EDOC Workshops 2014, Ulm, Germany, September 1-2, 2014*, pages 23–32.

[Hauder et al., 2014b] Hauder, M.; Pigat, S.; and Matthes, F. (2014b). "Research Challenges in Adaptive Case Management: A Literature Review". In *18th IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, EDOC Workshops 2014, Ulm, Germany, September 1-2, 2014*, pages 98–107.

[Haustermann, 2010] Haustermann, M. (2010). "Analyse von Workflows auf Basis von Petrinetzen". Bachelor's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Haustermann, 2017] Haustermann, M. (planned for 2017). *Verifizierung, Validierung und Testen von referenznetzbasierten Multiagentensystemen.* Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Havur et al., 2016] Havur, G.; Cabanillas, C.; Mendling, J.; and Polleres, A. (2016). "Resource Allocation with Dependencies in Business Process Management Systems". In Rosa, M. L.; Loos, P.; and Pastor, O., editors, *Business Process Management Forum - BPM Forum 2016, Rio de Janeiro, Brazil, September 18-22, 2016, Proceedings*, volume 260 of *Lecture Notes in Business Information Processing*, pages 3–19. Springer.

[Hawryszkiewycz and Debenham, 1998] Hawryszkiewycz, I. and Debenham, J. K. (1998). "A Workflow System Based on Agents". In *Database and Expert Systems Applications, 9th International Conference, DEXA '98, Vienna, Austria, August 24-28, 1998, Proceedings*, pages 135–144.

[Hayes-Roth, 1995] Hayes-Roth, B. (1995). "An Architecture for Adaptive Intelligent Systems". *Artificial Intelligence*, 72(1-2), pages 329–365.

[Hazzan and Dubinsky, 2008] Hazzan, O. and Dubinsky, Y. (2008). *Agile Software Engineering.* Springer-Verlag.

[He et al., 2014] He, L.; Chaudhary, N.; and Jarvis, S. A. (2014). "Developing security-aware resource management strategies for workflows". *Future Generation Comp. Syst.*, 38, pages 61–68.

[He et al., 2006] He, Y.; Yang, H.; He, W.; Zhang, W.; and He, X. (2006). "Flexible Workflow Driven Job Shop Manufacturing Execution and Automation Based on Multi Agent System". In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Hong Kong, China, 18-22 December 2006*, pages 695–699.

[Head et al., 2015] Head, B.; Hjorth, A.; Brady, C. E.; and Wilensky, U. (2015). "Evolving agent cognition with netlogo levelspace". In *Proceedings of the 2015 Winter Simulation Conference, Huntington Beach, CA, USA, December 6-9, 2015*, pages 3122–3123. IEEE/ACM.

[Heckel and Youngblood, 2010] Heckel, F. W. P. and Youngblood, G. M. (2010). "Multi-Agent Coordination Using Dynamic Behavior-Based Subsumption". In Youngblood, G. M. and Bulitko, V., editors, *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*. The AAAI Press.

[Hewelt et al., 2011] Hewelt, M.; Wagner, T.; and Cabac, L. (2011). "Integrating Verification into the PAOSE Approach". In Duvigneau, M.; Moldt, D.; and Hiraishi, K., editors, *Petri Nets and Software Engineering. International Workshop PNSE'11, Newcastle upon Tyne, UK, June 2011. Proceedings*, volume 723 of *CEUR Workshop Proceedings*, pages 124–135. CEUR-WS.org.

[Hewitt, 1977] Hewitt, C. (1977). "Viewing Control Structures as Patterns of Passing Messages". *Artificial Intelligence*, 8, pages 323–364.

[Hichami et al., 2015] Hichami, O. E.; Naoum, M.; Achhab, M. A.; Berrada, I.; and Mohajir, B. E. E. (2015). "An Algebraic Method for Analysing Control Flow of BPMN Models". *iJES*, 3(3), pages 20–26.

[Hindriks et al., 1999] Hindriks, K. V.; de Boer, F. S.; van der Hoek, W.; and Meyer, J. C. (1999). "Agent Programming in 3APL". *Autonomous Agents and Multi-Agent Systems*, 2(4), pages 357–401.

[Hinz et al., 2005] Hinz, S.; Schmidt, K.; and Stahl, C. (2005). "Transforming BPEL to Petri Nets". In van der Aalst, W. M. P.; Benatallah, B.; Casati, F.; and Curbera, F., editors, *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings*, volume 3649, pages 220–235.

[Hollingsworth, 1995] Hollingsworth, D. (1995). *The Workflow Reference Model*. Workflow Management Coalition, 2 Crown Walk, Winchester, Hampshire, UK.

[Hongchen and Meilin, 1999] Hongchen, L. and Meilin, S. (1999). "Application of agents in workflow management system". In *Fifth Asia-Pacific Conference on ... and Fourth Optoelectronics and Communications Conference on Communications,*, volume 2, pages 1068–1072 vol.2.

[Hoogers et al., 1995] Hoogers, P.; Kleijn, H.; and Thiagarajan, P. (1995). "A Trace Semantics for Petri Nets". *Information and Computation*, 117(1), pages 98 – 114.

[Hosman and Baikie, 2013] Hosman, L. and Baikie, B. (2013). "Solar-Powered Cloud Computing Datacenters". *IT Professional*, 15(2), pages 15–21.

[Hsieh and Lin, 2016] Hsieh, F. and Lin, J. (2016). "A self-adaptation scheme for workflow management in multi-agent systems". *J. Intelligent Manufacturing*, 27(1), pages 131–148.

[Huber et al., 2013] Huber, S.; Hauptmann, A.; Lederer, M.; and Kurz, M. (2013). "Managing Complexity in Adaptive Case Management". In *S-BPM ONE - Running Processes, 5th International Conference, S-BPM ONE 2013, Deggendorf, Germany, March 11-12, 2013. Proceedings*, pages 209–226.

[IEEE00, 2000] IEEE00 (2000). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE.

[IEEE07, 2007] IEEE07 (2007). *ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE.

[IEEE11, 2011] IEEE11 (2011). *ISO/IEC/IEEE Systems and software engineering – Architecture description*. IEEE.

[Indiono et al., 2016] Indiono, C.; Mangler, J.; Fdhila, W.; and Rinderle-Ma, S. (2016). "Rule-Based Runtime Monitoring of Instance-Spanning Constraints in Process-Aware Information Systems". In Debruyne, C.; Panetto, H.; Meersman, R.; Dillon, T. S.; eva Kühn; O'Sullivan, D.; and Ardagna, C. A., editors, *On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Rhodes, Greece, October 24-28, 2016, Proceedings*, volume 10033 of *Lecture Notes in Computer Science*, pages 381–399.

[Izquierdo et al., 2015] Izquierdo, L. R.; Olaru, D.; Izquierdo, S. S.; Purchase, S.; and Soutar, G. N. (2015). "Fuzzy Logic for Social Simulation Using NetLogo". *J. Artificial Societies and Social Simulation*, 18(4).

[Jacob, 2002] Jacob, T. (2002). "Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug". Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Jain et al., 1999] Jain, A. K.; IV, M. A.; and Singh, M. P. (1999). "Agents for Process Coherence in Virtual Enterprises". *Commun. ACM*, 42(3), pages 62–69.

[Jander, 2016] Jander, K. (2016). *Agile Business Process Management*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.

[Jander et al., 2016] Jander, K.; Braubach, L.; and Lamersdorf, W. (2016). "Distributed monitoring and workflow management for goal-oriented workflows". *Concurrency and Computation: Practice and Experience*, 28(4), pages 1324–1335.

[Jander et al., 2011] Jander, K.; Braubach, L.; Pokahr, A.; Lamersdorf, W.; and karl Josef Wack (2011). "Goal-Oriented Processes With GPMN". *International Journal on Artificial Intelligence Tools*, 20(06), pages 1021–1041.

[Jander and Lamersdorf, 2013] Jander, K. and Lamersdorf, W. (2013). "Jadex WfMS: Distributed Workflow Management for Private Clouds". In *Conference on Networked Systems, NetSys 2013, Stuttgart, Germany, March 11-15, 2013*.

[Janicki and Koutny, 1987] Janicki, R. and Koutny, M. (1987). "On Equivalent Execution Semantics of Concurrent Systems". In *Advances in Petri Nets 1987, Covers the 7th European Workshop on Applications and Theory of Petri Nets*, pages 89–103, Berlin Heidelberg New York. Springer-Verlag.

[Jankovic et al., 2015] Jankovic, M.; Ljubicic, M.; Anicic, N.; and Marjanovic, Z. (2015). "Enhancing BPMN 2.0 informational perspective to support interoperability for cross-organizational business processes". *Comput. Sci. Inf. Syst.*, 12(3), pages 1101–1120.

[Jannaber et al., 2016] Jannaber, S.; Karhof, A.; Riehle, D. M.; Thomas, O.; Delfmann, P.; and Becker, J. (2016). "Invigorating Event-driven Process Chains - Towards an integrated meta model for EPC standardization". In Betz, S. and Reimer, U., editors, *Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband*, volume 255 of *LNI*, pages 13–22. GI.

[Jansen and Bosch, 2005] Jansen, A. and Bosch, J. (2005). "Software Architecture as a Set of Architectural Design Decisions". In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pages 109–120.

[Jennings, 2000] Jennings, N. R. (2000). "On Agent-Based Software Engineering". *Artificial Intelligence*, 117(2), pages 277–296.

[Jennings et al., 1998] Jennings, N. R.; Norman, T. J.; and Faratin, P. (1998). "ADEPT: An Agent-Based Approach to Business Process Management". *SIGMOD Record*, 27(4), pages 32–39.

[Jennings et al., 2000] Jennings, N. R.; Norman, T. J.; Faratin, P.; O'Brien, P.; and Odgers, B. (2000). "Autonomous Agents for Business Process Management". *Applied Artificial Intelligence*, 14(2), pages 145–189.

[Jennings and Wooldridge, 1995] Jennings, N. R. and Wooldridge, M. J. (1995). "Applying Agent Technology". *Int. Journal of Applied Artificial Intelligence*, 9(4), pages 351–369.

[Jensen, 1981] Jensen, K. (1981). "Coloured Petri Nets and the Invariant Method". *Theoretical Computer Science*, 14(3), pages 317–336.

[Jensen, 1987] Jensen, K. (1987). "Coloured Petri Nets". In Brauer, W.; Reisig, W.; and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties*, volume 254, pages 248–299. Springer.

[Jensen and Kristensen, 2009] Jensen, K. and Kristensen, L. M. (2009). *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer-Verlag, Berlin Heidelberg New York, 1st edition.

[Jessen and Valk, 1987] Jessen, E. and Valk, R. (1987). *Rechensysteme - Grundlagen der Modellbildung*. Springer-Verlag.

[Jhu et al., 2015] Jhu, S.; Fan, C.; Yuan, S.; and Chang, Y. (2015). "Implementing a Workflow Agent on Federated Cloud". In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom 2015, Chengdu, China, December 19-21, 2015*, pages 915–920.

[Jingjing et al., 2004] Jingjing, H.; Yuanda, C.; and Zhen, Z. (2004). "Workflow management system based on agent for virtual enterprise". In *8th International Conference on Computer Supported Cooperative Work in Design*, volume 1, pages 373–378 Vol.1.

[Joeris, 2000] Joeris, G. (2000). "Decentralized and flexible workflow enactment based on task coordination agents". In *2nd International Bi-conference workshop on agent-oriented information systems (AOIS-2000@CAiSE*00), Stockholm*, pages 41—62. iCue Publishing.

[Jørgensen et al., 2015] Jørgensen, D. B.; Hallenborg, K.; and Demazeau, Y. (2015). "Environment for Telehealth Applications on Top of BDI4JADE". In [WIAT2015, 2015], pages 383–386.

[Kadar et al., 2014] Kadar, M.; Muntean, M.; Cretan, A.; and Jardim-Gonçalves, R. (2014). "Automated Negotiation with Multi-agent Systems in Business Processes". In *Intelligent Systems'2014 - Proceedings of the 7th International Conference Intelligent Systems IEEE IS'2014, September 24-26, 2014, Warsaw, Poland, Volume 1: Mathematical Foundations, Theory, Analyses*, pages 289–301.

[Kaes et al., 2015] Kaes, G.; Mangler, J.; Stertz, F.; Vigne, R.; and Rinderle-Ma, S. (2015). "ACaPlan - Adaptive Care Planning". In *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015.*, pages 11–15.

[Kalenkova et al., 2015] Kalenkova, A. A.; van der Aalst, W. M. P.; Lomazova, I. A.; and Rubin, V. A. (2015). "Process mining using BPMN: relating event logs and process models". *Software & Systems Modeling*, pages 1–30.

[Kappel et al., 2000] Kappel, G.; Rausch-Schott, S.; and Retschitzegger, W. (2000). "A framework for workflow management systems based on objects, rules and roles". *ACM Comput. Surv.*, 32(1es), pages 27.

[Karhof et al., 2016] Karhof, A.; Jannaber, S.; Riehle, D. M.; Thomas, O.; Delfmann, P.; and Becker, J. (2016). "On the de-facto Standard of Event-driven Process Chains: Reviewing EPC Implementations in Process Modelling Tools". In Oberweis, A. and Reussner, R. H., editors, *Modellierung 2016, 2.-4. März 2016, Karlsruhe*, volume 254 of *LNI*, pages 77–92. GI.

[Kay, 1984] Kay, A. (1984). "Computer Software". *Scientific American*, 251(3), pages 53–59.

[Keller et al., 1992] Keller, G.; Nüttgens, M.; and Scheer, A. (1992). *Semantische Prozessmodellierung auf der Grundlage "ereignisgesteuerter Prozessketten (EPK)"*. Institut für Wirtschaftsinformatik Saarbrücken: Veröffentlichungen des Instituts für Wirtschaftsinformatik. Inst. für Wirtschaftsinformatik.

[Khallouf and Winikoff, 2009] Khallouf, J. and Winikoff, M. (2009). "The goal-oriented design of agent systems: a refinement of Prometheus and its evaluation". *IJAOSE*, 3(1), pages 88–112.

[Kheldoun et al., 2015] Kheldoun, A.; Barkaoui, K.; and Ioualalen, M. (2015). "Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach". In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 55–71.

[Khomenko and Koutny, 2003] Khomenko, V. and Koutny, M. (2003). "Branching Processes of High-Level Petri Nets". In Garavel, H. and John, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 9th International Conference, TACAS 2003 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003 Warsaw, Poland, April 7–11, 2003 Proceedings*, pages 458–472. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kim et al., 2016] Kim, T. T. T.; Weiss, E.; Adensamer, A.; Ruhsam, C.; Czepa, C.; Tran, H.; and Zdun, U. (2016). "An Ontology-Based Approach for Defining Compliance Rules by Knowledge Workers in Adaptive Case Management - A Repair Service Management Case". In *20th IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2016, Vienna, Austria, September 5-9, 2016*, pages 1–8.

[Kirk et al., 2016] Kirk, J.; Mininger, A.; and Laird, J. E. (2016). "A Demonstration of Interactive Task Learning". In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 4248–4249.

[Klai and Ochi, 2015] Klai, K. and Ochi, H. (2015). "LTL Model Cheking of Service-Based Business Processes in the Cloud". In *39th Annual Computer Software and Applications Conference, COMPSAC Workshops 2015, Taichung, Taiwan, July 1-5, 2015*, pages 398–403.

[Klinik et al., 2017] Klinik, M.; Hage, J.; Jansen, J. M.; and Plasmeijer, R. (2017). "Predicting resource consumption of higher-order workflows". In Schultz, U. P. and Yallop, J., editors, *Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2017, Paris, France, January 18-20, 2017*, pages 99–110. ACM.

[Knuplesch et al., 2017] Knuplesch, D.; Reichert, M.; and Kumar, A. (2017). "A framework for visually monitoring business process compliance". *Inf. Syst.*, 64, pages 381–409.

[Köhler and Rölke, 2005] Köhler, M. and Rölke, H. (2005). "Reference and Value Semantics Are Equivalent for Ordinary Object Petri Nets". In Ciardo, G. and Darondeau, P., editors, *Proceedings of the 26th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, volume 3536 of *Lecture Notes in Computer Science*, pages 309–328. Springer-Verlag.

[Köhler-Bußmeier, 2009a] Köhler-Bußmeier, M. (2009a). "Hornets: Nets within Nets combined with Net Algebra". In Wolf, K. and Franceschinis, G., editors, *International Conference on Application and Theory of Petri Nets (ICATPN'2009)*, volume 5606 of *Lecture Notes in Computer Science*, pages 243–262. Springer-Verlag.

[Köhler-Bußmeier, 2009b] Köhler-Bußmeier, M. (2009b). *Koordinierte Selbstorganisation und selbstorganisierte Koordination: Eine formale Spezifikation reflexiver Selbstorganisation in Multiagentensystemen unter spezieller Berücksichtigung der sozialwissenschaftlichen Perspektive*. Habilitationsschrift, University of Hamburg.

[Köhler-Bußmeier and Heitmann, 2013] Köhler-Bußmeier, M. and Heitmann, F. (2013). "Complexity Results for Elementary Hornets". In Colom, J. M. and Desel, J., editors, *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 150–169. Springer-Verlag.

[Köhler-Bußmeier et al., 2009] Köhler-Bußmeier, M.; Wester-Ebbinghaus, M.; and Moldt, D. (2009). "A Formal Model for Organisational Structures behind Process-Aware Information Systems". *Trans. Petri Nets and Other Models of Concurrency*, 2, pages 98–114.

[Kok et al., 2010] Kok, J. N.; Lamprecht, A.-L.; and Wilkinson, M. D. (2010). "Tools in Scientific Workflow Composition". In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I*, pages 258–260. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Korhonen et al., 2002] Korhonen, J.; Pajunen, L.; and Puustjärvi, J. (2002). "Using Transactional Workflow Ontology in Agent Cooperation". In *AIM Workshop, First EurAsian Conference on Advances in ICT. Tehran, 2002*.

[Kotb, 2011] Kotb, Y. T. (2011). *Workflow-Net Based Cooperative Multi-Agent Systems*. PhD thesis, The University of Western Ontario, Electronic Thesis and Dissertation Repository. Paper 228.

[Kotb et al., 2012] Kotb, Y. T.; Beauchemin, S. S.; and Barron, J. L. (2012). "Workflow Nets for Multiagent Cooperation". *IEEE Trans. Automation Science and Engineering*, 9(1), pages 198–203.

[Kravari et al., 2016] Kravari, K.; Bassiliades, N.; and Governatori, G. (2016). "A policy-based B2C e-Contract management workflow methodology using semantic web agents". *Artif. Intell. Law*, 24(2), pages 93–131.

[Kriglstein et al., 2016] Kriglstein, S.; Leitner, M.; Kabicher-Fuchs, S.; and Rinderle-Ma, S. (2016). "Evaluation Methods in Process-Aware Information Systems Research with a Perspective on Human Orientation". *Business & Information Systems Engineering*, 58(6), pages 397–414.

[Kruchten, 2004] Kruchten, P. (2004). *The Rational Unified Process: An Introduction*. Addison-Wesley Object Technology Series. Addison-Wesley.

[Kruchten et al., 2006] Kruchten, P.; Obbink, H.; and Stafford, J. (2006). "The Past, Present, and Future for Software Architecture". *Software, IEEE*, 23(2), pages 22–30.

[Krzywinski et al., 2008] Krzywinski, A.; Chen, W.; and Helgesen, A. (2008). "Agent Architecture in Social Games - The Implementation of Subsumption Architecture in Diplomacy". In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA*.

[Kumar et al., 2013] Kumar, A.; Dijkman, R. M.; and Song, M. (2013). "Optimal Resource Assignment in Workflows for Maximizing Cooperation". In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 235–250.

[Kummer, 2001] Kummer, O. (2001). "Introduction to Petri Nets and Reference Nets". *Sozionik Aktuell*, 1, pages 1–9. ISSN 1617-2477.

[Kummer, 2002] Kummer, O. (2002). *Referenznetze*. Logos Verlag, Berlin.

[Kummer et al., 2016] Kummer, O.; Wienberg, F.; Duvigneau, M.; Cabac, L.; Haustermann, M.; and Mosteller, D. (2016). *Renew – User Guide (Release 2.5)*. University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg. Available at `www.renew.de` (last accessed May 28th, 2017).

[Kummer et al., 2003] Kummer, O.; Wienberg, F.; Duvigneau, M.; Köhler, M.; Moldt, D.; and Heiko (2003). "Renew – The Reference Net Workshop". In Veerbeek, E., editor, *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003).*, pages 99–102. Department of Technology Management, Technische Universiteit Eindhoven, Beta Research School for Operations Management and Logistics.

[Kummer et al., 2004] Kummer, O.; Wienberg, F.; Duvigneau, M.; Schumacher, J.; Köhler, M.; Moldt, D.; Rölke, H.; and Valk, R. (2004). "An Extensible Editor and Simulation Engine for Petri Nets: Renew". In Cortadella, J. and Reisig, W., editors, *Applications and Theory of Petri Nets 2004. 25th International Conference, ICATPN 2004, Bologna, Italy, June 2004. Proceedings*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493, Berlin Heidelberg New York. Springer.

[Kuo, 2004] Kuo, J.-Y. (2004). "A document-driven agent-based approach for business processes management". *Information and Software Technology*, 46(6), pages 373 – 382.

[Kurz, 2016] Kurz, M. (2016). "Automating BPMN Interchange Testing". In *42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, Limassol, Cyprus, August 31 - Sept. 2, 2016*, pages 331–338. IEEE Computer Society.

[Kuster et al., 2014] Kuster, C.; Küster, T.; Lützenberger, M.; and Albayrak, S. (2014). "Model-driven Development and Validation of Multi-agent Systems in JIAC V with the Agent World Editor". In Shakshuki, E. M. and Yasar, A., editors, *Proceedings of the 5th International Conference on Ambient Systems, Networks and Technologies (ANT 2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014), Hasselt, Belgium, June 2-5, 2014*, volume 32 of *Procedia Computer Science*, pages 920–927. Elsevier.

[Küster et al., 2015] Küster, T.; Lützenberger, M.; and Albayrak, S. (2015). "A Formal Description of a Mapping from Business Processes to Agents". In *Engineering Multi-Agent Systems - Third International Workshop, EMAS 2015, Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers*, pages 153–170.

[Laird et al., 1987] Laird, J. E.; Newell, A.; and Rosenbloom, P. S. (1987). "SOAR: an architecture for general intelligence". *Artificial Intelligence*, 33(1), pages 1–64.

[Lakos, 1995] Lakos, C. (1995). "From Coloured Petri Nets to Object Petri Nets". In De Michelis, G. and Diaz, M., editors, *Application and Theory of Petri Nets 1995: 16th International Conference Turin, Italy, June 26–30, 1995 Proceedings*, pages 278–297. Springer-Verlag, Berlin, Heidelberg.

[Lassen and van der Aalst, 2009] Lassen, K. B. and van der Aalst, W. M. (2009). "Complexity metrics for Workflow nets". *Information and Software Technology*, 51(3), pages 610 – 626.

[Lassen and van der Aalst, 2006] Lassen, K. B. and van der Aalst, W. M. P. (2006). "WorkflowNet2BPEL4WS: A Tool for Translating Unstructured Workflow Processes to Readable BPEL". In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I*, volume 4275 of *Lecture Notes in Computer Science*, pages 127–144. Springer.

[Legner and Wende, 2007] Legner, C. and Wende, K. (2007). "The Challenges of Inter-Organizational Business Process Design - A Research Agenda". In Österle, H.; Schelp, J.; and Winter, R., editors, *Proceedings of the Fifteenth European Conference on Information Systems, ECIS 2007, St. Gallen, Switzerland, 2007*, pages 106–118. University of St. Gallen.

[Lehmann and Moldt, 2004] Lehmann, K. and Moldt, D. (2004). "Modelling and Analysis of Agent Protocols with Petri Nets". In Lindemann, G.; Denzinger, J.; and Timm, I. J. e. a., editors, *Multiagent System Technologies: Second German Conference, MATES 2004, Erfurt, Germany, September 29-30, 2004. Proceedings*, volume 3187 of *Lecture Notes in Computer Science*, page 85, Berlin Heidelberg New York. Springer-Verlag.

[Leitner et al., 2015] Leitner, M.; Ma, Z.; and Rinderle-Ma, S. (2015). "A Cross-Layer Security Analysis for Process-Aware Information Systems". *CoRR*, abs/1507.03415.

[Lenz and Oberweis, 2001] Lenz, K. and Oberweis, A. (2001). "Modeling Interorganizational Workflows with XML Nets". *Proceedings of the Hawai'i International Conference On System Sciences, January 3-6, 2001, Maui, Hawaii*.

[Leppänen et al., 2017] Leppänen, T.; Lacasia, J. A.; Tobe, Y.; Sezaki, K.; and Riekki, J. (2017). "Mobile crowdsensing with mobile agents". *Autonomous Agents and Multi-Agent Systems*, 31(1), pages 1–35.

[Leymann, 2012] Leymann, F. (2012). "Linked Compute Units and Linked Experiments: Using Topology and Orchestration Technology for Flexible Support of Scientific Applications". In Heisel, M., editor, *Software Service and Application Engineering - Essays Dedicated to Bernd Krämer on the Occasion of His 65th Birthday*, volume 7365 of *Lecture Notes in Computer Science*, pages 71–80. Springer.

[Leymann et al., 2011] Leymann, F.; Fehling, C.; Mietzner, R.; Nowak, A.; and Dustdar, S. (2011). "Moving Applications to the Cloud: an Approach Based on Application Model Enrichment". *Int. J. Cooperative Inf. Syst.*, 20(3), pages 307–356.

[Lin, 2008] Lin, D. (2008). *Modeling and Coordination in Interorganizational Workflow*. PhD thesis, Kyoto University.

[Lindes and Laird, 2016] Lindes, P. and Laird, J. E. (2016). "Toward Integrating Cognitive Linguistics and Cognitive Language Processing". In *Proceedings of the 14th International Conference on Cognitive Modeling (ICCM). University Park, Pennsylvania*.

[Lindes et al., 2015] Lindes, P.; Lonsdale, D. W.; and Embley, D. W. (2015). "Ontology-Based Information Extraction with a Cognitive Agent". In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 558–564.

[Linson et al., 2015] Linson, A.; Dobbyn, C.; Lewis, G. E.; and Laney, R. C. (2015). "A Subsumption Agent for Collaborative Free Improvisation". *Computer Music Journal*, 39(4), pages 96–115.

[Liu et al., 2014] Liu, C.; Li, Y.; and Shen, W. (2014). "Integrated manufacturing process planning and control based on intelligent agents and multi-dimension features". *The International Journal of Advanced Manufacturing Technology*, 75(9), pages 1457–1471.

[Liu et al., 2016] Liu, G.; Reisig, W.; Jiang, C.; and Zhou, M. (2016). "A Branching-Process-Based Method to Check Soundness of Workflow Systems". *IEEE Access*, 4, pages 4104–4118.

[Liu et al., 2013] Liu, R.; Agarwal, S.; Sindhgatta, R.; and Lee, J. (2013). "Accelerating Collaboration in Task Assignment Using a Socially Enhanced Resource Model". In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 251–258.

[Liu and Iijima, 2015] Liu, Y. and Iijima, J. (2015). "Business process simulation in the context of enterprise engineering". *J. Simulation*, 9(3), pages 206–222.

[Lohmann, 2007] Lohmann, N. (2007). "A Feature-Complete Petri Net Semantics for WS-BPEL 2.0". In Dumas, M. and Heckel, R., editors, *Web Services and Formal Methods, 4th International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007. Proceedings*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer.

[Lohmann et al., 2008] Lohmann, N.; Massuthe, P.; Stahl, C.; and Weinberg, D. (2008). "Analyzing interacting WS-BPEL processes using flexible model generation". *Data Knowl. Eng.*, 64(1), pages 38–54.

[Lohrmann and Reichert, 2016] Lohrmann, M. and Reichert, M. (2016). "Effective application of process improvement patterns to business processes". *Software and System Modeling*, 15(2), pages 353–375.

[Lomazova and Schnoebelen, 2000] Lomazova, I. A. and Schnoebelen, P. (2000). "Some Decidability Results for Nested Petri Nets". In Bjøner, D.; Broy, M.; and Zamulin, A. V., editors, *Perspectives of System Informatics: Third International Andrei Ershov Memorial Conference, PSI'99 Akademgorodok, Novosibirsk, Russia July 6–9, 1999 Proceedings*, pages 208–220. Springer-Verlag, Berlin, Heidelberg.

[López-Mellado and Flores-Badillo, 2013] López-Mellado, E. and Flores-Badillo, M. (2013). "A Mobile Agent Based Approach for Business Processes Automation". *IJRAT*, 1(2), pages 1–10.

[Löwe et al., 2001] Löwe, W.; Ludwig, A.; and Schwind, A. (2001). "Understanding software - static and dynamic aspects". In *In 17th International Conference on Advanced Science and Technology*, pages 52–57.

[Lübke and van Lessen, 2017] Lübke, D. and van Lessen, T. (2017). "Modeling Test Cases in BPMN for Behavior-Driven Development". In Jürjens, J. and Schneider, K., editors, *Software Engineering 2017, Fachtagung des GI-Fachbereichs Softwaretechnik, 21.-24. Februar 2017, Hannover, Deutschland*, volume P-267 of *LNI*, pages 85–86. GI.

[Luck et al., 2003] Luck, M.; McBurney, P.; and Preist, C. (2003). *Agent Technology: Enabling Next Generation Computing*. AgentLink II.

[Ludäscher et al., 2009] Ludäscher, B.; Weske, M.; McPhillips, T.; and Bowers, S. (2009). "Scientific Workflows: Business as Usual?". In Dayal, U.; Eder, J.; Koehler, J.; and Reijers, H. A., editors, *Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, pages 31–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Luong et al., 2017] Luong, B. V.; Thangarajah, J.; and Zambetta, F. (2017). "A BDI Game Master Agent for Computer Role-Playing Games". *Computers in Entertainment*, 15(1), pages 4:1–4:16.

[Lützenberger et al., 2014] Lützenberger, M.; Konnerth, T.; Küster, T.; Tonn, J.; Masuch, N.; and Albayrak, S. (2014). "Engineering JIAC multi-agent systems". In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 1647–1648.

[Lützenberger et al., 2013] Lützenberger, M.; Küster, T.; Konnerth, T.; Thiele, A.; Masuch, N.; Heßler, A.; Keiser, J.; Burkhardt, M.; Kaiser, S.; and Albayrak, S. (2013). "JIAC V: A MAS framework for industrial applications". In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1189–1190.

[Lützenberger et al., 2016] Lützenberger, M.; Küster, T.; Masuch, N.; and Fähndrich, J. (2016). "Multi-Agent System in Practice: When Research Meets Reality". In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 796–805.

[Ly et al., 2015] Ly, L. T.; Maggi, F. M.; Montali, M.; Rinderle-Ma, S.; and van der Aalst, W. M. P. (2015). "Compliance monitoring in business processes: Functionalities, application, and tool-support". *Inf. Syst.*, 54, pages 209–234.

[Madhusudan, 2005] Madhusudan, T. (2005). "An agent-based approach for coordinating product design workflows". *Computers in Industry*, 56(3), pages 235–259.

[Maeda et al., 2007] Maeda, Y.; Kikuchi, H.; Izawa, H.; Ogawa, H.; Sugi, M.; and Arai, T. (2007). "“Plug & Produce" functions for an easily reconfigurable robotic assembly cell". *Assembly Automation*, 27(3), pages 253–260.

[Malinova et al., 2013] Malinova, M.; Dijkman, R. M.; and Mendling, J. (2013). "Automatic Extraction of Process Categories from Process Model Collections". In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pages 430–441. Springer.

[Mallek et al., 2015] Mallek, S.; Daclin, N.; Chapurlat, V.; and Vallespir, B. (2015). "Enabling model checking for collaborative process analysis: from BPMN to 'Network of Timed Automata"'. *Enterprise IS*, 9(3), pages 279–299.

[Mangler and Rinderle-Ma, 2014] Mangler, J. and Rinderle-Ma, S. (2014). "CPEE - Cloud Process Execution Engine". In *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014.*, page 51.

[Manmin and Huaicheng, 1999] Manmin, X. and Huaicheng, L. (1999). "Cooperative software agents for workflow management system". In *Fifth Asia-Pacific Conference on ... and Fourth Optoelectronics and Communications Conference on Communications,*, volume 2, pages 1063–1067 vol.2.

[Marin and Brena, 2005] Marin, C. A. and Brena, R. F. (2005). "Multiagent Architecture for Decentralized Workflow Process Execution". Technical report, Center for Intelligent Systems Tecnologico de Monterrey, Monterrey, Mexico.

[Markwardt, 2012] Markwardt, K. (2012). *Strukturierung petrinetzbasierter Multiagentenanwendungen am Beispiel verteilter Softwareentwicklungsprozesse*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Markwardt et al., 2009a] Markwardt, K.; Cabac, L.; and Reese, C. (2009a). "A Process-Oriented Tool-Platform for Distributed Development". In Moldt, D.; Augusto, J. C.; and Ultes-Nitsche, U., editors, *Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2009, In conjunction with ICEIS 2009, Milan, Italy, May 2009*, pages 44–52. INSTICC PRESS.

[Markwardt et al., 2009b] Markwardt, K.; Moldt, D.; and Wagner, T. (2009b). "Net Agents for Activity Handling in a WFMS". In Freytag, T. and Eckleder, A., editors, *16th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2009, Karlsruhe, Germany, September 25, 2009, Proceedings*, CEUR Workshop Proceedings.

[Marozzo et al., 2015] Marozzo, F.; Talia, D.; and Trunfio, P. (2015). "JS4Cloud: script-based workflow programming for scalable data analysis on cloud platforms". *Concurrency and Computation: Practice and Experience*, 27(17), pages 5214–5237.

[Marquard et al., 2015] Marquard, M.; Shahzad, M.; and Slaats, T. (2015). "Web-Based Modelling and Collaborative Simulation of Declarative Processes". In *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, pages 209–225.

[Mascheroni et al., 2010] Mascheroni, M.; Wagner, T.; and Wüstenberg, L. (2010). "Verifying Reference Nets By Means of Hypernets: A Plugin for Renew". In *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, pages 39–54.

[Mateo et al., 2015] Mateo, J. A.; Srba, J.; and Sørensen, M. G. (2015). "Soundness of Timed-Arc Workflow Nets in Discrete and Continuous-Time Semantics". *Fundam. Inform.*, 140(1), pages 89–121.

[Mazurkiewicz, 1987] Mazurkiewicz, A. (1987). "Trace theory". In Brauer, W.; Reisig, W.; and Rozenberg, G., editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency: Advances in Petri Nets 1986, Part II Proceedings of an Advanced Course Bad Honnef, 8.–19. September 1986*, pages 278–324. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Mazzara and Govoni, 2005] Mazzara, M. and Govoni, S. (2005). "A Case Study of Web Services Orchestration". In Jacquet, J.-M. and Picco, G. P., editors, *Coordination Models and Languages: 7th International Conference, COORDINATION 2005, Namur, Belgium, April 20-23, 2005. Proceedings*, pages 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg.

[McCarthy et al., 1955] McCarthy, J.; Minsky, M. L.; Rochester, N.; and Shannon, C. E. (1955). "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence". http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

[McGovern et al., 2006] McGovern, J.; Sims, O.; Jain, A.; and Little, M. (2006). *Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations*. Springer Netherlands, Dordrecht, The Netherlands.

[McMillan, 1992] McMillan, K. L. (1992). "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits". In Bochmann, G. and Probst, D. K., editors, *Computer Aided Verification: Fourth International Workshop, CAV '92 Montreal, Canada, June 29 – July 1, 1992 Proceedings*, pages 164–177. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Meghzili et al., 2016] Meghzili, S.; Chaoui, A.; Strecker, M.; and Kerkouche, E. (2016). "Transformation and validation of BPMN models to Petri nets models using GROOVE". In *2016 International Conference on Advanced Aspects of Software Engineering, ICAASE 2016, Constantine, Algeria, October 29-30, 2016*, pages 22–29. IEEE.

[Mendling et al., 2011] Mendling, J.; Weidlich, M.; and Weske, M., editors (2011). *Business Process Modeling Notation - Second International Workshop, BPMN 2010, Potsdam, Germany, October 13-14, 2010. Proceedings*, volume 67 of *Lecture Notes in Business Information Processing*. Springer.

[Meneguzzi and de Silva, 2015] Meneguzzi, F. and de Silva, L. (2015). "Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning". *Knowledge Eng. Review*, 30(1), pages 1–44.

[Merdan et al., 2013] Merdan, M.; Moser, T.; Sunindyo, W. D.; Biffl, S.; and Vrba, P. (2013). "Workflow scheduling using multi-agent systems in a dynamically changing environment". *J. Simulation*, 7(3), pages 144–158.

[Merritts, 2013] Merritts, R. A. (2013). *Online Deception Detection Using BDI Agents*. PhD thesis, Graduate School of Computer and Information Sciences Nova Southeastern University.

[Merz and Lamersdorf, 1999] Merz, M. and Lamersdorf, W. (1999). "Crossing Organizational Boundaries with Mobile Agents in Electronic Service Markets". *Integrated Computer-Aided Engineering*, 6(2), pages 91–104.

[Meyer, 2014] Meyer, B. (2014). *Agile! - The Good, the Hype and the Ugly*. Springer-Verlag.

[Mininger and Laird, 2016] Mininger, A. and Laird, J. E. (2016). "Interactively Learning Strategies for Handling References to Unseen or Unknown Objects". In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems*.

[Mislevics and Grundspenkis, 2012] Mislevics, A. and Grundspenkis, J. (2012). "Workflow based approach for designing and executing mobile agents". In *Digital Information Processing and Communications (ICDIPC), Second International Conference on*, pages 191–203.

[Mohan et al., 2012] Mohan, S.; Mininger, A.; Kirk, J.; and Laird, J. E. (2012). "Learning Grounded Language through Situated Interactive Instruction". In *Robots Learning Interactively from Human Teachers, Papers from the 2012 AAAI Fall Symposium, Arlington, Virginia, USA, November 2-4, 2012*, volume FS-12-07 of *AAAI Technical Report*. AAAI.

[Moldt, 1996] Moldt, D. (1996). *Höhere Petrinetze als Grundlage für Systemspezifikationen*. Dissertation, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Moldt, 2005] Moldt, D. (2005). "Petrinetze als Denkzeug". In Farwer, B. and Moldt, D., editors, *Object Petri Nets, Processes, and Object Calculi*, number FBI-HH-B-265/05 in Report of the Department of Informatics, pages 51–70, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Computer Science.

[Moldt et al., 2010] Moldt, D.; Quenum, J.; Reese, C.; and Wagner, T. (2010). "Improving a Workflow Management System with an Agent Flavour". In Duvigneau, M. and Moldt, D., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, number FBI-HH-B-294/10 in Bericht, pages 55–70, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Informatics.

[Moldt and Wienberg, 1997] Moldt, D. and Wienberg, F. (1997). "Multi-Agent-Systems based on Coloured Petri Nets". In Azéma, P. and Balbo, G., editors, *Application and Theory of Petri Nets 1997*, number 1248 in Lecture Notes in Computer Science, pages 82–101, Berlin Heidelberg New York. Springer Verlag.

[Monakova and Leymann, 2013] Monakova, G. and Leymann, F. (2013). "Workflow ART: a framework for multidimensional workflow analysis". *Enterprise IS*, 7(1), pages 133–166.

[Monett and Navarro-Barrientos, 2016] Monett, D. and Navarro-Barrientos, J. E. (2016). "Simulating the Fractional Reserve Banking using Agent-based Modelling with NetLogo". In Ganzha, M.; Maciaszek, L. A.; and Paprzycki, M., editors, *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.*, pages 1467–1470.

[Morandini et al., 2014] Morandini, M.; Dalpiaz, F.; Nguyen, C. D.; and Siena, A. (2014). "The Tropos Software Engineering Methodology". In *Handbook on Agent-Oriented Design Processes*, pages 463–490. Springer-Verlag.

[Mosteller, 2010] Mosteller, D. (2010). "Entwicklung eines Werkzeugs zur Modellierung der initialen Wissensbasen und Rollen-Abhängigkeiten in Multiagentenanwendungen im Kontext von PAOSE / MULAN". Bachelor's thesis, University of Hamburg, Department of Informatics.

[Mosteller, 2016] Mosteller, J. (2016). "Persistierung in agentenorientierten Anwendungen – Prototypische Implementierung im Kontext von Renew/Mulan". Master's thesis, University of Hamburg, Department of Informatics.

[Müller, 2016] Müller, E. (2016). "Bereitstellung eines JavaScript-basierten Frameworks für die Entwicklung von agentenorientierten Webanwendungen". Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Murata, 1989] Murata, T. (1989). "Petri Nets: Properties, Analysis and Applications". In *Proceedings of the IEEE*, pages 541–580. Published as Proceedings of the IEEE, volume 77, number 4.

[Murphy, 2000] Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition.

[Naoui et al., 2014] Naoui, M. A.; Mcheick, H.; and Kazar, O. (2014). "Mobile Agent approach based on mobile strategic environmental Scanning using Android and JADELEAP systems". In *IEEE 27th Canadian Conference on Electrical and Computer Engineering, CCECE 2014, Toronto, ON, Canada, May 4-7, 2014*, pages 1–7. IEEE.

[Narock and Yoon, 2015] Narock, T. and Yoon, V. Y. (2015). "An agent-based approach for capturing and linking provenance in geoscience workflows". *Computers & Geosciences*, 79, pages 58–68.

[Nazaruka et al., 2016] Nazaruka, E.; Ovchinnikova, V.; Alksnis, G.; and Sukovskis, U. (2016). "Verification of BPMN Model Functional Completeness by using the Topological Functioning Model". In Maciaszek, L. A. and Filipe, J., editors, *ENASE 2016 - Proceedings of the 11th International Conference on Evaluation of Novel Approaches to Software Engineering, Rome, Italy 27-28 April, 2016.*, pages 349–358. SciTePress.

[Neumann and Morgenstern, 1944] Neumann, J. V. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

[Nezhad and Swenson, 2013] Nezhad, H. R. M. and Swenson, K. D. (2013). "Adaptive Case Management: Overview and Research Challenges". In *IEEE 15th Conference on Business Informatics, CBI 2013, Vienna, Austria, July 15-18, 2013*, pages 264–269.

[Nouzri and Fazziki, 2015] Nouzri, S. and Fazziki, A. E. (2015). "A mapping from bpmn model to JADEX model". *Int. Arab J. Inf. Technol.*, 12(1), pages 77–85.

[Nunes et al., 2011] Nunes, I.; Lucena, C. J. P. D.; and Luck, M. (2011). "BDI4JADE: a BDI layer on top of JADE". In *International Workshop on Programming MultiAgent Systems*, pages 88–103.

[Nwana et al., 1999] Nwana, H. S.; Ndumu, D. T.; Lee, L. C.; and Collis, J. C. (1999). "ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems". In *Agents*, pages 360–361.

[O'Brien and Wiegard, 1998] O'Brien, P. and Wiegard, W. (1998). "Agent based process management: applying intelligent agents to workflow". *The Knowledge Engineering Review*, 13(2), pages 161–174.

[Oland et al., 2016] Oland, E.; Andersen, T. S.; and Kristiansen, R. (2016). "Subsumption architecture applied to flight control using composite rotations". *Automatica*, 69, pages 195–200.

[Omicini et al., 2006] Omicini, A.; Ricci, A.; and Zaghini, N. (2006). "Distributed Workflow upon Linkable Coordination Artifacts". In *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, pages 228–246.

[Ooi, 2003] Ooi, B. (2003). "A Multi-agent Approach to Business Processes Management in an Electronic Market". In *Intelligent Agents and Multi-Agent Systems, 6th Pacific Rim International Workshop on Multi-Agents, PRIMA 2003, Seoul, Korea, November 7-8, 2003, Proceedings*, pages 1–12.

[Ortmann, 2010] Ortmann, J. F. (2010). *Dienstbeschreibungsnetze als Modellierungstechnik für dienstbasierte Geschäftsprozesse*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Ouyang et al., 2007a] Ouyang, C.; van der Aalst, W. M.; Dumas, M.; ter Hofstede, A. H.; and Rosa, M. L. (2007a). "Service-Oriented Processes: An Introduction to BPEL". In Cardoso, J., editor, *Semantic Web Services: Theory, Tools and Applications*, chapter 8, pages 155–190. IGI Global.

[Ouyang et al., 2007b] Ouyang, C.; Verbeek, E.; van der Aalst, W. M. P.; Breutel, S.; Dumas, M.; and ter Hofstede, A. H. M. (2007b). "Formal semantics and analysis of control flow in WS-BPEL". *Sci. Comput. Program.*, 67(2-3), pages 162–198.

[Padgham et al., 2014] Padgham, L.; Thangarajah, J.; and Winikoff, M. (2014). "Prometheus Research Directions". In *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*, pages 155–171. Springer-Verlag.

[Padgham and Winikoff, 2002] Padgham, L. and Winikoff, M. (2002). "Prometheus: A Methodology for Developing Intelligent Agents". In *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, pages 174–185.

[Pan and Choi, 2016] Pan, A. and Choi, T. (2016). "An agent-based negotiation model on price and delivery date in a fashion supply chain". *Annals OR*, 242(2), pages 529–557.

[Pantoja et al., 2016] Pantoja, C. E.; Jr., M. F. S.; Lazarin, N. M.; and Sichman, J. S. (2016). "ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming". In *Engineering Multi-Agent Systems - 4th International Workshop, EMAS 2016, Singapore, Singapore, May 9-10, 2016, Revised, Selected, and Invited Papers*, pages 136–155.

[Parsons et al., 2002] Parsons, S.; Gmytrasiewicz, P.; and Wooldridge, M., editors (2002). *An Introduction to Game Theory and Decision Theory*, volume 5 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer-Verlag, Berlin Heidelberg New York.

[Passos and Julia, 2013] Passos, L. M. S. and Julia, S. (2013). "Qualitative Analysis of Interorganizational WorkFlow Nets Using Linear Logic: Soundness Verification". In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 667–673.

[Perez-Diaz et al., 2015] Perez-Diaz, F.; Zillmer, R.; and Groß, R. (2015). "Firefly-Inspired Synchronization in Swarms of Mobile Agents". In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 279–286.

[Perry and Wolf, 1992] Perry, D. E. and Wolf, A. L. (1992). "Foundations for the Study of Software Architecture". *SIGSOFT Software Engineering Notes*, 17(4), pages 40–52.

[Petri, 1962] Petri, C. A. (1962). *Kommunikation mit Automaten*. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn.

[Pokahr, 2007] Pokahr, A. (2007). *Programmiersprachen und Werkzeuge zur Entwicklung verteilter agentenorientierter Softwaresysteme*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany.

[Pokahr and Braubach, 2009] Pokahr, A. and Braubach, L. (2009). "From a Research to an Industrial-Strength Agent Platform: Jadex V2". In Hans Robert Hansen, Dimitris Karagiannis, H.-G. F., editor, *Business Services: Konzepte, Technologien, Anwendungen - 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, pages 769–778. Österreichische Computer Gesellschaft Wien.

[Pokahr and Braubach, 2011] Pokahr, A. and Braubach, L. (2011). "Active Components: A Software Paradigm for Distributed Systems". In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 141–144.

[Pokahr and Braubach, 2013] Pokahr, A. and Braubach, L. (2013). "The Active Components Approach for Distributed Systems Development". *Int. J. Parallel Emerg. Distrib. Syst.*, 28(4), pages 321–369.

[Pokahr et al., 2010] Pokahr, A.; Braubach, L.; and Jander, K. (2010). "Unifying Agent and Component Concepts". In Dix, J. and Witteveen, C., editors, *Multiagent System Technologies: 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings*, pages 100–112. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Pokahr et al., 2013] Pokahr, A.; Braubach, L.; and Jander, K. (2013). "Jadex: A Generic Programming Model and One-Stop-Shop Middleware for Distributed Systems". *Praxis der Informationsverarbeitung und Kommunikation*, 36(2), pages 149–150.

[Pokahr et al., 2003] Pokahr, A.; Braubach, L.; and Lamersdorf, W. (2003). "Jadex: Implementing a BDI-Infrastructure for JADE Agents". *EXP - in search of innovation (Special Issue on JADE)*, 3(3), pages 76–85.

[Pokahr et al., 2005] Pokahr, A.; Braubach, L.; and Lamersdorf, W. (2005). "Jadex: A BDI Reasoning Engine". In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. Springer-Verlag.

[Poveda and Schumann, 2016] Poveda, G. and Schumann, R. (2016). "An Ontology-Driven Approach for Modeling a Multi-agent-Based Electricity Market". In *Multiagent System Technologies - 14th German Conference, MATES 2016, Klagenfurt, Österreich, September 27-30, 2016. Proceedings*, pages 27–40.

[Preisler, 2013] Preisler, T. (2013). "Policy-Based Approach for Non-Functional Requirements in Self-adaptive and Self-Organizing Systems". In *Multiagent System Technologies - 11th German Conference, MATES 2013, Koblenz, Germany, September 16-20, 2013. Proceedings*, pages 420–423.

[Premm and Kirn, 2015] Premm, M. and Kirn, S. (2015). "A Multiagent Systems Perspective on Industry 4.0 Supply Networks". In *Multiagent System Technologies - 13th German Conference, MATES 2015, Cottbus, Germany, September 28-30, 2015, Revised Selected Papers*, pages 101–118.

[Priese and Wimmel, 1998] Priese, L. and Wimmel, H. (1998). "A uniform approach to true-concurrency and interleaving semantics for Petri nets". *Theoretical Computer Science*, 206(1–2), pages 219–256.

[Prisecaru and Jucan, 2008] Prisecaru, O. and Jucan, T. (2008). "Interorganizational Workflow Nets: a Petri Net Based Approach for Modelling and Analyzing Interorganizational Workflows". In Barjis, J.; Narasipuram, M. M.; Rittgen, P.; Hunt, E.; Franch, X.; and Coletta, R., editors, *Proceedings of the 4th International Workshop on Enterprise & Organizational Modeling and Simulation held in conjunction with the CAiSE'08 Conference, Montpellier, France, June 16–17, 2008*, pages 64–78.

[Puviani et al., 2015] Puviani, M.; Cabri, G.; Capodieci, N.; and Leonardi, L. (2015). "Building Self-adaptive Systems by Adaptation Patterns Integrated into Agent Methodologies". In *Agents and Artificial Intelligence - 7th International Conference, ICAART 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers*, pages 58–75.

[Queiroz and Leitão, 2016] Queiroz, J. and Leitão, P. (2016). "Agent-Based Data Analysis Towards the Dynamic Adaptation of Industrial Automation Processes". In *Technological Innovation for Cyber-Physical Systems - 7th IFIP WG 5.5/SOCOLNET Advanced Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2016, Costa de Caparica, Portugal, April 11-13, 2016, Proceedings*, pages 99–106.

[Rachdi et al., 2016a] Rachdi, A.; En-Nouaary, A.; and Dahchour, M. (2016a). "Liveness and Reachability Analysis of BPMN Process Models". *CIT*, 24(2), pages 195–207.

[Rachdi et al., 2016b] Rachdi, A.; En-Nouaary, A.; and Dahchour, M. (2016b). "Verification of Common Business Rules in BPMN Process Models". In Abdulla, P. A. and Delporte-Gallet, C., editors, *Networked Systems - 4th International Conference, NETYS 2016, Marrakech, Morocco, May 18-20, 2016, Revised Selected Papers*, volume 9944 of *Lecture Notes in Computer Science*, pages 334–339. Springer.

[Rantrua et al., 2013] Rantrua, A.; Gleizes, M. P.; and Hanachi, C. (2013). "Flexible and Emergent Workflows Using Adaptive Agents". In *Computational Collective Intelligence. Technologies and Applications - 5th International Conference, ICCCI 2013, Craiova, Romania, September 11-13, 2013, Proceedings*, pages 185–194.

[Rao, 1996] Rao, A. S. (1996). "AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language". In de Velde, W. V. and Perram, J. W., editors, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer.

[Rao and Georgeff, 1991] Rao, A. S. and Georgeff, M. P. (1991). "Modeling Rational Agents within a BDI-Architecture". In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991.*, pages 473–484.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). "BDI Agents: From Theory to Practice". In *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319.

[Recker and Mendling, 2016] Recker, J. and Mendling, J. (2016). "The State of the Art of Business Process Management Research as Published in the BPM Conference". *Business & Information Systems Engineering*, 58(1), pages 55–72.

[Reese, 2003] Reese, C. (2003). "Multiagentensysteme: Anbindung der petrinetzbasierten Plattform CAPA an das internationale Netzwerk Agentcities". Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Reese, 2010] Reese, C. (2010). *Prozess-Infrastruktur für Agentenanwendungen*, volume 3 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin.

[Reese et al., 2008] Reese, C.; Wester-Ebbinghaus, M.; Dörges, T.; Cabac, L.; and Moldt, D. (2008). "Introducing a Process Infrastructure for Agent Systems". In Dastani, M.; Fallah, A. E.; ao Leite, J.; and Torroni, P., editors, *LADS'007 Languages, Methodologies and Development Tools for Multi-Agent Systems*, volume 5118 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Revised Selected and Invited Papers.

[Reichert, 2000] Reichert, M. (2000). *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis, Universität Ulm.

[Reichert et al., 2009a] Reichert, M.; Bauer, T.; and Dadam, P. (2009a). "Flexibility for Distributed Workflows". In *Handbook of Research on Complex Dynamic Process Management: Techniques for Adaptability in Turbulent Environments*.

[Reichert et al., 2009b] Reichert, M.; Dadam, P.; Rinderle-Ma, S.; Jurisch, M.; Kreher, U.; and Göser, K. (2009b). "Architectural Principles and Components of Adaptive Process Management Technology". In *Lecture Notes in Informatics (LNI)*. PRIMIUM - Process Innovation for Enterprise Software.

[Reichert et al., 2005] Reichert, M.; Rinderle, S.; Kreher, U.; and Dadam, P. (2005). "Adaptive Process Management with ADEPT2". In Aberer, K.; Franklin, M. J.; and Nishio, S., editors, *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 1113–1114. IEEE Computer Society.

[Reichert et al., 2009c] Reichert, M.; Rinderle-Ma, S.; and Dadam, P. (2009c). "Flexibility in Process-Aware Information Systems". *Trans. Petri Nets and Other Models of Concurrency*, 2, pages 115–135.

[Reichert and Weber, 2012] Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems*. Springer-Verlag.

[Reisig, 1987] Reisig, W. (1987). "Place/Transition Systems". In Brauer, W.; Reisig, W.; and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties: Advances in Petri Nets 1986, Part I Proceedings of an Advanced Course Bad Honnef, 8.–19. September 1986*, pages 117–141. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Reisig, 2013] Reisig, W. (2013). *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer-Verlag, Berlin Heidelberg New York.

[Reuter and Dadam, 2013] Reuter, C. and Dadam, P. (2013). "Navigieren statt modellieren - Flexible Prozessgestaltung durch Endanwender". *Informatik Spektrum*, 36(4), pages 359–370.

[Ricci et al., 2002] Ricci, A.; Omicini, A.; and Denti, E. (2002). "Virtual Enterprises and Workflow Management As Agent Coordination Issues". *Int. J. Cooperative Inf. Syst.*, 11(3), pages 355–380.

[Riemann, 2015] Riemann, U. (2015). "Benefits and Challenges for BPM in the Cloud". *IJOCI*, 5(1), pages 32–61.

[Rimassa et al., 2006] Rimassa, G.; Greenwood, D. A. P.; and Kernland, M. E. (2006). "The Living Systems Technology Suite: An Autonomous Middleware for Autonomic Computing". In *2006 International Conference on Autonomic and Autonomous Systems (ICAS 2006), 16-21 July 2006, Silicon Valley, California, USA*, page 33. IEEE Computer Society.

[Rinderle et al., 2007] Rinderle, S.; Jurisch, M.; and Reichert, M. (2007). "On Deriving Net Change Information From Change Logs - The DELTALAYER-Algorithm". In *Proc. 12th Conf. Datenbanksysteme in Business, Technologie und Web (BTW'07), Aachen, Germany.*

[Rinderle, 2004] Rinderle, S. B. (2004). *Schema Evolution in Process Management Systems.* PhD thesis, Universität Ulm, Abt. Datenbanken und Informationssysteme.

[Rinderle-Ma et al., 2006] Rinderle-Ma, S.; Kreher, U.; Lauer, M.; Dadam, P.; and Reichert, M. (2006). "On Representing Instance Changes in Adaptive Process Management Systems". In *First IEEE Workshop on Flexibility in Process-aware Information Systems (ProFlex'06)*, pages 297–302, Manchester. IEEE Computer Society Press.

[Rivera-Villicana et al., 2016] Rivera-Villicana, J.; Zambetta, F.; Harland, J.; and Berry, M. (2016). "Towards a BDI Player Model for Interactive Narratives". In Kaminka, G. A.; Fox, M.; Bouquet, P.; Hüllermeier, E.; Dignum, V.; Dignum, F.; and van Harmelen, F., editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1648–1649. IOS Press.

[Roa et al., 2016] Roa, J.; Chiotti, O.; and Villarreal, P. D. (2016). "Specification of behavioral anti-patterns for the verification of block-structured Collaborative Business Processes". *Information & Software Technology*, 75, pages 148–170.

[Rodriguez et al., 2014] Rodriguez, S.; Gaud, N.; and Galland, S. (2014). "SARL: A General-Purpose Agent-Oriented Programming Language". In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, August 11-14, 2014 - Volume III*, pages 103–110. IEEE Computer Society.

[Roeser and Kern, 2015] Roeser, T. and Kern, E. (2015). "Surveys in business process management - a literature review". *Business Proc. Manag. Journal*, 21(3), pages 692–718.

[Rogge-Solti et al., 2016] Rogge-Solti, A.; Senderovich, A.; Weidlich, M.; Mendling, J.; and Gal, A. (2016). "In Log and Model We Trust? A Generalized Conformance Checking Framework". In [Rosa et al., 2016], pages 179–196.

[Rojek, 2016] Rojek, G. (2016). "Agents Retaining and Reusing of Experience Applied to Control of Semi-continuous Production Process". In Rutkowski, L.; Korytkowski, M.; Scherer, R.; Tadeusiewicz, R.; Zadeh, L. A.; and Zurada, J. M., editors, *Artificial Intelligence and Soft Computing - 15th International Conference, ICAISC 2016, Zakopane, Poland, June 12-16, 2016, Proceedings, Part I*, volume 9692 of *Lecture Notes in Computer Science*, pages 729–738. Springer.

[Rölke, 2004] Rölke, H. (2004). *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications.* Logos Verlag, Berlin.

[Rosa et al., 2016] Rosa, M. L.; Loos, P.; and Pastor, O., editors (2016). *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science.* Springer.

[Rozanski and Woods, 2005] Rozanski, N. and Woods, E. (2005). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley Professional.

[Russel and Norvig, 1995] Russel, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

[Russell and ter Hofstede, 2009a] Russell, N. and ter Hofstede, A. H. M. (2009a). "newYAWL: Towards Workflow 2.0". *Trans. Petri Nets and Other Models of Concurrency*, 2, pages 79–97.

[Russell and ter Hofstede, 2009b] Russell, N. and ter Hofstede, A. H. M. (2009b). "Surmounting BPM challenges: the YAWL story". *Computer Science - R&D*, 23(2), pages 67–79.

[Russell et al., 2006] Russell, N.; ter Hofstede, A. H. M.; van der Aalst, W. M. P.; and Mulyar, N. (2006). "Workflow Control-Flow Patterns: A Revised View". Technical report, BPMcenter.org.

[Russell et al., 2005] Russell, N.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.; and Edmond, D. (2005). "Workflow Resource Patterns: Identification, Representation and Tool Support". In Pastor, O. and Falcão e Cunha, J., editors, *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005. Proceedings*, pages 216–232. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Russell et al., 2016] Russell, N.; van van der Aalst, W. M.; and ter Hofstede, A. H. M. (2016). *Workflow Patterns: The Definitive Guide*. The MIT Press.

[Russell et al., 2009] Russell, N. C.; van der Aalst, W. M. P.; and ter Hofstede, A. H. M. (2009). "Designing a Workflow System Using Coloured Petri Nets". In Jensen, K.; Billington, J.; and Koutny, M., editors, *Transactions on Petri Nets and Other Models of Concurrency III*, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[Sackmann and Kittel, 2015] Sackmann, S. and Kittel, K. (2015). "Flexible Workflows and Compliance: A Solvable Contradiction?!". In *BPM - Driving Innovation in a Digital World*, pages 247–258. Springer-Verlag.

[Said et al., 2016] Said, I. B.; Chaâbane, M. A.; Bouaziz, R.; and Andonoff, E. (2016). "A Version-based Approach to Address Flexibility of BPMN Collaborations and Choreographies". In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 2: ICE-B, Lisbon, Portugal, July 26-28, 2016.*, pages 31–42.

[Saleem et al., 2014] Saleem, M.; Chung, P. W. H.; Fatima, S.; and Dai, W. (2014). "A cross organisation compatible workflows generation and execution framework". *Knowl.-Based Syst.*, 56, pages 1–14.

[Samiri et al., 2017] Samiri, M. Y.; Najib, M.; El Fazziki, A.; and Boukachour, J. (2017). "Toward a Self-Adaptive Workflow Management System Through Learning and Prediction Models". *Arabian Journal for Science and Engineering*, 42(2), pages 897–912.

[Sandhu et al., 1996] Sandhu, R. S.; Coyne, E. J.; Feinstein, H. L.; and Youman, C. E. (1996). "Role-based access control models". *Computer*, 29(2), pages 38–47.

[Santos-Pereira et al., 2013] Santos-Pereira, C.; Augusto, A. B.; Cruz-Correia, R.; and Correia, M. E. (2013). "A secure RBAC mobile agent access control model for healthcare institutions". In *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems*, pages 349–354.

[Savarimuthu et al., 2005] Savarimuthu, B. T. R.; Purvis, M.; and Purvis, M. K. (2005). "Different Perspectives on Modeling Workflows in an Agent Based Workflow Management System". In *Knowledge-Based Intelligent Information and Engineering Systems, 9th International Conference, KES 2005, Melbourne, Australia, September 14-16, 2005, Proceedings, Part IV*, pages 208–214.

[Scheer and Nüttgens, 2000] Scheer, A. and Nüttgens, M. (2000). "ARIS Architecture and Reference Models for Business Process Management". In van der Aalst, W. M. P.; Desel, J.; and Oberweis, A., editors, *Business Process Management, Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer.

[Schleinzer, 2007] Schleinzer, B. (2007). "Flexible und hierarchische Multiagentensysteme – Modellierung und prototypische Erweiterung von Mulan und Capa". Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Schleinzer et al., 2008] Schleinzer, B.; Cabac, L.; Moldt, D.; and Duvigneau, M. (2008). "From Agents and Plugins to Plugin-Agents, Concepts for Flexible Architectures". In *New Technologies, Mobility and Security, 2008. International Conference, NTMS '08, Tangier, Morocco. Electronical proceedings*, pages 1–5. IEEE Xplore.

[Schmidt, 2016] Schmidt, A. (2016). "Integration von ActiveMQ in eine Petrinetz-basierte Agentenumgebung". Bachelor's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Schmidt, 2000a] Schmidt, K. (2000a). "LoLA - A Low Level Analyser". In Nielsen, M. and Simpson, D., editors, *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000 Aarhus, Denmark, June 26–30, 2000 Proceedings*, pages 465–474. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Schmidt, 2000b] Schmidt, K. (2000b). "LoLA: A Low Level Analyser". In *ICATPN*, pages 465–474.

[Schmitz, 2016] Schmitz, D. (2016). "Neugestaltung von Lernmaterialien zur Unterstützung der Lehrenden und Lernenden in der petrinetzbasierten und agentenorientierten Softwareentwicklung". Master's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Schmitz et al., 2016] Schmitz, D.; Moldt, D.; Cabac, L.; Mosteller, D.; and Haustermann, M. (2016). "Utilizing Petri Nets for Teaching in Practical Courses on Collaborative Software Engineering". In *16th International Conference on Application of Concurrency to System Design, ACSD 2016, Toruń, Poland, June 19-24, 2016*, pages 74–83. IEEE Computer Society.

[Schmolke, 2015] Schmolke, P. (2015). " Examination of adaptation concepts and mechanisms of agent-based AI-processes for multi-agent systems on the basis of Petrinets and the PAOSE process". Master's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Schönig and Zeising, 2015] Schönig, S. and Zeising, M. (2015). "The DPIL Framework: Tool Support for Agile and Resource-Aware Business Processes". In *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015.*, pages 125–129.

[Schulte et al., 2015] Schulte, S.; Janiesch, C.; Venugopal, S.; Weber, I.; and Hoenisch, P. (2015). "Elastic Business Process Management: State of the art and open challenges for BPM in the cloud". *Future Generation Comp. Syst.*, 46, pages 36–50.

[Schumm et al., 2009] Schumm, D.; Karastoyanova, D.; Leymann, F.; and Nitzsche, J. (2009). "On Visualizing and Modelling BPEL with BPMN". In Müller, H.; Chen, J.; Cafaro, M.; Park, J. H.; and Abdennadher, N., editors, *IEEE Proceedings of the 4th International Workshop on Workflow Management (IWWM2009)*, pages 80–87, Los Alamitos, California. IEEE Computer Society.

[Scott et al., 2016] Scott, N.; Livingston, M.; Hart, A.; Wilson, J.; Moore, D.; and Dietze, P. (2016). "SimDrink: An Agent-Based NetLogo Model of Young, Heavy Drinkers for Conducting Alcohol Policy Experiments". *J. Artificial Societies and Social Simulation*, 19(1).

[Shaw and Garlan, 1996] Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Shoham, 1989] Shoham, Y. (1989). "Time for Action: On the Relation Between Time, Knowledge and Action". In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, pages 954–959, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Shoham, 1993] Shoham, Y. (1993). "Agent-oriented Programming". *Artificial Intelligence*, 60(1), pages 51–92.

[Sibertin-Blanc, 1994] Sibertin-Blanc, C. (1994). "Cooperative Nets". In Valette, R., editor, *Application and Theory of Petri Nets 1994: 15th International Conference Zaragoza, Spain, June 20–24, 1994 Proceedings*, pages 471–490. Springer-Verlag, Berlin, Heidelberg.

[Sidorova and Isik, 2010] Sidorova, A. and Isik, O. (2010). "Busness process research: a cross-disciplinary review". *Business Process Management Journal*, 16(4), pages 566–597.

[Sidorova and Stahl, 2013] Sidorova, N. and Stahl, C. (2013). "Soundness for Resource-Constrained Workflow Nets Is Decidable". *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(3), pages 724–729.

[Silva et al., 2003] Silva, V.; Garcia, A.; Brandão, A.; Chavez, C.; Lucena, C.; and Alencar, P. (2003). "Taming Agents and Objects in Software Engineering". In Garcia, A.; Lucena, C.; Zambonelli, F.; Omicini, A.; and Castro, J., editors, *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

[Singh et al., 2016] Singh, D.; Padgham, L.; and Logan, B. (2016). "Integrating BDI Agents with Agent-Based Simulation Platforms". *Autonomous Agents and Multi-Agent Systems*, 30(6), pages 1050–1071.

[Steckel et al., 2013] Steckel, T.; Kersting, T.; and Nüßer, W. (2013). "Towards Supporting Mobile Business Processes in Non-deterministic Agricultural Environments by Using Agent-Based Technologies". *KI*, 27(4), pages 359–362.

[Sterling and Taveter, 2009] Sterling, L. and Taveter, K. (2009). *The Art of Agent-Oriented Modeling*. The MIT Press.

[Stiefelmann, 2016] Stiefelmann, A. (2016). "Entwicklung und Untersuchung von Model-Checker-Prototypen für ausgewählte beschränkte Referenznetze als Plugin für den Referenznetzsimulator Renew". Bachelor's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Stormer and Knorr, 2001] Stormer, H. and Knorr, K. (2001). "PDA- and Agent-based Execution of Work of Tasks". In *GI Jahrestagung (2)*, pages 968–972.

[Strembeck and Rinderle-Ma, 2013] Strembeck, M. and Rinderle-Ma, S. (2013). "Security and Privacy in Business Processes: A Posteriori Analysis Techniques". *it - Information Technology*, 55(6), pages 247–254.

[Stroppi et al., 2015] Stroppi, L. J. R.; Chiotti, O.; and Villarreal, P. D. (2015). "Defining the resource perspective in the development of processes-aware information systems". *Information & Software Technology*, 59, pages 86–108.

[Sturm and Shehory, 2014] Sturm, A. and Shehory, O. (2014). "Agent-Oriented Software Engineering: Revisiting the State of the Art". In *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*, pages 13–26. Springer-Verlag.

[Suh et al., 2001] Suh, Y.; Namgoong, H.; Yoo, J.; and Lee, D. (2001). "Design of a Mobile Agent-Based Workflow Management System". In *Mobile Agents for Telecommunication Applications, Third International Workshop, MATA 2001, Montreal, Canada, August 14-16, 2001, Proceedings*, pages 93–102.

[Swenson, 2010] Swenson, K. D. (2010). *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way that Knowledge Workers Get Things Done*. Landmark books. Meghan-Kiffer Press.

[Taylor et al., 2009] Taylor, R. N.; Medvidovic, N.; and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing.

[Tell and Moldt, 2005] Tell, V. and Moldt, D. (2005). "Ein Petrinetzsystem zur Modellierung selbstmodifizierender Petrinetze". In Schmidt, K. and Stahl, C., editors, *Proceedings of the 12th Workshop on Algorithms and Tools for Petri Nets (AWPN 05)*, pages 36–41. Humboldt Universität zu Berlin, Fachbereich Informatik.

[Tenschert and Lenz, 2016] Tenschert, J. and Lenz, R. (2016). "Agora - Speech-Act-Based Adaptive Case Management". In *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.*, pages 61–66.

[ter Hofstede et al., 2010] ter Hofstede, A. H. M.; van der Aalst, W. M. P.; Adams, M.; and Russell, N., editors (2010). *Modern Business Process Automation - YAWL and its Support Environment*. Springer.

[Thomssen, 2017] Thomssen, F. A. (2017). "Unterstützung der Analyse von Prozessen in PAOSE-Lehrprojekten – Konzeption, Bereitstellung und Nutzung spezieller Werkzeuge für Process-Mining-Verfahren". Masterarbeit, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Tiplea et al., 2015] Tiplea, F. L.; Bocaneala, C.; and Chirosca, R. (2015). "On the Complexity of Deciding Soundness of Acyclic Workflow Nets". *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 45(9), pages 1292–1298.

[Tiplea and Leahu, 2016] Tiplea, F. L. and Leahu, I. (2016). "The Reversible Released Form of Petri Nets and Its Applications to Soundness of Workflow Nets". *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(2), pages 303–312.

[Topçu and Yilmaz, 2014] Topçu, O. and Yilmaz, L. (2014). "Agent-supported simulation for coherence-driven workflow discovery and evaluation". In *Proceedings of the 2014 Winter Simulation Conference, Savannah, GA, USA, December 7-10, 2014*, pages 419–428.

[Trappey et al., 2009] Trappey, C. V.; Trappey, A. J. C.; Huang, C.; and Ku, C. C. (2009). "The design of a JADE-based autonomous workflow management system for collaborative SoC design". *Expert Syst. Appl.*, 36(2), pages 2659–2669.

[Tsui and Karam, 2010] Tsui, F. and Karam, O. (2010). *Essentials of Software Engineering*. Jones & Bartlett Learning.

[Valk, 1987] Valk, R. (1987). "Modelling of Task Flow in Systems of Functional Units". Technical Report FBI-HH-B-124/87, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Valk, 1996] Valk, R. (1996). "On Processes of Object Petri Nets". Report of the Department of Informatics FBI-HH-B-185/96, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Valk, 1998] Valk, R. (1998). "Petri Nets as Token Objects - An Introduction to Elementary Object Nets". In Desel, J. and Silva, M., editors, *19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal*, number 1420 in Lecture Notes in Computer Science, pages 1–25, Berlin Heidelberg New York. Springer-Verlag.

[Valk, 2000] Valk, R. (2000). "Relating Different Semantics for Object Petri Nets, Formal Proofs and Examples". Technical Report FBI-HH-B-226, University of Hamburg, Department for Computer Science Report/00.

[Valk, 2004] Valk, R. (2004). "Object Petri Nets – Using the Nets-within-Nets Paradigm". In Desel, J.; Reisig, W.; and Rozenberg, G., editors, *Advances in Petri Nets: Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, Berlin Heidelberg New York.

[van der Aalst, 1996] van der Aalst, W. M. P. (1996). "Structural Characterizations of Sound Workflow Nets". Computing Science Reports 96/23, Eindhoven University of Technology,, Eindhoven.

[van der Aalst, 1997] van der Aalst, W. M. P. (1997). "Verification of Workflow Nets". In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, ICATPN '97, pages 407–426, London, UK, UK. Springer-Verlag.

[van der Aalst, 1998a] van der Aalst, W. M. P. (1998a). "Modeling and Analyzing Interorganizational Workflows". In *1st International Conference on Application of Concurrency to System Design (ACSD '98), 23-26 March 1998, Fukushima, Japan*, pages 262–272. IEEE Computer Society.

[van der Aalst, 1998b] van der Aalst, W. M. P. (1998b). "The Application of Petri Nets to Workflow Management". *Journal of Circuits, Systems, and Computers*, 8(1), pages 21–66.

[van der Aalst, 1999a] van der Aalst, W. M. P. (1999a). "Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets". *Systems Analysis - Modelling - Simulation*, 34(3), pages 335–367.

[van der Aalst, 1999b] van der Aalst, W. M. P. (1999b). "Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow". *Inf. Syst.*, 24(9), pages 639–671.

[van der Aalst, 2010] van der Aalst, W. M. P. (2010). "Business Process Simulation Revisited". In *Enterprise and Organizational Modeling and Simulation - 6th International Workshop, EOMAS 2010, held at CAiSE 2010, Hammamet, Tunisia, June 7-8, 2010. Selected Papers*, pages 1–14.

[van der Aalst, 2013] van der Aalst, W. M. P. (2013). "Business Process Management: A Comprehensive Survey". *ISRN Software Engineering*, 2013, pages 1 – 37.

[van der Aalst, 2015] van der Aalst, W. M. P. (2015). "Business Process Management As the "Killer App" for Petri Nets". *Software & Systems Modeling*, 14(2), pages 685–691.

[van der Aalst et al., 2004a] van der Aalst, W. M. P.; Aldred, L.; Dumas, M.; and ter Hofstede, A. H. M. (2004a). "Design and Implementation of the YAWL System". In Persson, A. and Stirna, J., editors, *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, volume 3084 of *Lecture Notes in Computer Science*, pages 142–159. Springer.

[van der Aalst et al., 2006] van der Aalst, W. M. P.; Dumas, M.; Ouyang, C.; Rozinat, A.; and Verbeek, H. M. W. (2006). "Choreography Conformance Checking: An Approach based on BPEL and Petri Nets". In Leymann, F.; Reisig, W.; Thatte, S. R.; and van der Aalst, W. M. P., editors, *The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[van der Aalst et al., 2005] van der Aalst, W. M. P.; Jørgensen, J. B.; and Lassen, K. B. (2005). "Let's Go All the Way: From Requirements Via Colored Workflow Nets to a BPEL Implementation of a New Bank System". In Meersman, R.; Tari, Z.; Hacid, M.; Mylopoulos, J.; Pernici, B.; Babaoglu, Ö.; Jacobsen, H.; Loyall, J. P.; Kifer, M.; and Spaccapietra, S., editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, volume 3760 of *Lecture Notes in Computer Science*, pages 22–39. Springer.

[van der Aalst et al., 2010] van der Aalst, W. M. P.; Lohmann, N.; Massuthe, P.; Stahl, C.; and Wolf, K. (2010). "Multiparty Contracts: Agreeing and Implementing Interorganizational Processes.". *Computer Journal*, 53(1), pages 90–106.

[van der Aalst et al., 1999] van der Aalst, W. M. P.; Moldt, D.; and Wienberg, F. (1999). "Enacting Interorganizational Workflows using Nets in Nets". In Becker, J.; zur Mühlen, M.; and Rosemann, M., editors, *Proceedings of the 1999 Workflow Management Conference*, volume 70 of *Working Paper Series of the Department of Information systems*, pages 117–136, Muenster, Germany. University of Muenster.

[van der Aalst and ter Hofstede, 2005] van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). "YAWL: yet another workflow language". *Inf. Syst.*, 30(4), pages 245–275.

[van der Aalst et al., 2003] van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.; and Barros, A. P. (2003). "Workflow Patterns". *Distributed and Parallel Databases*, 14(1), pages 5–51.

[van der Aalst and van Dongen, 2013] van der Aalst, W. M. P. and van Dongen, B. F. (2013). "Discovering Petri Nets from Event Logs". In Jensen, K.; van der Aalst, W. M. P.; Balbo, G.; Koutny, M.; and Wolf, K., editors, *Transactions on Petri Nets and Other Models of Concurrency VII*, pages 372–422. Springer Berlin Heidelberg, Berlin, Heidelberg.

[van der Aalst and van Hee, 2002] van der Aalst, W. M. P. and van Hee, K. (2002). *Workflow Management - Models, Methods, and Systems*. The MIT Press.

[van der Aalst et al., 1994] van der Aalst, W. M. P.; van Hee, K. M.; and Houben, G.-J. (1994). "Modelling Workflow Management Systems with High-Level Petri Nets". In de Michelis, G.; Ellis, C.; and Memmi, G., editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50.

[van der Aalst et al., 2011] van der Aalst, W. M. P.; van Hee, K. M.; ter Hofstede, A. H.; Sidorova, N.; Verbeek, E. H.; Voorhoeve, M.; and Wynn, M. T. K. (2011). "Soundness of workflow nets: classification, decidability, and analysis". *Formal Aspects of Computing*, 23(3), pages 333–363.

[van der Aalst et al., 2009] van der Aalst, W. M. P.; van Hee, K. M.; ter Hofstede, A. H. M.; Sidorova, N.; Verbeek, E. H.; Voorhoeve, M.; and Wynn, M. T. K. (2009). "Soundness of Workflow Nets with Reset Arcs". In Jensen, K.; Billington, J.; and Koutny, M., editors, *Transactions on Petri Nets and Other Models of Concurrency III*, pages 50–70. Springer Berlin Heidelberg, Berlin, Heidelberg.

[van der Aalst et al., 2004b] van der Aalst, W. M. P.; Weijters, T.; and Maruster, L. (2004b). "Workflow mining: discovering process models from event logs". *IEEE Transactions on Knowledge and Data Engineering*, 16(9), pages 1128–1142.

[van der Aalst and Weske, 2013] van der Aalst, W. M. P. and Weske, M. (2013). "Reflections on a Decade of Interorganizational Workflow Research". In Bubenko, J.; Krogstie, J.; Pastor, O.; Pernici, B.; Rolland, C.; and Sølvberg, A., editors, *Seminal Contributions to Information Systems Engineering*, pages 307–313. Springer Berlin Heidelberg.

[van Dongen et al., 2016] van Dongen, B. F.; Carmona, J.; and Chatain, T. (2016). "A Unified Approach for Measuring Precision and Generalization Based on Anti-alignments". In [Rosa et al., 2016], pages 39–56.

[van Dongen et al., 2005] van Dongen, B. F.; de Medeiros, A. K. A.; Verbeek, H. M. W.; Weijters, A. J. M. M.; and van der Aalst, W. M. P. (2005). "The ProM Framework: A New Era in Process Mining Tool Support". In Ciardo, G. and Darondeau, P., editors, *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer.

[van Dongen et al., 2012] van Dongen, B. F.; Desel, J.; and van der Aalst, W. M. P. (2012). "Aggregating Causal Runs into Workflow Nets". In Jensen, K.; van der Aalst, W. M.; Marsan, M. A.; Franceschinis, G.; Kleijn, J.; and Kristensen, L. M., editors, *Transactions on Petri Nets and Other Models of Concurrency VI*, pages 334–363. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Vasilecas et al., 2016] Vasilecas, O.; Kalibatiene, D.; and Lavbic, D. (2016). "Rule- and context-based dynamic business process modelling and simulation". *Journal of Systems and Software*, 122, pages 1–15.

[Venero, 2014] Venero, M. L. F. (2014). "Verifying Cross-Organizational Workflows Over Multi-Agent Based Environments". In *Enterprise and Organizational Modeling and Simulation - 10th International Workshop, EOMAS 2014, Held at CAiSE 2014, Thessaloniki, Greece, June 16-17, 2014, Selected Papers*, pages 38–58.

[Verbeek et al., 1999] Verbeek, E. H.; Basten, T.; and van der Aalst, W. M. P. (1999). "Diagnosing Workflow Processes Using Woflan". *The Computer Journal*, 44, pages 2001.

[Verbeek and van der Aalst, 2000] Verbeek, H. M. W. E. and van der Aalst, W. M. P. (2000). "Woflan 2.0: A Petri-Net-Based Workflow Diagnosis Tool". In *ICATPN*, pages 475–484.

[Visser et al., 2016] Visser, S.; Thangarajah, J.; Harland, J.; and Dignum, F. (2016). "Preference-based reasoning in BDI agent systems". *Autonomous Agents and Multi-Agent Systems*, 30(2), pages 291–330.

[Völzer, 2010] Völzer, H. (2010). "An Overview of BPMN 2.0 and Its Potential Use". In [Mendling et al., 2011], pages 14–15.

[vom Brocke and Rosemann, 2010a] vom Brocke, J. and Rosemann, M. (2010a). *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*. Springer Publishing Company, Incorporated, 1st edition.

[vom Brocke and Rosemann, 2010b] vom Brocke, J. and Rosemann, M. (2010b). *Handbook on Business Process Management 2: Strategic Alignment, Governance, People and Culture*. Springer Publishing Company, Incorporated, 1st edition.

[von Rosing et al., 2015] von Rosing, M.; White, S.; Cummins, F.; and de Man, H. (2015). "Business Process Model and Notation - BPMN". In von Rosing, M.; von Scheel, H.; and Scheer, A., editors, *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I*, pages 429–453. Morgan Kaufmann/Elsevier.

[Wagner, 2009a] Wagner, T. (2009a). "A Centralized Petri Net- and Agent-based Workflow Management System". In [Duvigneau and Moldt, 2009], pages 29–44.

[Wagner, 2009b] Wagner, T. (2009b). "Modeling of a Centralized Petri Net- and Agent-based Workflow Management System". Bachelor's thesis (equiv.), University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Wagner, 2009c] Wagner, T. (2009c). "Prototypische Realisierung einer Integration von Agenten und Workflows". Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Wagner, 2010] Wagner, T. (2010). "Prototypische Realisierung einer Integration von Agenten und Workflows". In *Informatiktage 2010 - Fachwissenschaftlicher Informatik-Kongress 19. und 20. März 2010, B-IT Bonn-Aachen International Center for Information Technology in Bonn*, volume S-9 of *LNI*, pages 101–104. GI.

[Wagner, 2012] Wagner, T. (2012). "Agentworkflows for Flexible Workflow Execution". In [Cabac et al., 2012], pages 199–214.

[Wagner and Cabac, 2013] Wagner, T. and Cabac, L. (2013). "Advantages of a Full Integration between Agents and Workflows". In Moldt, D., editor, *Modeling and Business Environments MODBE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 353–354. CEUR-WS.org.

[Wagner and Moldt, 2011] Wagner, T. and Moldt, D. (2011). "Approaching the Integration of Agents and Workflows". In Bergenthum, R. and Desel, J., editors, *Algorithmen und Werkzeuge für Petrinetze. 18. Workshop AWPN 2011, Hagen, September 2011. Tagungsband*, pages 55–61.

[Wagner and Moldt, 2015a] Wagner, T. and Moldt, D. (2015a). "Integrating Agent Actions and Workflow Operations". In Müller, J. P.; Ketter, W.; Kaminka, G.; Wagner, G.; and Bulling, N., editors, *Multiagent System Technologies - 13th German Conference, MATES 2015, Cottbus, Germany, September 28-30, 2015, Revised Selected Papers*, volume 9433 of *Lecture Notes in Computer Science*, pages 61–78. Springer.

[Wagner and Moldt, 2015b] Wagner, T. and Moldt, D. (2015b). "Workflow Management Principles for Interactions Between Petri Net-Based Agents". In Devillers, R. and Valmari, A., editors, *Application and Theory of Petri Nets and Concurrency - 36th International Conference, Petri Nets 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, volume 9115 of *Lecture Notes in Computer Science*, pages 329–349. Springer-Verlag.

[Wagner and Moldt, 2016] Wagner, T. and Moldt, D. (2016). "Revisiting Workflow Interactions for Petri Net-Based Agents (Extended Abstract)". In Mendling, J. and Rinderle-Ma, S., editors, *Proceedings of the 7th International Workshop on Enterprise Modeling and Information Systems Architectures, EMISA 2016: Fachgruppentreffen der GI-Fachgruppe Entwicklungsmethoden für Informationssysteme und deren Anwendung, Vienna, Austria, October 3-4, 2016.*, volume 1701 of *CEUR Workshop Proceedings*, pages 9–12. CEUR-WS.org.

[Wagner et al., 2016a] Wagner, T.; Moldt, D.; and Köhler-Bußmeier, M. (2016a). "From eHornets to Hybrid Agent and Workflow Systems". In Cabac, L.; Kristensen, L. M.; and Rölke, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'16, Toruń, Poland,*

June 20-21, 2016. Proceedings, volume 1591 of *CEUR Workshop Proceedings*, pages 307–326. CEUR-WS.org.

[Wagner et al., 2012] Wagner, T.; Quenum, J.; Moldt, D.; and Reese, C. (2012). "Providing an Agent Flavored Integration for Workflow Management". In Jensen, K.; Donatelli, S.; and Kleijn, J., editors, *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900 of *Lecture Notes in Computer Science*, pages 243–264. Springer-Verlag, Berlin Heidelberg New York.

[Wagner et al., 2016b] Wagner, T.; Schmitz, D.; and Moldt, D. (2016b). "Paffin: Implementing an Integration of Agents and Workflows". In *Proceedings of the 14th European Conference on Multi-Agent Systems (2016)*, volume 14, page tba. Springer. Postproceedings to be published in 2017.

[Wang et al., 2010] Wang, C.; Huang, P. S.; and Wang, F. (2010). "A Hierarchical Timed Coloured Petri Nets for BPMN-based Process Analysis". In *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, July 1 - July 3, 2010*, pages 417–420. Knowledge Systems Institute Graduate School.

[Wang et al., 2005a] Wang, J.-W.; Li, C.-C.; and Wang, F.-J. (2005a). "Dynamic activities on an agent-based workflow management system". In *The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.*, pages 122–.

[Wang et al., 2005b] Wang, M.; Wang, H.; and Xu, D. (2005b). "The Design of Intelligent Workflow Monitoring with Agent Technology". *Know.-Based Syst.*, 18(6), pages 257–266.

[Wang et al., 2005c] Wang, S.; Shen, W.; and Hao, Q. (2005c). "Agent based workflow ontology for dynamic business process composition". In *Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, CSCWD 2005, Volume 1, May 24-26, 2005, Coventry, UK*, pages 452–457.

[Waters et al., 2015] Waters, M.; Padgham, L.; and Sardiña, S. (2015). "Improving domain-independent intention selection in BDI systems". *Autonomous Agents and Multi-Agent Systems*, 29(4), pages 683–717.

[Weber et al., 2013] Weber, B.; Rinderle, S.; and Reichert, M. (2013). "Change Patterns and Change Support Features in Process-Aware Information Systems". In *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*, pages 381–395. Springer-Verlag.

[Wei and Blake, 2013] Wei, Y. and Blake, M. B. (2013). "Adaptive Service Workflow Configuration and Agent-Based Virtual Resource Management in the Cloud". In *2013 IEEE International Conference on Cloud Engineering, IC2E 2013, San Francisco, CA, USA, March 25-27, 2013*, pages 279–284.

[Weijters and van der Aalst, 2001] Weijters, A. and van der Aalst, W. M. P. (2001). "Process Mining Discovering Workflow Models from Event-Based Data". In *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 283–290.

[Werner-Stark et al., 2014] Werner-Stark, A.; Dulai, T.; and Abraham, G. (2014). "Modeling of an agent system to support the management of cooperating and rival resources for business workflows". In *4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications, SIMULTECH 2014, Vienna, Austria, August 28-30, 2014*, pages 407–412.

[Weske, 2012] Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures*. Springer Publishing Company, Incorporated, 2nd edition.

[Wester-Ebbinghaus, 2010] Wester-Ebbinghaus, M. (2010). *Von Multiagentensystemen zu Multiorganisationssystemen – Modellierung auf Basis von Petrinetzen*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg. Available at `http://www.sub.uni-hamburg.de/opus/volltexte/2011/4974/` (last accessed May 28th, 2017).

[Wester-Ebbinghaus et al., 2009] Wester-Ebbinghaus, M.; Köhler-Bußmeier, M.; and Moldt, D. (2009). "From Multi-Agent to Multi-Organization Systems: Utilizing Middleware Approaches". In Artikis, A.; Picard, G.; and Vercouter, L., editors, *Engineering Societies in the Agents World IX*, volume 5485 of *Lecture Notes in Computer Science*, pages 46–65.

[Wester-Ebbinghaus et al., 2010] Wester-Ebbinghaus, M.; Moldt, D.; and Köhler-Bußmeier, M. (2010). "Modeling Organizational Units as Modular Components of Systems of Systems". *Trans. Petri Nets and Other Models of Concurrency*, 4, pages 174–198.

[Wfmc12, 2012] Wfmc12 (2012). *Workflow Management Coalition Workflow Standard Process Definition Interface – XML Process Definition Language*. Workflow Management Coalition.

[Wfmc99, 1999] Wfmc99 (1999). *Terminology & Glossary*. Workflow Management Coalition, 2 Crown Walk, Winchester, Hampshire SO23 8BB, United Kingdom.

[Whitestein15, 2015] Whitestein15 (2015). *Goal-Oriented Business Process Management*. Whitestein Technologies AG.

[WIAT2015, 2015] WIAT2015 (2015). *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015, Singapore, December 6-9, 2015 - Volume II*. IEEE Computer Society.

[Wilensky, 1999] Wilensky, U. (1999). "NetLogo". Technical report, Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. Available via `http://ccl.northwestern.edu/netlogo/` (last accessed May 28th, 2017).

[Wilensky and Rand, 2015] Wilensky, U. and Rand, W. (2015). *An Introduction to Agent-Based Modelling - Modelling Natural, Social, and Engineered Complex Systems with Netlogo*. The MIT Press, Cambridge, Massachusetts - London, England.

[Wincierz, 2016] Wincierz, M. (2016). "Erweiterung des PAOSE Softwareentwicklungsansatzes um ein Testkonzept und Bereitstellung von Plugins zu dessen technischer Umsetzung". Bachelor's thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Wincierz, 2017] Wincierz, M. (2017). "A Tool Chain for Test-driven Development of Reference Net Software Components in the Context of CAPA Agents". In Moldt, D.; Cabac, L.; and Rölke, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'17, Zaragoza, Spain, June 25-26, 2017. Proceedings*, volume 1846 of *CEUR Workshop Proceedings*, pages 201–220. CEUR-WS.org.

[Winikoff, 2005] Winikoff, M. (2005). "JACK^{TM} Intelligent Agents: An Industrial Strength Platform". In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 175–193. Springer-Verlag.

[Winikoff, 2016] Winikoff, M. (2016). "How Testable Are BDI Agents? An Analysis of Branch Coverage". In *Autonomous Agents and Multiagent Systems - AAMAS 2016 Workshops, - Best Papers - , Singapore, Singapore, May 9-10, 2016, Revised Selected Papers*, pages 90–106.

[Wolf, 2016] Wolf, K. (2016). "Running LoLA 2.0 in a Model Checking Competition". *T. Petri Nets and Other Models of Concurrency*, 11, pages 274–285.

[Wooldridge et al., 2000] Wooldridge, M.; Jennings, N. R.; and Kinny, D. (2000). "The Gaia Methodology for Agent-Oriented Analysis and Design". *Autonomous Agents and Multi-Agent Systems*, 3(3), pages 285–312.

[Wooldridge, 2009] Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. Wiley, 2nd edition edition.

[Wynn et al., 2009] Wynn, M. T.; Verbeek, H. M. W.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.; and Edmond, D. (2009). "Soundness-preserving Reduction Rules for Reset Workflow Nets". *Inf. Sci.*, 179(6), pages 769–790.

[Xu et al., 2003] Xu, Q.; Qiu, R. G.; and Xu, F. (2003). "Agent-based workflow approach to the design and development of cross-enterprise information systems". In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: Washington, D.C., USA, 5-8 October 2003*, pages 2633–2638.

[Yamaguchi and Ahmadon, 2016] Yamaguchi, S. and Ahmadon, M. A. B. (2016). "Properties and Decision Procedure for Bridge-Less Workflow Nets". *IEICE Transactions*, 99-A(2), pages 509–512.

[Yan et al., 2015] Yan, J.; Hu, D.; Liao, S. S. Y.; and Wang, H. (2015). "Mining Agents' Goals in Agent-Oriented Business Processes". *ACM Trans. Management Inf. Syst.*, 5(4), pages 20:1–20:22.

[Yan et al., 2010] Yan, Z.; Reijers, H. A.; and Dijkman, R. M. (2010). "An Evaluation of BPMN Modeling Tools". In [Mendling et al., 2011], pages 121–128.

[Yao and Logan, 2016] Yao, Y. and Logan, B. (2016). "Action-Level Intention Selection for BDI Agents". In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 1227–1236.

[Zeng et al., 2001] Zeng, L.; Ngu, A. H. H.; Benatallah, B.; and O'Dell, M. (2001). "An agent-based approach for supporting cross-enterprise workflows". In *Twelfth Australasian Database Conference, ADC2001, Bond University, Queensland, Australia, January 29 - February 1, 2001*, pages 123–130.

[Zhao et al., 2007] Zhao, Z.; Belloum, A.; de Laat, C.; Adriaans, P. W.; and Hertzberger, B. (2007). "Distributed execution of aggregated multi domain workflows using an agent framework". In *2007 IEEE International Conference on Services Computing - Workshops (SCW 2007), 9-13 July 2007, Salt Lake City, Utah, USA*, pages 183–190.

[Zhao et al., 2016] Zhao, Z.; Paschke, A.; and Zhang, R. (2016). "A rule-based agent-oriented approach for supporting weakly-structured scientific workflows". *J. Web Sem.*, 37-38, pages 36–52.

[Zhuge et al., 2002] Zhuge, H.; Chen, J.; Feng, Y.; and Shi, X. (2002). "A Federation-agent-workflow Simulation Framework for Virtual Organisation". *Inf. Manage.*, 39(4), pages 325–336.

[Zytniewski, 2016] Zytniewski, M. (2016). "Integration of knowledge management systems and business processes using multi-agent systems". *IJCIStudies*, 5(2), pages 180–196.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, 29.06.2017

Thomas Wagner