Universität Hamburg

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Department Informatik
Arbeitsbereich Theoretische Grundlagen der Informatik

Dissertation

Prozess-Infrastruktur für Agentenanwendungen

vorgelegt von
Christine Reese
aus Hamburg

Juli 2009

Gutachter der Dissertation:	Dr. Daniel Moldt Prof. Dr. Rüdiger Valk Prof. Dr. Norbert Ritter
Vorsitzender des Promotionsausschusses:	Prof. Dr. Christopher Habel
Datum der Disputation:	9. Dezember 2009
Vorsitzender der Prüfungsausschusses:	Prof. Dr. Bernd Page

Danksagung

An dieser Stelle möchte ich all jenen danken, die zur Entstehung dieser Arbeit beigetragen haben. Die vorliegende Arbeit wurde in den ersten zwei Jahren von der Universität Hamburg im Rahmen eines Promotionsstipendiums gefördert. Bis zur Bewilligung haben schon viele Kräfte zusammengewirkt, so dass diese Arbeit ihren Anfang nehmen konnte und damit eine ganze Lebensphase.

Sehr herzlich möchte ich Dr. Daniel Moldt, Professor Dr. Rüdiger Valk und Professor Dr. Norbert Ritter für die Betreuung und Begutachtung dieser Arbeit danken.

Ich schätze das freie, offene und freundliche Klima im Arbeitsbereich TGI und danke den Kollegen, Mit-Promovenden und der ganzen TGI-Gruppe. Daniel hat eine fantastische Begeisterungsfähigkeit, durch die in den engagierten Diskussionen die Visionen und Ideen Gestalt angenommen haben. Ich möchte speziell Lawrence Cabac, Michael Duvigneau, Kolja Markwardt, Jan Ortmann und Matthias Wester-Ebbinghaus danken. Immer wieder gab es fruchtbare Diskussionen und Ideenaustausch, besonders zu den gemeinsamen Veröffentlichungen und alljährlich zu den AOSE-Lehreprojekten.

Viele Studenten haben durch ihre Teilnahme im AOSE-Projekt, ihre Baccalaureats- und Diplomarbeiten zu dieser Arbeit beigetragen. Namentlich möchte ich Timo Carl, Wojciech Laka und Thomas Wagner erwähnen, deren Ausarbeitungen vor, während und nach dem Hauptteil meiner Arbeit entstanden und ganz unmittelbar mit der vorliegenden Arbeit verknüpft sind.

Und noch jemanden möchte ich namentlich erwähnen und vielen Dank sagen: Reinhard Zierke ist hilfsbereit und großzügig und als Systemadministrator eine wertvolle Unterstützung für mich gewesen.

Zusammenfassung

Verteilte und heterogene Systeme sind von besonderem Interesse in der Informatik. Die daraus resultierende hohe Komplexität von Anwendungen macht die explizite Darstellung bestimmter Sichten notwendig, insbesondere einer Prozesssicht und einer Sicht auf gekapselte autonome Teilsysteme. Diese Arbeit stellt eine Infrastruktur vor, die explizite Koordination von Prozessen in agentenbasierten Anwendungen bietet.

Die Prozesssicht ist eine wichtige Sicht auf ein komplexes System und prozessorientierter Entwurf ist eine Möglichkeit, ein großes System erfolgreich zu entwerfen. Das heißt, die Prozesse brauchen eine explizite Repräsentation, wodurch Beschreibbarkeit und Änderbarkeit der Anwendung verbessert werden. Besonders wichtig ist dabei der Entwurf von systemübergreifenden Prozessen, also über verschiedene Teilsysteme verteilte Prozesse des übergreifenden Systems. Die Ausführung der Prozesse muss flexibel steuerbar und beobachtbar sein. Petrinetze und Workflow-Technologie bieten auf jeweils eigene Weise Unterstützung für diese Anforderungen.

Die Zusammenarbeit zwischen unabhängigen, autonomen Einheiten wie Organisationen braucht weitere Unterstützung auf konzeptioneller und technischer Ebene. Agententechnologie liefert wichtige Lösungsmöglichkeiten und es existieren Architekturvorschläge für offene Agentennetze. Verschiedene Europäische Projekte entwickeln offene Infrastruktur für Agenten. Eine agentenbasierte Anwendung nutzt diese Infrastruktur mittels Verzeichnisdiensten und indem Agenten untereinander kommunizieren. Solch eine Anwendung kann selbst offen sein und einen übergeordneten Dienst anbieten, z.B. eine Umgebung, in der andere Entwickler beteiligt sind.

Innerhalb der Agententechnik stellt die Gestaltung von Abläufen eine besondere Herausforderung dar, da die zugrundeliegenden Prozesse nebenläufig und verteilt sind. Übergeordnete Prozesse und die Koordination von Prozess-Teilen sind in einer solchen Infrastruktur für Agenten meist weder explizit repräsentiert noch sonstwie unterstützt. Das Ziel dieser Arbeit ist die explizite Unterstützung komplexer Prozesse von Anwendungen im Kontext der offenen Agentenumgebungen, also sowohl die explizite Repräsentation der Prozesse als auch die steuerbare und beobachtbare Ausführung, ohne jedoch dabei die Autonomie der Komponenten als Basiseigenschaft der Agententechnologie zu verlieren.

Diese Arbeit trägt zur Lösung dieser Probleme und Anforderungen bei, indem eine verteilte Infrastruktur-Schicht zur Verwaltung von Prozessen eingeführt wird, also eine *Prozessinfrastruktur*. Die Ergebnisse umfassen Konzepte und einen lauffähigen Prototyp unter Verwendung von Workflow-Technologie, wodurch die Koordination in verteilten agentenbasierten Anwendungen unterstützt wird. Zur Visualisierung abstrakter Konzepte werden Petrinetze verwendet, ebenso zur Spezifikation von Agentenverhalten und für die Präzedenzrelation in Workflows. Die verwendeten Petrinetze können direkt ausgeführt werden, die Laufzeitumgebung ist vollständig mit Referenznetzen implementiert.

Ein wichtiges Anwendungsgebiet einer solchen Prozessinfrastruktur sind unternehmensübergreifende Anwendungen. Als Ausblick liefert die vorliegende Arbeit die konzeptionelle und technische Basis für ein Rahmenwerk für interorganisationales Geschäftsprozessmanagement.

Abstract

Distributed and heterogenous systems are of special interest within computer sciences. Due to the resulting complexity of systems, an explicit representation of certain views becomes necessary, e.g. a process view and a view on encapsulated autonomous entities. This work introduces an infrastructure for explicit process coordination for agent-based applications.

The process view is one important view on a complex system, and process oriented design is a way to successfully design a system. This means that processes need an explicit representation to improve description and changeability of an application. One special point that has to be handled is the design of systems spanning processes, i.e. processes that are distributed over separate parts of the overall system. Monitoring and controlling are needed for the execution of processes. Petrinets and workflow technology provide each in their own way support for such requirements.

Cooperation between independent, autonomous entities (e.g. organisations) needs further support in terms of concepts and technology. Agent technology provides important solutions and architectural proposals for open agent networks. Within different European projects, infrastructures for agents were created that form open environments accessible via Internet. An agent-based application can use this infrastructure through the directory services provided and by communicating with the other agents. Such an application may again be open, providing a comprehensive service or an application platform where agents of other developers and owners may participate.

Whithin agent technology, a special challange is the design of processes because the processes are concurrent and distributed. The overall process and the coordination of process parts of an application is usually not represented or supported in an infrastructure for agents. The objective of this work is to provide explicit support for complex processes within the context of open agent environments, meaning both explicit representations of processes as well as controlling and monitoring for active processes, without loosing the autonomy of the system components.

This work tackles these problems and requirements by defining a distributed infrastructure layer for the management of processes, a *process infrastructure*. Results include concepts and a running prototype which uses workflow technology and supports coordination in distributed agent applications. Petri nets are used to visualize abstract concepts as well as for the specification of agent behavior and the precedence relation of workflows. And even more the utilized Petri nets can directly be executed. The runtime environment is completely realized using Reference Nets.

One important application area of the process infrastructure includes applications that run across several companies. As a future prospect, this work provides basics for an inter-organizational business process management framework consisting of conceptual and technical solutions.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Definitionen und Begriffsbestimmungen

Kapitel 1

Einleitung

Bei verteilten und heterogenen Systemen müssen ab einer gewissen Größe sowohl die Struktur als auch das Verhalten (die internen Prozesse) explizit dargestellt werden, um die wachsende Komplexität handhabbar zu machen. Komplexe Anwendungen bedürfen einer mächtigen Strukturierung zur Entwurfszeit und einer Infrastruktur zur expliziten Unterstützung ihrer Prozesse zur Laufzeit.

Abstrakt lässt sich der Sachverhalt mit Petrinetzen erfassen: Ein Netz mit seinen Stellen, Transitionen und Kanten bildet die (formal erfassbare) Struktur eines Systems. Jedes Netzelement bildet eine klar abgegrenzte lokale Einheit, da es ausschließlich von den direkt verbundenen Netzelementen abhängig ist. So lassen sich auch verteilte Systeme mit unabhängigen Systemkomponenten mit ihren Wechselwirkungen exakt erfassen. Die Schaltvorgänge im Petrinetz werden als Petrinetzprozesse dargestellt, in denen die Unabhängigkeit sichtbar bleibt. Das Verhalten eines modellierten verteilten Systems kann also durch die Menge aller möglichen Prozesse eines Petrinetzes beschrieben werden.

Verlässt man die formale Ebene und schaut auf die konkrete Ebene der Umsetzungen, so liefert die Agententechnologie wichtige Beiträge zu der Thematik der komplexen Systeme und ihrer Strukturierung. Eine Anwendung wird in unabhängige und gekapselte Einheiten untergliedert, so dass mit dem Begriffsgerüst der Agententechnologie vorhandenes Wissen und konkret vorhandene Technologie aus den Bereichen der Verteilten Systeme und der Softwaretechnik, aber auch aus den Bereichen der Künstlichen Intelligenz, Spieltheorie und Sozialtheorie integriert werden kann, wenn es für die aktuell betrachtete Anwendung von Vorteil ist.

Die Workflowtechnologie liefert dagegen wichtige und ausgereifte Beiträge zur Prozessmodellierung und -ausführung. Die Konzepte zur Prozessunterstützung sind sehr mächtig, sie ermöglichen Strukturierung, Entwurf, Intanziierung, Ausführung, Beobachtung, und Änderung von Prozessdefinitionen. Mit Hilfe von Workflows werden ursprünglich Geschäftsprozesse unterstützt, also der Arbeits- und Dokumentenfluss zwischen Mitarbeitern in einem Unternehmen koordiniert. In dieser Arbeit werden mit dem Prozess-Begriff insbesondere die Vorgänge innerhalb von komplexen Anwendungen bezeichnet. Beim Entwurf und bei der Ausführung von verteilten und heterogenen Systemen ist eine weitreichende Prozessunterstützung besonders wichtig. Die Workflowtechnologie eignet sich in ihren allgemeinen Konzepten auch für diesen Anwendungsbereich: die Prozessunterstützung für verteilte und heterogene Systeme.

Sowohl die Agenten- als auch Workflowtechnologie bieten eine *Infrastruktur* (Laufzeitumgebung im weitesten Sinne) für die Ausführung von Anwendungen an und zielen so auf die Unterstützung komplexer Anwendungen. Zugespitzt lassen sich beide Gebiete wie folgt gegenüberstellen: Eine Agenten-Infrastruktur instanziiert strukturelle Komponenten in Form von Agenten und führt sie auf Plattformen aus; eine Prozess-

Infrastruktur instanziiert anwendungsinterne Prozesse in Form von Workflows und führt sie in einer Workflowengine aus. Eine Ausprägung für eine Agenten-Infrastruktur sind die offenen Agentennetze, in denen zwischen Agenten über offene Verzeichnisdienste vermittelt wird. Die Komponenten einer Anwendung sollen sich hier dynamisch zusammenschließen lassen. Eine Ausprägung für Prozess-Infrastruktur sind die Workflowmanagementsysteme.

Ab einer gewissen Größe und Komplexität von Anwendungen bieten jedoch für sich genommen weder die Agententechnologie, noch die Workflowtechnologie und erst recht nicht die Petrinetze ausreichende Unterstützung für den Entwurf und die Ausführung verteilter und heterogener Anwendungen. Bei den Petrinetzen fehlt die vereinheitlichende Strukturierung: Selbst bei hierarchischen Modellen oder wenn eine objektorientierte Programmiersprache hinzugezogen wird, ermöglichen erst zusätzliche strukturbildende Konzepte wie die der Agententechnologie größere Systementwürfe. Bei der Agententechnologie mangelt es an der Prozessunterstützung, insbesondere zur Laufzeit: Die Agenteninteraktionen werden zwar mit Hilfe von Diensten strukturiert und mit Interaktionsprotokollen genormt, dies steht jedoch insbesondere zur Laufzeit weit hinter den Möglichkeiten der Workflowtechnologie zurück. Die agenteninternen Prozesse werden im übergreifenden Begriffsgerüst der Agententechnologie kaum adressiert. Man kann zwar die grundlegende Ausführungsform der Agenten als reaktive oder deliberative Agenten verwenden, um mögliche agenteninterne Prozesse daraus abzuleiten (z.B. wurden diese Ausführungsformen von Rölke [Rölke 2004] als Petrinetze modelliert, so dass die Prozesse daraus formal ableitbar sind), allerdings ist diese Betrachtungsweise höchstens ein Ansatzpunkt, um die anwendungsspezifischen internen Prozesse explizit gestalten zu können und bei der Ausführung von ähnlichen Möglichkeiten wie bei der Workflowtechnologie zu profitieren.

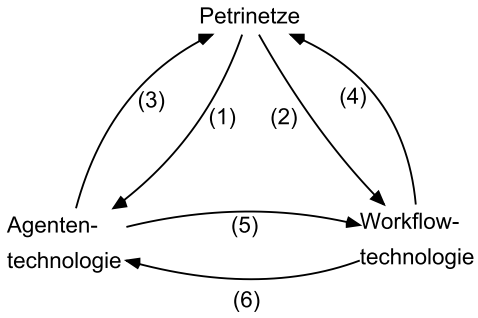


Abbildung 1.1: Wechselwirkungen der Technologien

Abbildung 1.1 zeigt ein Dreieck aus den Technologien: Petrinetze, Agenten und Workflows. (1,2) Sowohl Agententechnologie als auch Workflowtechnologie werden mit Petrinetzen modelliert, bereichert und formalisiert. (3) Für die Programmierung mit Petrinetzen bietet die Agententechnologie wichtige Strukturierungsmöglichkeiten. (4) Die Workflowtechnologie liefert z.B. Konzepte für Netzklassen zur Bereicherung der Petrinetze. (5) Die Workflowtechnologie wird in vielen aktuellen Projekten mit Konzepten der Agententechnologie flexibilisiert und für verteilte Szenarien aufbe-

reitet. (6) Die Bereicherung der Agententechnologie durch die Workflowtechnologie ist noch wenig verbreitet und hängt eng mit der in dieser Arbeit gewählten Betrachtungsweise zusammen, bei der mit Workflows die internen Prozesse einer komplexen Anwendung bezeichnet werden. Diese Arbeit setzt an dieser Stelle an und zielt auf den Aufbau einer Prozess-Infrastruktur für Agentenanwendungen. Petrinetze dienen als „Denkzeug“ (nach [Moldt 2005]), als Modellierungswerkzeug und als Programmiersprache zur Spezifikation von Verhalten. Als technische Ausgangspunkte werden für diese Arbeit das Referenznetzwerkzeug Renew von Kummer et al. ([Kummer et al. 2008, Kummer 2002]), das petrinetzbasierte Agenten-Rahmenwerk Capa von Duvigneau ([Duvigneau et al. 2003]) und das petrinetzbasierte Workflowmanagementsystem von Jacob ([Jacob et al. 2002]) herangezogen. Die Umsetzung des Prototyps folgt dem Software-Entwicklungsansatz Paose (petrinetzbasierte agentenorientierte Software-Entwicklung, [Cabac et al. 2007a]). Die prozessorientierte Vorgehensweise aus Paose wird bei der Ideenentwicklung eine wichtige Rolle spielen.

Ziel dieser Arbeit ist eine Infrastruktur zur Verwaltung von Prozessen in Agentenanwendungen. Die Infrastruktur erhält den Namen Pia (Prozess-Infrastruktur für Agentenanwendungen) und wird in Form einer Architektur entworfen (Apia). Die grundlegende Idee ist dabei, die Workflowtechnologie so zu integrieren, dass die Unterstützung von Prozessen zur Entwurfs- und Laufzeit übernommen werden kann, dabei aber die Agenten als vorrangiges Strukturierungs- und Abstraktionsmerkmal erhalten bleiben.

Um das Ziel der Arbeit zu erreichen, werden die notwendigen Grundlagen aus den Gebieten der Petrinetze, Agenten und Workflows sowie die verwendeten Prozessmodellierungstechniken aus der UML in Kapitel 2 vorgestellt, ebenso einige verwandte Arbeiten aus dem Bereich der verteilten und flexiblen Workflowmanagementsysteme.

In Kapitel 3 werden die grundlegenden Konzepte für Pia erarbeitet und als vorbereitende konzeptionelle Zusammenfassung vorgestellt. Das Kapitel beginnt bei der Festlegung der zentralen Begriffe für diese Arbeit und beleuchtet dann den Bezug der Konzepte zu der anvisierten Technologie. Es folgen die Vorarbeiten zur Fragmentierung von Workflow-Petrinetzen und die Einführung in den Entwurfsansatz Paose. Schließlich wird in einem Überleitungsabschnitt die Vision der Arbeit aufgegriffen und auf der Grundlage der eingeführten Konzepte und Begriffe als Ziel ausbuchstabiert.

Der zentrale Vorschlag für die Konzeption der Prozess-Infrastruktur für Agentenanwendungen wird in in Kapitel 4 systematisch ausgearbeitet und vorgestellt. Nach einer Motivation der Ideen zur Beziehung der Agenten- und Workflow-Technologie werden fünf Integrationsstufen vorgestellt. Die Bandbreite führt von einfachsten Systemen mit zwei Architekturschichten bis hin zu der Vision einer abstrakten, Agenten- und Workflow-Technologie vereinheitlichenden Prozess-Infrastruktur mit vielen Architekturschichten.

Kapitel 5 diskutiert Realisierungen der eingeführten Konzepte und die notwendigen Arbeiten zur Verbesserung der technischen Umgebungen, insb. Capa. Neben den Grundzügen des Prototyps für Pia werden daher das Rahmenwerk Capa und ein konkretes Workflowmanagementsystem präsentiert. Die Vorstellung eines konkreten Workflow-Agenten rundet die technische Seite der Arbeit ab.

Kapitel 6 liefert als Abschluss eine umfassende Diskussion der Arbeitsergebnisse und ihrer wechselseitigen Verknüpfungen und Möglichkeiten zur Anknüpfung weiterer Arbeiten, abgerundet durch kurz angerissene Anwendungsbeispiele.

Kapitel 7 liefert eine Zusammenfassung in Kurzform und schließt mit einem kurzen Ausblick, der weitere mögliche Anschlüsse für die Arbeit aufzeigt.

Kapitel 2

Prozesse, Agenten und Workflows

Die Petrinetze, die Agententechnologie und die Workflowtechnologie bilden in dieser Arbeit die wesentlichen Bestandteile für Apia, einer Architektur für eine Prozess-Infrastruktur für Agentenanwendungen. Die Petrinetze sind das Mittel der Wahl zur Spezifikation von Verhalten und Prozessen, zusammen mit den Sequenzdiagrammen aus UML. Die Workflows liefern die Konzepte und Technologien zur expliziten Verwaltung von Prozessen und Zuordnung der Ressourcen zur Ausführung von Prozessschritten. Die Agenten sind einerseits die Zieltechnologie und liefern andererseits wichtige Konzepte und Technologien für gekapselte autonome Einheiten.

Dieses Kapitel liefert die Grundlagen und einen Einblick in die Begriffswelten der jeweiligen Techonologien, soweit es für die weitere Arbeit notwendig ist. Das Kapitel schließt mit einem Abschnitt, in dem verwandte Technologien vorgestellt werden.

2.1 Prozessmodellierung und Petrinetze

Komplexe Anwendungen können entlang ihrer internen Prozesse gegliedert sein, sowohl beim Entwurf als auch bei der Implementation. Dieser Abschnitt stellt einige geläufige Möglichkeiten zur Darstellung von Prozessen und Prozessdefinitionen vor.

2.1.1 Überblick

In der Systemspezifikation werden zustands- und prozessorientierte Modelle verwendet, z.B. Endliche Automaten, Statecharts, Petrinetze, Use Case-Diagramme, UML-Aktivitätsdiagramme, Ereignisgesteuerte Prozessketten (EPK) oder Workflownetze. Alle genannten Techniken werden von Laue und Liedke ausführlich in ihren Aspekten Pragmatik, Syntax und Semantik in [Laue und Liedtke 2000] vorgestellt. Neben den graphischen Modellierungstechniken gibt es auch rein formale Techniken wie die Prozesskalküle (z.B. [Milner et al. 1992] für Prozesse, die auch Mobilität involvieren).

Aus der UML bieten sich zur Beschreibung von Prozessen hauptsächlich Sequenzdiagramme an, aber auch Activity-Diagramme und in eingeschränkter Hinsicht Communication-Diagramme (mit ihren nummerierten Nachrichten) und Timing-Diagramme können Prozesse darstellen. Im Agentenumfeld werden die Sequenzdiagramme vielfach verwendet und heißen dort Interaktionsdiagramme. Sequenzdiagramme können zunächst nur einen festen Ablauf darstellen, es gibt jedoch Notationen für alternative Abläufe und Schleifen. Erst in der Form der Interaktions-*Protokoll*diagramme aus der AUML (UML für Agenten) werden zusätzliche Notationen für Verzweigungen bereitgestellt. Eine Darstellung der Syntax für die in dieser Arbeit verwendeten Interaktionsdiagramme und Interaktionsprotokolldiagramme folgt in Abschnitt 2.1.2.

Obwohl mit UML 2 vermehrt formale Aspekte in die Gestaltung der Notation eingeflossen sind, bleibt die Prozessmodellierung mit UML informell und eher bezogen auf die Modellierung als auf die Implementation. Eine wesentlich implementationsnähere und sehr verbreitete Prozessbeschreibung sind Workflows, welche mit Kapitel 2.3 einen eigenen ausführlichen Abschnitt erhalten.

Petrinetze werden in dieser Arbeit sowohl zur Modellierung als auch zur Programmierung eingesetzt. Durch ihre exakte formale und operationale Semantik sind Petrinetze für Modelle mit starker Prozessorientierung besonders gut geeignet, insbesondere können Konzepte der Nebenläufigkeit, Unabhängigkeit, Präzedenz und Konfliktsituationen präzise dargestellt werden.

Der Begriff *Prozess* ist im Bereich der Petrinetze formal definiert. Mit Petrinetz-Prozessen kann das Verhalten eines Petrinetzes in Form von kausal geordneten Mengen beschrieben werden. Ein Prozess in diesem Sinne wird durch ein Petrinetz aus der Klasse der Kausalnetze dargestellt, worin es keine Verzweigung an Stellen gibt (also keine Konflikte). Die Einführung und die formale Definition folgt im Abschnitt 2.1.3.

Um Petrinetze zur Programmierung umfangreicher Systeme verwenden zu können, braucht man mächtigere Darstellungsformen. Es gibt Klassen von Petrinetzen mit verschiedenen Eigenschaften, z.B. können im Gegensatz zu Stellen-/Transitionsnetzen bei gefärbten Netzen auch Typen („Farben“) wie Integer verwendet werden. Referenznetze sind eine Form von gefärbten Netzen und haben als Petrinetze höherer Ordnung verglichen mit sonstigen gefärbten Petrinetzen einige wichtige Erweiterungen. Zunächst ermöglichen sie die Modellierung auf Basis des Konzepts Netze-in-Netzen nach Valk [Valk 1998], wobei ein äußeres Systemnetz als Token Instanzen von Netzen (den Objektnetzen) erlaubt. Zur Kommunikation zwischen Netzen erlauben synchrone Kanäle die Verschmelzung von Transitionen für die Dauer eines Schaltvorgangs (bidirektionaler und synchroner Austausch). Außerdem können in Referenznetzen beliebige Java-Anschriften verwendet werden, so dass von petrinetzbasierter Programmierung gesprochen werden kann. Abschnitt 2.1.4 führt die Prinzipien der Referenznetze ein.

Zur Programmierung steht der Multiformalismen-fähige Editor und Simulator Renew als integrierte Entwicklungsumgebung mit vielfältiger Programmierunterstützung zur Verfügung, z.B. fassen vordefinierte Netzkomponenten wiederkehrende Teilnetze zu Komponenten zusammen, welche über Paletten beim manuellen und automatischen Erzeugen von Referenznetzen zur Verfügung stehen. Renew bietet auch Plugins für UML-Interaktionsprotokolldiagramme an. Einige wichtige Aspekte für die Programmierung mit Petrinetzen werden im Abschnitt 2.1.5 vorgestellt.

Petrinetze können für ganz verschiedene Repräsentationsformen von Prozessen als Definition einer Semantik verwendet werden, z.B. auch für die UML-Interaktionsdiagramme und Protokolldiagramme. Immer wenn genau ein möglicher Prozess erfasst werden soll, kann dieser mit einem Kausalnetz repräsentiert oder modelliert werden. Wenn mehrere alternative Prozesse erfasst werden sollen, können diese mit einem Petrinetz repräsentiert oder modelliert werden. Auch für komplexe Systeme wie z.B. mehrere Referenznetze, kann ein spezieller Prozess in Form eines Kausalnetzes angegeben werden, allerdings muss die Methode dazu erst von den S / T-Netzen auf die Referenznetze übertragen werden. Im letzten Abschnitt zum Thema Petrinetze, im

Abschnitt 2.1.6, wird diese Übertragung vorgenommen, sowie eine Petrinetz-Semantik für Interaktionsdiagramme und Protokolldiagramme angegeben.

2.1.2 UML-Sequenzdiagramme

Dieser Abschnitt stellt aus der UML die Sequenzdiagramme vor, die in der Agentenorientierung in Form von *Interaktionsdiagrammen* und *Interaktionsprotokolldiagrammen* (AUML) verwendet werden. Eine vollständige Einführung in UML 2 gibt z.B. das Buch von Bernd Oesterreich [Oesterreich 2005].

Definition 1 (Sequenz) *Eine Sequenz zeigt eine Reihe von Nachrichten, die eine ausgewählte Menge von Beteiligten (Objekten und Akteuren) in einer zeitlich begrenzten Situation austauscht, wobei der zeitliche Ablauf betont wird.* [Oesterreich 2005, S. 331] ◇

In einem Sequenzdiagramm werden Objekte und Akteure durch gestrichelte senkrechte Linien (den sog. Lebenslinien) mit einem Bezeichner am oberen Ende dargestellt. Die Zeit verläuft von oben nach unten. Die Nachrichten werden mit einer Beschriftung der Form „Nachricht(Argumente)“ als waagerechte Pfeile dargestellt. Die Pfeilspitze zeigt die Art der Nachricht an: gefüllte Pfeilspitze für synchrone Nachrichten, offene Pfeilspitze für asynchrone Nachrichten. Antwortnachrichten können optional dargestellt werden mit einer gestrichelten Linie und einer offenen Pfeilspitze. Eine „Aktivierung“ in Form einer Verbreiterung der Lebenslinie durch einen rechteckigen Balken kann optional angeben, welches Objekt die Programmkontrolle besitzt oder aktiv ist.

Objektkonstruktion wird durch einen Nachrichtenpfeil dargestellt, der auf einen Objekt- oder Akteursbezeichner trifft. Objektdestruktion wird durch ein Kreuz am Ende der Lebenslinie dargestellt.

Eine weitere Möglichkeit ist die Verschachtelung von Sequenzdiagrammen durch Referenzierung. Dabei wird anstelle des referenzierten Sequenzdiagramms ein Kasten über die Lebenslinien der daran beteiligten Objekte oder Akteure gelegt mit einer Referenz darin.

Die Darstellung alternativer Abläufe ist zunächst nicht der Zweck von Sequenzdiagrammen, für Schleifen und „kleine“ alternative Abläufe, wo dem größeren Körper des Diagramms eine oder zwei Nachrichten als Alternativen gegenüber stehen, kann man entsprechende Elemente verwenden. Ein Kasten um einige Nachrichten herum mit dem Bezeichner „alt“ steht für Alternativen, daneben gibt es Schlüsselwörter wie „loop“ und „opt“ für Schleifen bzw. optionale Elemente.

Der Abschnitt im Buch [Oesterreich 2005, S. 331–335] endet mit dem Hinweis, dass die Bedeutung von Sequenzdiagrammen eher nachrangig bleibe und die Aktivitätsdiagramme zur vollständigen Spezifikation von Abläufen besser geeignet seien.

Als weniger formale Einführung von Sequenzdiagrammen mit größerer Betonung der Benutzung ist [Ambler 2004, Kap. UML Sequence Diagrams, S. 321–334] von Scott Ambler wesentlich besser geeignet. Ambler schreibt, dass Sequenzdiagramme typischerweise für Nutzungsszenarios, Methodenlogik und Dienstlogik verwendet werden und sich mit einem entsprechenden Detaillierungsgrad auch zur visuellen Programmierung eignen. Im Rahmen der vorliegenden Arbeit werden Sequenzdiagramme

tatsächlich zur visuellen, modellbasierten Programmierung verwendet, allerdings werden hier Elemente aus AUML (Agent UML) verwendet, welche die Spezifikation von mehreren alternativen Abläufen wesentlich übersichtlicher gestalten. AUML wird von Odell et al. eingeführt in [Odell et al. 2000]. In dieser Form der visuellen Programmierung geht es nicht mehr darum, einen beispielhaften Ablauf darzustellen, sondern alle möglichen Abläufe zu spezifizieren, das Sequenzdiagramm wird spezialisiert zu einem *Protokolldiagramm*.

Die neuen Elemente aus der AUML erlauben es, sowohl die Lebenslinien als auch die Nachrichtenpfeile aufzuspalten und zusammenzufassen, um so Alternativen im Kommunikationsablauf darzustellen.

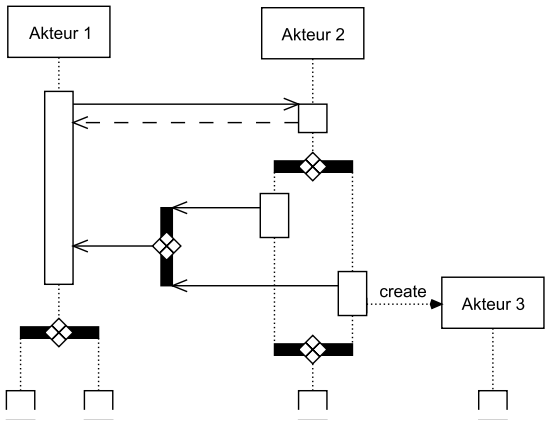


Abbildung 2.1: Fragment eines Protokolldiagramms nach AUML

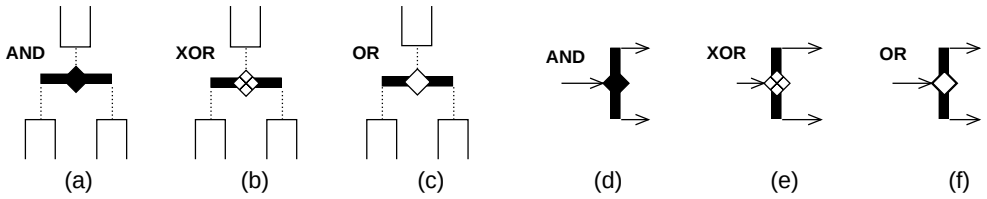


Abbildung 2.2: Protokolldiagramm-Elemente aus AUML

In Abbildung 2.1 ist ein AUML-Interaktionsdiagramm mit den in dieser Arbeit typischen Elementen dargestellt. Die schwarzen Balken mit dem X stehen für eine exklusiv-Oder-Situation. Daneben gibt es noch die Varianten für Und und einfaches Oder, sie sind in Abbildung 2.2 in je einer horizontalen und vertikalen Variante dargestellt. Die jeweils umgekehrte Richtung ist ebenso möglich (Lebenslinien bzw. Nachrichten zusammenfassen).

2.1.3 Petrinetze

Mit Petrinetzen lassen sich zentrale Begriffe der Verteilten Systeme formal und gleichzeitig intuitiv verständlich fassen. Dadurch sind Petrinetze besonders gut für die Modellierung von verteilten Systemen und Prozessen in verteilten Systemen geeignet.

Die folgenden Prinzipien der Petrinetze orientieren sich an der einführenden Darstellung in [Girault und Valk 2003, S. 9ff].

Das Prinzip der Dualität In Petrinetzen werden aktive Elemente (wie Ereignisse, Aktionen, Ausführung von Anweisungen, Übermitteln von Nachrichten) als Transitionen und passive Elemente (Bedingungen, Zustände, Betriebsmittel, Kanäle) als Stellen modelliert.

Für die Modellierung von verteilten Systemen ist es sehr hilfreich, sowohl passive als auch aktive Elemente explizit darzustellen.

Das Prinzip der Lokalität Das Verhalten einer Transition wird ausschließlich durch ihre Lokalität bestimmt. Zur Lokalität einer Transition gehören ihre Eingangs- und Ausgangsstellen sowie die Transition selbst.

Für die Modellierung von verteilten Systemen ist es sehr hilfreich, den Bereich der Vorbedingungen und Auswirkungen von aktiven Elementen explizit zu beschreiben.

Das Prinzip der Nebenläufigkeit Transitionen mit disjunkter Lokalität finden unabhängig (nebenläufig) statt.

Für die Modellierung von verteilten Systemen ist es sehr hilfreich, auch die Unabhängigkeit von aktiven Elementen explizit prüfen zu können.

Zustand: Markierung Der Zustand eines Petrinetzes kann durch eine *Markierung* beschrieben werden. Eine Markierung beschreibt den Zustand für jede Stelle.

In einem verteilten System kann man grundsätzlich nicht auf den aktuellen Zustand einer entfernten Einheit zugreifen (das ist eine wesentliche Eigenschaft verteilter Systeme, siehe Begriffsbestimmung 2 auf Seite 61). Mit der Markierung eines Petrinetzes besteht die Möglichkeit, aus der globalen Perspektive den Zustand eines verteilten Systems zu beschreiben.

Das Prinzip der graphischen Darstellung Es sind nur wenige graphische Elemente notwendig, um für alle Eigenschaften von Petrinetzen eine (je nach Gestaltung) intuitiv verständliche Notation der vorgestellten Prinzipien zu erreichen. Die Elemente eines Petinetzes sind die folgenden:

Die *Stellen* eines Petrinetzes werden als Kreise dargestellt und die *Transitionen* als Rechtecke; Stellen und Transitionen sind durch *Kanten* (Pfeile) verbunden. Stellen können *Marken* enthalten: allgemein durch einen Punkt notiert, im hier verwendeten Werkzeug Renew durch eckige Klamern „[]“ notiert. Die Bestückung des Netzes mit Marken ist seine *Markierung*. Zusätzlich kann den Kanten noch eine Gewichtung zugeordnet sein, die angibt, wieviele Marken jeweils zu bewegen sind. Wenn außerdem noch eine Anfangsmarkierung definiert ist, handelt es sich um ein sog. S/T-Netz (Stellen-

Transitionen-Netz), einer geläufigen einfachen Art von Petrinetzen. Die Ausführung eines Petrinetzes besteht in der Änderung der Markierung gemäß der *Schaltregel*; die Stellen, Transitionen und Kanten zwischen ihnen bleiben unverändert.

Für die Modellierung von verteilten Systemen ist eine einfache und dabei ausdrucksstarke Modellierungssprache sehr hilfreich.

Das Prinzip der formaltextuellen Darstellung Zu jeder graphischen Darstellung eines Petrinetzes gibt es eine formaltextuelle Darstellung und umgekehrt.

Für die Simulation und Verifikation von verteilten Systemen ist eine äquivalente formaltextuelle Darstellung zur graphischen Darstellung sehr hilfreich. Diese Arbeit befasst sich hauptsächlich mit Konzepten und hat nicht zum Ziel, eine konkrete Anwendung zu verifizieren oder einen Simulator zu entwerfen. Aus diesem Grunde wird die formaltextuelle Darstellung von Petrinetzen nur andeutungsweise eingeführt.

Definition 2 (Petrinetz) *Ein* Netz *ist ein Tripel* $N = (S, T, F)$*, wobei*

- S *eine Menge von* Stellen*,*
- T *eine disjunkte Menge von* Transitionen *und*
- F *die* Flussrelation *für die Kanten darstellt mit* $F \subseteq (S \times T) \cup (T \times S)$.

*(Nach: [*Girault *und* Valk *2003, S. 14])* ◇

Für ein S/T-Netz werden dem Tupel noch eine Wichtungsfunktion $W : F \to \mathbb{N} \setminus \{0\}$ und eine Anfangsmarkierung $M_0 : S \to \mathbb{N}$ hinzugefügt.

Verhalten: Schaltregel Eine Transition in einem Petrinetz ist *aktiviert*, wenn jede Eingangsstelle mindestens eine Marke enthält. Eine aktivierte Transition kann *schalten* oder *feuern*, indem von jeder Eingangsstelle eine Marke entfernt wird und in jede Ausgangsstelle eine Marke plaziert wird. Die Transition im Bild 2.3 kann feuern, indem sie die Marken von **a** und **P** entfernt und eine Marke in **Q** legt. Danach ist sie nicht mehr aktiviert.

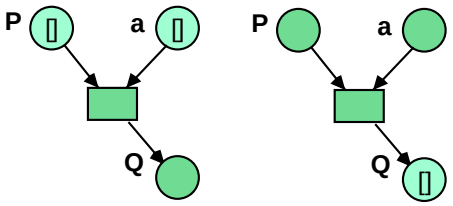


Abbildung 2.3: Transition vor und nach dem Schalten

Prozess In einem Netz entstehen *Schaltfolgen* durch mehrfaches Schalten von Transitionen. In einem parallelen oder verteilten System hat aber eine globale Auflistung aller Vorgänge in einer Schaltfolge nur begrenzten Wert, weil die Information über die (Un-) Abhängigkeit der Schaltvorgänge dabei verloren geht. Deshalb werden Prozesse in Netzen mit Hilfe von Kausalnetzen beschrieben. Kausalnetze beschreiben

nebenläufige Schaltvorgänge als unabhängige Ereignisse und definieren so eine partielle Ordnung auf den Schaltvorgängen (im Gegensatz zur totalen Ordnung einer Schaltfolge).

Definition 3 (Kausalnetz) *Ein Netz $N = (S, T, F)$ heißt* Kausalnetz, *wenn*

- *N zyklenfrei ist und*
- *alle Stellen höchstens eine Eingangs- und Ausgangstransition haben.*

*Nach: [*Jessen *und* Valk *1987, S. 26]* ◇

Definition 4 (Petrinetzprozess) *Ein asynchroner Prozess eines S/T-Netzes N ist durch ein markiertes Kausalnetz N_K und einer Abbildung von den Elementen aus N_K auf die Elemente von N beschrieben.*

*Die formale Definition findet sich z.B. in [*Jessen *und* Valk *1987, S. 45]* ◇

Konfliktsituationen In einem Petrinetz können mehrere mögliche Prozesse beschrieben werden, indem die Alternativen mittels einer Stellenverzweigung modelliert werden. Konfliktsituationen können zu Verklemmungen führen. Konflikte können entweder durch Anschriften wie z.B. guards eindeutig aufgelöst werden oder unterspezifiziertes oder nichtdeterministisches Verhalten darstellen.

Präzedenz und Unabhängigkeit Mit einem Petrinetz kann für Anwendungsprozesse die Präzedenz der Einzelschritte definiert werden, also die direkte Nachfolge (Abhängigkeit) von Netzelementen. Das betrifft formal nur die direkt benachbarten Netzelemente, also ist jede Transition nur von ihren Eingangsstellen abhängig und die Markierung jeder Stelle nur von den Transitionen in ihrer Umgebung abhängig.

Die indirekte Abhängigkeit von Netzelementen über mehrere Schritte hinweg wird graphisch erkennbar durch eine lineare Verkettung von Netzelementen. Formal lässt sich diese indirekte Abhängigkeit durch die Relation *li (in Linie)* für jedes gegebene (endliche) Kausalnetz berechnen.

Nicht direkt verbundene Netzelemente sind grundsätzlich unabhängig voneinander (siehe „Prinzip der Nebenläufigkeit“). Graphisch ist dies nur durch das Fehlen einer verbindenden Kante „dargestellt“. Für die mittelbare Unabhängigkeit, also solche Netzelemente, die auch über mehrere Schritte unabhängig voneinander sind, gibt es die Relation *co (concurrent, nebenläufig)*, welche für jedes gegebene (endliche) Kausalnetz berechnet werden kann.

Definition 5 (*li* und *co*) *Sei $N = (S,\ T,\ F)$ ein Kausalnetz. Dann ist*
$< := F^+$ die transitive Hülle von F und
li *die symmetrische und reflexive Hülle von $<$:*

$$li := \ \widehat{<} = < \ \cup <^{-1} \cup \mathrm{id}_{|\ (S \cup T)}$$

und

$$co := \ \overline{li} \cup \ \mathrm{id}_{|\ (S \cup T)}$$

*(Nach: [*Jessen *und* Valk *1987, S. 27])* ◇

Mit den Relationen *li* („on a line“) und *co* („concurrent“) besteht die Möglichkeit, die Abhängigkeit oder Unabhängigkeit von Netzelementen explizit zu behandeln.

Arten von Petrinetzen S/T-Netze wurden bereits als Petrinetze mit einer Wichtungsfunktion und einer Anfangsmarkierung eingeführt. Eine erweiterte Art von Petrinetzen sind *gefärbte* Petrinetze, in denen Marken verschiedene Eigenschaften haben. Die Farbe einer Marke beschreibt den *Typ*, den der *Wert* einer Marke annehmen kann. Statt Punkte in einer Stelle zu haben, können also Zahlen darin liegen oder Strings oder Werte von komplexeren Typen. In einem gefärbten Petrinetz kommen vielfältige Beschriftungsmöglichkeiten zum Aufbau des Netzes hinzu. Stellen und Kanten können getypt werden, an Transitionen können in Form von Guards zusätzliche Bedingungen für die Aktivierung notiert werden. Gefärbte Petrinetze werden vielfach für Modellierung, Simulation, Spezifikation und Implementierung verwendet. Als Standardwerk sei hier auf [Jensen 1992] verwiesen.

Der nächste Schritt wird durch die *Referenznetze* gebildet. Im Vergleich zu den S/T-Netzen kommen neue Arten von Beschriftungen und Marken hinzu, sowie die Instanziierung von Netzen und eine starke Verbindung zu Java. Referenznetze werden im nächsten Abschnitt vorgestellt. Eine moderne Einführung in die gesamte Thematik der Verwendung von Petrinetzen zur Modellierung, Verfikation und Anwendungsentwicklung ist in [Girault und Valk 2003] zu finden.

2.1.4 Referenznetze

Referenznetze wurden 2002 von Olaf Kummer eingeführt (siehe [Kummer 2002]). Referenznetze sind eng mit dem Editor und Simulationswerkzeug Renew verbunden, beides wird im Handbuch zu Renew ([Duvigneau et al. 2002]) ausführlich vorgestellt[1].

Das Prinzip der Instanziierung Bei objektorientierten Programmiersprachen verhalten sich Objekte zu Klassen so, wie bei Referenznetzen die Netzinstanz zum (Referenz-) Netz. Netzinstanzen können dynamisch zur Laufzeit erzeugt und vernichtet werden. Das Netz definiert die Netzstruktur mit Stellen, Transitionen, Kanten, Anschriften und einer Anfangsmarkierung. Bei der Instanziierung können Parameter zur Initialisierung der Netzinstanz übergeben werden. Jede erzeugte Netzinstanz hat einen eigenen Zustand und schaltet unabhängig von (nebenläufig zu) anderen Netzinstanzen. Wie bei einfachen Petrinetzen auch, schalten die Transistionen innerhalb einer Netzinstanz unabhängig voneinander, wenn ihre Lokalitäten disjunkt sind (siehe Prinzip der Nebenläufigkeit).

Das Prinzip der Referenzsemantik Bezeichner in Referenznetzen sind immer Referenzen auf Objekte oder Netzinstanzen. Es können also Alias-Bezeichner von verschiedenen Orten auf dasselbe Objekt oder dieselbe Netzinstanz verweisen. Beim Schalten einer Transition werden Referenzen in Form von Marken auf die Ausgangsstellen gelegt und können von anderen Transitionen (welche die betreffende Stelle als Eingangsstelle haben) weiterverwendet werden.

[1] Die jeweils aktuelle Version von Renew sowie Dokumentation und Artikel sind im Internet verfügbar – [Renew 2008]

Bei der Verwendung von Referenznetzen zur Modellierung von Systemen können auch andere Mechanismen wie Wertsemantik simuliert werden, indem z.B. immer genau eine Referenz auf ein Objekt oder Netz existiert.

Das Prinzip der synchronen Kommunikation Transitionen in Referenznetzen können über Rendezvous-Synchronisation während eines Schaltvorgangs kommunizieren. Die betroffenen Transitionen verschmelzen für einen Schaltvorgang zu einer Transition, deren Lokalität durch die Vereinigung der Lokalitäten der betroffenen Transitionen gebildet wird. Es können somit prinzipiell alle Daten des Schaltvorgangs an den beteiligten Transitionen verwendet werden.

Konkret wird die synchrone Kommunikation mit *synchronen Kanälen* realisiert. Eine Hälfte eines synchronen Kanals (der *Downlink*) besteht aus einer Anschrift an einer Transition mit einer Referenz auf eine Netzinstanz (z.B. `this`), einem Kanalnamen und einer Parameterliste. Die andere Hälfte eines synchronen Kanals (der *Uplink*) besteht aus einer Transitionsanschrift mit demselben Kanalnamen und einer dem Typ und der Anzahl nach passenden Parameterliste. Der Schaltvorgang einer Transition mit Downlink ist realisiert, indem in der referenzierten Netzinstanz nach einem passenden Uplink an einer aktivierten Transition gesucht wird. Sobald die Suche erfolgreich ist, schalten beide Transitionen synchron unter bidirektionalem Datentransfer.

Bei der Modellierung von verteilten Systemen kann eine asynchrone Kommunikation angemessen sein. Alle asynchronen Kommunikationsformen (wie Nachrichtensynchronisation) können durch Pufferstellen und entsprechend verwendete Kanalanschriften simuliert werden.

Das Konzept der hierarchischen Schachtelung Für große Systeme ist die Hierarchisierung, also die Kombination eines großen Systems aus mehreren überschaubaren kleineren Systemen, ein wesentliches Mittel der Strukturierung. Es gibt verschiedene Formen der Hierarchisierung von Petrinetzen. Eine Form basiert auf der Ersetzung von Tranisitionen durch Subnetze und auf der Verschmelzung von Stellen (vgl. [Jensen 1992, S. 90–99]).

Eine zweite Form der Hierarchisierung basiert darauf, Netze als Marken in anderen Netzen zu erlauben und mittels synchroner Kommunikation Wechselwirkungen zwischen dem äußeren Netz (Systemnetz) und dem Marken-Netz (Objektnetz) zu erlauben (siehe [Valk 1998]). Diese zweite Form begründet bei den Referenznetzen das Netze-in-Netzen-Paradigma, welches in dieser Arbeit von grundlegender Bedeutung ist. Das Netze-in-Netzen-Paradigma bildet zusammen mit dynamischer Instanzierung, Referenzsemantik und synchroner Kommunikation sehr mächtige und flexible Modellierungsmöglichkeiten.

Mit Netzen in Netzen können Subsysteme in übergeordneten Systemen modelliert werden, z.B. aktive Einheiten in einem Ortssystem, Dokumente in einem Arbeitsablauf, Dinge in Behältern oder Plugins in einer dynamisch konfigurierbaren Anwendung.

Für die Umsetzung werden Netze dynamisch von einer schaltenden Transition instanziiert und die Netzreferenzen werden als Marken verwendet. Die Netzinstanzen kommunizieren anhand von synchronen Kanälen. Um Nebeneffekte und Fernwirkungen grundsätzlich zu vermeiden, ist vorzugsweise immer nur exakt eine Referenz auf

ein untergeordnetes Netz zu halten und so äquivalentes Verhalten zur Wertsemantik zu erreichen.

Zur Programmierung mit Petrinetzen ist die Werkzeugunterstützung wesentlich, worauf im Folgenden eingegangen wird.

2.1.5 Programmieren mit Petrinetzen

Ursprünglich dienen Petrinetze zur Modellierung von Sachverhalten und mit dem Aufkommen von Petrinetzwerkzeugen auch der Simulation. Ein Petrinetz ist dann ein Modell. Mit steigender Rechenleistung und der hier aufgeführten Werkzeugunterstützung können Petrinetze auch zur Programmierung verwendet werden. Der Simulator eines Petrinetzwerkzeuges wird dann die virtuelle Maschine zur *Ausführung von Programmen*. Die Petrinetze sind also ursprünglich Modelle für einen bestimmten Modellierungszweck. Bei der Programmierung mit Petrinetzen bilden diese die Implementation und zur Ausführungszeit entsteht durch die Netze ein System, welches mit der realen Umwelt interagiert.

Für Referenznetze gibt es das Werkzeug Renew. Angefangen beim Simulator-Kern und beim Editor-Kern bietet Renew eine Vielzahl von Plugins. Grundsätzlich wird ein mit dem Editor spezifiziertes Netz vor der Instanziierung compiliert und in einem grafikfreien Format als *Schattennetz* repräsentiert. Ein Schattennetz wird ähnlich einer Java-Klasse nach Bedarf geladen und instanziiert.

Der Simulator-Kern wird durch eine Vielzahl von Formalismen verfeinert, die jeweils ihren eigenen Compiler bereitstellen. Weitere Simulations-Funktionen sind Breakpoints, speicherbarer Simulationszustand und Remote Simulation.

Der Editor-Kern wird durch Werkzeuge für Grafiken ohne operationale Semantik (z.B. für Kommentare und Gliederung in einem Petrinetz, aber auch für UML-Diagramme) und durch Paletten für spezifische Formalismen (z.B. spezielle Kanten) ergänzt. Weitere Merkmale sind Exportfunktionen für verschiedene Grafik- und Petrineztformate (eps, pdf, svg bzw. Woflan, Macao) und Editor-Funktionen zur Layout-Unterstützung.

Diese Eigenschaften begründen die Praxistauglichkeit von Renew als Entwicklungsumgebung. Einige weitere wichtige Punkte dazu sind die folgenden:

Java-Anbindung Referenznetze verfügen über eine enge Anbindung an Java, das bedeutet, dass beliebige Java-Objekte als Marken verwendet werden können, es können beliebige Java-Methoden beim Schalten von Transitionen aufgerufen werden und Netzinstanzen und Java-Objekte können über synchrone Kanäle Informationen austauschen (dazu muss die Netzinstanz in einer sog. stub-Klasse beschrieben werden).

Nebenläufige Ausführung Referenznetze werden tatsächlich nebenläufig ausgeführt. Das Schalten einer Transition ist ein atomarer Vorgang, der ganz oder gar nicht stattfindet, aber auch eine zeitliche Ausdehnung hat. Während in Renew eine Transition schaltet, kann eine andere aktivierte Transition mit ihrem Schaltvorgang starten (auch dieselbe Transition, falls sie mehrfach aktiviert ist). Nur die Bindungssuche wird synchron durchgeführt, d.h. während der Bindungssuche starten keine Transitionen

ihren Schaltvorgang. Die Bindungssuche ist aber unabhängig von gerade schaltenden Transitionen. Die Bindungssuche resultiert in einer Menge von aktivierten Transitionen, woraus eine Transition für den nächsten Schritt ausgewählt wird. Im Single-Step-Modus wird die nächste Bindungssuche erst durch den Programmierer angestoßen, ansonsten kann die Bindungssuche wieder starten, sobald der Schaltvorgang einer Transition in einem eigenen Thread gestartet wurde. Diese Punkte sind deshalb wichtig, weil spezielle Phänomene nebenläufiger Systeme, die durch verschränkte Ausführung von Vorgängen entstehen, auch in einer Simulation mit Renew auftreten. Insofern ist die Abbildung verteilter Systeme angemessen.

Netzkomponenten und Codegeneratoren Zur Entwicklung umfangreicher Anwendungen und wiederkehrender Funktionen gibt es Unterstützung in Form von Netzkomponenten und anderen Codegeneratoren in Renew. Die Netzkomponenten ([Cabac et al. 2003]) sind vordefinierte Netze, welche so entworfen wurden, dass sie besonders einfach aneinandergefügt werden können. Netzkomponenten werden über eine Werkzeugleiste in Renew in das aktuelle Netz eingefügt.

Codegeneratoren können z.B. aufgrund eines UML-Diagramms mehrere Netzkomponenten zusammenfügen. Je nach Anwendungsgebiet und Gestaltung der Netzkomponenten sind dann noch manuelle Anpassungen notwendig oder das generierte Netz ist dann lauffähig.

2.1.6 Petrinetze als Semantik für Modellierungstechniken

In der Systemspezifikation werden verschiedene Modellierungstechniken angewandt. Petrinetze vereinen Struktur und Verhalten, Grafik und formale Präzision sowie Zustands- und prozessorientierte Sicht auf Systeme. Aus diesem Grund sind die Petrinetze besonders gut geeignet, eine Semantik für verschiedene Modellierungstechniken anzugeben. Moldt hat dies für die Modellierungstechniken der Strukturierten Analyse und Vorläufer der UML-Techniken ausgearbeitet ([Moldt 1996]). Für die Modellierung von Geschäftsprozessen sind vor allem die Ereignisprozessketten (EPK, [Keller et al. 1992, Scheer und Thomas 2005]) sehr verbreitet. Als Beispiel für die Verwendung von Petrinetzen für ereignisgesteuerte Prozessketten sei hier die Arbeit von Rodenhagen angegeben ([Rodenhagen 1997]).

BPEL (*Business Process Execution Language*, [OASIS 2009]) ist eine verbreitete Sprache zur Verknüpfung von WebServices zu Workflows („Orchestrierung“). Schon kurz nach der Standardisierung im Jahr 2003 durch OASIS wurde die Übertragung von BPEL in Petrinetze von Karsten Wolf (geb. Schmidt), Christian Stahl und Sebastian Hinz bearbeitet: [Schmidt und Stahl 2004], [Hinz et al. 2005]. Später wird diese Übertragung vervollständigt ([Lohmann 2008]) und mit einer alternativen Petrinetz-Semantik für BPEL verglichen ([Lohmann et al. 2009]).

Die in dieser Arbeit relevante Beziehung zwischen Workflows und Petrinetzen wird später im Abschnitt 2.3.4 anhand der Workflow-Petrinetze nach van der Aalst gesondert aufgegriffen. Für die vorliegende Arbeit sind jedoch noch zwei spezielle Aspekte der Petrinetzsemantik vorbereitend notwendig: eine Petrinetzsemantik für Interaktionsdiagramme und Protokolldiagramme und zweitens die Erzeugung eines Kausal-

netzes auf der Basis von mehreren Referenznetzen. Beide Punkte werden im Folgenden dargestellt.

Petrinetzsemantik für einfache Sequenzdiagramme Zu den UML-Interaktionsdiagrammen und zu den AUML-Interaktionsprotokolldiagrammen lässt sich eine Petrinetzsemantik angeben. Zu einem gegebenen Diagramm wird für jede Rolle eine Stelle zum Petrinetz hinzugefügt. Jede Nachricht wird in eine Netzkomponente wie in Abbildung 2.4 übersetzt und an die den Rollen entsprechenden Stellen mit Kanten verbunden. Auf diese Weise entstehen vertikal die „Lebenslinien“ für die Rollen

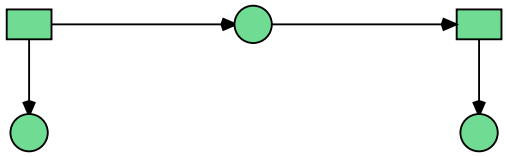


Abbildung 2.4: Netzkomponente für eine Nachricht in einem Interaktionsdiagramm

und aus einem Interaktionsdiagramm wird ein Kausalnetz erzeugt. Bei den AUML-Protokolldiagrammen werden für die zusätzlichen Konstrukte weitere Netzkomponenten benötigt. Für die Split-Konstrukte zeigt Abbildung 2.5 passende Netzkomponenten. Die Join-Konstrukte sind entsprechend zu entwerfen. Beim OR-Join müsste im

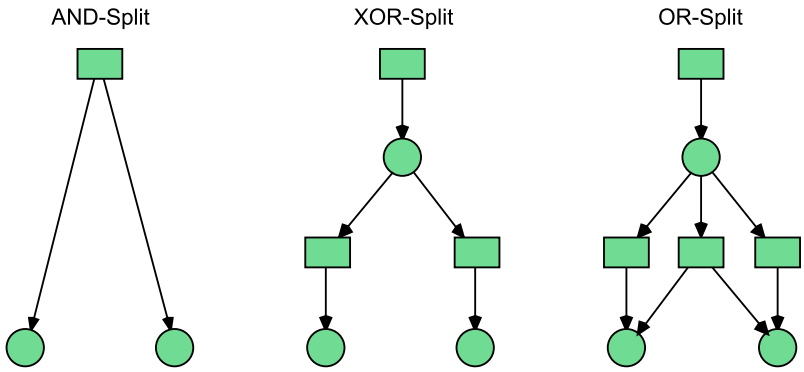


Abbildung 2.5: AUML-Elemente als Netzkomponenten

Petrinetz sichergestellt sein, dass die Join-Transition nur einmal schaltet, auch wenn beide Zweige aktiviert sind. In der für diese Arbeit relevanten Praxis werden aber OR-Split und OR-Join nicht notwendig.

Für Nachrichten sind die Konstrukte entsprechend. Im Rahmen dieser Arbeit wird für Nachrichten nur die XOR-Join-Variante benutzt. Die Anzahl der eingehenden Nachrichtenpfeile spezifiziert so die Anzahl der eingehenden Nachrichten für die betrachtete Interaktion. Abbildung 2.6 stellt die entsprechende Netzkomponente vor. Die Semantik für *exklusiv-oder* ergibt sich allerdings nur, wenn dem Konstrukt ordnungsgemäß ein passender Split *exklusiv-oder* in der Lebenslinie vorausgeht.

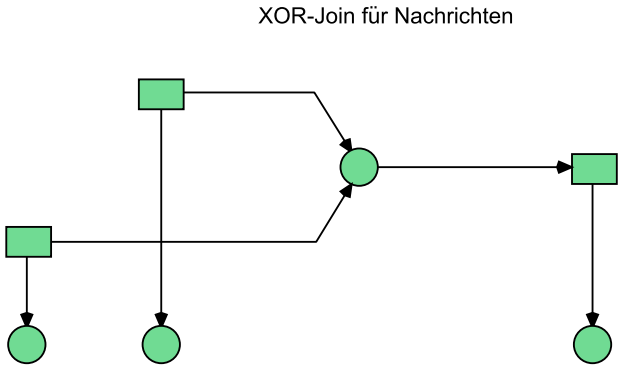


Abbildung 2.6: XOR-Join für Nachrichten

Zusammenfassend wird hier festgehalten, dass für UML-Interaktionsdiagramme ein Kausalnetz angegeben werden kann und dass für AUML-Protokolldiagramme ein Petrinetz angegeben werden kann.

Erzeugung eines Kausalnetzes aus Referenznetzen Die bekannte Abwicklung eines Petrinetzes zu einem Kausalnetz (beschrieben z.B. in [Jessen und Valk 1987, S. 43f]), wird für diese Arbeit auf die Erzeugung eines Kausalnetzes aus Referenznetzen übertragen. Für die vorliegende Arbeit ist es ausreichend, diese Übertragung in Form einer intuitiv verständlichen Beschreibung vorzunehmen.

Bei der Darstellung von Prozessen in Referenznetzen müssen die synchronen Kanäle besonders beachtet werden. Da später im Abschnitt 3.5 auf Seite 96 ein solches Kausalnetz erstellt wird, wird an dieser Stelle vorbereitend die eigens dafür entwickelte Vorgehensweise angegeben.

Um aus mehreren angegebenen Referenznetzen ein ausführbares Kausalnetz zu erstellen, geht man in Renew folgendermaßen vor:

1. Füge für jede Marke eine passende (markierte) Stelle in das Kausalnetz ein (im Beispiel sind das die beiden Marken in der Initialisierungsstelle im Netz `Plattform` (Abbildung 3.12c auf Seite 98)).
2. Instanziiere das Hauptnetz im Schritt-Modus (Strg-i) (im Beispiel ist dies ebenfalls das Netz `Plattform`.)
3. Bis das Kausalnetz die gewünschte Länge erreicht hat oder bis keine Transition mehr aktiviert ist, wiederhole die angegebenen Schritte.
   a) Wähle eine aktivierte Transition und feuere sie in der Netzinstanz (rechter Mausklick). Evtl. müssen dazu die erzeugten Netzinstanzen geöffnet werden (rechter Mausklick auf die Marke, linker Mausklick auf die dann angezeigte Netzreferenz).
   b) Markiere im Netz (nicht in der Netzinstanz) alle Kanten an der zu schaltenden Transition, kopiere und füge in das Kausalnetz ein (Strg-c und Strg-v). Die Transition und die verbundenen Stellen werden automatisch mit eingefügt. Falls ein synchroner Kanal am Schaltvorgang beteiligt ist, wiederhole

den Kopiervorgang für alle beteiligten Transitionen in den jeweils entsprechenden Netzen.

c) Verbinde die eingefügten Netz-Schnipsel mit dem bisherigen Kausalnetz entsprechend den Regeln für die Entwicklung von Kausalnetzen: Ersetze alle Eingangsstellen für jede Marke durch eine entsprechende im Kausalnetz vorhandene Stelle und stelle die Ausgangsstellen pro Marke einzeln dar. Alle Kanten sind einfache Kanten. Verbinde die synchronisierten Transitionen eines Schaltvorganges mit einer gestrichelten senkrechten Linie.

d) Für Transitionen mit leerem Vorbereich füge eine markierte Stelle in grau hinzu (auch für Transitionen, deren Aktivierung über einen synchronen Kanal gesteuert wird).

4. Lösche alle Anschriften an Kanten und Transitionen, die nicht Namen sind.
5. Vergröbere evtl. transitions- oder stellenberandete Bereiche (Bereiche mit synchronen Kanälen sind mit Bedacht zu vergröbern).

Das erzeugte Kausalnetz kann schrittweise ausgeführt werden – allerdings muss dabei „von Hand“ darauf geachtet werden, dass die Transitionen, deren Aktivierung über einen synchronen Kanal gesteuert wird, erst dann schalten, wenn der zugehörige Downlink auch schaltet.

2.2 Agenten

Komplexe Anwendungen können auch entlang von strukturellen Komponenten gegliedert sein. Die Metapher des Agenten ist besonders ausdrucksstark. Die hier vorgestellten Agenten werden in der Agententechnologie verwendet, dabei handelt es sich um Software-Agenten. Unter dem Begriff Agententechnologie werden seit etwa fünfzehn Jahren verschiedene Technologien zu einer neuen Abstraktionsweise für die Software-Entwicklung zusammengefasst:

- [Wooldridge und Jennings 1995]
- [Jennings und Wooldridge 1998]
- [Jennings 2000]

In ausführlichen Schulungsangeboten wurde die Agententechnologie der Software-Entwicklungsgemeinde zugänglich gemacht (z.B. in der European Agent Systems Spring School (easss) [Luck et al. 2003a]

- mit allgemeiner Einführung ([Wooldridge und Luck 2003]) und Themen wie
- AOSE ([Bergenti et al. 2003, Jennings 2003]),
- intelligente Agenten ([Klusch 2003])
- und lernende Agenten ([Wiering 2003])

Im Jahr 2003 erschien auch ein weitgreifender Ausblick für die Agententechnologie als Ganzes ([Luck et al. 2003b]), in dem kurz-, mittel- und langfristig mögliche Entwicklungen analysiert werden.

Die Wurzeln der Agententechnologie liegen vor allem in der *Künstlichen Intelligenz* (KI) und in der *Softwaretechnik*. In der KI sind Agenten „intelligente“ Einheiten, die Informationen verarbeiten können. In der Softwaretechnik sind Agenten ein Konzept zur Abstraktion und Kapselung für verteilte, offene oder anderweitig komplexe Systeme. Der Begriff Agent bezeichnet im einfachsten Fall eine gekapselte Softwareeinheit, die asynchron mit der Umgebung interagiert. Agenten befinden sich auf Plattformen,

welche durch eine Kommunikations-Infrastruktur untereinander verbunden sind und das Multi-Agenten-System (MAS) als Ganzes ausmachen. Eng damit verwandt ist der Aspekt der Umsetzung von agentenorientierter Software (AOSE für Agentenorientierte Software-Entwicklung). Eine tiefgreifende aktuelle Darstellung der Aspekte Architekturen, Methoden, Programmiersprachen und Werkzeuge für die Entwicklung verteilter agentenorientierter Softwaresysteme wurde mit den Arbeiten [Pokahr 2007] und [Braubach 2007] bereitgestellt.

2.2.1 Umfeld der Agententechnologie

Die Agententechnologie hat deutliche Beziehungen zu einer ganzen Reihe von weiteren Fachgebieten, so zur Künstlichen Intelligenz, zu den Sozialwissenschaften, zu Verteilten Systemen und zur Spieltheorie.

- Künstliche Intelligenz: Für einen Agenten werden eine Wissensbasis und Ziele modelliert; diese bestimmen zusammen mit seiner Wahrnehmung der Umgebung (sei es über Nachrichten oder Sensoren) sein spezifisches Verhalten. Adaption in Form von Lernfähigkeit oder Flexibilität spielt in diesem Zusammenhang eine wichtige Rolle. Planvolles Handeln und Priorisierung können reaktiv (Subsumption), rein proaktiv (deliberative Agenten, z.B. nach dem BDI-Modell) oder als Kombination umgesetzt werden (hybride Entscheidungsarchitektur).

- Sozialwissenschaften: Die Fragen nach dem Entstehen, Aufbau und Funktionsweise von Organisationen, Gruppen, Rollen, Fähigkeiten und Verantwortung können mit Agententechnologie modelliert werden. Die Sprechakttheorie und Kommunikationsmuster werden andererseits auch als Metaphern für Multi-Agenten-Systeme in der Softwaretechnik verwendet.

- Verteilte Systeme: Der Umgang mit Nebenläufigkeit und technische Kommunikationsprotokolle mit Eigenschaften wie Verlässlichkeit und Verklemmungsfreiheit sind für Multi-Agenten-Systeme von großer Bedeutung.

- Spieltheorie: Der Agent wird als Einheit mit eigenen Interessen modelliert: Wie entsteht Koordinierung, wenn sie nicht „von oben“ vorgegeben ist? Es werden Mechanismen für Kooperation und Konkurrenz benannt und modelliert und so auch für die Softwaretechnik als Metaphern verfügbar gemacht.

- Softwaretechnik: Die Gliederung von komplexen Anwendungen mit gekapselten Einheiten ist essentiell für die Softwaretechnik. Die Agententechnologie liefert Metaphern von besonders hohem Abstraktionsgrad.

- Theoretische Informatik: Abstraktion, Prinzipien und Konzepte sind Gegenstand der theoretischen Informatik und wichtige Bestandteile der Agententechnologie. Die Mobilitätskalküle dienen zur formalen Modellierung mobiler Bestandteilen in verteilten Systemen. So bekommt die theoretische Informatik durch die Agententechnologie einen Rahmen, der sie mit den anderen genannten Gebieten verknüpft.

Jedes dieser Gebiete profitiert von der Agententechnologie, weil das jeweilige Spezialwissen in einen größeren Zusammenhang gestellt wird. Jedes dieser Gebiete entwickelt Methoden zur strukturierten Beschreibung der jeweiligen Aspekte und zu ihrer Umsetzung in Programmcode. So ist z.B. die Autonomie zentraler Bestandteil aller beteiligten Gebiete. Eine Möglichkeit, Autonomie im Code darzustellen ist ein eigener Thread für jeden Agenten, der ihn eigenständig auf seine Umwelt einwirken lässt. Es ist für die Anwendung der Agententechnologie nicht notwendig, sich in allen angrenzenden Gebieten auszukennen. Diese Arbeit befasst sich mit softwaretechnischen und konzeptionellen Aspekten der Agententechnologie.

2.2.2 FIPA

In der Softwaretechnik kommunizieren Agenten asynchron mittels *Nachrichten* in einer Agentenkommunikationssprache unter Verwendung von Begriffen aus einer anwendungsspezifischen *Ontologie*. Die dazu notwendige Standardisierung von Agententechnologie bezüglich Lebenszyklus und Kommunikationsstandards wird von dem IEEE-Gremium FIPA [FIPA 23 2005, FIPA 2005] betrieben (z.B. die Agentenkommunikationssprache FIPA-ACL). Die Kommunikation zwischen Agenten wird in *Interaktionsprotokollen* spezifiziert (syntaktisch wurden diese im Abschnitt 2.1.2 vorgestellt). Agenten werden auf *Plattformen* ausgeführt, diese dienen als Laufzeitumgebung und als logische Orte in einem Multi-Agenten-System. Nachrichten werden als Handlungen verstanden und entsprechend mit einem *Performativ* gekennzeichnet Z.B. wird eine Aufforderung mit einem *Request* eingeleitet (d.h. der Empfänger soll etwas tun) und eine Informationsnachricht mit einem *Inform* (d.h. es ist dem Sender egal, was der Empfänger mit der Information tut).

Ein wichtiger Faktor bei den Beziehungen zwischen Agenten sind *Dienste*, die von Agenten angeboten und genutzt werden. Ein Agent hat in einer Agentenanwendung bestimmte *Rollen* inne und bietet spezifische Dienste an. Um in einer dynamischen Umgebung passende Dienstanbieter zu finden, nutzen Agenten einen (möglicherweise verteilten) *Verzeichnisdienst*. Von der FIPA standardisierte Dienste sind der Directory Facilitator (DF) als Verzeichnisdienst und das Agent Management System (AMS) als Instanz für Erzeugung, Migration, evtl. Einschränkung von Agenten auf Plattformen. Die bekannteste Implementierung der FIPA-Standards in Form eines Rahmenwerkes zur Entwicklung und Ausführung von Agenten ist JADE ([Bellifemine et al. 2009]). Die FIPA listet weitere Plattformen auf ihrer Webseite auf[2]. Nguyen et al. und Ricordel et al. liefern zwei Beispiele für Arbeiten, in denen mehrere Agentenplattformen einander gegenübergestellt und verglichen werden. Nguyen et al. vergleichen speziell FIPA-kompatible Rahmenwerke nach technischen Kriterien wie Kompatibilität, Mobilität, Sicherheit, Verfügbarkeit und Benutzbarkeit ([Nguyen et al. 2002]), während Ricordel et al. vollständige Entwicklungsansätze vergleichen, bei denen die jeweilige Unterstützung in den Phasen Analyse, Entwurf, Umsetzung und Betrieb einbezogen wird ([Ricordel und Demazeau 2000]).

[2] `http://www.fipa.org/resources/livesystems.html`

2.2.3 Capa

Neben anderen Agentenplattformen wie April Agent Platform, FIPA-OS, Grasshopper, JADE, LEAP und ZEUS ist Capa ([Duvigneau et al. 2003]) eine FIPA-kompatible Agentenplattform oder vielmehr ein Rahmenwerk zur Spezifikation und Ausführung von Agenten auf Basis von Referenznetzen. Capa bietet zur Laufzeit einen Nachrichtentransportdienst, welcher FIPA-kompatible Nachrichten über Internetverbindungen versenden und empfangen kann; einen AMS-Agenten nach FIPA zur Erzeugung und Verwaltung von Agenten; einen DF-Agenten nach FIPA als Verzeichnisdienst; einen Viewer („MulanViewer“) zur Anzeige und Steuerung laufender Agenten mit ihren internen Prozessen und einen Sniffer zur detaillierten Nachrichtenverfolgung. Capa bietet einen erweiterbaren Standard-Agenten, der eine Wissensbasis mit Schlüssel-Wert-Paaren zur Verfügung hat (die Werte sind Java-Objekte). In der Wissensbasis wird einerseits festgelegt, wie der Agent auf eine ankommende Nachricht reagiert und welche Aktionen der Agent proaktiv, also ohne einen Anstoß durch eine Nachricht ausführt. Andererseits kann der Agent anwendungsbezogene Informationen ablegen.

Ein spezieller Agent wird in Capa durch eine initiale *Wissensbasis* und durch *Protokollnetze* sowie durch *Java-Hilfsklassen* implementiert. Die initiale Wissensbasis kann entweder in einem einfach editierbaren Datei-Format angegeben werden oder mit einem graphischen Editor („KBE“ für KnowledgeBase Editor) für alle Agenten einer Anwendung spezifiziert werden. Die Protokollnetze sind Referenznetze, die mittels synchroner Kanäle die Schnittstellen für den Zugriff auf die Wissensbasis und für das Senden und Empfangen von Nachrichten nutzen. Ein Protokollnetz spezifiziert das Verhalten des Agenten im Laufe einer Konversation. In Protokollnetzen werden immer wieder gleichartige Bausteine verwendet, z.B. Senden und Empfangen von Nachrichten, bedingte Verzweigung u.s.w.. Für diese Aufgaben stehen in Renew Netzkomponenten zur Verfügung. Die oben erwähnten Hilfsklassen in Java können beliebige anwendungsspezifische Funktionen erfüllen, insbesondere steht die Ontologie, also das Begriffsgerüst der Anwendung in Form von Java-Klassen als Speicher- und Nachrichtenformat zur Verfügung.

2.2.4 Mulan

Die FIPA definiert ein Referenzmodell für Agenten als autonome Einheiten, die auf Plattformen laufen und Nachrichten austauschen. Dieses Referenzmodell wurde von Rölke [Rölke 2004] mit Referenznetzen formalisiert und ausgearbeitet. Das Referenzmodell Mulan (Multi-Agenten-Netze) ist insbesondere die Implementationsgrundlage für Capa, also gelten die folgenden Aussagen für Capa mit. Mulan ist aber auch Grundlage für andere Ansätze wie organisationsorientierte Software, zur Formalisierung von sozialen Theorien, für eine Formalisierung von Webservices [Moldt et al. 2005] und für Plugin-Architekturen [Cabac et al. 2005b]. Die folgende Beschreibung bezieht sich rein auf den Agenten-Aspekt.

Mulan stellt vier Ebenen eines geschachtelten Systems in Beziehung zueinander. Die Ebenen sind mit Referenznetzen als ausführbares Agentensystem modelliert. Die oberste Ebene bildet ein verteiltes System ab, wobei jede Stelle einen Ort model-

liert und jede Transition eine Kommunikationsverbindung zwischen Orten modelliert. Das Ortsnetz muss nicht vollständig vermascht sein. Die Kommunikationsverbindungen können auch zur Migration von Agenten dienen. Auf der zweiten Ebene sind Agentenplattformen modelliert mit der Fähigkeit, Agenten zu erzeugen, zu migrieren und Nachrichten zwischen Agenten zu vermitteln. Die dritte Ebene modelliert den einzelnen Agenten mit folgenden Fähigkeiten: asynchroner Nachrichtenempfang und -versand, reaktiv und proaktiv instanziiertes Verhalten und Verwaltung von Wissen. Auf der vierten Ebene werden diese Funktionen implementiert, dabei bilden die Referenznetze für das Verhalten (die Protokollnetze) den wesentlichen anwendungsspezifischen Teil der Implementierung.

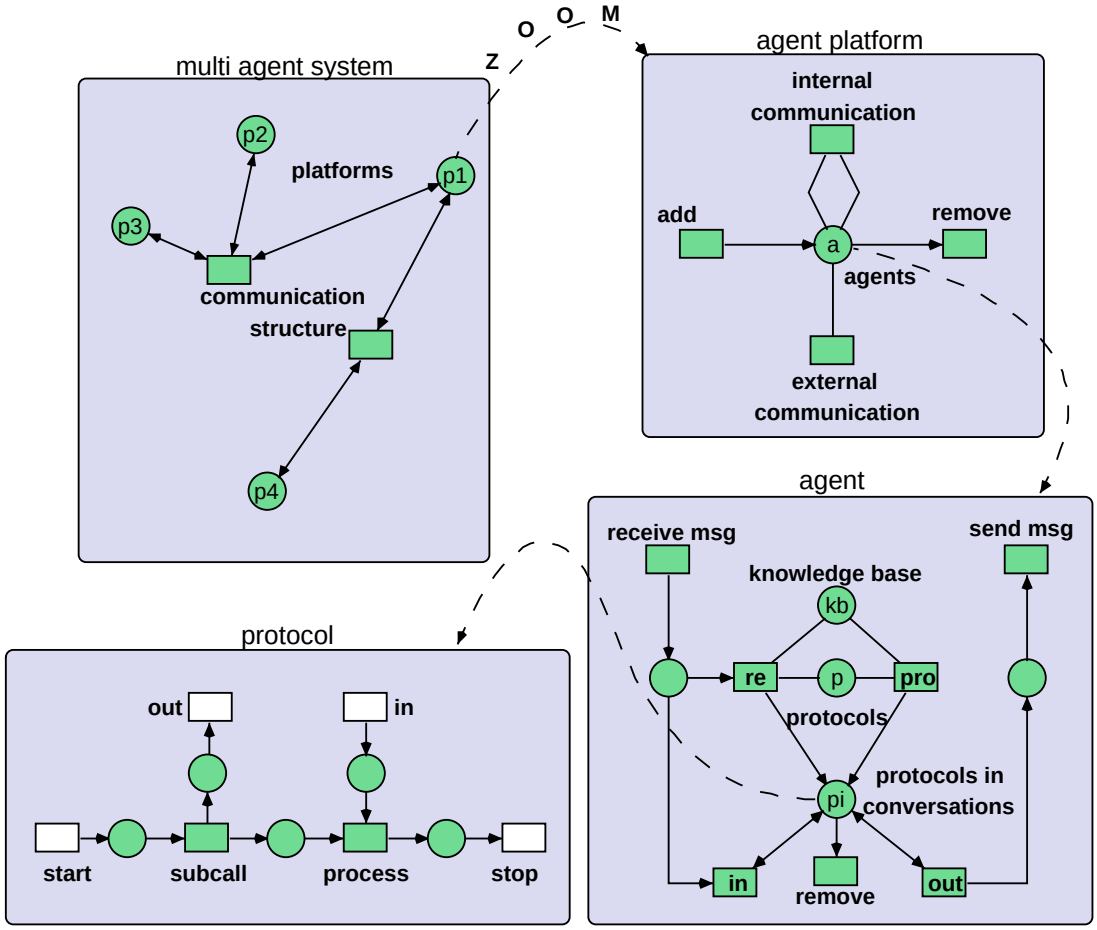


Abbildung 2.7: MULAN: Abstraktes Modell. Aus: [RÖLKE 2004, S. 157]

Abbildung 2.7 zeigt die vier Ebenen von MULAN nach Rölke [RÖLKE 2004]: die äußerste Ebene ist das Multi-Agenten-System mit Orten als Stellen und Nachrichtenkanälen als Transitionen. An jedem Ort läuft genau eine Plattform, die als Marke (genauer: als Referenz auf eine Netzinstanz) modelliert ist.

Auf der zweiten Ebene ist jede Agentenplattform auf die gleiche Weise modelliert: In einer zentralen Stelle sind die Agenten als Marken (Netzreferenzen) enthalten. Die Plattform kann neue Agenten erzeugen, indem die entsprechende Transition eine neue Marke erzeugt. Im lauffähigen Modell wird dieses Verhalten mit Benutzer- oder Agenteninteraktionen gesteuert. Eine weitere Transition kann Agenten beenden, d.h. eine Marke wird von der zentralen Stelle entfernt. Innerhalb einer Plattform können Nachrichten zwischen Agenten direkt ausgetauscht werden, für die plattformübergreifende Kommunikation wird die entsprechende Transition über synchrone Kanäle (die Kanalanschriften sind im Diagramm nicht dargestellt) mit der passenden Transition auf der obersten Ebene verknüpft. Jede der Marken auf der zentralen Stelle im Plattform-Netz repräsentiert einen Agenten.

Das Agenten-Netz ist die dritte Ebene von Mulan. Hier werden die Bestandteile der vierten Ebene koordiniert: Die Wissensbasis, die Fabrik (nicht dargestellt) und die Protokollnetze. Die Transitionen für Nachrichteneingang und Nachrichtenausgang werden über synchrone Kanäle an die Kommunikationstransitionen im Plattform-Netz gekoppelt. Eine eingehende Nachricht kann entweder einer bestehenden Konversation (Protokollnetzinstanz) zugeordnet werden (Transition `in`) oder die Instanziierung eines Protokollnetzes durch die Fabrik unter Nutzung der Wissensbasis bewirken (Transition `re` für reaktives Verhalten). Ein Agent kann auch ohne äußeren Anstoß, also *proaktiv*, mittels Fabrik und Wissensbasis ein Protokoll instanziieren (Transition `pro`). Ein abgearbeitetes Protokoll wird über die Transition `remove` entfernt. Im Agentennetz liegt in der Stelle `knowledge base` genau eine Marke mit einer Referenz auf die Instanz des Wissensbasis-Referenznetzes. Auf der Stelle `protocols in conversations` liegt dagegen eine variable Anzahl von Marken. Jede Marke ist eine Referenz auf eine Protokollnetzinstanz.

Stellvertretend für die vierte Ebene von Mulan ist ein Protokollnetz dargestellt. Die Transition `start` ist über synchrone Kanäle mit den beiden erzeugenden Transitionen (`pro` und `re`) im Agentennetz verbunden. Die Transition `stop` ist über einen synchronen Kanal mit der Transition `remove` im Agentennetz verbunden. Auch die Übermittlung von Nachrichten geschieht mittels synchronen Kanälen.

Diese Darstellung ist schematisch und lässt einige Details weg, so haben z.B. die Protokollnetzinstanzen über einen nicht dargestellten synchronen Kanal im Agentennetz auch Zugriff auf die Wissensbasis. Im Abschnitt 3.5 wird ein vereinfachtes, aber lauffähiges Mulan mit allen nötigen Anschriften und der Funktionsweise der Kanäle vorgestellt.

In Capa kommt der FIPA-kompatible Nachrichtenaustausch über das Internet hinzu, so dass das Systemnetz, die oberste Ebene von Mulan wegfällt zugunsten eines real verteilten Systems. Capa implementiert die Plattform in Form eines speziellen Agenten, dem Plattform-Agenten als Kombination aus Referenznetz und Plattform-Javaobjekt. Die Agenten-Ebene ist in Capa und Mulan weitgehend identisch, d.h. auch in Capa sind die Protokollnetze als geschachtelte Netze im Agentennetz als Marken enthalten.

Einige angrenzende praktische Punkte zu Capa werden im Verlauf der Arbeit noch detailliert aufgegriffen: (a) Der Standard-Agent von Capa wird in dieser Arbeit um eine Schnittstelle für Entscheidungskomponenten erweitert (Darstellung des erweiter-

ten Agenten in Abschnitt 3.4.1, Details zur Umsetzung im Abschnitt 5.2.1). (b) Die in [Rölke 2004] vorgestellte Migration von Mulan-Agenten ist in Capa umgesetzt und wird in dieser Arbeit im Abschnitt 5.4.4 konkret ausformuliert und dargestellt. (c) Schließlich gehört zu Capa auch ein Software-Entwicklungsansatz: „Paose“ – Petrinetzbasierte Agentenorientierte Software-Entwicklung. Die wesentlichen Begriffe und Konzepte werden im nächsten Abschnitt vorgestellt; Paose wird dann im Abschnitt 3.4 ausführlich vorgestellt.

2.2.5 PAOSE

Um komplexe Software zu entwickeln, braucht man einen softwaretechnischen Entwicklungsansatz. Ein Ansatz zur Software-Entwicklung enthält nach Moldt die Facetten Werkzeuge, Vorgehen, Techniken / Methoden, Organisation und Anwendungsfeld ([Moldt 1996]).

Das *Anwendungsfeld* beschreibt dabei, für welche Arten von Anwendungen der Ansatz geeignet ist. Die konkrete Anwendung ist natürlich erst bei einem konkreten Projekt bekannt. In diesem Sinne ist ein generischer Ansatz nie vollständig, weil ohne ein konkretes Projekt keine Software entwickelt wird und jedes konkrete Projekt hat gewisse Auswirkungen auf die anderen Aspekte des Entwicklungsansatzes. Die *Werkzeuge* für Paose sind Renew mit diversen Plugins zur Unterstützung grafischer Modellierung, Capa als Rahmenwerk für die Agentenimplementierung, der Ontologie-Editor Protégé[3]. Wichtige Modelldiagramme als Vertreter der Facette *Methoden / Techniken* sind die bereits im Abschnitt 2.1.2 vorgestellten AUML-Interaktionsprotokolldiagramme, UML-Use-Case-Diagramme sowie stärker von Paose geprägte Diagrammtypen wie z.B. Multi-Agenten-Diagramme. Die *Organisations*-Facette von Paose schlägt eine starke organisatorische Strukturierung des Entwicklerteams nach Agenten und Interaktionen vor (die sog. Matrixorganisation). Das *Vorgehen*, um von ermittelten Anforderungen zu einem laufenden System zu gelangen, wird in Paose in die iterierten Phasen Grobentwurf, Umsetzung in drei parallelen Schritten (Verhalten, Agenten und Ontologie) und Integration gegliedert. Die Einzelheiten zu den Modelldiagrammen und zur Organisation im Ansatz Paose werden aus zwei Gründen erst später im Abschnitt 3.4 vorgestellt. Erstens ist die Entscheidungskomponente als ein wichtiger Aspekt des Agentenentwurfs erst im Rahmen dieser Arbeit hinzugekommen und die Motivation dazu soll vor der Darstellung des Ansatzes erfolgen; zweitens bildet die Matrixorganisation des Entwicklerteams ein zentrales Konzept für den Entwurf der Prozess-Infrastruktur, so dass diesem Konzept an zentraler Stelle eine Darstellung zukommt.

2.2.6 Agentcities, Ace und openNet

Verteilte Systeme können aus unterschiedlich stark gekoppelten Subsystemen zusammengesetzt sein. Ein Verzeichnisdienst kann dazu verwendet werden, Systeme zu entkoppeln. Dies ist das Prinzip für CORBA ([OMG 2003]) und diverse darauf aufsetzende oder ähnlich konzipierte Middleware. Für die Agententechnologie wird ein ent-

[3] Protégé: http://protege.stanford.edu/

sprechender Dienst von den *Agentennetzen* bereitgestellt, wobei die Kommunikation zur Registrierung und Suche über (FIPA-) Agentennachrichten stattfindet.

In den Anfängen der Standardisierung für Agentenkommunikation durch die FIPA wurden die Agentennetze eher als offene Testumgebungen bereitgestellt. Das europäische Projekt „Agentcities“ hatte danach bereits das weiterführende Ziel, die allgemeinen Möglichkeiten von offenen Agentenumgebungen als eine alternative Technologie zu WebServices auszuloten, wobei sich Agenten auf einer höheren konzeptuellen Ebene als WebServices bewegen. Eines der Ergebnisse von Agentcities ist das gleichnamige offene Agentennetz. Ein zentraler Knoten beherbergt dabei globale Verzeichnisse für agentenbasierte Dienste und für Kooperationen zwischen Agenten. Das Nachfolgeprojekt „openNet“ (allerdings kein EU-Projekt) führt einen hierarchischen und skalierbaren Ansatz ein. Beide Agentennetze spezifizieren spezielle Agenten, die das eigentliche Netz ausmachen und die von jeder teilnehmenden Agentenplattform bereitgestellt werden müssen.

Der folgende Abschnitt stellt Agentcities vor und openNet wird im darauf folgenden Abschnitt vorgestellt.

Agentcities und Ace

Agentcities [ac-org 2003] bestand aus zwei EU-Projekten im Rahmenprogramm FP-06, von denen Agentcities.NET [ac-net 2003] für die Vernetzung und Förderung von Forschungsprojekten rund um offene und kommerzielle Agentenprojekte zuständig war und Agentcities.RTD [ac-rtd 2003] für die Entwicklung eines Prototyps für ein offenes Agentennetz im Internet mitsamt einer ausreichend großen Menge an sinnvollen und potentiell kommerziellen Agentendiensten darin.

Die Architektur des Netzwerkes ([Constantinescu et al. 2003]) legt fest, dass der zentrale Knoten des Agentennetzes ein **Verzeichnis der angemeldeten Agentenplattformen** beherbergt, wo auch Kontaktinformationen zur Forschungsgruppe verzeichnet sind. Das Verzeichnis wird per Web-Formular von der Forschungsgruppe mit Daten gefüllt. Der online-Status der Plattformen wird mittels einer *Ping*-Anfrage in FIPA-ACL ermittelt und dargestellt. Das **Agentenverzeichnis** wird dynamisch mit Daten gefüllt, indem in regelmäßigen Abständen die AMS-Agenten aller registrierten Plattformen nach den laufenden Agenten gefragt werden. Das **Diensteverzeichnis** wird per automatischer periodischer Anfrage bei den DF-Agenten der registrierten Plattformen mit Daten gefüllt und jeweils aktuell dargestellt. Alle Verzeichnisdienste werden über JSP-Seiten menschenlesbar dargestellt und stehen für FIPA-konforme Suchanfragen in FIPA-ACL zur Verfügung, so dass z.B. für die Verknüpfung von einfachen Diensten zu komplexeren Diensten die Anfragen beim zentralen Verzeichnisdienst herangezogen werden können.

Die technischen Voraussetzungen, um einen Agenten mit seinen Diensten in Agentcities zu publizieren, sind also: eine FIPA-konforme Agentenplattform (d.h. Kommunikation in FIPA-ACL, AMS- und DF-Agent), eine feste IP-Adresse, damit die Registrierung auch über inaktive Zeiten erhalten bleibt und schließlich ein Ping-Agent, der die regelmäßigen Anfragen des Plattform-Verzeichnisdienstes beantwortet.

Das technische Ziel des Agentennetzes Agentcities lag damit auch in der Konkretisierung der FIPA-Standardisierungen, da bei der Interoperabilität heterogener Agen-

tenplattformen die Hindernisse im Detail stecken. Deshalb war die einzige strenge Voraussetzung für die Teilnahme die FIPA-konforme Ping-Kommunikation. Durch das Netz wurden die Spezifikationen der FIPA also um einen de-Facto-Standard ergänzt: Die Agentenplattform JADE setzte die Maßstäbe, da die zentralen Verzeichnisse mit JADE realisiert waren.

Das konzeptionelle Ziel war die Betrachtung von Agentenplattformen als Orte (*Cities*), an denen jeweils mehrere Agenten beherbergt sind und ihre Dienste ausführen. Die dynamische Kombination und Konfiguration über die Verzeichnisdienste stand dabei im Mittelpunkt.

Capa ist an das Agentennetz Agentcities über das Plugin Ace ([Reese 2003]) angebunden.

openNet

Der Nachfolger von Agentcities ist openNet [Willmott 2005a, Willmott 2005b], leider ohne eigene EU-Förderung. Hier lag der Schwerpunkt auf der Skalierbarkeit des Agentennetzes. Die eigentliche Funktion der Verzeichnisdienste wurde nicht neu entwickelt, sondern als prinzipiell verfügbar angenommen, aufbauend auf den Ergebnissen von Agentcities.

Das technische Ergebnis war eine Netzarchitektur mit speziellen Agenten, die von jedem Netzknoten zur Verfügung gestellt werden, um einen verteilten Verzeichnisdienst jeweis auf dem aktuellen Stand zu halten. Eine Anfrage an diesen Verzeichnisdienst wird dann als Suche im Netz ausgeführt, was allerdings im Rahmen des Projektes nicht ausbuchstabiert wurde.

Um den aktuellen Status des Netzes und damit für jede Plattform zu erhalten, gibt es drei Agententypen, die miteinander diesen Dienst erbringen: Die **PS-Agenten** (*Platform Service*) repräsentieren je eine Agentenplattform und geben Informationen über IP-Adresse, Ports und unterstützte Nachrichtenrepräsentationen, aber auch Standortinformationen wie globale Koordinaten in Breiten- und Längengrad und Informationen über das zugehörige Forschungsteam. Die **NS-Agenten** (*Network Service*) stoßen proaktive Tests an, indem sie an einige bekannte PS-Agenten Aufforderungen verschicken, *andere* Plattformen zu testen und das Ergebnis an den NS-Agenten zurückzuschicken. Die NS-Agenten schreiben dann aktuelle Informationen zum Netz-Zustand an die **DS-Agenten** (Domain-Service). Zur Anmeldung wendet sich ein PS-Agent an die veröffentlichte Adresse des passenden DS-Agenten um sich zu registrieren. Die DS-Agenten sind hierarchisch einander zugeordnet. So entsteht eine hierarchische Gliederung aller angemeldeten Plattformen.

Mittels der globalen Koordinaten konnte eine dynamische Sicht auf den Status der weltweit verteilten Agentenplattformen erzeugt werden. Konzeptionell soll eine Suche über die Domänen eingeschränkt werden können. Mangels großflächiger Beteiligung kam es jedoch nicht zu relevanter Informationsdichte.

Das Capa-Plugin Ace wird von Wojciech Laka im Rahmen dieser Arbeit für openNet erweitert ([Laka 2007]).

2.3 Workflows

Komplexe verteilte Anwendungen bedürfen einer Prozesssicht, um die Komplexität und Wechselwirkungen zwischen lokalen Prozessen handhabbar zu machen. Im Abschnitt 2.1 wurden bereits die Petrinetze und UML-Sequenzdiagramme als für diese Arbeit relevante prozessorientierte Sichten vorgestellt. Workflowtechnologie wird in dieser Arbeit zur *Spezifikation, Definition, Ausführung und Steuerung komplexer Abläufe* in Anwendungen verwendet. Der Anwendungskontext wird dabei weiter gefasst als das ursprünglich bei der Workflowtechnologie der Fall ist und ist insbesondere nicht auf Geschäftsprozesse eingeschränkt. Vielmehr werden die grundlegenden Konzepte zum Umgang mit Prozessdefinitionen und deren Instanziierung verwendet.

2.3.1 Einführung

Ursprünglich kommt die Workflowtechnologie aus der *Verwaltung von Formularen*, wobei ein Vorgang (eng. case) von verschiedenen Abteilungen eines Unternehmens bearbeitet wird. Die Workflowmanagement-Coalition (*WfMC*) hat als Gremium mit mehreren Industriepartnern ein Referenzmodell für Workflowtechnologie formuliert [WfMC 1995]. Das Ziel des Referenzmodells ist einerseits eine gemeinsame Begriffsbildung und andererseits deutlich herausgearbeitete Schnittstellen zwischen den Bestandteilen eines Workflowmanagementsytems, um langfristig eine Interoperabilität zwischen Produkten von verschiedenen Anbietern zu ermöglichen. Folgend der *WfMC* [WfMC 1995, S. 10] gibt es vielfältige Einsatzgebiete für Workflows: Workflows werden zur Teamunterstützung (*Groupware*) oder Projektunterstützung eingesetzt und Werkzeuge für das *Business Process Reengineering* werden für die Analyse, Modellierung und (Re-) Definition von Geschäftsprozessen eingesetzt und bieten eine Möglichkeit, die Effekte abzuschätzen, die bei einer Änderung der Geschäftsprozesse auftreten können. Ein weiteres Thema sind *transaktionale Workflows*, durch die transaktionsbasierte Anwendungen ausgehend von der ehemals zentralisierten Arbeitsweise auf mehrere Rechner verteilt werden können.

Die *WfMC* definiert:

> **Workflow** „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.“ [WfMC 1997, S. 385].
>
> **Workflowmanagementsystem (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.“ [WfMC 1997, S. 386].

Workflows und alle angrenzenden Themen werden sowohl in der Literatur als auch in der Praxis von zahlreichen Personen, Gruppen und Organisationen intensiv bearbeitet. Wil van der Aalst hat auf dem akademischen Gebiet der Workflows zahlreiche

fundamentale Beiträge geleistet. Peter Dadam und Manfred Reichert sind besonders aktiv auf dem Gebiet der adaptiven Workflows. Im Folgenden werden Teile aus dem Tutorial *Business Process Management demystified* ([Aalst 2003]) aufgegriffen.

Ein Workflow wird in drei Dimensionen dargestellt: Kontrollflussdimension, Ressourcendimension und Anwendungsfalldimension (siehe Abbildung 2.8). Ein Workflow wird aus *Tasks* mit einer bestimmten Präzedenzrelation zusammengesetzt; ein ausführbarer Task in einer Workflow-Instanz mit seinen Falldaten (engl. *case*) heißt *Workitem*; ein Workitem zusammen mit allen notwendigen Ressourcen, welches einem Benutzer zur Bearbeitung zugeordnet ist, heißt *Activity*.

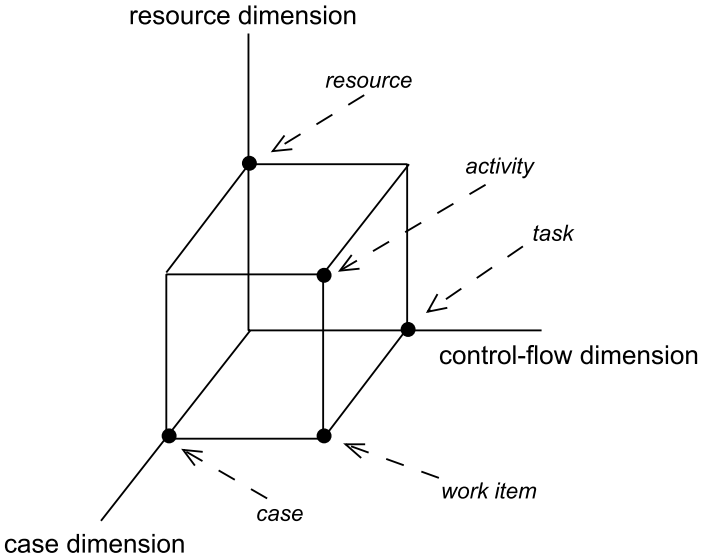


Abbildung 2.8: Drei Dimensionen eines Workflows. Aus: [Aalst 2003, S. 8]

Aalst benennt weiter fünf Perspektiven, also Sichten auf die Spezifikation eines Workflows: (1) In der Kontrollflussperspektive werden die Abhängigkeiten der Teilaufgaben (*Tasks*) definiert; (2) in der Ressourcenperspektive werden die Fähigkeiten und Berechtigungen eines Benutzers als Rollen modelliert, so dass die Workitems einer Workflow-Instanz einem passenden Benutzer zur Bearbeitung zugeordnet werden; (3) in der Datenperspektive wird definiert, welche Daten und Datenformate zum Informationsaustausch zwischen den Akteuren und Teilaufgaben genutzt werden. Es wird dabei unterschieden zwischen Kontrolldaten (nur für die Ausführung des Workflows notwendig) und produktiven Daten (diese exisitieren unabhängig vom Workflow). (4) In der Taskperspektive werden Tasktypen und ihre Teilaufgaben beschrieben; (5) In der operationalen Perspektive werden die fachlichen Operationen, die auf den Produktionsdaten möglich sind, definiert.

Die Ausführung von Activities wird in einem Workflowmanagementsystem verwaltet (engl. *managed*), d.h. die Ausführung kann überwacht und gesteuert werden. In vielen WFM-Systemen bekommen die Benutzer eine ihrer Rolle angepasste aktuelle

Liste vorhandener Workitems, aus der sie eine Aufgabe zur Bearbeitung auswählen oder zugewiesen bekommen.

Die Einleitung zu Workflows wird mit zwei Abschnitten zur Analyse und zu Definitionssprachen für den Kontrollfluss von Workflows abgeschlossen.

2.3.2 Analyse von Workflows

Werkzeuge für die Analyse von Workflows sind z.B. Woflan [Aalst 2008] und LoLa (*A Low Level Petri net Analyzer*, [Wolf 2008]). Für die Analyse können z.B. formale Eigenschaften von Petrinetzen verwendet werden sowie speziellere Eigenschaften für Workflows, wie sie z.B. von Aalst [Aalst 2003] anhand der Workflownetze definiert werden. Workflowpetrinetze und ihre Subklassen und Erweiterungen werden weiter unten im Abschnitt 2.3.4 vorgestellt.

Zur Entwurfszeit erlaubt ein WFMS die Spezifikation der fünf Perspektiven. Zur Laufzeit wird entsprechend der ankommenden Falldaten und der Verfügbarkeit von Ressourcen die Ausführung von Workflows vom WFMS verwaltet. Die älteren WFM-Systeme erlauben nur für die Kontrollfluss- und die Ressourcen-Perspektive eine explizite Spezifikation. Die übrigen Perspektiven werden implizit innerhalb der Kontrollflussdefinition durch Annotationen spezifiziert (vgl. [Aalst et al. 1999, S. 3]).

Für die Analyse von Workflows dominiert ebenfalls die Kontrollflussdefinition als Gegenstand der Betrachtung. Bei der Analyse von Workflows wird von den übrigen Perspektiven abstrahiert: Im Normalfall wird ein einem WFM-System für jeden Task genau eine Ressource zugeordnet, also sind Verklemmungen durch überkreuzende Belegungen ausgeschlossen. Aus diesem Grund kann von der Ressourcenperspektive abstrahiert werden. Weiter wird angenommen, dass jeder Konflikt im Kontrollfluss erst anhand der Falldaten aufgelöst wird. Deshalb kann für die Analyse von einer nichtdeterministischen Entscheidung ausgegangen und so von der Datenperspektive abstrahiert werden. Die Task- und die operationale Perspektive zielen auf die Effekte in der Außenwelt, welche für die Analyse von Workflows nicht betrachtet werden, deshalb werden Tasks als atomar betrachtet und Operationen innerhalb eines Tasks werden nicht berücksichtigt (eine Ausnahme bilden verfeinerte Tasks, die selbst eine Workflowdefinition enthalten und vom WFMS ausgeführt werden).

Definitionssprachen für den Kontrollfluss

Es gibt eine Vielzahl von Definitionssprachen für Workflows. Aalst et al. haben eine Sammlung von Workflowpatterns vorgestellt ([Aalst et al. 2003]), anhand derer die Ausdrucksstärke von Definitionssprachen bzgl. des Kontrollflusses verglichen werden kann. Aalst legt dabei Wert darauf, dass die jeweiligen Konstrukte direkt in der jeweiligen Sprache verfügbar sind und nicht aus einfacheren Konstrukten zusammengesetzt werden müssen.

Beispiele für Definitionssprachen und für Produkte mit einer eigenen Sprache sind:

- Workflownetze [Aalst 2003] dienen ausschließlich zur Modellierung und Analyse der Prozessperspektive und zur Definition der formalen Eigenschaften von Workflows (siehe Abschnitt 2.3.4).

- YAWL („Yet another workflow language“) [Aalst und Hofstede 2005]. Ausgehend von Workflownetzen soll YAWL eine vollständige Workflowdefinitionssprache für alle fünf Perspektiven sein.
- Ereignisgesteuerte Prozessketten (EPK) [Scheer und Thomas 2005] sind in ein Gesamtkonzept zur Modellierung von Unternehmensprozessen eingebunden (ARIS — Architektur integrierter Informationssysteme).
- Kommerzielle WFM-Systeme wie z.B. Staffware [Tibco 2008] oder COSA [Cosa 2008] haben eine eigene Prozessdefinitionssprache, die dann jeweils mit einem entsprechenden Modellierungswerkzeug und einer Ausführungsumgebung verbunden sind.
- XPDL (XML Process Definition Language, [WfMC 2008]) ist ein von der *WfMC* standardisiertes Format für die Prozessdefinition.
- Die Business Process Modeling Notation (BPMN) [White 2004] (aktuell bei der OMG [Object Management Group (OMG) 2008]) ist eine graphische Modellierungssprache für Geschäftsprozesse und kann in XPDL übersetzt werden.

Aalst hat in [Aalst et al. 2003] 15 Produkte anhand der Workflowpatterns verglichen und einander gegenübergestellt. Die Beschreibung von YAWL wird ebenfalls anhand der Patterns bewertet [Aalst und Hofstede 2005].

Zu Workflows folgt Abschnitt 2.3.3 mit einer Einführung in das Referenzmodell der *WfMC*, da ein Workflowmanagementsystem in die Prozess-Infrastruktur eingebunden wird. Die Abschnitte 2.3.4 und 2.3.5 setzen die Workflows für diese Arbeit in Bezug zu Petrinetzen und zu Agenten.

2.3.3 Das Referenzmodell der *WfMC*

Das Referenzmodell der *WfMC* [WfMC 1995] bietet ein Vokabular zur Einordnung von Workflowmanagementsystemen und hat sich in seiner Grundform seit 1995 nicht geändert. Abbildung 2.9 zeigt eine Übersicht der Schnittstellen und Softwarekomponenten des Referenzmodells. Der Kern eines WFM-Systems ist der *Workflow Enactment Service* (WFES). Dieser bildet die Laufzeitumgebung und ist für die kontrollierte Ausführung eines Workflows zuständig. Der WFES kann eine oder mehrere *Workflow Engines* benutzen. Jede Workflow Engine bearbeitet einen bestimmten Teil eines Workflows und verwaltet einen bestimmten Teil der Ressourcen. Die *Process Definition Tools* sind Werkzeuge zur Spezifikation und Analyse von Workflows und Ressourcen und werden zur Entwurfszeit benutzt. Solche Werkzeuge können auch für das Business Process Reengineering geeignet sein. Viele WFM-Systeme bieten drei verschiedene Werkzeugtypen an: Ein graphisches Werkzeug zur Prozessspezifikation, ein Werkzeug für die Spezifikation des Organisationsmodells (Ressourcenklassen) und ein Simulationswerkzeug zur Analyse eines gegebenen Workflows. Letzteres kann auch lediglich in einer Exportschnittstelle zu einem verbreiteten und ausgefeilten Simulationswerkzeug bestehen.

Der Benutzer interagiert zur Laufzeit mit dem Workflowsystem über die *Workflow Client Applications*. Ein Beispiel dafür ist der verbreitete Posteingangskorb, in dem aktuelle Workitems angeboten werden. Der Benutzer kann ein Workitem auswählen und die zugehörige Aufgabe zu einem spezifischen Fall ausführen. Die Workflowengine

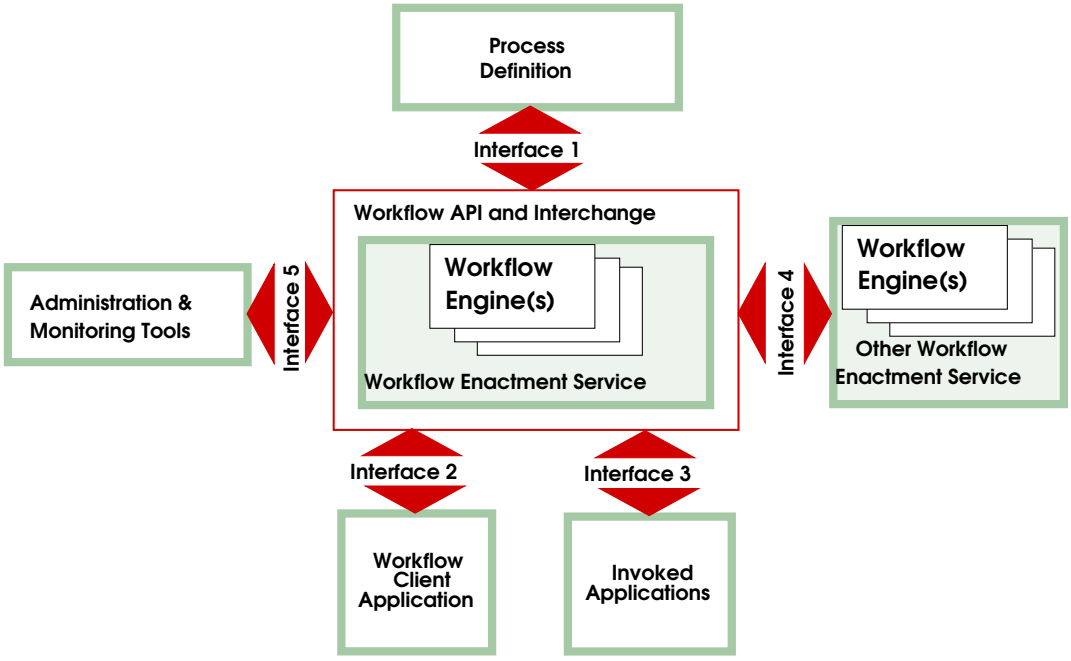


Abbildung 2.9: Das Referenzmodell der *WfMC*. Nach: [WfMC 1995, S. 20].

kann auch direkt Anwendungen aufrufen, welche die Aufgabe dann ohne Benutzerinteraktion ausführen können. Die Werkzeuge *Administration and Monitoring* können z.B. benutzt werden, um den Fortschritt eines Falls zu beobachten oder um Engpässe bei der Abarbeitung zu erkennen, aber auch für Konfiguration, um Aufgaben direkt Ressourcen zuzuordnen oder Ausnahmefälle zu regeln. Ein Workflowsystem kann mit anderen Workflowsystemen gekoppelt werden. Um die fünf Schnittstellen aus Abbildung 2.9 zu standardisieren, zielt die *WfMC* auf eine integrierte Schnittstelle, die *Workflow Application Programming Interface* (WAPI). Zur WAPI gehören dann sowohl die Funktionsaufrufe als auch die Austauschformate.

2.3.4 Workflows und Petrinetze

Petrinetze sind zur Darstellung von Workflows besonders gut geeignet, wie Aalst schon früh offiziell festgestellt hat ([Aalst 1996]). In diesem Abschnitt werden zunächst die Workflownetze nach Aalst vorgestellt sowie einige Eigenschaften, die zur Analyse von Workflows verwendet werden. Anschließend werden zwei Arbeiten zu Workflows anhand von Referenznetzen vorgestellt: erstens ein allgemeingültiges Workflow-Modell, in dem die fünf Perspektiven mittels Referenznetzen miteinander in Beziehung gesetzt werden und zweitens das referenznetzbasierte WFMS von Thomas Jacob, welches später in dieser Arbeit als Ausgangspunkt und Grundlage für ein agentenbasiertes WFMS verwendet wird.

Workflownetze nach Aalst

In diesem Abschnitt werden die wesentlichen Punkte zu Workflownetzen nach Aalst aus [Aalst 2003] dargestellt. Ein Workflownetz ist ein Petrinetz mit einer Quelle (eine Stelle ohne Eingangstransitionen) und einer Senke (eine Stelle ohne Ausgangstransitionen), engl. *source place* und *sink place*. Alle anderen Netzknoten müssen auf einem Pfad von der Quelle zur Senke liegen. Mit einem Workflownetz wird das Verhalten für einen einzelnen Vorgang modelliert.

Typischerweise werden Petrinetz-Eigenschaften wie Lebendigkeit, Beschränktheit, Sicherheit und starker Zusammenhang betrachtet. Stärkeren Workflow-Bezug haben Eigenschaften wie Wohlgeformtheit, Free-Choice-Netze, *Soundness* (allg. Korrektheit) und Wohlstrukturiertheit. Je nach gewählter Subklasse von Petrinetzen sind diese Eigenschaften entscheidbar oder unentscheidbar (für den allgemeinen Fall), und falls sie entscheidbar sind, lassen sie sich z.B. mit polynomiellem Aufwand entscheiden oder nur mit exponentiellem Aufwand. Der Nachweis der genannten Eigenschaften ist für große, komplexe oder für automatisch erzeugte Petrinetze besonders interessant.

Die Korrektheit von Workflows wird formal anhand der *Soundness*-Eigenschaft definiert: Ein Workflownetz ist **sound**, wenn (1.) von jeder erreichbaren Markierung die Senke markiert werden kann, und wenn (2.) alle anderen Stellen leer sind, sobald die Senke markiert ist und es (3.) keine toten Transitionen bzgl. der Startmarkierung (eine Marke in der Quelle) gibt.

Aalst [Aalst 2003] definiert drei Subklassen von Workflownetzen anhand von *strukturellen* Merkmalen mit jeweils erwünschten positiven Eigenschaften, die sich im Gegensatz zur *Soundness* an der Struktur des Netzes entscheiden lassen.

- Free-Choice Workflownetze, also Workflownetze mit der Free-Choice-Eigenschaft: Für je zwei Transitionen gilt, dass entweder keinerlei Überschneidungen hinsichtlich ihrer Eingangsstellen existieren oder beide Transitionen dieselbe Menge an Eingangsstellen aufweisen.
- Wohlstrukturierte Workflownetze: Jeder UND-Split wird durch einen entsprechenden UND-Join ergänzt und jeder ODER-Split wird durch einen ODER-Join ergänzt. Diese Eigenschaft wird auch für das erweiterte Workflownetz gefordert, bei dem die Senke und die Quelle durch eine zusätzliche Transition verbunden sind.
- S-Coverable Workflownetze: Das erweiterte Workflownetz lässt sich vollständig mit S-Komponenten überdecken. Eine S-Komponente ist ein stark zusammenhängendes Subnetz des Workflownetzes, das als ein Dokumenten- oder Materialfluss interpretiert werden kann: Jede Transition hat exakt eine Eingangs- und eine Ausgangsstelle. Das Workflownetz ist dann eine Kombination solcher Flüsse.

Die S-Coverable-Eigenschaft ist eine Verallgemeinerung der free-choice und wohlstrukturierten Workflownetze und gilt als Basis-Eigenschaft eines korrekten Workflows. Für free-choice und wohlstrukturierte Workflownetze lässt sich die *Soundness* jeweil polynomiell nachweisen und sie sind auch für sich genommen i.A. erwünschte Eigenschaften eines Workflows.

Referenznetze eignen sich sowohl zur Modellierung als auch als Grundlage der Implementierung eines WFMS. Für beide Fälle wird im Folgenden ein Beispiel vorgestellt.

Modellierung eines WFMS mit Referenznetzen

Aalst, Moldt, Valk und Wienberg haben das Zusammenspiel der fünf Perspektiven in einem ein WFMS mit Referenznetzen modelliert ([Aalst et al. 1999]). Darin sind die im Abschnitt 2.3.1 genannten fünf Perspektiven auf einen Workflow explizit als Referenznetze *(Objektnetze)* modelliert und ihr Zusammenspiel ist in einem *Systemnetz* modelliert, das so dem Enactment Service entspricht. Die Objektnetze sind über synchrone Kanäle mit dem Systemnetz verbunden.

Das Objektnetz für die *Kontrollflussperspektive* enthält ein Workflownetz. Gegenüber den Workflownetzen nach Aalst (z.B. [Aalst 2003]) sind diese Workflownetze transitionsberandet, d.h. statt den Stellen Quelle und Senke gibt es zwei Transitionen als Quelle und Senke mit entsprechenden synchronen Kanälen. In der Kontrollflussperspektive werden die Tasks in ihren Abhängigkeiten dargestellt. Es gibt *stille* Tasks, welche keine Interaktion mit dem restlichen WFMS benötigen, z.B. für einen Parallel-Split. Die anderen Tasks tragen Anschriften für synchrone Kanäle und werden vom Systemnetz mit den übrigen Perspektiven synchronisiert. Ein Task, der auch fehlschlagen kann und dann ein Rollback braucht, muss explizit modelliert werden, indem drei Transitionen verwendet werden. Abbildung 2.10 zeigt als Beispiel einen

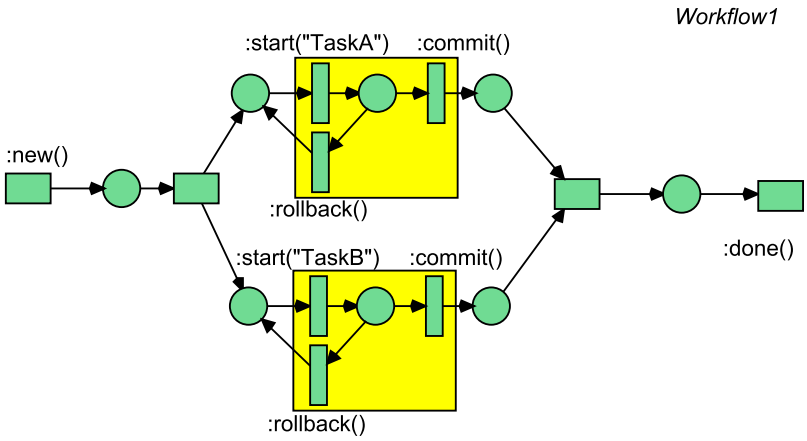


Abbildung 2.10: Beispiel: Objektnetz für die Kontrollflussperspektive.
Aus: [Aalst et al. 1999].

einfachen Kontrollfluss, in dem zwei parallele Tasks mit Rollback-Funktion modelliert sind und zwei stille Tasks für den Parallel-Split und -Join. Nachdem einer der Tasks mit Rollback-Funktion über den Kanal `start` gestartet wurde, kann er entweder über den Kanal `commit` erfolgreich beendet werden oder über den Kanal `rollback` zurückgesetzt werden.

Für die *Ressourcenperskpektive* wird in [Aalst et al. 1999] ein einfaches Netz angegeben, welches eine Instanz einer Ressource mit genau einer Rolle modelliert. Die möglichen Operationen (`new`, `use` und `release`) sind mit synchronen Kanälen realisiert. Dieses Objektnetz ist als generische Ressource bereits für viele Szenarien ausreichend, es kann aber auch mit geringem Aufwand erweitert werden, um z.B. mehrere Rollen zu erfüllen.

Für die *Datenperspektive* wird ein Objektnetz als Datenbehälter mit den Operationen `create`, `read`, `update` und `remove` angegeben. Es können Daten beliebigen Typs in Form von Schlüssel-Wert-Paaren abgelegt werden. Diese Datenbehälter werden nur für die Steuerungsdaten eines Workflows zu verwendet, da die Anwendungsdaten in der Operationsperspektive modelliert werden.

In der *Taskperspektive* werden die Einzelschritte für die Ausführung eines Tasks spezifiziert. Zu Beginn werden die erlaubten Rollen für diesen Task spezifiziert, dann werden nacheinander verschiedene fachliche Operationen aufgerufen und es wird auf Datenobjekte aus der Datenperspektive zugegriffen. Diese einzelnen Aktionen sind ähnlich wie in einem Workflownetz spezifiziert, wobei an den Transitionen mit Hilfe von synchronen Kanälen (in Form von Downlinks) die eigentlichen Aktionen durchgeführt werden. Das Tasknetz hat eine `Start`- und eine `End`-Transition und kann optional eine `Fail`-Transition haben. Diese bewirkt, dass zur Laufzeit in der Kontrollflussperspektive der Task zurückgesetzt werden kann.

In der *Operationsperspektive* wird ein einfaches Objektnetz angegeben mit den Operationen `new` und `perform`. An der Transition `perform` müssen im konkreten Fall die Anschriften ergänzt werden, welche die eigentliche Operation durchführen.

Als letzte Perspektive wird das *Systemnetz* angegeben. Hier werden die Objektnetze instanziiert und die synchronen Kanäle sinnvoll verbunden. Es wird deutlich gemacht, dass die Objektnetze je nach Anwendungskomplexität beliebig komplex aufgebaut sein können, so dass z.B. eine Operation nicht immer verfügbar ist, indem das entsprechende Objektnetz in einen inneren Zustand wechselt, in dem die `perform`-Transition nicht aktiviert ist.

Implementierung eines WFMS mit Referenznetzen

Jacob, Kummer, Moldt und Ultes-Nitsche haben ein benutzbares WFMS mit rmi-Clients auf Basis von Referenznetzen realisiert ([Jacob et al. 2002]). Die Autoren verwenden aus dem vorigen Paper [Aalst et al. 1999] das Konzept, drei Transitionen zur Modellierung eines rücksetzbaren Tasks zu verwenden. Die Vergröberung dieser Transitionen ist eine *Task-Transition* und wird in Renew zwischen die Darstellung im Editor und der ausführbaren Repräsentation übersetzt. Eine Verfeinerung ist in Abbildung 2.11 beispielhaft für drei Eingangsstellen und zwei Ausgangsstellen dargestellt. In dieser Implementierung gilt das Referenznetzwerkzeug Renew mit dem Workflow-Plugin als Enactment Service, d.h. alle Task-Transitionen in einer laufenden Simulation werden von einem globalen Listener überwacht. Sobald eine Task-Transition aktiviert ist, werden die angemeldeten Benutzer entsprechend ihren Rollen über eine rmi-Schnittstelle benachrichtigt und können aus ihrer Liste von Workitems auf einem Palmtop z.B. eine Aufgabe auswählen. Diese Auswahl bewirkt, dass die Eingangstransition der Task-Transition schaltet und dass das Workitem aus den Listen der übrigen Benutzer entfernt wird. Je nach Typ des Tasks führt der Benutzer die Aufgabe unterschiedlich aus:

- der Benutzer füllt ein Formular aus und klickt am Ende OK (erfolgreich bearbeitet) oder Cancel (Rollback im Workflow),
- er arbeitet mit einer externen Anwendung (z.B. einem Texteditor, der mit einem Standardtext vorbelegt ist)

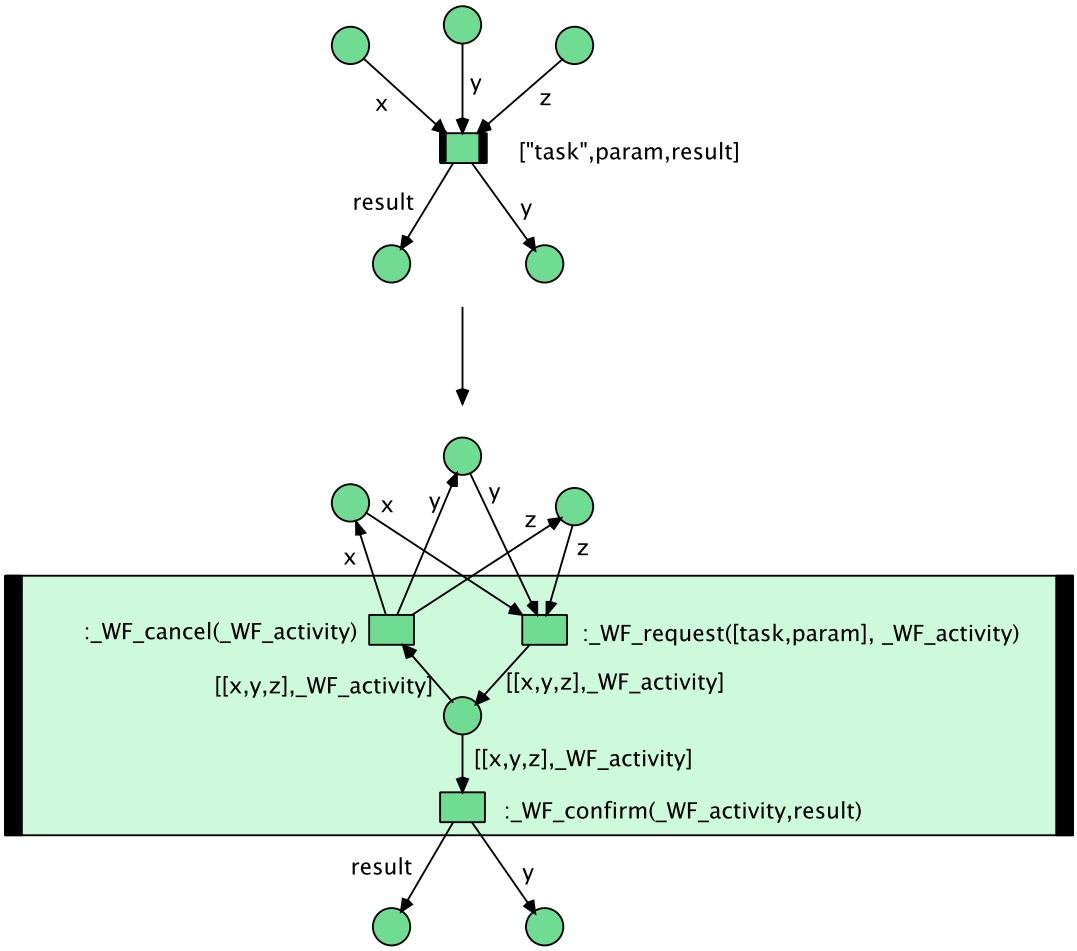


Abbildung 2.11: Task-Transition und ihre Verfeinerung.
Aus: [Jacob et al. 2002].

- oder er bearbeitet die Aufgabe manuell und kann die Bearbeitung in seiner Liste von Workitems als erfolgreich oder als nicht erfolgreich bearbeitet kennzeichnen.

Weitere verfügbare Task-Typen sind automatische Tasks, die ohne Benutzerinteraktion ausgeführt werden können und Workflow-Tasks, für die ein Subworkflow instanziiert und ausgeführt wird. Der im Papier [Jacob et al. 2002] angegebene Beispiel-Workflow hat einen Telefonversand als Anwendungsgebiet und wird im Abschnitt 6.3.1 vorgestellt.

Dieses WFMS wird später in dieser Arbeit als Grundlage für ein agentenbasiertes WFMS verwendet, indem es zunächst von einem Agenten gekapselt wird. Anschließend werden die Kernfunktionen der Task-Transition mit Listener-Mechanismus als eigenes Plugin gekapselt und im agentenbasierten WFMS verwendet. Das agentenbasierte WFMS bildet dann einen zentralen Bestandteil in der Prozess-Infrastruktur.

2.3.5 Workflows und Agenten

Die Workflowtechnologie war schon früh Gegenstand der Agentenorientierung. Adept [Alty et al. 1994] ist ein Beispiel für eine frühe Verwendung von Agentenkonzepten für flexibles Workflowmanagement und wird weiter unten in diesem Abschnitt vorgestellt. Grundsätzlich können Agenten verwendet werden, um ein Workflowmanagementsystem *funktional äquivalent* ([Buhler 2004]) nachzubauen, um dann durch Erweiterung der Flexibilität innerhalb der Agenten ein flexibles Workflowmanagement zu erreichen. Buhlers Arbeit wird ebenfalls vorgestellt. Purvis et al. [Purvis et al. 2005] benutzen ein FIPA-kompatibles Agentenrahmenwerk und ein Petrinetzwerkzeug, um das agentenbasierte WFMS JBees zu entwickeln.

Nach Yan et al. ([Yan et al. 2001]) heißt ein WFMS *agent-enhanced*, solange ein zentrales WFMS besteht, welches die Erzeugung und Vernichtung der beteiligten Agenten kontrolliert. Diese Agenten kommunizieren kaum untereinander. Drei Beispiele sind: (a) Benutzeragenten, die die Schnittstelle zum WFMS personalisieren, (b) Agenten für automatische Tasks oder (c) Schnittstellenagenten, die den Aufruf von Fremdanwendungen kapseln und flexibilisieren. Dagegen heißt ein WFMS *agent-based*, wenn die beteiligten Agenten unabhängig voneinander existieren und je einen Teil der Prozessdefinition kontrollieren und ausführen. Als eine Folge nennen die Autoren, dass die vollständige Prozessdefinition keine explizite Repräsentation mehr hat, sondern teilweise in die Agenten eingebettet ist. Die beteiligten Agenten haben dieselbe Verantwortung wie ein WFMS und kommunizieren untereinander. In diesem Sinne wird später in dieser Arbeit ein agentenbasiertes WFMS entwickelt. Die vollständige explizite Prozessdefinition soll dabei aber erhalten bleiben.

Es folgt aus dem extrem umfangreichen Feld der flexiblen oder agentenbasierten Workflowmanagementsysteme beispielhaft die Darstellung der Arbeiten dreier Arbeitsgruppen.

Flexible Workflowausführung (Buhler)

Buhler bearbeitet in seiner Dissertation und dazugehörigen Veröffentlichungen flexible Workflowausführung (workflow enactment) mit Hilfe der Agentenorientierung.

- Dissertation: [Buhler 2004]
- [Buhler und Vidal 2005]
- [Vidal et al. 2004]

Der erste Schritt in diese Richtung ist eine Umgebung zur Ausführung von Workflows, die funktional äquivalent zu einer vorgegebenen, statischen Workflowausführung ist, aber erhöhte Laufzeitkomplexität aufweist. Mit Abbildung 2.12 ordnet Buhler die Schritte zur voll flexiblen Komposition von Aufgaben nach zunehmender Komplexität an.

Die von Buhler umgesetzte Workflowausführung ist wie folgt aufgebaut: Ein Workflow ist in BPEL4WS dargestellt. Die BPEL-Beschreibung wird mit Hilfe von „Building Blocks“ manuell in ein Petrinetz übersetzt. Die Informationen, die nicht direkt dargestellt werden können, werden als Annotationen im Petrinetz vermerkt. Anschließend wird das Petrinetz partitioniert, entsprechend der beteiligten Partner. An den Schnittstellen werden die konsumierten und produzierten Token erfasst. Diese Infor-

mation wird zur Konfiguration von Task-Agenten genutzt, um den Workflow koordiniert auszuführen.

Jeder Task wird durch einen Agenten bearbeitet, der zur Laufzeit über einen DF gefunden wird. Alle diese Task-Agenten („Distributed Workflow Agent“) sind gleich implementiert und rufen entsprechend ihrer Konfiguration einen Webservice auf, der den Task letztlich ausführt.

Das erzeugte Petrinetz kann bei Bedarf mit formalen Methoden z.B. auf Deadlocks untersucht und als terminierender Workflow verifiziert werden kann. Durch die asynchrone Kommunikation mit den Taskagenten kann eine Verteilung des Workflows erreicht werden. Die Intelligenz der Agenten soll dafür genutzt werden können, dynamisch den geeigneten Webservice aufzurufen oder auch den in BPEL4WS vorgegebenen Workflow zu ändern.

Fazit: Diese Lösung ist bei einer erhöhten Laufzeitkomplexität funktional äquivalent zur statischen Ausführung eines BPEL4WS-Workflows (vgl. den Stern in Abbildung 2.12). Buhler hat also eine zusätzliche Indirektionsschicht aus Agenten zwischen Workflowbeschreibung und Workflowausführung eingeführt. Damit legt er den Implementationsort für Flexibilisierung bei der Workflowausführung fest. Petrinetze werden lediglich zur automatischen Analyse der Schnittstellen zwischen Tasks in einer BPEL4WS-Beschreibung verwendet.

Das Konzept der funktional äquivalenten Umsetzung ist ein guter Ansatz, um eine neue Technologie einzuführen, weil die Anforderungen durch ein Originalsystem exakt vorgegeben sind. Diese Lösung ist nach der Einteilung von Yan et al ([Yan et al. 2001]) *agent-enhanced*, wobei Buhler selbst keine tatsächliche Verbesserung (engl. *enhancement*) erstellt, sondern nur den Implementationsort dafür angibt.

JBees (Purvis)

JBees ist ein flexibles Rahmenwerk für verteilte Workflowsysteme, das von der Arbeitsgruppe um Maryam und Martin Purvis entwickelt wurde unter Verwendung von WebServices, Agententechnologie und Petrinetzen.

Das Ziel ist, Softwarewerkzeugen und Prozessen in einer verteilten und dynamischen Umgebung Interoperation zu ermöglichen. Petrinetze werden benutzt, um die Interaktion zwischen verschiedenen Teilprozessen zu modellieren (siehe [Purvis et al. 2001]).

Zunächst übertragen Purvis et al. die allgemein gehaltenen Konstrukte aus der Agentenorientierung auf Workflowmanagementsysteme. In [Purvis et al. 2005] wird eine Agentenplattform beschrieben, wobei die Agenten die „autonomen Komponenten“ des WFMS sind, der Petrinetz-Editor wird „Workflow modeler“ genannt, die Ausführungsumgebung für Petrinetze ist die „Workflow Engine“ und die Petrinetze sind „Modelle“ oder „Interaktionsprotokolle“, das Nachrichtentransportsystem ist der „Conversation Manager“.

Die anvisierte Anwendungsdomäne ist eine flexible Software-Entwicklungsumgebung und stimmt so mit der im Ausblick dieser Arbeit genannten Vision einer verteilten, agentenbasierten und workflowunterstützten Software-Entwicklungsumgebung überein.

Eine frühe Einführung in JBees gibt [Fleurke et al. 2003]. Zur Adaptivität von Workflows werden einige allgemeine Aussagen gemacht: Abhängig vom Änderungsbe-

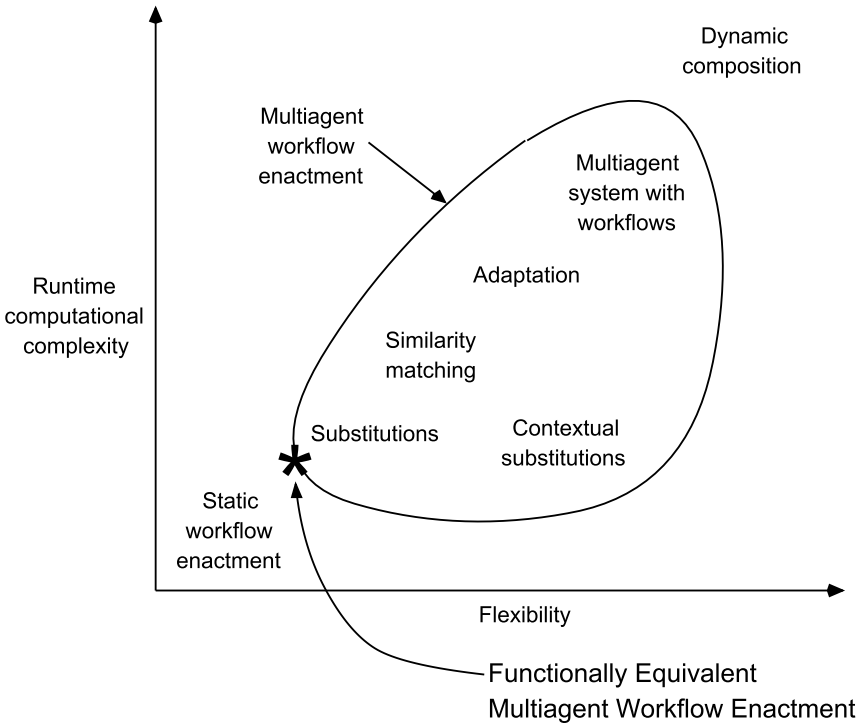


Abbildung 2.12: Flexibles Workflow-Enactment mit Agenten (aus: [Buhler 2004, S. 91])

reich, und dem Ausmaß, in dem eine Änderung auf laufende Fälle angewendet werden soll, muss entschieden werden, wann eine Änderung als „sicher“ gilt. Der einfachste Fall ist, wenn die Änderung nur „frische“ Fälle betreffen soll. Es wird eine verbesserte Version eines Algorithmus von Aalst benutzt, der die „kleinste Änderungsregion“ bestimmt.

Stellenberandete Petrinetze werden in JBees benutzt, um die Präzedenzrelation von Tasks in Workflows darzustellen. Zu jedem Task gehört ein Subnetz, welches die benötigten Ressourcen im verteilten Fall anfordert. Zur Ausführung von gefärbten Petrinetzen wird der Simulator JFern benutzt, welcher erweitert wurde um hierarchische Netze definieren und ausführen zu können.

Jeder Agent führt eine Instanz von JFern aus. Ankommende Nachrichten werden in eine Nachrichteneingangsstelle des Workflow-Netzes gelegt, dort werden von JFern die nächsten Schritte ausgeführt. Marken, die in der Nachrichtenausgangsstelle landen, werden als Agenten-Nachrichten übermittelt ([Purvis et al. 2005]). Die Petrinetze und Nachrichtenstrukturen, die zu einer Interaktion gehören, können selbst in einer Nachricht im XML-Format verschickt werden. Ein Agent kann entscheiden, ob er das gesendete Petrinetz („Protokoll“) instanziieren möchte (ebenfalls [Purvis et al. 2005]). Workflowpatterns und Kommunikationspatterns können in JBees realisiert werden (siehe [Purvis et al. 2004]), die Einbindung von WebService-Technologie wird in [Savarimuthu et al. 2005] behandelt.

Die folgenden Angaben stammen aus [Savarimuthu et al. 2004]. Die Agenten für JBees sind mit dem Rahmenwerk Opal ([Purvis et al. 2002a, Purvis et al. 2002b]) implementiert. Der Begriff Monitoring wird hier im Sinne von Datensammlung und graphischer Darstellung verwendet, Controlling bezieht sich auf Warnhinweise, die durch vorher spezifizierte Kriterien wie z.B. zu große Wartezeiten ausgelöst werden. Das Monitoring wird von einem Simulationsagenten durchgeführt, der einen Workflow bei vorgegebenen Bearbeitungszeiten und Ankunftsrate simuliert und dabei Daten sammelt, wie z.B. Durchsatz, Auslastung und Wartezeit. Für einen Beispielworkflow wird die Auswirkung verschiedener Ankunftsraten und verfügbarer Ressourcen verglichen; als Ergebnis der Simulation wird deutlich dargestellt, dass der Beispielworkflow nicht skaliert, in dem Sinne dass bei höheren Ankunftsraten die Bearbeitungszeit nicht durch eine erhöhte Anzahl verfügbarer Ressourcen ausgeglichen werden kann.

Fazit: Purvis et al. arbeiten bzgl. Ausgangspunkt, Methode und Fernziel mit einem sehr ähnlichen Ansatz wie dem in meiner Arbeit gewählten Ansatz für ein WFMS. Den Ausgangspunkt bilden Petrinetze und Agenten. Der Fokus liegt bei Purvis et al. allerdings deutlich stärker auf der Simulation und Analyse von Workflows, während in der vorliegenden Arbeit die Workflows für eine allgemein verfügbare Prozess-Infrastruktur verwendet werden.

ADEPT (Reichert, Dadam)

Als Beispiel für ein weit fortgeschrittenes Projekt zu adaptiver Workflowausführung wird nun ADEPT vorgestellt. ADEPT wurde 2004–2007 im Rahmen des Projekts AristaFlow ([AristaFlow 2004–2007]) von Baden-Württemberg gefördert und es wurde eine Kooperation mit mehreren Industriepartnern aufgebaut.

ADEPT („Application Development based on Encapsulated Premodeled Process Templates“) ist das Rahmenwerk für adaptive Unterstützung von Geschäftsprozessen aus der Gruppe um Manfred Reichert, Peter Dadam und Thomas Bauer. Die Autoren ordnen ADEPT in das Feld der „Process Aware Information Systems“ (PAIS) [Dumas et al. 2005] ein und benennen die zugehörigen Themen: Workflowmanagement (WFM), Geschäftsprozessmanagement (BPM), Enterprise Application Integration (EAI) und Service Oriented Architectures (SOA) (vgl. [Dadam et al. 2007]).

Die Prozessdefinitionen enthalten den Kontroll- und den Datenfluss zwischen Tasks. Den Tasks werden Funktionen aus einem Funktionsrepository zugeordnet. Die Funktionen können elektronische Formulare, ausführbare Dateien, WebServices, Java Libraries und Funktionsaufrufe an bestehenden Systemen (legacy code) kapseln.

Mehrere Teilprojekte von ADEPT integrieren jeweils einen Aspekt in den Gesamtzusammenhang: In ADEPT$_{base}$ ([Reichert und Dadam 1997]) wird eine operationale Semantik für ein graphbasiertes Prozessmetamodell bereitgestellt, das auch eine Analyse auf den Prozessdefinitionen z.B. bzgl. Deadlocks und eine Basis an Flexibilität erlaubt.

Ein Workflowmanagementsystem heißt *adaptiv*, wenn zur Laufzeit Änderungen an Workflowinstanzen vorgenommen werden können. Es wird unterschieden zwischen Änderungen, die alle Instanzen betreffen und, wo angebracht, automatisch durchgeführt werden („Schema-Evolution“) und Änderungen an einzelnen Instanzen,

die keine Auswirkungen auf andere Instanzen desselben Workflows haben („Ad-hoc-Changes“). Ad-hoc-Changes und Schema-Evolution schließen sich in ADEPT nicht aus. Die Flexibilität wird mit ADEPT$_{flex}$ explizit ausgearbeitet: der ADEPT$_{flex}$-Kalkül bietet einen Satz an Modifikationsoperatoren, mit denen sich die Struktur, die Attribute oder der Status von sich in Ausführung befindlichen Prozess-Instanzen abändern lassen (z.B. in [Reichert und Dadam 1998, Reichert et al. 2003], vgl. [Dadam und Reichert 2004, Abschnitt 3.2]). Mit ADEPT$_{flex}$ können abhängig vom Zustand einer Workflowinstanz dynamische Änderungen vorgenommen werden:

- in laufende Workflow-Instanzen Tasks hinzugefügt werden oder
- aus ihnen entfernt werden,
- Tasks können übersprungen werden mit der Option, sie später noch auszuführen,
- man kann in einen inaktiven Teil des Workflows springen,
- unabhängige Tasks in eine feste Reihenfolge bringen
- oder synchronisierte Tasks entkoppeln und
- man kann einen Workflow-Teil mehrfach ausführen oder zurücksetzen.

Die Veröffentlichung [Reichert et al. 1999] beschreibt mit ADEPT$_{distribution}$ unter anderem, wie die dynamischen Änderungen mit einer verteilten Ausführung der Workflows kombiniert werden können, wobei auch die Performanz beachtet wird.

ADEPT$_{time}$ ist ein Zweig, in dem zeitliche Abhängigkeiten zwischen Tasks integriert werden, z.B. externe Termine oder ein zeitlicher Mindest- oder Höchstabstand zwischen zwei Tasks.

Das Augenmerk für ADEPT2 in [Dadam et al. 2007] liegt auf der Beschreibung der möglichen Interaktion mit dem Benutzer beim Entwurf von Workflows und bei der Änderung von laufenden Workflows und der Übertragung von Änderungen auf laufende Instanzen anhand von Beispielen aus der klinischen Umgebung.

Reichert et al. (in [Reichert und Dadam 1998, Reichert und Dadam 1997]) geben einige Details zum Prozessmodell an: Der Zustand einer Workflowinstanz ist aus (a) den Zuständen der Knoten und Kanten, (b) den Werten der Datenelemente und (c) der Ausführungshistorie (inklusive Änderungshistorie) zusammengesetzt. Ein Task hat fünf mögliche Zustände: Not_Activated, Activated, Running, Completed und Skipped. Eine Kante hat die Zustände Not_Signaled, False_Signaled und True_Signaled.

Z.B. wird ein Und-Task im Zustand Not_Activated aktiviert, wenn alle eingehenden Kanten True_Signaled sind. Seine ausgehenden Kanten werden dann ebenfalls True_Signaled. Ein Und-Task wird übersprungen (er bekommt den Zustand Skipped), wenn eine der eingehenden Kanten im Zustand False_Signaled ist. Die ausgehenden Kanten werden dann ebenfalls False_Signaled, wodurch Folgezustände ggf. ebenfalls übersprungen werden.

Fazit: ADEPT ist ein erfolgreiches Forschungsprojekt, welches adaptive Ausführung von Workflows mit praktischen Aspekten der Verteilung, Handhabung und Effizienz verknüpft. Die Erfolge führten zur Gründung der AristaFlow GmbH und wurden in [Dadam et al. 2009] zusammengefasst veröffentlicht. „Der [ADEPT-]Prototyp zeigt aber auch, dass solche High-End-WfMS große Software-Systeme sind, die leicht die Code- Komplexität eines High-End-DBMS erreichen werden“. Diese Erkenntnis aus

[Dadam und Reichert 2004] sagt deutlich aus, wie weit der Weg von theoretischen Konzepten zu einer industrietauglichen Version ist. Mein eigenes Thema, für Agentenanwendungen eine umfassende Prozesssicht verfügbar zu machen, kann in einem späteren Stadium bzgl. der Adaptivität sehr von dem ADEPT-Ansatz profitieren, zumal Rölke bereits Vorüberlegungen zur prinzipiellen Umsetzung der Änderungsformen mit Referenznetzen angestellt hat (ohne Veröffentlichung). Dadam und Reichert beziehen sich dagegen höchstens am Rande auf Agententechnologie. ADEPT zielt vor allem auf Prozesse der realen Welt (also Geschäftsprozesse, Workflows). Mit meiner Prozess-Infrastruktur sollen insbesondere und vor allem die internen Prozesse einer Anwendung bei der Entwicklung und Ausführung unterstützt werden.

2.4 Verwandte Themen

Das Grundlagenkapitel zu den Themen Prozesse, Agenten und Workflows findet hier seinen Abschluss. Im Prozessteil wurden die Petrinetze, Referenznetze sowie UML-Interaktionsdiagramme mit Erweiterungen aus AUML vorgestellt. Für den Begriff Prozess wurde in Bezug auf Petrinetzprozesse das Ausdrucksmittel „Kausalnetz mit einer Abbildung zum Originalnetz“ angegeben. Die Aspekte zur Programmierung mit Referenznetzen (Werkzeugunterstützung, Java-Anbindung u.s.w.) beschließen den Prozessteil.

Im Agententeil wurden nach einer Einführung in das Umfeld der Agententechnologie die hier relevanten Begriffe eingeführt: die FIPA aus der IEEE wurde als Standardisierungsgremium für Kommunikationsstandards vorgestellt, dann das verwendete Agenten-Rahmenwerk Capa, welches auf Mulan mit den vier Ebenen (Umgebung – Plattform – Agent – Protokoll) aufbaut. Mulan ist eine Formalisierung des Agentenreferenzmodells von der FIPA mit Referenznetzen. Der Entwicklungsansatz Paose mit den fünf Facetten eines Ansatzes wurde vorgestellt und schließlich die FIPA-konformen Agentennetze Agentcities und openNet.

Für die Workflows ist die *WfMC* das Standardisierungsgremium und das Referenzmodell für Workflows wurde vorgestellt: Der Workflow-Enactment-Service und die Workflowengines bilden den Kern eines Workflowmanagementsystems, welches fünf Schnittstellen hat: Administrations-, Definitions-, Benutzer- und Anwendungsschnittstelle sowie eine Schnittstelle zu weiteren Workflow-Enactment-Services. Der Bezug zwischen Workflows und Petrinetzen wird durch die Workflownetze von Aalst hergestellt und indem eine Modellierung des Workflow-Referenzmodells mit Referenznetzen vorgestellt wird. Der Bezug zwischen Agenten und Workflows wird anhand von diversen Literaturstellen hergestellt, insbesondere werden die Arbeitgruppen um Dadam (mit ADEPT), Purvis (mit JBees) und Arbeiten von Buhler (funktional äquivalente Umsetzung eines WFMS mit Agenten) und Yan et al. (agentenbasierte vs. agenten-*enhanced* WFM-Systeme) vorgestellt.

Abschließend werden noch einige angrenzende Themen zur flexiblen Prozesssteuerung und verbreiteten Technologien betrachtet. Einen Sonderfall bei der Ablaufsteuerung und Zuordnung von Ressourcen bilden die flexiblen Fertigungssysteme (siehe z.B. [Leymann und Roller 2000]). Die Ressourcenperspektive, von der für die Analyse von Workflows i.A. abstrahiert wird, steht hier an zentraler Stelle: die Vermeidung

von Verklemmungen aufgrund von Ressourcenbelegungen ist ein wichtiger Gegenstand der Betrachtung bei flexibler Fertigung. Flexible Fertigungssysteme sind nicht weiter Bestandteil der Betrachtungen in der vorliegenden Arbeit.

Sowohl bei den Agenten als auch bei den Workflows treten WebServices als alternative und ergänzende Technologie deutlich in den Vordergrund. Die WebServices bilden die allgemeinere Technologie, sie sind breiter verfügbar, dafür aber auch auf einem konzeptionell niedrigeren Niveau.

Grid-Technologie ist ein Sammelbegriff für die Vision von Rechnerverbänden, die gemeinsam ein großes System bilden. Insbesondere die Agentennetze verfolgen in gewissem Sinne die Vision eines Service-Grids.

2.4.1 WebServices

WebServices sind die Elemente einer dienstorientierten Architektur (engl. service oriented architecture, SOA). Über das WSRF (Web Service Resource Framework) können mit WebServices auch statusbehaftete Dienste realisiert werden. Technisch gesehen sind WebServices zustandslose Dienste, die über eine URI (Uniform Resource Identifier) identifiziert werden und mittels SOAP[4] ansprechbar sind. Ein WebService wird in der Web Service Description Language (WSDL) beschrieben. Mehrere WebServices können z.B. mittels BPEL (Business Process Execution Language) koordiniert aufgerufen werden. WebServices bilden einen Standard für asynchrone Kommunikation. In diesem Sinne werden WebServices nahezu in jedem Projekt für verteilte, heterogene, adaptive Systeme eingebunden.

Savarimuthu, Purvis et al. ([Savarimuthu et al. 2005]) bearbeiten mit JBees (siehe Abschnitt 2.3.5) u.a. die agentenbasierte Integration von WebServices in WFM-Systeme und untersuchen die agentenbasierte Komposition von WebServices im Kontext von Lieferketten ([Savarimuthu et al. 2006]). Buhler benutzt Agenten und WebServices für eine Architektur zur verteilten Workflowausführung ([Buhler 2004], siehe auch Abschnitt 2.3.5). Singh benutzt temporale Logiken zur Komposition von WebServices, ebenfalls mit dem Ziel der verteilten Workflowausführung [Singh 2003]. Böhme und Saar bearbeiten die Integration heterogener Dienste im Rahmen des EU-Projektes ASG (Adaptive Services Grid) ([Böhme und Saar 2005]). Moldt, Offermann und Ortmann konzipieren eine petrinetzbasierte Architektur für WebServices in einem Modell zur dynamischen Komposition von Web Services für Workflows (vgl. [Moldt et al. 2005] als Fortführung von [Offermann 2003]). Auch Agentcities und JADE integrieren WebServices (für das Agentennetz siehe [Dale et al. 2003], für JADE siehe [Whitestein Technologies and the JADE Board 2005]). Blake bearbeitet z.B. in [Blake 2002] agentenbasierte WFM-Systeme für organisationsübergreifende Workflows und benutzt dazu ebenfalls WebServices.

[4] SOAP, ursprünglich für *Simple Object Access Protocol*, wird offiziell seit Version 1.2 nicht mehr als Akronym gebraucht, da es erstens (subjektiv) keineswegs einfach (Simple) ist und da es zweitens nicht (nur) dem Zugriff auf Objekte (Object Access) dient. *Quelle*: Artikel SOAP. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 9. Oktober 2008, 12:30 UTC. URL: `http://de.wikipedia.org/w/index.php?title=SOAP&oldid=51649460` (Abgerufen: 27. Oktober 2008, 12:42 UTC)

Aus dieser Aufstellung wird deutlich, dass WebServices und Workflows zentrale Technologien im Umfeld der agentenorientierten, verteilten und adaptiven Systeme sind. Sehr viele Forschungsprojekte beziehen WebServices in ihre Arbeit ein. Bei Agentcities und JADE geht es dabei eher um alternative Kommunikationsstandards (SOAP und WSDL statt FIPA-ACL mit Dienstbeschreibungen), dagegen steht in den Projekten zur Dienstkomposition im Zusammenhang mit Workflows ein WebService für einen Task in einem Workflow, der dynamisch gebunden und asynchron aufgerufen wird.

2.4.2 Grid

Der Grundgedanke des Grid Computing ist die Bereitstellung von Ressourcen zur gegenseitigen Nutzung in einem Netzwerk von Teilnehmern. Dies ist auch der Grundgedanke der im Abschnitt 2.2.6 vorgestellten Agentennetze Agentcities und openNet. Dort werden Agentendienste in stets aktualisierten Verzeichnissen global verfügbar gemacht. Grid Computing (vgl. [Husemann et al. 2007]) wird ursprünglich für extrem umfangreiche Berechnungen im wissenschaftlichen Umfeld verwendet (ein sog. *Computing Grid*). Andere Formen sind *Data Grid* zum kontrollierten Austausch und zur Verwaltung von großen und verteilten Datenmengen oder *Service Grid*, über das nicht nur einfache Ressourcen wie Rechenzeit oder Speicherplatz angeboten werden können, sondern auch beliebig komplexe Dienste. Technisch betrachtet kann ein Service Grid aus statusbehafteten Web Services entsprechend dem Web Services Resource Framework (WSRF) aufgebaut sein. In diesem Sinne bieten auch Agentcities und openNet eine technische Spezifikation für ein Service Grid an. „Ein visionäres globales Service Grid könnte die inhaltliche Vielfalt des World Wide Web erreichen“ ([Husemann et al. 2007, S. 1]). Im Rahmen des EU-Projektes Adaptive Services Grid [ASG 2004–2007] wird u.a. die Integration heterogener Dienste am Beispiel von CORBA und .NET ([Böhme und Saar 2005]) bearbeitet. Auch diese Fragestellung nach der dynamischen Komposition wird in der Agenten-Gemeinschaft durch standardisierte und semantisch angereicherte Nachrichtenmechanismen behandelt, z.B. mit Interaktionsprotokollen und einer dazugehörigen Ontologie.

Der Grundgedanke des Grid Computing, speziell aber in der Ausprägung des Service Grid, ist also den Zielen der Agententechnologie nach frei verfügbaren und dynamisch kombinierbaren Diensten stark verwandt.

Foster et al. [Foster et al. 2001] geben eine Einführung in die Grid-Konzepte und stellen dieses mit dem Konzept der virtuellen Organisationen in Beziehung. Das EU-Projekt ESPRIT CrossFlow ([CrossFlow 1998–2000], [Grefen et al. 2002]) fördert die koordinierte Zusammenarbeit zwischen Unternehmen und legt den Schwerpunkt auf die Ausgliederung von Prozessteilen, dazu werden mehrere WFM-Systeme verknüpft. Grefen (siehe [Grefen 2002]) ordnet verschiedene Herangehensweisen für die Kombination von Transaktionen und Workflows (Atomizität und Prozessspezifikation) in eine Taxonomie ein, unter anderem auch Petrinetze. Reichert, Bauer und Dadam beziehen die allgemeinen Herausforderungen für adaptive Workflows auf unternehmensübergreifende Workflows (vgl. [Reichert et al. 1999]).

Agenten, Grid, Workflows, WebServices: Die Community und die Ziele im Detail sind jeweils verschieden, die grundlegende Zielsetzung dabei z.T. sehr ähnlich: die flexible und verlässliche Koordination komplexer, verteilter oder heterogener Systeme.

Kapitel 3

Grundlagen für den Entwurf der Prozess-Infrastruktur PIA

Im vorigen Kapitel wurde deutlich, dass die Unterstützung von verteilten, heterogenen oder offenen Systemen als breit verfolgtes Ziel von verschiedenen Richtungen her mit verschiedenen Ansätzen bearbeitet wird. Die Ergebnisse konvergieren in Richtung allgemeiner flexibler Infrastrukturen für übergreifende Prozesse. Lediglich für die Programmierung mit Agenten fehlt eine einheitliche integrierte Sicht auf übergreifende und interne Prozesse insbesondere zur Laufzeit. Die hier entwickelte Prozess-Infrastruktur soll hierzu für agentenübergreifende Prozesse beitragen.

Im vorigen Kapitel wurde die technologische Grundlage dafür gelegt. Dieses Kapitel legt nun die konzeptionelle Grundlage.

Zunächst werden in Abschnitt 3.1 die spezifischen Begriffe festgelegt, die im Rahmen dieser Arbeit eine wichtige Rolle spielen, insbesondere Insfrastruktur, Prozess und Prozess-Infrastrukur.

Abschnitt 3.2 stellt einige konzeptionelle Aspekte für die Prozess-Infrastruktur vor. Die im Rahmen der vorliegenden Arbeit veröffentlichten Ergebnisse zur Fragmentierung von Workflows werden in Abschnitt 3.3 vorgestellt.

Der Entwurfsansatz PAOSE wurde bereits im Abschnitt 2.2.5 kurz vorgestellt. Der PAOSE-Ansatz wird im Rahmen dieser Arbeit an zwei Punkten erweitert: im Abschnitt 3.2 wird eine notwendige Erweiterung des CAPA-Agenten eingeführt und das prozessorientierte Vorgehen des PAOSE-Ansatzes wird im Rahmen dieser Arbeit von der organisatorischen Ebene auf die Ausführungsebene erweitert. Im Abschnitt 3.4 wird der PAOSE-Ansatz inklusive des erweiterten CAPA-Agenten vorgestellt.

In Abschnitt 3.5 wird die Darstellung eines Prozesses in einem komplexen System genauer untersucht, indem eine Interaktion zweier Agenten in einem stark vereinfachten MULAN als Kausalnetz und als Interaktionsdiagramm dargestellt werden. Am Ende des Abschnitts werden noch Möglichkeiten zur Gestaltung eines Workflows für die Beispielinteraktion vorgestellt.

Im letzten Abschnitt (3.6) wird ausgehend von dieser Zusammenstellung der Bogen zum Ziel dieser Arbeit geschlagen, bevor im nächsten Kapitel die Integration als solche durchgeführt wird.

3.1 Zentrale Begriffe

Begriffe wie Infrastruktur, Prozess und System werden auch umgangssprachlich verwendet. Für formale Arbeiten werden dann formale Definitionen ausgearbeitet, wie z.B. der Begriff Prozess für ein Petrinetz. Die vorliegende Arbeit ist eher konzeptioneller Natur als formal. Deshalb wurde für diese Arbeit die Form der *Begriffsbestimmung* gewählt, um explizit auszudrücken, was unter einem Begriff im Rahmen dieser Arbeit zu verstehen ist.

3.1.1 System und Architektur

In diesem Abschnitt werden die Begriffe System und Architektur mit einigen zugehörigen Begriffen im Allgemeinen festgelegt. Die Begriffe werden anschließend zum konkreten Verständnis für diese Arbeit zuerst auf Referenznetze und dann auf Agentenanwendungen angewendet. Bei den Agentenanwendungen werden einige spezielle Architekturausprägungen nach Duvigneau [Duvigneau 2002] vorgestellt.

System und verteiltes System

Nach DIN 66201 Teil 1 (DIN 1981, S. 159) wird ein System folgendermaßen definiert (Zitat übernommen aus [Moldt 1996, S. 27f]):

> [Ein System ist] Im Sinne dieser Norm eine abgegrenzte Anordnung von aufeinander einwirkenden Gebilden. Solche Gebilde können sowohl Gegenstände als auch Denkmethoden und deren Ergebnis (z.B. Organisationsformen, mathematische Methoden, Programmiersprachen) sein. Diese Anordnung wird durch eine Hüllfläche von ihrer Umgebung abgegrenzt oder abgegrenzt gedacht.
>
> Anmerkung: Durch zweckmäßiges Zusammenfügen und Unterteilen von solchen Systemen können größere und kleinere Systeme entstehen.

Die Bestimmung entspricht der in DIN 19226. Der Begriff System ist außerdem gebräuchlich als eine Aussage über den gleichen Bauplan, durch den eine Gruppe von Bauteilen oder Geräten miteinander verwandt ist, z.B. Bausteinsystem, Gerätesystem.

Begriffsbestimmung 1 (System)
Ein System ist eine abgrenzbare Anordnung von aufeinander einwirkenden Gebilden. Meistens und insbesondere in den folgenden Begriffsbestimmungen bezeichnet ein System ein abgrenzbares Softwaregebilde, welches eine IT-Infrastruktur gemäß Begriffsbestimmung 13 auf Seite 66 nutzt. ◇

Typische „Gebilde“ in einem Software-System sind Architekturschichten, Subsysteme und instanziierte Einheiten.

In der folgenden Begriffsbestimmung zum verteilten System wird bewusst auf den Begriff des Ortes verzichtet, um Probleme mit logischen Orten und mit der Referenzsemantik zu umgehen. Das wesentliche Merkmal eines verteilten Systems ist, dass aus der Perspektive eines Subsystems der Zustand eines benachbarten Subsystems oder der Zustand des globalen Systems nur unvollständig zugreifbar ist.

Wenn in einem real verteilten System Algorithmen und Protokolle angewendet werden, welche über bestimmte Aspekte des globalen Systems sichere Kenntnis in die lokalen Subsysteme bringen, hat das resultierende System wesentliche Eigenschaften eines nicht verteilten Systems. In einem nicht verteilten System können sich Zustandsänderungen eines Subsystems unmittelbar und uneingeschränkt ausbreiten.

Begriffsbestimmung 2 (Verteiltes System)
Ein verteiltes System ist ein System bestehend aus Subsystemen, welche untereinander in Beziehung stehen und Wechselwirkungen untereinander haben. Der globale Zustand eines verteilten Systems lässt sich aus der Perspektive eines Subsystems nicht vollständig zugreifen. ◇

Managementsystem und Architektur

Für diese Arbeit wird der Begriff des Workflowmanagementsystems (vgl. die Definition auf S. 41) zum Begriff Managementsystem verallgemeinert. Zum Vergleich ist die Defintion eines Workflowmanagementsystems noch einmal angegeben:

> 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.

Workflows werden zu instanziierbaren Einheiten verallgemeinert. Die „workflow participants“ werden zu einer *darüberliegenden Schicht* verallgemeinert und die Benutzung von IT-Werkzeugen und Anwendungen zu einer *darunterliegenden Schicht* verallgemeinert.

Begriffsbestimmung 3 (Managementsystem)
Ein Managementsystem ist ein System zur Definition, Instanziierung und Ausführungsverwaltung von Einheiten. Ein Teil eines Managementsystems ist das Rahmenwerk zur Definition der Einheiten. Ein Managementsystem hat Schnittstellen zu angrenzenden Schichten in der Anwendungsarchitektur und interagiert über diese Schnittstellen mit der darüberliegenden Schicht und mit der darunterliegenden Schicht. ◇

Ein Managementsystem stellt also Managementfunktionen und ein Rahmenwerk für Systeme aus instanziierbaren Einheiten bereit (mit Definition, Verwaltung und Laufzeitumgebung sowie der Festlegung von technischen Möglichkeiten, wiederkehrenden Strukturen und Funktionen, Lebenszyklus der Einheiten).

Begriffsbestimmung 4 (Rahmenwerk)
Der Teil eines Managementsystems, der zur Entwurfszeit verwendet wird. Dies umfasst die Schnittstellen zur Definition der instanziierbaren Einheiten und legt so wiederkehrende Strukturen und ihre Gemeinsamkeiten fest. ◇

Begriffsbestimmung 5 (Laufzeitumgebung)
Der Teil eines Managementsystems, der zur Laufzeit verwendet wird. Die Laufzeitumgebung definiert den Lebenszyklus der instanziierbaren Einheiten. Dadurch sind wiederkehrende Strukturen im Verhalten einer Anwendung festgelegt. ◇

Im Gegensatz zu den wiederverwendbaren Gebilden des Managementsystems, der Laufzeitumgebung und des Rahmenwerkes, ist der Begriff der Anwendung wie folgt zu verstehen:

Begriffsbestimmung 6 (Anwendung)
Eine Anwendung ist ein System mit einem speziellen Zusammenhang in der realen Welt (Anwendungszusammenhang). Eine Anwendung wird vorrangig direkt von menschlichen Benutzern verwendet. ◇

Begriffsbestimmung 7 (Architektur)
Eine Architektur beschreibt die grundlegende Konstruktion eines Systems. Dazu gehören die Elemente des Systems, deren Aufgaben und die Zusammenarbeit der Elemente, d.h. die Schnittstellen und Interaktionen zwischen den Elementen. ◇

Referenznetz-Anwendungen

In diesem Abschnitt werden die bisher eingeführten Begriffe auf Referenznetz-Anwendungen angewendet.

Bei der Programmierung mit Referenznetzen gehören der Editor und evtl. vorhandene Netzkomponenten zum *Rahmenwerk*. Die *Laufzeitumgebung* wird durch den Simulator mit seinen verschiedenen Formalismen gegeben, aber auch durch die Möglichkeiten zur Beobachtung und Steuerung der Ausführung, also Breakpoints oder der Single-Step-Modus mit seinen Interaktionsmöglichkeiten bei der Bindungssuche. Eine *Anwendung* besteht aus mehreren Referenznetzen (und evtl. Java-Klassen, Konfigurationsdateien u.s.w.). Die *Architektur* der Anwendung bestimmt das Zusammenspiel der Referenznetze und definiert Ebenen und Einheiten der Anwendung in Form von System- und Objektnetzen. Das *Managementsystem* für Referenznetzanwendungen verwaltet Referenznetze als *instanziierbare Einheiten*, die in Form von Schattennetzen vom Simulator bei Bedarf geladen, instanziiert und ausgeführt werden. Das Dateiformat für graphische Referenznetze bildet die Schnittstelle zum Rahmenwerk zur Definition der Referenznetze. Eine Interaktion mit der *darüberliegenden Schicht* ist z.B. in Form von manuellen Transitionen gegeben, d.h. Transitionen, für die die Bindungssuche ausschließlich manuell vom Benutzer angestoßen werden kann. Die *darunterliegende Schicht* ist Java mit eigenem Rahmenwerk, eigener Laufzeitumgebung und Dateiformaten.

Agentenanwendungen

In diesem Abschnitt werden die eingeführten Begriffe auf das Umfeld der Agenten angewendet.

Ein *Agentensystem* ist ein System mit Agenten als Bestandteilen. Der Begriff Multi-Agenten-System (MAS) wird sehr vielfältig verwendet. In dieser Arbeit ist Multi-Agenten-System synonym zu Agentensystem zu verstehen. Eine *Agentenanwendung* ist ein Agentensystem mit einem speziellen Anwendungszusammenhang.

Abbildung 4.5 zeigt, dass die Grenzen einer betrachteten Agentenanwendung unabhängig von Plattformgrenzen gezogen werden können. Das Agentennetz umfasst alle

Agenten und Agentenplattformen, die in den Verzeichnissen des Agentennetzes eingetragen sind. Als Agentensystem kann hier sowohl das gesamte Agentennetz bezeichnet werden, als auch eine einzelne Plattform, als auch die eingezeichnete Agentenanwendung.

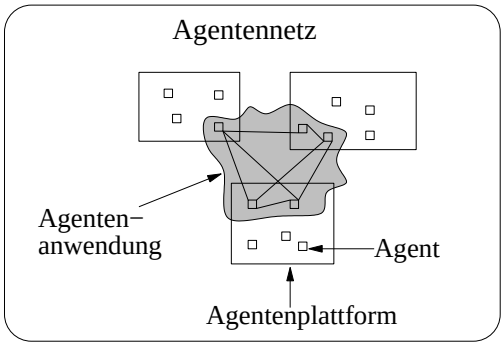


Abbildung 3.1: Agentenanwendung im Agentennetz. Geändert aus [Reese 2003]

Der Begriff der *Architektur* wird im Agentenumfeld sehr überfrachtet verwendet. So kann z.B. *Agent Architecture* für ein Rahmenwerk wie JADE[1] stehen oder *BDI Architecture* kann für die Aussage stehen, dass Agenten nach dem BDI-Modell entwickelt werden[2]. Deshalb ist es sinnvoll, einige spezialisierte Architekturbegriffe aufzugreifen (vgl. [Duvigneau 2002]).

Begriffsbestimmung 8 (Architektur eines Agentensystems)
Gemäß Begriffsbestimmung 7 beschreibt die Architektur eines Agentensystems seine Konstruktion.

Wird im Zusammenhang mit einem Agentensystem von Architektur gesprochen, ist jedoch meistens nur ein Aspekt gemeint. Das kann z.B. die Konstruktion der einzelnen Agenten sein oder die Konstruktion einer Agentenplattform. Die gesamte Architektur eines Agentensystems hat demnach viele Aspekte.

Im Folgenden wird zwischen vier Aspekten differenziert, die spezifisch für Agentensysteme sind: Entscheidungsarchitektur, Plattformarchitektur, Kommunikationsarchitektur und Anwendungsarchitektur. ◇

Die folgenden Bestimmungen zum Architekturbegriff sind angelehnt an die Glossar-Einträge in [Reese 2003, S. 142–156].

Begriffsbestimmung 9 (Entscheidungsarchitektur eines Agenten)
Dieser spezielle Aspekt einer Architektur in einem Agentensystem legt fest, wie ein Agent (re-)agiert und zu seinen Entscheidungen kommt. Entscheidungsarchitekturen werden vor allem in der (verteilten) Künstlichen Intelligenz betrachtet.

Ein Beispiel für eine Entscheidungsarchitektur ist die BDI-Architektur. ◇

[1] JADE im Internet: [Bellifemine et al. 2009]

[2] Das BDI-Modell wurde von Braubach ([Braubach 2007]) ausführlich vorgestellt, erweitert und zusammen mit Pokahr ([Pokahr 2007]) für JADE in Form eines Rahmenwerkes samt Programmiersprache unter dem Namen „Jadex“ umgesetzt ([Jadex 2009]).

In Capa ist die Entscheidungsarchitektur durch die Form der Wissensbasis mit den Protokoll-Einträgen gegeben. Die Protokollfabrik instanziiert das passende Protokollnetz zu einer ankommenden Nachricht (reaktive Entscheidung). Ein Protokollnetz kann wiederum die Instanziierung weiterer Protokollnetze veranlassen (proaktive Entscheidung). Vergleiche Abschnitt 2.2.4 zu Mulan.

Nebenbemerkung: Aus der Tatsache, dass hier eine Aussage über einen Aspekt der *Architektur eines Agenten* gemacht wird, lässt sich gemäß der Begriffsbestimmung für Architektur folgern: Ein Agent ist ein System. Dies ist für die vorliegende Arbeit eine wesentliche Aussage. Sie bildet die Grundlage für zwei wichtige Punkte, die im nächsten Abschnitt (3.2) angesprochen werden: agenteninterne Kommunikation und geschachtelte Strukturen.

Begriffsbestimmung 10 (Plattformarchitektur)
Dieser spezielle Aspekt einer Architektur in einem Agentensystem beschreibt den technischen Aufbau einer Agentenplattform mit allen Diensten, die den Agenten zur Verfügung gestellt sind. Diese Sicht ist eher softwaretechnischen Ursprungs.

Die Plattformarchitektur wird durch die Wahl eines Rahmenwerkes festgelegt, z.B. JADE oder Capa. ◇

In Capa gehören zur Plattformarchitektur der Plattform-Agent, AMS und DF, der Transportdienst mit seinen Schnittstellen und auch die Repräsentation von Nachrichten mit Java-Klassen.

Begriffsbestimmung 11 (Kommunikationsarchitektur eines Agentensystems)
Dieser spezielle Aspekt einer Architektur in einem Agentensystem beschreibt Vorgaben für die Kommunikationssprachen und die Interaktion zwischen Agenten.

Die Spezifikationen der FIPA definieren im Wesentlichen eine Kommunikationsarchitektur (u.a. [FIPA 08 2003, FIPA 26 2003, FIPA 37 2003, FIPA 61 2003], sowie [FIPA 67 2003, FIPA 70 2003, FIPA 71 2003, FIPA 75 2003, FIPA 84 2003, FIPA 85 2003, FIPA 93 2003]). ◇

Zur Kommunikationsarchitektur zählt in Capa die Umsetzung der zentralen FIPA-Spezifikationen, z.B. in Form eines Transformators, welcher zwischen der internen Java-Repräsentation von Nachrichten und dem standardisierten String-Format der FIPA vermitteln kann.

Begriffsbestimmung 12 (Anwendungsarchitektur für Agentenanwendungen)
Dieser spezielle Aspekt einer Architektur in einem Agentensystem beschreibt, wie ein agentenbasiertes System aus einzelnen Agenten zusammengefügt ist, um einen Anwendungszusammenhang zu konstruieren, d.h. wie Agenten entweder ein übergeordnetes Ziel erreichen oder ihre lokalen Ziele im Mittel gut erreichen können. Dabei können die detaillierteren Architekturaspekte (Entscheidungsarchitektur, Plattformarchitektur und Kommunikationsarchitektur) einbezogen sein. ◇

Bei der Programmierung mit Capa bilden der Editor für Referenznetze, Java, das Wissensbasis-Format, die API von Capa und FIPA sowie der Standard-Agent zusammen das *Rahmenwerk*. Die *Laufzeitumgebung* umfasst die Agentenplattform, den

Nachrichtentransport, die Verwaltung des Agenten-Lebenszyklus' durch den AMS („Agent Management System", standardisierter FIPA-Agent), und den DF („Directory Facilitator"). Eine *Agentenanwendung* besteht aus einem oder mehreren Agenten (mit ihren Wissensbasen, Protokollnetzen, Java-Hilfsklassen etc.). Die *Anwendungsarchitektur* beschreibt das Zusammenspiel der Agenten sowie die Schichten von Interaktionsprotokollen (mehrfach verwendbare, allgemeine Interaktionen vs. spezielle Behandlung einer Situation in der Anwendung). Das *Managementsystem* für Agentenanwendungen verwaltet also Agenten als instanziierbare Einheiten mit einer Schnittstelle zum Rahmenwerk und zur Laufzeitumgebung.

3.1.2 Infrastruktur

In diesem Abschnitt wird zuerst auf den Begriff Infrastruktur im Allgemeinen eingegangen, um ihn dann gegen den Begriff Middleware abzugrenzen. Der Begriff Infrastruktur hat eine enge Beziehung zum Begriff der Schicht in einer Architektur. Schließlich wird der Begriff zum konkreten Verständnis für diese Arbeit auf Agentennetze angewendet.

Der Begriff Infrastruktur setzt sich aus lat. *infra* (unter) und *Struktur* zusammen. Zunächst bezeichnet er den Unterbau des gesellschaftlichen Miteinanders, also einerseits die *technische* Infrastruktur wie Versorgung (Gas, Wasser, Strom, Müll, ...), Verkehr (Straßen, öffentliche Verkehrsmittel) und Kommunikation (Telefon, Fernsehen); andererseits die *soziale* Infrastruktur mit Bildung, Gesundheit, Recht usw. Diese Sichtweise wird von van Laak ausführlich betrachtet [Laak 1999].

Für Organisationen gibt es den spezialisierten Begriff der IT-Infrastruktur, der besonders seit der Verbreitung der ITIL (IT-Infrastructure Library) Verbreitung findet. Die IT-Infrastruktur bezeichnet den Unterbau der IT, also im allgemeinsten Sinne die Gesamtheit aller Gebäude, Kommunikationsdienste (Netzwerk), Maschinen (Hardware) und Programme (Software), die zum Betrieb einer automatisierten Informationsverarbeitung („IT") zur Verfügung gestellt werden[3]. Ein Anwender kann die IT-Infrastruktur nutzen, aber nicht verändern oder direkt beeinflussen.

In einem vielschichtigen System beginnt der Unterbau je nach Sichtweise an einer anderen Schicht (und bezeichnet dann alle untenliegenden Schichten zusammen): Ein Anwendungsbenutzer kann die Programme, die er benutzt, nicht verändern; sie gehören zur Infrastruktur aus seiner Sichtweise. Für einen Anwendungsentwickler beginnt die Infrastruktur beim Betriebssystem und der Programmiersprache, die er benutzt; er kann diese nicht verändern, sondern nur benutzen; ein Betriebssystementwickler kann die Hardware, für die er programmiert, nicht verändern. Es hängt also davon ab, wo in einer vielschichtigen IT-Struktur gerade der Fokus liegt und wo entsprechend dieser Sichtweise die IT-Infrastruktur anfängt. Insofern ist der Begriff Infrastruktur relativ zu verstehen und es wird damit die Existenz von mindestens zwei Schichten impliziert: die *untergeordnete Schicht* ist die Infrastruktur für die *übergeordnete Schicht*. Die übergeordnete Schicht nutzt eine Infrastruktur, die sie nicht direkt ver-

[3] Artikel IT-Infrastruktur. In: Wikipedia, Die freie Enzyklopädie. URL: `http://de.wikipedia.org/w/index.php?title=IT-Infrastruktur\&oldid=33298861` (Abgerufen: 3. Dezember 2007)

ändern kann. Die Infrastruktur bietet bestimmte Funktionalität an und behält dabei die Planungshoheit über ihre internen Wege und Prozesse.

Beziehung des Begriffes Schicht zum Begriff Infrastruktur

Die Semantik von Architekturschichten ist in [Lilienthal 2007] so definiert, dass keinesfalls eine unterliegende Schicht auf eine höhere Schicht zugreifen darf, dass aber in den meisten Fällen eine Schicht alle unterliegenden Schichten nutzen darf. Im konkreten Fall können Subsysteme (nach [Lilienthal 2007]) ihre eigenen Schichten besitzen, auf die von den das Subsystem nutzenden Schichten nicht zugegriffen werden darf. Im weiteren Verlauf des Textes wird explizit darauf hingewiesen, wenn eine Schicht nur von der direkt darüberliegenden Schicht direkt angesprochen wird.

Die in dieser Arbeit entwickelten Funktionen und Dienste sind für sich genommen eine Schicht: sie bauen auf existierenden Funktionen auf und werden von übergeordneten Schichten einer Anwendung genutzt. Da aber auf die existierende Infrastruktur der Agentennetze wie Agentcities aufgebaut wird und die Dienste der existierenden Infrastruktur erweitert werden, kann das Ergebnis berechtigterweise als Infrastruktur bezeichnet werden. Agentcities nutzt seinerseits die Infrastuktur, die durch das Internet bereitgestellt wird erweitert die Dienste des Internets. Das Internet nutzt und erweitert die Dienste der Infrastruktur für Strom und für Telekommunikation.

Insofern stehen die Begriffe Infrastruktur und Schicht in einer engen Beziehung zueinander. Im Gegensatz zu einer Schicht umfasst eine Infrastruktur jeweils alle unterliegenden Schichten mit.

In einem verteilten System fallen die Begriffe Schicht und Infrastruktur noch stärker zusammen. Eine verteilte Schicht muss zur Erbringung ihrer Dienste die Infrastruktur erweitern, welche die Verteilung ermöglicht. Deshalb kann eine verteilte Schicht aus dem Standpunkt des Benutzers als eine Infrastruktur bezeichnet werden.

Begriffsbestimmung 13 (Infrastruktur)
Infrastruktur bezeichnet den Unterbau eines Systems. Entsprechend dem Software-Charakter der in dieser Arbeit betrachteten Systeme handelt es sich meistens um eine IT-Infrastruktur, d.h. die Gesamtheit aller zur Ausführung eines Systems nötigen Strukturen (mit ihren internen Prozessen). Meistens wird jedoch nicht die gesamte IT-Infrastruktur, sondern nur ein Aspekt daraus adressiert.

Eine Infrastruktur ist eine verteilte Schicht mit einer speziellen Aufgabe und einer Schnittstelle zur Benutzung.

Eine verteilte Schicht in einer Software-Architektur mit einer speziellen Aufgabe, deren Prozesse von den übergeordneten Schichten nicht direkt beeinflusst werden können, wird nach ihrer Aufgabe benannt und bildet einen Teil der Infrastruktur für die übergeordneten Schichten. Entsprechend den Architekturschichten kann die Infrastruktur eines Systems gegliedert betrachtet werden. ◇

In diesem Sinne ist eine „Prozess-Infrastruktur“ eine verteilte Schicht in einer Software-Architektur und bildet einen nach seiner Aufgabe benannten Teil der Infrastruktur für ein Software-System. Die Begriffe Prozess und Prozess-Infrastruktur werden später einzeln eingeführt.

Abgrenzung zum Begriff Middleware

Für diese Arbeit wird der Begriff Infrastruktur im Gegensatz zum Begriff Middleware verwendet:

> middleware: software that mediates between an application program and a network.[4]

Middleware dient eher dazu, die Infrastruktur zu verbergen und die Kommunikation in einem verteilten System auf einer höheren Ebene zu ermöglichen (als es ohne Middleware möglich wäre). Das unterliegende Netz ist dann für die Kommunikationspartner *transparent*, d.h. es ist für die Kommunikationspartner unerheblich, ob sie sich lokal auf demselben Netzknoten befinden oder ob sie remote miteinander kommunizieren. In diesem Sinne bildet der Nachrichtentransport einer Agentenplattform eine Middleware für die Agenten. Für die Agenten ist es transparent, ob sich ein Kommunikationspartner lokal oder remote befindet. Ein Agent *kann* durch einen Vergleich der Adressen feststellen, ob sein Kommunikationspartner sich auf einer anderen Plattform befindet als er selbst. Für die Kommunikation als solche ist dieser Sachverhalt jedoch unerheblich.

Die in dieser Arbeit entworfenen Dienste und Funktionen haben nicht zum Ziel, die Kommunikation zwischen Anwendungskomponenten (hier Agenten) zu vereinfachen oder das unterliegende Netz transparent zu machen, deshalb ist Middleware ein ungeeigneter Begriff.

Infrastruktur in Agentenanwendungen

In diesem Abschnitt wird kurz beleuchtet, wie der Begriff Infrastruktur im Agentenumfeld verstanden werden kann. Gemäß der obigen Ausführungen ist eine Infrastruktur eine verteilte Schicht mit einer speziellen Aufgabe und einer Schnittstelle zur Benutzung.

Agentcities und openNet sind Infrastrukturen für Agentenanwendungen.

Beide bilden eine verteilte Schicht. Die jeweilige Aufgabe und Schnittstelle wird im Folgenden zusammengefasst (vgl. auch Abschnitt 2.2.6 zum Thema der Agentennetze).

Die Aufgabe von Agentcities und openNet ist leicht unterschiedlich: bei Agentcities liegt der Fokus auf der Funktionsweise und Verwendung von zentralen Verzeichnisdiensten für dynamisch vernetzte Agentenanwendungen, bei openNet liegt der Fokus auf der hierarchischen und skalierbaren Pflege und Nutzung von verteilten Verzeichnisdiensten. Beide zielen auf stets verfügbare aktuelle Informationen über den Betriebsstatus der beteiligten Agentenplattformen, so dass Dienste dynamisch kombiniert werden können.

Das System, welches diese Infrastruktur benutzt, ist in beiden Fällen eine Agentenanwendung. Die Schnittstelle zur Benutzung von *Agentcities* besteht aus verschiedenen Voraussetzungen: Jede beteiligte Agentenplattform muss FIPA-ACL-Nachrichten empfangen und versenden können, weiter müssen ein AMS und ein DF nach FIPA

[4] `http://www.vcs.de/uploads/media/IRT_Symposium_Bregenz_Middleware.pdf`, abgerufen am 5. 10. 2008

implementiert sein und ein Ping-Agent. Die Agenten der Agentenanwendung müssen ihre Dienste beim lokalen DF publizieren. Um nun konkret einen Dienst dynamisch aufzufinden, müssen die zentralen Verzeichnisdienste mit einer passenden Nachricht angefragt werden. Für die Benutzung von *openNet* muss eine Plattform ebenfalls einen speziellen Agenten bereitstellen, den Plattform-Service-Agenten. Dieser soll auch die Fähigkeit implementieren, andere Plattformen auf Erreichbarkeit und Funktionalität zu testen. Auf diese Weise kann jede beteilige Plattform zur der Infrastruktur beitragen, die sie selbst nutzt.

Agentcities und openNet erbringen ihren Nutzen ihrerseits mit einer gewissen Infrastruktur, so ist es z.B. wesentlich, dass die technischen Dienste unter festen IP-Adressen erreichbar sind. Ein Ziel der Verzeichnisdienste ist die dynamische Kombination heterogener Dienste zu höherwertigen Diensten. Dieses ist erst sinnvoll möglich, wenn ausreichend viele Teilnehmer die Infrastruktur nutzen (so dass die Verzeichnisdienste auch gefüllt und damit nützlich sind).

3.1.3 Prozess

Der Begriff Prozess wird sowohl umgangssprachlich als auch formal sehr vielfältig verwendet. In dieser Arbeit wird ein ausdrucksstarker Prozessbegriff für Prozesse in verteilten Agentensystemen benötigt, d.h. der Prozessbegriff soll präzise erfassbar sein und für verteilte Systeme geeignet sein, in dem Sinne, dass auch unabhängige Ereignisse in ihrem jeweiligen Zusammenhang dargestellt werden können.

Der Prozessbegriff für Petrinetze vereint graphische Darstellung, maximale Nebenläufigkeit und eine präzise Semantik. Der Prozessbegriff für Petrinetze erfüllt damit die Anforderungen und wird deshalb im Folgenden für die verschiedenen Verwendungen des Begriffs Prozess im Bereich der verteilten Agentensysteme zugrundegelegt.

Begriffsbestimmung 14 (Prozess)
Die Grundlage für einen Prozess bildet ein System. Ist das System ein Petrinetz oder kann als Petrinetz modelliert werden oder als Petrinetz modelliert gedacht werden, so entspricht der Prozessbegriff dem des Petrinetzprozesses (Definition 4 auf Seite 25).

Entsprechend der Gliederung eines Systems können sowohl Teilprozesse als auch der Gesamtprozess eines Systems betrachtet werden. Diese Begriffe werden später in der Begriffsbestimmung 17 auf Seite 73 ergänzend vorgestellt. ◇

Für Petrinetzprozesse ist das System durch ein Petrinetz vollständig beschrieben. Ein Petrinetzprozess kann mit einem Kausalnetz dargestellt werden, also mit einem Petrinetz ohne Verzweigungen und Schleifen, in dem genau eine Abwicklung möglich ist. Die Definitionen für Petrinetz und Kausalnetz wurden im Abschnitt 2.1.3 vorgestellt.

Der Prozessbegriff für reale Anwendungen entspricht dem Prozessbegriff für Petrinetze, nur dass das jeweilige System aus mehreren verschiedenartigen Bestandteilen zusammengesetzt ist und nicht unbedingt abgeschlossen ist. Deshalb sind für diese Arbeit einige Begriffe für die Darstellung und allgemeiner für die Repräsentation von Prozessen in Anwendungen notwendig.

Um die Einordnung zu erleichtern, werden als Beispiel die Prozesse in Agentennnetzen den Begriffsbestimmungen vorangestellt. Nach dem Beispiel werden die Begriffe

→Prozessbeschreibung, →Prozessdefinition und →Prozessdarstellung für die vorliegende Arbeit eingeführt. Der Pfeil wird in diesem Abschnitt zur Betonung der jeweilig bestimmten Begriffe verwendet.

Prozesse in Agentennetzen

In Agentcities oder openNet sind mehrere Agentenanwendungen zu einem System verbunden. Der Gesamtprozess bei der Ausführung besteht aus vielen verschiedenen unabhängigen und interagierenden Prozessen: ein Teilprozess für jeden Agenten und für jede Plattform. Interaktionen sind agentenübergreifende Prozesse. Über einen Verzeichnisdienst bildet sich ein Prozess mit besonders vielen beteiligten Agenten: Alle Eintragungen beeinflussen eine später stattfindende Suche. Auch die speziellen Agenten und Plattformen der Verzeichnisdienste und Netzwerkagenten (z.B. Domain-Service- und Network-Service-Agenten bei openNet) haben ihre eigenen Prozesse, die mit den Prozessen der übrigen Agenten Wechselwirkungen haben.

→Prozessbeschreibung: Bei der Entwicklung einer Anwendung für Agentcities werden (je nach Entwicklungsansatz) anwendungsbezogene Prozesse zusammen mit anderen Aspekten der Anwendung entworfen und spezifiziert. Prozesse werden als Prozessbeschreibungen repräsentiert, z.B. als Interaktionsprotokoll. Die Standard-Interaktionsprotokolle der FIPA können als Beschreibung von agentenübergreifenden Prozessen gezählt werden. →Prozessdefinition: Die Prozesse und alle weiteren Bestandteile der Anwendung werden implementiert. Die Interaktionsprotokolle bilden die Grundlage für die Implementierung von Agentendiensten. Erst mit der Implementation sind die tatsächlich möglichen Prozesse der Anwendung festgelegt (definiert). →Prozessdarstellung: Die einzelnen Agenten der Anwendung werden gestartet und ausgeführt. Der Gesamtprozess von Agentcities wird dadurch weiter abgewickelt. Die Darstellung von Teilprozessen aus dem Gesamtprozess ist nicht Teil der von Agentcities oder openNet angebotenen Dienste. Es werden allerdings in Agentcities zu Debug-Zwecken Logs zur Verfügung gestellt, die die eingehenden und ausgehenden Nachrichten für den AMS der zentralen Plattform protokollieren und so als Darstellung eines Teilprozesses dienen. Die tatsächlich stattfindenden Interaktionen als agentenübergreifende Prozesse werden aber nicht explizit dargestellt. Ebenso fehlt eine Darstellung von agenteninternen Prozessen in Agentcities, eine solche Darstellung kann aber von einzelnen Agentenplattformen bereitgestellt werden.

Repräsentationsformen für Prozesse in Anwendungen

Nach dem Beispiel im vorigen Abschnitt werden die Repräsentationsformen im Folgenden für diese Arbeit festgelegt. Für die vorliegende Arbeit werden drei Gruppen von Repräsentationsformen für Prozesse nach ihrem Zweck unterschieden: →Prozessbeschreibung, →Prozessdefinition und →Prozessdarstellung.

Aus dem Bereich der Modellierung werden einige Begriffe verwendet, um zwischen Technik, konkretem Modell und Instanzen zu unterscheiden: Eine *Technik* gibt Elemente und Beziehungen vor, z.B. sind die Petrinetze eine Technik mit Elementen wie Transitionen, Stellen, Kanten, Anschriften. Ein *Modell* ist eine konkrete Ausprägung, z.B. ein bestimmtes Petrinetz. Hier sind keine mentalen Modelle gemeint, sondern ein

Modell ist immer eine diskrete und endliche Darstellung für einen durch das Modell repräsentierten Inhalt. Eine *Instanz* benutzt das Modell wie ein Objekt eine Klasse benutzt, z.B. kann in RENEW ein Petrinetz instanziiert und ausgeführt werden.

Repräsentationsform für Prozesse	Gruppe von Tätigkeiten für eine Anwendung
Prozess-Definition	Implementation
(Instanz) Prozess-Darstellung	Ausführung Steuerung Nutzung Beobachtung
Prozess-Beschreibung	Analyse Spezifikation Entwurf Auswertung

Abbildung 3.2: Prozesse in den Phasen einer Anwendung

In den verschiedenen Tätigkeiten im Lebenszyklus einer Anwendung (Entwurf, Implementation, Ausführung, Beobachtung...) werden Prozesse mit einer jeweils geeigneten Technik repräsentiert. In Abbildung 3.2 sind die typischen Tätigkeiten im Lebenszyklus einer Anwendung in drei Gruppen dargestellt: erstens die Tätigkeiten, die direkt der Implementation einer Anwendung dienen, zweitens die Tätigkeiten während der Ausführung der Anwendung und drittens die Tätigkeiten, die in keine der vorigen Gruppen fallen. Der zeitliche Aspekt für diese drei Gruppen ist hier nicht wesentlich, deshalb ist hier bewusst auf den Begriff Phase verzichtet worden. Für jede Gruppe ist in der ersten Spalte ein zugeordneter Begriff angegeben, mit dem alle Repräsentationsformen für Prozesse, also alle Modelle einer Gruppe, egal mit welcher Technik sie erstellt sind, gleichermaßen addressiert werden können.

Im Folgenden wird der →Prozessbeschreibung, der →Prozessdefinition und der →Prozessdarstellung je ein eigener Abschnitt gewidmet.

Beschreibung von Anwendungsprozessen (→Prozessbeschreibung)

Aus den Tätigkeiten zur Analyse, zum Entwurf oder zur Spezifikation von Anwendungen resultieren →Prozessbeschreibungen. Typische Techniken sind Interaktionsdiagramme für den Entwurf von Agentenanwendungen oder Petrinetze zur Modellierung und Analyse von bestimmten Systemeigenschaften.

Der Begriff der →Prozessbeschreibung ist hier bewusst so allgemein gewählt, weil die Tätigkeiten des Entwurfs, der Analsye und der Spezifikation sehr unterschiedlich exakte Prozessrepräsentationen zum Gegenstand haben können. Ihnen allen gemeinsam ist, dass der Zweck ihrer Erstellung noch nicht die Implementation einer Anwendung ist. Durch den Begriff der Prozessbeschreibung sollen keine Annahmen über die Vollständigkeit und Exaktheit der Prozesse in der Beschreibung assoziiert werden. Eine Prozessbeschreibung repräsentiert möglicherweise mehrere alternative Prozesse.

Die vorhandenen Prozessbeschreibungen decken die erwünschten oder gar die möglichen Prozesse einer Anwendung nicht unbedingt vollständig ab:

- Es handelt sich um skizzenhafte, nicht exakte Beschreibungen
- Die Prozesse werden nur bis zu einem gewissen Detailgrad entworfen, bevor mit der Implementierung begonnen wird

- Die Anwendung wird nicht oder nicht durchgehend prozessorientiert entworfen
- Die Anwendung wird in einer Schichtenarchitektur entworfen. Wenn z.B. eine Workflowanwendung entworfen wird, so werden einzelne Workflows entworfen, zur vollständigen Beschreibung der Anwendung gehören jedoch weitere Informationen über Ressourcen und Leistungen des unterliegenden Workflowmanagementsystems.

Begriffsbestimmung 15 (Prozessbeschreibung)
Eine →Prozessbeschreibung hat die Semantik eines Petrinetzes und kann einen oder mehrere Prozesse repräsentieren. Eine →Prozessbeschreibung wird für Zwecke der Analyse, des Entwurfs, der Spezifikation oder ähnliche Zwecke erstellt. Eine →Prozessbeschreibung ist ein Modell einer geeigneten Technik. ◇

Eine →Prozessbeschreibung orientiert sich an ihrem spezifischen Zweck. Die verwendete Technik hat einen für den Zweck geeigneten Abstraktionsgrad. In einer Anwendung mit Schichtenarchitektur ist eine →Prozessbeschreibung meistens einer bestimmten Schicht zuzuordnen.

Definition von Anwendungsprozessen (→Prozessdefinition)

Die Tätigkeiten zur Implementation umfassen alle Tätigkeiten, um die Anwendung gegenüber einem Computer in einem ausführbaren oder interpretierbaren Format zu definieren.

Die Implementation legt alle prinzipiell möglichen Prozesse der Anwendung fest. In diesem Sinne wird in dieser Arbeit die Implementation einer Anwendung auch als ihre vollständige *Prozessdefinition* verstanden.

Der Begriff der Prozessdefinition für die Implementation ist hier aus der Begriffsbildung im Workflow-Umfeld angelehnt (vgl. [WfMC 1995, S. 52]: „Process Definiton – The computerised representation of a process[...]“). Eine Prozessdefinition kann mehrere alternative Prozesse in einer Definition zusammenfassend repräsentieren.

Typische Techniken zur Implementation sind Programmiersprachen wie Java, die *process definition tools* eines Workflowmanagementsystems oder spezielle Workflowdefinitionssprachen wie BPEL. Im Zusammenhang dieser Arbeit werden auch Referenznetze zur Implementation verwendet sowie nicht prozessorientierte Techniken, z.B. ein Format zur Definition der initialen Wissensbasen von Agenten. Im Zusammenhang von offenen Agentennetzen wie Agentcities oder openNet sind die Techniken zur Implementation besonders vielfältig und umfassen insbesondere auch nicht prozessorientierte Techniken, wie z.B. logische Regeln.

Wenn in einem speziellen Teil der Implementation (bezogen auf eine bestimmte Schicht der Architektur) die Festlegung der möglichen Prozesse im Vordergrund steht, wird im Folgenden von einer *expliziten Prozessdefinition* gesprochen. Ein Beispiel sind Workflows in einer Workflowanwendung. Ein wesentliches Merkmal einer expliziten Prozessdefinition ist, dass sie an einem Ort (z.B. in einer Datei, in einem Diagramm) zusammenhängend vorliegt und nicht auf verschiedene Bestandteile der Implementation verteilt ist.

Beim prozessorientierten Entwurf von Anwendungen kommt es vor, dass die Beschreibung der möglichen Prozesse für einen bestimmten Zusammenhang beim Entwurf explizit vorliegt, um dann für die Implementierung in mehrere Teile zu zerfallen. Diese Teile zusammengenommen bilden dann die *implizite Prozessdefinition* für den betrachteten Zusammenhang. Ein Beispiel dafür ist eine Agentenanwendung, die mit Hilfe von Interaktionsprotokollen[5] entworfen wird. Zur Implementation werden aus einem Interaktionsprotokoll mehrere Protokollnetze für CAPA-Agenten generiert. In diesem Fall steht beim Entwurf die Menge der erlaubten Prozesse im Rahmen einer Interaktion im Vordergrund der Betrachtung, bei der Implementation geht dieser Zusammenhang verloren, obwohl er implizit weiterhin vorhanden ist: die Protokollnetze implementieren zusammengenommen die gewünschte Interaktion. Um diesen Sachverhalt ausdrücken zu können, wird zwischen *expliziter Prozessdefinition* und *impliziter Prozessdefinition für einen gegebenen Zusammenhang* unterschieden. Eine explizite Prozessdefinition existiert in einem Stück und kann dynamisch instanziiert werden.

Die möglichen Prozesse einer Anwendung können auch durch nicht-prozessorientierte Bestandteile der Implementation festgelegt sein: Ontologie, Regeln, Strukturfestlegungen. Diese sind zunächst allgemeine Bestandteile der Implementation. Sobald durch diese Bestandteile ein spezieller Prozess-Zusammenhang implementiert wird, können diese Bestandteile der Implementation auch Bestandteile einer *impliziten Prozessdefinitionen* sein.

Begriffsbestimmung 16 (Prozessdefinition)
Die Gesamtheit der Implementation bildet die →Prozessdefinition einer Anwendung, weil durch die Implementation alle möglichen Prozesse der Anwendung festgelegt (definiert) sind.

Eine Einheit (Diagramm oder Text) der Implementation, in der die erlaubten Prozesse im Vordergrund stehen, so dass sich als semantische Beschreibung ein Petrinetz daraus erzeugen lässt, bildet eine ***explizite Prozessdefinition****. Eine explizite Prozessdefinition ist ein Modell einer gegebenen Technik.*

Teile der Implementation, die entlang von Strukturkomponenten (z.B. Agenten) oder nach logischen Gesichtspunkten organisiert sind, können durch ihr Zusammenwirken eine ***implizite Prozessdefinition im Zusammenhang der übergreifenden Prozesse*** *bilden.*

Beispiele für Formate einer **expliziten Prozessdefinition** sind Referenznetze (insb. in Form von Protokollnetzen für MULAN-Agenten), Workflow-Referenznetze, BPEL- oder BPMN-Dateien. Bei den Interaktionsprotokollen (AIPs) stehen zwar die erlaubten Prozesse im Vordergrund, AIPs sind aber in CAPA nicht Bestandteil der Implementation. Deshalb zählen AIPs hier lediglich als →Prozessbeschreibung.

Ein Beispiel für eine **implizite Prozessdefinition** wurde bereits angegeben: im Zusammenhang einer Interaktion bilden die beteiligten Protokollnetze durch ihr Zusammenwirken eine *implizite Prozessdefinition im Zusammenhang einer Interaktion.*

Ein weiteres Beispiel für eine implizite Prozessdefinition findet sich im Zusammenhang eines synchronen Kanals in Referenznetzen. Der synchrone Kanal wird durch

[5] Interaktionsprotokoll = Agenteninteraktionsprotokolldiagramm, also ein Modell der Technik AIP.

Anschriften an zwei oder mehr Transitionen implementiert. Erst durch ihr Zusammenwirken entsteht der Kanal.

In einer Anwendung mit Schichtenarchitektur können die expliziten Prozessdefinitionen einer Schicht zugeordnet werden.

Wenn die gleichen Techniken für die Spezifikation und für die Implementation einer Anwendung verwendet werden, wechselt die Zuordnung eines Modells zwischen Prozessbeschreibung und Prozessdefinition je nach aktueller Tätigkeit und Blickwinkel. So kann z.B. ein Petrinetz eine Prozessbeschreibung sein, wenn es als Grobentwurf für eine Anwendung gedacht ist. Wenn dieses Petrinetz in Renew instanziiert und ausgeführt wird, um sein Verhalten zu testen, dann ist es die Prozessdefinition für das gerade laufende System, also in diesem Falle eine Prozessdefinition für einen frühen Prototypen der geplanten Anwendung.

Abschließend ist zu betonen, dass sowohl implizite als auch explizite Prozessdefinitionen eben Prozessdefinitionen sind, also Teil der Implementation einer Anwendung. Bei einer impliziten Prozessdefinition ist nur die Betrachtungsebene eine andere als die Ebene, auf der implementiert wird. Die Implementation wird durch implizite Prozessdefinitionen nicht ungenau, sondern nur schwieriger zu verstehen in Bezug auf einen bestimmten, implizit definierten Zusammenhang.

Bezüglich der Wohlgeformtheit von Prozessdefinitionen und evtl. erwünschter Eigenschaften (z.B. Verklemmungsfreiheit, Terminierbarkeit oder Lebendigkeit) werden hier bewusst keine Aussagen gemacht. Die Ergebnisse aus dem Bereich der Petrinetze und Workflow-Petrinetze mit Begriffen, wie z.B. Soundness (vgl. Abschnitt 2.3.4 auf Seite 46), sind für explizite Prozessdefinitionen uneingeschränkt anwendbar, weil für explizite Prozessdefinitionen gefordert ist, dass sie eine Petrinetz-Semantik aufweisen.

Im Abschnitt 2.1.6 wurden einige Beispiele für die Übersetzung von Prozessdefinitionstechniken in Petrinetze angegeben.

Der Gesamtprozess einer laufenden Anwendung

Nachdem nun die Gliederung der Implementation einer Anwendung mittels expliziter und impliziter Prozessdefinitionen erläutert wurde, können die Begriffe Gesamtprozess und Teilprozess für eine laufende Anwendung bestimmt werden:

Begriffsbestimmung 17 (Gesamtprozess und Teilprozess)
Der Gesamtprozess einer laufenden (evtl. verteilten) Anwendung ist ein Prozess wie in Begriffsbestimmung 14 und entsteht aus einem sich abwickelnden System, das in seiner Gesamtheit betrachtet wird. Die laufende Anwendung ist das (ggf. verteilte) System, aus dem sich der Gesamtprozess zur Laufzeit entwickelt. Das System besteht aus dem Zusammenspiel der Implementation, der IT-Infrastruktur, der Konfiguration und äußeren Einflüssen wie z.B. Eingaben von Benutzern. Der Gesamtprozess einer laufenden Anwendung umfasst auch „uninteressante“ und unvorhergesehene Ereignisse. Der Gesamtprozess einer laufenden Anwendung bezieht beliebig fein granulare Ereignisse mit ein. Es gibt keine vollständige Darstellung für den Gesamtprozess einer laufenden Anwendung.

Ein Teilprozess ist ein beliebiger Ausschnitt des Gesamtprozesses. Sinnvolle Ausschnitte sind insbesondere durch die expliziten Prozessdefinitionen der Anwendung gegeben, z.B. die Protokollnetzinstanzen eines Capa*-Agenten.* ◇

Im laufenden System werden auf der Grundlage der Implementation Instanzen erzeugt: instanziierter Quellcode, instanziierte Workflows, instanziierte Petrinetze. Eine Instanz durchläuft exakt einen ausgewählten Prozess aus der Menge der definierten Prozesse. Der Prozess einer Instanz ist ein Teilprozess des gesamten Anwendungsprozesses. In dieser Arbeit ist mit dem Begriff Prozess in den meisten Fällen ein Teilprozess eines Systems gemeint.

Darstellung von Anwendungsprozessen (→Prozessdarstellung)

Eine →Prozessdarstellung wird aufgrund der Instanzen von Beobachtungswerkzeugen für den Benutzer erzeugt. Eine →Prozessdarstellung repräsentiert einen Ausschnitt aus dem Gesamtprozess einer laufenden Anwendung bis zum Zeitpunkt der Beobachtung. Mehrere Prozessdarstellungen können zu einer *aggregierten Prozessdarstellung* zusammengefasst werden, was hier aber nicht weiter betrachtet werden soll.

Im Folgenden soll der Unterschied zwischen expliziter und impliziter →Prozessdarstellung deutlich werden. Explizite Prozessdarstellungen sind Modelle einer gegebenen Technik. In den nächsten Absätzen werden typische Techniken für implizite und explizite →Prozessdarstellungen anhand von Beispielen beschrieben.

Im einfachsten Fall reicht eine Darstellung des Ergebnis-Zustands (z.B. der Kontostand nach einer Transaktion). Der Benutzer schließt daraus, dass der Prozess zu seiner Zufriedenheit stattgefunden hat.

Der Benutzer kann auch aus einer Reihe von nacheinander angezeigten Zuständen auf Details des Prozesses schließen. Dies ist z.B. der Fall, wenn in Renew eine Netzinstanz angezeigt wird und darin die jeweils markierten Stellen und schaltenden Transitionen hervorgehoben werden. Dies sind Zustandsdarstellungen, können aber auch als flüchtige und *implizite* Prozessdarstellungen bezeichnet werden.

Eine übliche Technik für *explizite* →Prozessdarstellung ist die Erzeugung von Listen. Listen von Prozessschritten, Ereignissen oder Zuständen werden in Log-Dateien oder auch im Simulation-Trace von Renew verwendet. Der Simulation-Trace sammelt global alle Schaltvorgänge einer Simulation in Form einer Schaltfolge inklusive Informationen zur jeweiligen Bindung. Agentcities hat für die zentralen Verzeichnisdienst-Agenten eine Log-Datei mit den kürzlich empfangenen Nachrichten zu Debug-Zwecken zur Verfügung gestellt.

Eine weitere Technik für explizite →Prozessdarstellung ist die Erzeugung von Diagrammen wie z.B. Interaktionsdiagramme[6].

[6] Man beachte den Unterschied zwischen den Begriffen:
- Interaktionsdiagramm (für einen speziellen Ablauf ohne Alternativen)
- Protokolldiagramm (auch: Interaktionsprotokoll; AIP; zur Beschreibung mehrerer möglicher Abläufe)
- Protokollnetze, das sind die anwendungsspezifischen Referenznetze, mit denen in Mulan und Capa das Verhalten eines Agenten implementiert wird

Wenn es kein passendes Werkzeug zur Erzeugung einer geeigneten →Prozessdarstellung gibt, kann diese auch manuell erzeugt werden. Zu diesem Zweck wurde im Abschnitt 2.1.6 beschrieben, wie aus mehreren Referenznetzen ein Kausalnetz für einen Prozess erzeugt werden kann. Im Abschnitt 3.5.3 wird dieses Vorgehen dann auf ein vereinfachtes MULAN angewendet.

Im Abschnitt 2.1.3 wurde deutlich, dass für die Darstellung eines Petrinetzprozesses das erzeugende System N, ein Kausalnetz Nk und eine Abbildung zwischen den jeweiligen Netzelementen benötigt wird. Wie sieht diese Beziehung zum erzeugenden System bei der Darstellung von Anwendungsprozessen aus? Das erzeugende System ist die laufende Anwendung und eine →Prozessdarstellung soll einen Ausschnitt des Gesamtprozesses repräsentieren.

Die →Prozessdarstellung hat eine Kausalnetz-Semantik, d.h. jede vorhandene explizite Prozessdarstellung kann in ein Kausalnetz übersetzt werden. Die Qualität der verwendeten Prozessdarstellung bestimmt, ob im Kausalnetz z.B. unabhängige Ereignisse auch unabhängig dargestellt werden. Eine explizite Prozessdarstellung ist ein Modell einer gegebenen Technik und daher endlich und diskret. Eine Prozessdarstellung repräsentiert außerdem einen einzigen Prozess. Damit ist sichergestellt, dass eine explizite Prozessdarstellung in ein Kausalnetz übertragbar ist. Die Übersetzung zwischen Interaktionsdiagrammen und Kausalnetzen wurde im Abschnitt 2.1.6 dargestellt.

Begriffsbestimmung 18 (Prozessdarstellung)
Eine explizite Prozessdarstellung D gehört zu einem Prozess P, welcher von einem System S erzeugt wurde. D ist ein Modell einer gegebenen Technik mit der Semantik eines Kausalnetzes. Falls S ein Petrinetz ist, gilt die Definition 4 auf Seite 25. Wenn nicht anders angegeben, steht →Prozessdarstellung für explizite Prozessdarstellung. Wo nötig, wird die explizite Prozessdarstellung gegen eine implizite oder aggregierte Prozessdarstellung abgegrenzt. ◇

In einer Anwendung mit Schichtenarchitektur kann eine →Prozessdarstellung möglicherweise einer Schicht zugeordnet werden. Ein Prozess in einem Software-System kann beliebig lang werden. Unendliche Prozesse können für formale Zwecke auch als Kausalnetz angegeben werden, da für Kausalnetze nicht die Endlichkeit gefordert ist. Eine →Prozessdarstellung im Sinne der obigen Begriffsbestimmung ist jedoch stets endlich, da gefordert ist, dass der dargestellte Prozess von einem System erzeugt wurde. Ansonsten handelt es sich allgemein um eine →Prozessbeschreibung.

Zusammenfassung

Rückblickend auf die vorigen Abschnitte werden die Kernpunkte hier zusammengefasst: Ein Prozess wird durch eine →Prozessbeschreibung, eine →Prozessdefinition oder eine →Prozessdarstellung repräsentiert.

Durch eine →Prozessbeschreibung werden erwünschte und wesentliche Prozesse einer Anwendung repräsentiert, z.B. um die Prozesse zu analysieren oder um die erwünschten Prozesse für die spätere Implementation zu spezifizieren.

Durch die Implementation sind alle möglichen Prozesse einer Anwendung festgelegt, deshalb wird die Implementation auch die →Prozessdefinition einer Anwendung genannt. Wenn die Implementation durch eine Architektur in Schichten oder Subsysteme strukturiert ist, so ist das Anwendungssystem entsprechend strukturiert. Ein Prozess in dem Anwendungssystem kann dann ebenfalls strukturiert betrachtet werden.

Mit einer →Prozessdarstellung wird ein bestimmter Anwendungsprozess repräsentiert. Es können nur solche Prozesse in einer →Prozessdarstellung repräsentiert werden, die tatsächlich in einer laufenden Anwendung aufgetreten sind (sonst handelt es sich um eine →Prozessbeschreibung).

Das System, welches den Gesamtprozess einer laufenden Anwendung erzeugt, besteht aus der Implementation, der gesamten IT-Infrastruktur sowie äußeren Einflüssen wie Benutzereingaben.

Die Grundlage für die Begriffe Prozess, →Prozessbeschreibung, →Prozessdefinition und →Prozessdarstellung stützt sich auf Petrinetzprozesse, wie sie im Abschnitt 2.1.3 vorgestellt wurden. Bei der Modellierung mit Petrinetzen ist die Abgrenzung zwischen den Begriffen für die Repräsentation von Prozessen in ihrem Gebrauchszusammenhang (also →Prozessbeschreibung, →Prozessdefinition und →Prozessdarstellung) nicht so scharf, weil die Elemente einer Prozessrepräsentation zu den Elementen einer anderen Prozessrepräsentation eine definierte Beziehung haben. So kann eine ausreichend vollständige Prozessbeschreibung direkt als Prozessdefinition verwendet werden, oder die Prozessdefinition ist eine Verfeinerung der Prozessbeschreibung. Eine Prozessdarstellung mit Hilfe eines Kausalnetzes bedarf einer formalen Beziehung der Transitionen und Stellen zu den Transitionen und Stellen in der Prozessdefinition. In der Agentenorientierten Software-Entwicklung können diese Beziehungen nicht so allgemein betrachtet werden. Hier ist die Prozessdarstellung im allgemeinen Fall vollständig unabhängig von der Form der Prozessdefinition.

Mit Paose wird die Technik der Petrinetze sowohl für Prozessbeschreibungen als auch Prozessdefinitionen und Prozessdarstellungen verwendet. Dadurch sind die Repräsentationen in ihrer Funktion evtl. schwierig zu unterscheiden.

Bei der Entwicklung von Agentenanwendungen ist es eine Besonderheit, nur kleine Prozessabschnitte explizit zu definieren und ein Gesamtverhalten daraus emergieren zu lassen. Ein wichtiger Aspekt ist dabei, wie man sicherstellt, dass sich nur erwünschte Prozesse ergeben können.

Ein System, welches Prozesse erzeugt, kann im Prinzip auch eine Instanz einer Prozessbeschreibung sein und zur Analyse des Verhaltens herangezogen werden. Im Rahmen dieser Arbeit wird eine Prozessbeschreibung in dem Moment, wo sie instanziiert werden soll, als eine →Prozessdefinition betrachtet.

Entsprechend der Strukturiertheit eines Systems bezieht sich jede Prozessdarstellung und jede Prozessdefinition auf eine bestimmte Schicht oder ein bestimmtes Subsystem im Anwendungssystem. In diesem Sinne ist es auch möglich, von Teilen des Anwendungsprozesses als den „agenteninternen Prozessen“ oder den „plattformübergreifenden Prozessen“ zu sprechen. Dies wird im Abschnitt 3.5 anhand eines Beispiels aufgegriffen:

Im Abschnitt 3.5 wird ein Prozess im stark vereinfachten Mulan vollständig mit Stellen und geschalteten Transitionen in allen beteiligten Referenznetzen als Kau-

salnetz dargestellt. Das Kausalnetz erreicht ein beträchtliche Größe. Als →Prozessbeschreibung auf einer wesentlich abstrakteren Ebene werden daneben die ausgetauschen Nachrichten zwischen Agenten dargestellt.

3.1.4 Prozess-Infrastruktur

Nachdem nun die Begriffe Prozess und Infrastruktur für sich betrachtet wurden, wird in diesem Abschnitt auf den Begriff Prozess-Infrastruktur eingegangen; zuerst im Allgemeinen und dann zum konkreten Verständnis für diese Arbeit in Bezug auf Agentennetze.

Eine Prozess-Infrastruktur (synonym ist die Schreibweise ohne Bindestrich: Prozessinfrastruktur) ist entsprechend der Begriffsbestimmung Infrastruktur (Nr. 13 auf Seite 66) eine verteilte Schicht in einer Software mit einer speziellen Aufgabe. Die Aufgabe der Prozess-Infrastruktur besteht darin, die Prozesse der darüberliegenden verteilten Schicht zu verwalten. Dazu umfasst die Schnittstelle der Prozess-Infrastruktur eine Schnittstelle zur Übergabe der Prozessdefinitionen der übergeordneten Schicht, setzt diese nötigenfalls in eine explizite Prozessdefinition um und instanziiert diese nach Bedarf. Für solche Prozessschritte, die von der übergeordneten Schicht bearbeitet werden müssen, gibt es ebenfalls eine Schnittstelle. Weiter stellt die Prozess-Infrastruktur der übergeordneten Schicht Informationen zum Zustand der laufenden Prozesse und geeignete Prozessdarstellungen zur Verfügung. Die Prozess-Infrastruktur stellt der übergeordneten Schicht eine Schnittstelle zur Steuerung und evtl. zur Manipulation der Prozesse (z.B. abbrechen, ändern, Teilschritte überspringen...) zur Verfügung.

Begriffsbestimmung 19 (Prozess-Infrastruktur)
Eine Prozess-Infrastruktur ist eine Infrastruktur zur Verwaltung der Prozesse einer höherliegenden Schicht in einem Software-System. ◇

Jedes Workflowmanagementsystem ist eine Prozess-Infrastruktur für die darauf aufbauende Workflowanwendung. Die in dieser Arbeit konzipierte Prozess-Infrastruktur für Agentenanwendungen „Pia“ leistet diese Art von Prozess-Unterstützung für Agentenanwendungen. Da dieser Abschnitt lediglich der Begriffsbestimmung dient, wird die konkretere Fassung einer *Prozessinfrastruktur für Agentenanwendungen* auf später verschoben (Abschnitt 3.6).

Damit endet der erste Grundlagenabschnitt mit dem Ergebnis von Begriffsbestimmungen zum Prozess, zur Infrastruktur, Prozess-Infrastruktur, zu System und Architektur.

Es ist klar, dass die Verwendung der zentralen Begriffe die Konzepte für einen Lösungsansatz stark beeinflusst. Ebenso hat die gewählte technische Grundlage einen starken Einfluss auf die Konzepte eines Lösungsansatzes, da einige Möglichkeiten klar auf der Hand liegen, während andere nur über Umwege zu realisieren sind. Die Tendenz der Lösung wird also je nach technischer Grundlage verschieden sein.

Der nächste Abschnitt beleuchtet einige Konzepte und technische Überlegungen, die einen besonderen Einfluss auf den Lösungsansatz in dieser Arbeit haben.

3.2 Zentrale Entwurfskonzepte in Bezug auf die technische Umsetzung

Die gewählte technische Umgebung zur Konzeption, Entwurf und Umsetzung eines Systems hat immer einen Einfluss auf die Art der Lösung, ja selbst auf die Art, über das Problem nachzudenken. Daniel Moldt hat in diesem Sinne die Petrinetze als ein *Denkzeug*, ein „Werkzeug" zum Nachdenken dargestellt (vgl. [Moldt 2005]). In diesem Sinne stellt dieser Abschnitt einige konzeptionelle Besonderheiten und Überlegungen in Bezug auf die technische Umsetzung vor. Zu jedem Punkt gibt es einen Text, welcher für das Verständnis der weiteren Arbeit nötig ist. Darin finden sich z.T. Verweise auf nähere Betrachtungen in Unterabschnitten oder direkt auf den entsprechenden Abschnitt im Implementierungskapitel 5.

Petrinetze zur Darstellung von Prozessen Aus der jetzigen Perspektive der vorliegenden Arbeit kann die Rolle der Petrinetze für diese Arbeit noch einmal zusammengefasst werden: Petrinetze liegen jeder *Prozessdarstellung* in Form von Kausalnetzen zugrunde. Abschnitt 3.5 verdeutlicht z.B. einen Prozess in einem einfachen Mulan. Andere Prozessdarstellungen wie die Interaktionsdiagramme werden auf Kausalnetze zurückgeführt, so dass eine einheitliche Betrachtung möglich ist (Abschnitt 2.1.6).

Petrinetze werden zur *Systemspezifikation* in den vier Ebenen von Mulan verwendet, zur Verhaltensspezifikation von Agenten in Form von Protokollnetzen sowie zur Spezifikation von Workflows in Form von Workflow-Referenznetzen. Mit Renew ist auch die *Ausführungsumgebung* auf Petrinetze ausgerichtet und ermöglicht so weitgehende Beobachtungs- und Inspektionsmöglichkeiten, welche durch integrierte Werkzeuge (Mulan-Viewer, Sniffer) ergänzt werden.

Beweise über Netzeigenschaften wie die Korrektheit von Workflownetzen und die Terminierung von Protokollnetzen werden in dieser Arbeit nicht behandelt, dazu müssten geeignete Subklassen von Referenznetzen verwendet werden.

Agenteninterne Kommunikation Das Agenten-Referenznetz in Capa koordiniert die Referenznetze Wissensbasis, Fabrik und die Protokollnetze, indem entsprechende synchrone Kanäle zur Verfügung gestellt werden. Die Protokollnetze können sich untereinander über die Wissensbasis koordinieren, oder indem der Agent sich selbst eine Nachricht schickt. In dem Fall, wo ein Agent selbst ein komplexes System repräsentiert, ist jedoch eine weitere explizite Unterstützung der agenteninternen Kommunikation nötig. Beispiele sind Agenten mit einer GUI oder Agenten, die ein Altsystem kapseln. Beide Fälle traten im Rahmen dieses Projektes auf[7]. Die Nachrichten von außen müssen in solchen Fällen übersetzt und weitergeleitet werden. Ein anderes Beispiel ist ein spezieller Lebenszyklus eines Agenten.

Ein Agent als Repräsentant eines komplexen Systemteils muss deshalb ein definiertes agenteninternes Kommunikationsrahmenwerk aufweisen. In Capa wird dies durch eine von mir eingeführte Entscheidungskomponente (DC für *Decision Component*) und einer Kanalschnittstelle im Agenten-Referenznetz dargestellt. Die Entscheidungskomponente eines Agenten ist ein Referenznetz welches (a) bidirektionale,

[7] Der User-Agent des agentenbasierten WFMS hat eine GUI. Im ersten Prototyp wurde der WFES von Thomas Jacob durch einen Agenten gekapselt

anwendungsspezifische Kommunikationskanäle zu Protokollnetzen bereitstellt, (b) Zugriff auf die Wissensbasis des Agenten hat und (c) Verbindung zu Altsystemen oder einer GUI-Komponente des Agenten herstellen kann. Die Verwendung der Schnittstelle für Entscheidungskomponenten wird im Abschnitt 3.4 im Zusammenhang mit dem Entwicklungsansatz PAOSE dargestellt, die Umsetzung der Schnittstelle für Entscheidungskomponenten wird Abschnitt 5.2.1 aufgegriffen.

Geschachtelte Strukturen Referenznetze eignen sich besonders gut zur Modellierung von geschachtelten Strukturen, indem für jedes Subnetz nur genau eine Referenz existiert und damit die Referenzsemantik und die Wertsemantik zusammenfallen. In diesem Sinne wurde MULAN entworfen.

Wenn man die grundsätzliche Idee der geschachtelten Strukturen auf Agenten überträgt, dann wird ein Agent durch mehrere interne Agenten implementiert. Der übergreifende Agent funktioniert wie eine Plattform für die internen Agenten.

Rölke in [RÖLKE 2004] beschreibt diese Möglichkeit für weitere Ebenen in MULAN, indem die Agentenplattform durch einen speziellen Agenten realisiert wird, und so ein Plattform-Agent dann sowohl als Plattform (nach Innen für seine enthaltenen Agenten) als auch als Agent (nach Außen mit eigenen Zielen, evtl. migrationsfähig) betrachtet werden kann.

Für CAPA-Agenten ist der erste Schritt in diese Richtung, die Protokollnetzinstanzen als Agenten zu betrachten, welche untereinander mittels der Entscheidungskomponente (DC) kommunizieren und die Wissensbasis als gemeinsame Ressource nutzen. Protokollnetzinstanzen können sich auch ACL-Nachrichten über den externen Kanal schicken. Benjamin Schleinzer setzt eine umfassendere Schachtelung in seiner Diplomarbeit [SCHLEINZER 2007] um.

Von der anderen Richtung kommend, kann ein Multi-Agenten-System als ein verteilt implementierter Agent betrachtet werden, entweder mit oder sogar ohne eine explizite Repräsentierung dieses einen Agenten.

Konkret wird in dieser Arbeit ein Workflowmanagementsystem-Agent (WFMS-Agent) aus mehreren spezialisierten Agenten (z.B. WFES – Workflow-Enactment-Service) zusammengesetzt. Seine Bestandteile werden als *innerhalb* des WFMS-Agenten angenommen, welcher allerdings nicht der einzige nach Außen kommunizierende Teil ist. Eine Ebene höher können mehrere WFMS-Agenten kombiniert werden und bilden so die verteilte Implementierung eines übergeordneten WFMS-Agenten. In einem Agentennetz wird dann ein Verzeichnisdienst zur dynamischen Addressierung der verteilt implementierten WFMS-Agenten benutzt.

Abschnitt 5.1.1 greift die Umsetzung mit Referenznetzen, den Aufbau eines lokalen WFMS als geschachtelte Struktur, dann den Aufbau eines verteilten WFMS als geschachtelte Struktur und schließlich die Projektion der Kommunikationswege auf lokale Kommunikation auf.

Netzwerkanbindung Die Infrastruktur, die durch ein Agentennetzwerk zur Veröffentlichung und Auffindung von Agentendiensten bereitgestellt wird, ist für ein verteiltes WFMS ein wesentlicher Bestandteil. Die Anbindung an ein existierendes Agentennetzwerk ist von Vorteil, um keine eigene Lösung zu erarbeiten, wo es schon „ge-

brauchsfertige“ Lösungen gibt. Die wesentliche Vorbedingung für die Anbindung an ein Agentennetzwerk ist der FIPA-konforme Nachrichtentransportdienst, welcher in Capa von Anfang an integraler Bestandteil ist. Capa-Agenten mittels des Plugins Ace können in Agentcities teilnehmen. Im Rahmen dieses Projektes wurde Ace erweitert, um auch an openNet teilhaben zu können. Die Anbindung wird im Abschnitt 5.2.2 erläutert. Die im Prototypen gewählte Lösung eines „zentralen DF“ (ZDF) wird ebenfalls in Abschnitt 5.2.2 vorgestellt. Der ZDF ist auf die besonderen Bedürfnisse im Informatik-Subnetz abgestimmt.

Abonnierbare Dienste Für ein verteiltes Workflowmanagementsystem ist die dynamische Adressierung ein besonders wichtiger Faktor. Dabei reicht die einmalige Anfrage nicht aus: bei einem abonnierbaren Suchdienst wird die Suche weitergeleitet und asynchron treffen Antworten ein, deren Anzahl und Zeitverzögerung gezielt eingeschränkt werden können. Allgemein können abonnierbare Dienste Informations- oder Warn-Dienste sein, von denen nach bestimmten Kriterien wie Zeitpunkt, Zeitintervall, Änderungen oder Erreichen bestimmter Zustände Nachrichten verschickt werden. Für solche Abonnements gilt, dass in einer dynamischen Umgebung grundsätzlich die Dienstanbieter ausfallen oder neue hinzukommen können. Im Falle eines Workflowmanagementsystems wird z.B. ein Benutzer jeweils aktuelle Informationen über verfügbare Workitems abonnieren.

Die Verwaltung von Abonnements umfasst allgemein vier Fälle: 1. die einfache Suche nach Anbietern (wie bei einem Verzeichnisdienst), 2. die persistente Suche (in jedem Falle asynchron zu betrachten, weil diese Suchform bis zur Kündigung für jede passende Veränderung eine neue Nachricht liefert), 3. die Vermittlung eines Abonnement-Anbieters an einen Dienstkonsumenten und 4. die Vermittlung aller Anbieter an einen Konsumenten. Der in Capa realisierte *Subscription Manager* sorgt auch für den Ersatz bei einem ausgefallenem Anbieter für dessen Konsumenten. Die Umsetzung in Capa wird in Abschnitt 5.2.3 aufgegriffen. Es handelt sich dabei um eine Ergänzung des von der FIPA spezifizierten Directory Facilitator (DF) als grundlegender Verzeichnisdienst um Funktionen eines Subscription Managers nach einem Vorschlag von MBala et al. [MBala et al. 2005].

Erzeuger und erzeugter Agent Die Beziehung zwischen zwei Agenten, von denen einer den anderen erzeugt hat, bedarf spezieller Beachtung, weil der erzeugte Agent seinem Erzeuger besonderes Vertrauen schenken soll. In Capa wird dieses nicht direkt unterstützt, deshalb wird im Rahmen dieser Arbeit ein Interaktionsmuster entwickelt, bei dem ein reaktives Initialisierungsprotokoll beim erzeugten Agenten nur genau einmal aufgerufen werden kann (vom Erzeuger). Der Absender wird dann als Erzeuger in der Wissensbasis eingetragen. Abschnitt 5.2.4 beschreibt diesen Vorgang eingehender.

Agenteninteraktionen und Workflows Agenteninteraktionen bilden anwendungsbezogene und agentenübergreifende Prozesse ab. Ein Workflow dient ebenfalls dem Zweck, einen Prozess mit mehreren beteiligten Strukturkomponenten explizit abzubilden. Diese beiden Techniken haben jedoch verschiedene Modellierungsschwerpunkte.

Ein Interaktionsprotokoll ist graphisch nach den beteiligten Rollen geordnet; die logische Reihenfolge der ausgetauschten Nachrichten ist gut erkennbar. Agenten gelten als autonom: alternative Prozesse werden als Schnittstelle beschrieben, der Grund für die jeweilige Entscheidung wird evtl. nicht dargestellt. Es ist (abgesehen von Kommentaren) im Interaktionsdiagramm auch nicht erkennbar, was die inhaltlichen Tätigkeiten der beteiligten Komponenten ausmacht und welches die Merkmale dieser Tätigkeiten sind (z.B. benötigte Informationen und Wirkungen).

Ein Workflow ist graphisch nach der Reihenfolge der Tätigkeiten geordnet; die Merkmale der Tätigkeiten sind durch die Namen und Typen der Tätigkeiten klar definiert (wenn auch nicht unbedingt im selben Diagramm). Im Workflow wird dagegen keine Kommunikation zwischen beteiligten Komponenten abgebildet, vielmehr kommunizieren die beteiligten Komponenten jeweils nur mit dem WFMS. Die Komponenten gelten nicht als autonom, die Auswahl von alternativen Prozessen liegt beim WFMS und die Entscheidungsgrundlage ist Bestandteil der Workflow-Definition.

Abschnitt 3.5.5 zeigt einige mögliche Übersetzungen zwischen einem einfachen Interaktionsprotokoll und einem Workflow auf. Auf dem Weg vom typischen Interaktionsprotokoll zum typischen Workflow ändert sich die Semantik des beschriebenen Prozesses.

Fragmentierung Petrinetze werden zunächst lokal an einem Ort ausgeführt. Für die verteilte Ausführung, z.B. für Workflows, sind gesonderte technische Lösungen notwendig. Eine Möglichkeit dazu ist die Fragmentierung. Zur Fragmentierung von Workflow-Petrinetzen existieren Vorarbeiten, die im nächsten Abschnitt beschrieben werden.

Es werden stellenberandete Fragmente eingeführt, welche in Stern-Topologie einem Steuerungs-Netz zugeordnet sind. Das Steuerungsnetz repräsentiert den übergreifenden Workflow. Konfliktsituationen werden mit Hilfe eines verteilten Sperr-Algorithmus sicher gelöst, so dass keine inkonsistenten Zustände auftreten. Die Einzelheiten zu den Anforderungen und Ergebnissen der Workflow-Fragmentierung werden nun im Folgenden dargestellt.

3.3 Fragmentierung von Workflow-Referenznetzen

Die Fragmentierung von Workflow-Referenznetzen im Hinblick auf ein agententenorientiertes Workflowmanagementsystem wurde von Timo Carl bearbeitet [Carl 2004] und in nachfolgenden Papieren veröffentlicht [Reese et al. 2005, Reese et al. 2006c].

Die folgende Darstellung gibt den Inhalt von Kapitel 3 aus [Reese et al. 2006c] wieder.

In Petrinetzen werden die Abhängigkeiten zwischen Netzelementen lokal in Form von Kanten definiert. Zur Synchronisation müssen jeweils nur lokal benachbarte Netzelemente betrachtet werden. An den Fragmentierungsmechanismus werden hier die folgenden Anforderungen gestellt:

(1) Workflowfragmente sollen abgesehen von der Synchronisation an den Rändern unabhängig voneinander sein.

(2) Die Fragmentierung soll beliebig sein, insbesondere soll ein XOR-Split möglich sein (und damit alle wesentlichen Workflow-Patterns unterstützt sein).
(3) Der Rand eines Fragments soll beliebig komplex sein, d.h. die Anzahl der Ein- und Ausgangskanten soll frei wählbar sein. Auch Schleifen sollen möglich sein.
(4) Die Semantik des verteilten Workflows soll dieselbe Semantik sein wie der ungeteilte Workflow sie hat. Insbesondere soll es nicht notwendig sein, zusätzliche Netzelemente zu zeichnen. Automatische, nicht sichtbare Verfeinerung der Randelemente eines Fragment-Netzes sind akzeptabel.

Der Rand der Fragmente

Grundsätzlich kann der Rand eines Fragments auf drei verschiedene Weisen definiert werden: geteilte Kanten, Rand-Stellen oder Rand-Transitionen. Geteilte Kanten grenzen den Entwurf und die Fragmentierung von Workflownetzen in keiner Weise ein, allerdings sind die Kosten für die Koordination ziemlich hoch: Für jede Bindungssuche müssen verteilte Synchronisation und Datenaustausch stattfinden.

Die Fragmentierung an Stellen kann durch eine automatische Stellenverfeinerung realisiert werden, in der die Synchonisation implementiert wird. Die Rand-Stelle wird so zu einer verteilten Stelle, die in jedem angrenzenden Fragment eine Kopie hat. Jede Änderung der Markierung muss unteilbar sein, um inkonsistente Zustände zu verhindern.

Die Fragmentierung an Transitionen entspricht am Ehesten der Intuition, weil die Transitionen die aktiven Elemente eines Netzes sind, an denen sowieso Datentransfer stattfindet. In diesem Falle gibt es keinen Konflikt und damit ist keine verteilte Transaktion notwendig. Sobald das Schalten initiiert ist, wird die Aktion isoliert und kann parallel zu anderen Vorgängen ausgeführt werden. Zur Realisierung kann die Rand-Transition verfeinert werden. Mit dieser Methode ist allerdings die Fragmentierung nicht beliebig, insbesondere kann ein XOR nicht an einer Rand-Transition realisiert werden. Außerdem wird im Falle von transitionsberandeten Fragmenten wie bei den geteilten Kanten für jede Bindungssuche aufwändige Kommunikation notwendig. Aus diesem Grunde werden Fragmente für Workflownetze stellenberandet modelliert.

Erzeugung der Fragmente

Der Workflow-Entwickler markiert die beabsichtigten Rand-Stellen im Workflow-Netz. Diese werden dann als Schnittpunkte verwendet, als Schnittstellen zwischen Fragmenten. Jedes Workflownetz muss genau einen Startpunkt und ein oder mehrere explizite Endpunkte aufweisen. Von diesen Fixpunkten ausgehend, kann das Workflownetz in Fragmente zerlegt werden.

Jede Rand-Stelle muss Verbindungen zu mindestens zwei verschiedenen Fragmenten aufweisen, ansonsten muss eine Warnung ausgegeben werden, da die Intention des Entwicklers, an dieser Stelle den Workflow zu teilen, nicht erfüllt werden kann.

Der nun beschriebene Algorithmus kann zur Fragmentierung und zur Konsistenzprüfung verwendet werden. Abbildung 3.3 zeigt ein Ausgangsnetz und die daraus erzeugten Fragment-Netze. Die fett berandeten Transitionen wurden im Abschnitt 2.3.4 als Task-Transitionen eingeführt, der Doppelpfeil markiert Rand-Stellen.

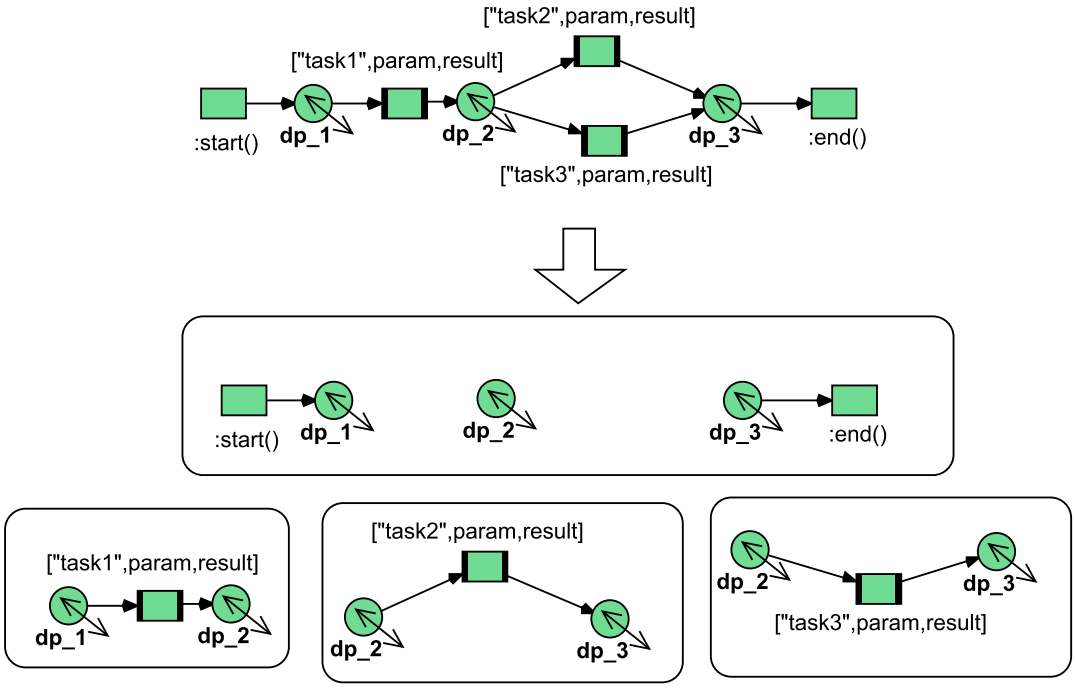


Abbildung 3.3: Beispiel für Workflow-Fragmentierung. (Neu gestaltet nach: [Reese et al. 2006c])

Algorithmus Eingabe: Ein Workflow-Petrinetz mit markierten Rand-Stellen.

(1) Transitionen mit der Anschrift `:start()` oder `:end()` dürfen keine eingehenden bzw. ausgehenden Kanten haben, ansonsten ist das Netz inkonsistent.

(2) Transitionen mit der Anschrift `:start()` oder `:end()` müssen mit genau einer Rand-Stelle verbunden sein, ansonsten ist das Netz inkonsistent.

(3) Betrachte alle Kanten als ungerichtet und alle Stellen und Transitionen als unbenannte Knoten. Benenne alle Rand-Knoten individuell (so dass Namenskonflikte vermieden werden). Vervielfache jeden Rand-Knoten entsprechend der angrenzenden Kanten und verbinde mit jeder Kante exakt einen der erzeugten Knoten. Im Beispiel werden von der Stelle `dp_1` (Abk. für *distributed place*) zwei Kopien erzeugt und von der Stelle `dp_2` werden drei Kopien erzeugt.

(4) Für jeden nicht bearbeiteten Randknoten suche nun alle durch Kanten verbundenen Randknoten. Sammle diese Listen von verbundenen Randknoten für jedes Fragment. Kommt dann ein Name nur in einer Liste vor, so stellt er keine Verbindung zwischen zwei Fragmenten dar und dieser Randknoten ist inkonsistent.

(5) Verbinde nun alle Fragmente mit `:start()` und `:end()`-Anschriften zu einem Netz und füge von jedem Randknoten eine Kopie hinzu, um das Steuerungsnetz zu erhalten.

(6) Betrachte nun wieder Transitionen und Stellen getrennt und speichere jedes Fragment in einem eigenen Netz. Verbinde die Rand-Stellen mit gleichen Namen quer über alle Netze: Markiere die erste Rand-Stelle im Steuerungsnetz und setze in

jedes damit verbundene Fragment (am gemeinsamen Namen der Stelle erkennbar) eine synchronisierte Kopie der Stelle.

Ergebnis: Eine Fehlermeldung, falls das Netz inkonsistent ist, ansonsten disjunkte Fragmente (abgesehen von Rand-Stellen), welche zusammengenommen das Ausgangsnetz ergeben, plus das Steuerungsnetz. Das Steuerungsnetz enthält die Start- und Endpunkte des Workflows, alle Rand-Stellen und ihre Koordination. Sobald eine der Randstellen markiert wird, werden alle betroffenen Fragmente aktiviert (das sind alle Fragmente, welche eine mit der Stelle verbundene Eingangs- oder Testkante haben). Sobald eine Ende-Transition des Worfklows im Steuerungsnetz aktiviert ist, wird der Workflow als bearbeitet angesehen.

Aktivierung von Fragmenten und Terminierung des Workflows

Im Allgemeinen terminiert ein Workflow, sobald ein Endknoten erreicht ist.

Bezüglich der Aktivierung gilt: eine Transition ist aktiviert, wenn ihre Vorbedingungen (Markierung, Typen, Wächter) erfüllt sind, ein Petrinetz ist aktiviert, wenn mindestens eine Transition aktiviert ist. Weiterhin ist eine Netzinstanz in Renew passiv, wenn sie nicht mehr referenziert wird, keine Transition gerade schaltet und keine Transition aktiviert ist. In diesem Falle wird die Netzinstanz vom Garbage Collector entfernt, sobald der Speicher anderweitig benötigt wird.

Im verteilten Fall, den wir hier betrachten, können die Referenzen prinzipiell nicht so leicht verfolgt werden. Anders als im lokalen Fall muss eine Netzinstanz explizit (z.B. mit einer speziellen Stop-Transition) angehalten werden. Deshalb wird ein Workflow-Fragment explizit aktiviert, sobald eine der Rand-Stellen zum ersten Mal markiert wird. Das Fragment wird erst dann de-aktiviert, wenn im Kontrollnetz die Ende-Transition schaltet. Vom Kontrollnetz her werden die Fragmente benachrichtigt, so dass sie ihre eigene Ende-Transition schalten.

Aus diesem Grund müssen Workflownetze und Fragmente so gestaltet sein, dass nach dem Schalten der Ende-Transition keine andere Transition mehr schaltet. Diese Anforderung ist Teil der Korrektheit (engl. soundness) von Workflows. Möglicherweise ist diese Eigenschaft für generierte Workflowfragmente auch nachweisbar, was an dieser Stelle nicht weiter betrachtet wird.

3.4 Entwurfsansatz: PAOSE

Das anvisierte System mit Workflowtechnologie zur Unterstützung von übergreifenden Prozessen in Agentenanwendungen weist eine hohe Komplexität auf. Um diese Komplexität beim Entwurf handhabbar zu machen, ist ein Entwurfsansatz notwendig. In dieser Arbeit wird der Entwurf mit dem Ansatz Paose (Petrinetzbasierte agentenorientierte Software-Entwicklung) durchgeführt. Gleichzeitig stammen einige Anstöße für die in dieser Arbeit entwickelten Ideen aus der stetigen Verwendung, Verfeinerung, Klärung und schrittweise weiterentwickelten Werkzeugunterstützung des Paose-Ansatzes: Insbesondere die Matrixorganisation des Entwicklerteams findet in der Vision von integrierter Agenten- und Prozesssicht ihre Fortsetzung.

Die folgenden Abschnitte skizzieren die wichtigsten Elemente von Paose. Zuerst werden die minimal notwendigen Bestandteile für die Anwendungsentwicklung mit

CAPA im Entwurf und in der Implementation beschrieben: Wissensbasis, Protokollnetze, Entscheidungskomponenten und Startskript. Im zweiten Abschnitt wird die im PAOSE-Ansatz wesentliche Organisation des Entwicklerteams in Form einer Matrix-Struktur vorgestellt. Der dritte Abschnitt stellt die verwendeten Diagrammtypen für den Entwurf und die Implementierung von Anwendungen mit PAOSE vor.

Im PAOSE-Ansatz werden vor allem die folgenden Diagrammtypen verwendet: Use-Case-Diagramme und Agenten-Interaktionsprotokolldiagramme (AIP) aus (A)UML, außerdem Multi-Agenten-Diagramme, die auf Klassendiagrammen der UML basieren. Alle Diagrammtypen werden durch Plugins für RENEW unterstützt, wobei das AIP-Plugin und das Multi-Agenten-Diagramm-Plugin auch Code generieren können.

3.4.1 Anwendungsentwicklung mit Capa

Zu den hier angegebenen Elementen bei der Anwendungsentwicklung mit CAPA werden weder die verwendeten Werkzeuge noch das Vorgehen geschildert: Diese wichtigen Aspekte werden in den hierauf folgenden Abschnitten vorgestellt. Hier soll ein Überblick über das Rahmenwerk CAPA zur Anwendungsentwicklung gegeben werden.

Entwurf

Die minimal notwendigen Elemente für den Entwurf einer Anwendung mit CAPA sind die Folgenden: Um mit CAPA eine Anwendung zu implementieren, wird die Anwendung in Agentenrollen (oder Agententypen; im Folgenden kurz: Agenten) gegliedert. Jeder Agent bekommt Dienstbeschreibungen zugeordnet, um die speziellen Aufgaben und Abhängigkeiten eines Agenten darzustellen. Es werden weiterhin die Interaktionen entworfen sowie eine Ontologie (ein Begriffsgerüst) für die Anwendung.

Zusammenspiel der Elemente

Das Agenten-Referenznetz von CAPA koordiniert die Bestandteile eines Agenten.

Die allgemeine Form von MULAN mit seinen vier Ebenen zur Strukturierung von Multi-Agenten-Systemen (System, Plattform, Agent und Protokollnetz) wurde bereits im Abschnitt 2.2.4 vorgestellt. Die Entscheidungskomponente fehlt darin; sie wurde im Abschnitt 3.2 unter dem Stichwort „Agenteninterne Kommunikation“ als eine notwendige Ergänzung beschrieben. Die Umsetzung der Entscheidungskomponente wird später im Abschnitt 5.2.1 anhand eines graphisch überarbeiteten Agentennetzes (Abbildung 5.3 auf Seite 155) im Einzelnen vorgestellt.

Das stark abstrahierte Agentennetz aus Abbildung 2.7 auf Seite 36 ist nun in Abbildung 3.4 auf der nächsten Seite ausführlicher dargestellt und wird im folgenden Text in den wesentlichen Elementen erläutert.

Implementation

Die minimal notwendigen Elemente für die Implementation einer Anwendung mit CAPA sind die Folgenden:

- Wissensbasen: Für jeden Agenten wird eine initiale Wissensbasis in Form einer Konfigurationsdatei (wis-Format oder xml-Format) angegeben. Darin sind

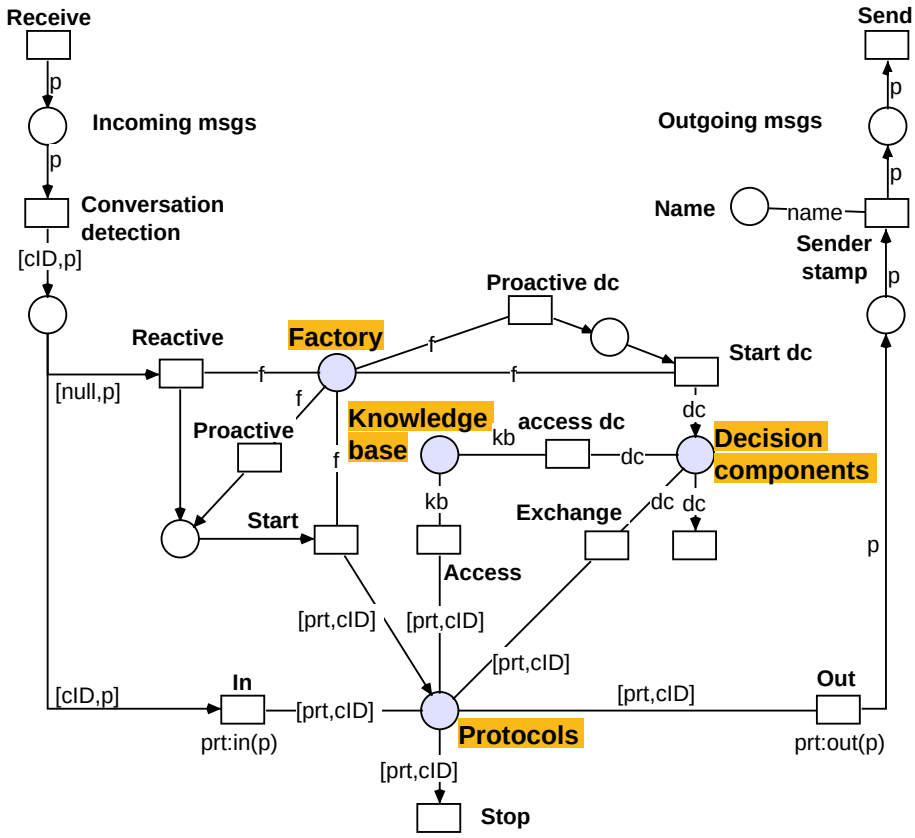


Abbildung 3.4: Agentennetz. Aus: [Cabac et al. 2007a]

die Dienstbeschreibungen, das reaktive und proaktive Verhalten und initiales Wissen konfiguriert. Zur Laufzeit liegt die aktuelle Wissensbasis in der Stelle „Knowledge base“.

- Protokollnetze: Das Verhalten des Agenten und die Verwendung der Wissensbasis werden durch eine Reihe von Protokollnetzen in Form von Referenznetzen implementiert. Die Instanzen der aktuell aktiven Protokollnetze liegen in der Stelle „Protocols“.

- Entscheidungskomponenten: Entscheidungskomponenten implementieren, wie die Protokollnetze auch, anwendungsspezifisches Verhalten. Eine Netzinstanz einer Entscheidungskomponente kann von Protokollnetzinstanzen angefragt werden, um zu den ansonsten statischen und workflow-artigen Eigenschaften der Protokollnetze zusätzliche Flexibilität hinzuzufügen. Entscheidungskomponenten können auch proaktives Verhalten auslösen, indem sie die Fabrik veranlassen, ein Protokollnetz zu instanziieren. Auf diese Weise kann eine Planungskomponente im Sinne der KI als Entscheidungskomponente in einen Agenten eingebunden werden. Eine Entscheidungskomponente kann auch externe Werkzeuge,

Programme oder alten Code, sowie eine GUI kapseln. Die externe Rückmeldung wird dann in proaktives Agentenverhalten umgesetzt.

Die zur Laufzeit aktuell instanziierten Entscheidungskomponenten liegen in der Stelle „Decision components“.

- Ontologie: Die Ontologie liegt in Form von Java-Klassen vor. Die Java-Klassen bieten Hilfsmethoden zum Umgang mit Nachrichten. Die Ontologie ist im Agentennetz nicht abgebildet. Die Java-Klassen können für die Implementierung der initialen Wissensbasis, für die Protokollnetze und für die Entscheidungskomponenten verwendet werden.

- Graphische Benutzungsoberfläche: Falls die Anwendung GUI-Bestandteile hat, werden diese Bestandteile in Java implementiert und über eine generische Entscheidungskomponente an einen Agenten gekoppelt. Die GUI ist im Agentennetz nicht sichtbar, die entsprechende Entscheidungskomponente liegt zur Laufzeit in der Stelle „Decision components“.

- Anwendungsstart: Beim Start einer Anwendung wird der Java-Klassenpfad und der RENEW-Netzpfad konfiguriert sowie diverse Parameter für CAPA und Unterplugins von CAPA. Der Anwendungsstart kann in unterschiedlicher Weise implementiert sein: in einem Startskript (als Shellskript), in einem Referenznetz, in Form einer speziellen Multi-Agenten-System-Konfiguration, die aus dem Werkzeug KBE gestartet wird. Eine Anwendung kann aber auch über Menüeinträge von CAPA in der RENEW-GUI gestartet werden. Der Anwendungsstart ist im Agentennetz nicht sichtbar. Sobald ein Agent gestartet wird, kann der Agent aufgrund von Einträgen in seiner Wissensbasis auf der Stelle „Knowledge base“ eine Entscheidungskomponente instanziieren, eine proaktive Aktion auslösen oder auf ankommende Nachrichten reagieren. In allen Fällen wird dazu die Fabrik in der Stelle „Factory“ aktiv. Die Fabrik ist für jeden Agenten gleich und muss nicht für eine spezielle Anwendung implementiert werden.

  Zum Anwendungsstart gehört auch die Realisierung des Deployments. Dazu wird für jeden logischen Ort eine CAPA-Plattform und entsprechende Agenten gestartet.

3.4.2 Organisation des Entwicklerteams

Die Organisation des Entwicklerteams hat im PAOSE-Ansatz eine besondere Bedeutung: Das Entwicklerteam wird selbst als Multi-Agenten-System verstanden. Dieses Leitbild bestimmt die Ausrichtung, Gewohnheiten, Vorgehensweisen und Begrifflichkeiten in der Teamarbeit, indem es das Augenmerk auf eigenständige Teammitglieder mit ihren Zielen und Lösungsstrategien in Beziehung zum Projektziel setzt. Das Leitbild wird in Form einer Matrixorganisation auf ein konkretes Entwicklerteam angewendet, indem Agenten und Interaktionen von unabhängigen Teams entwickelt werden. Die Kommunikation zwischen den Teams wird mit Versionskontrolle und einer Web-basierten Plattform unterstützt. Die folgenden Unterabschnitte führen weitere Einzelheiten dazu an.

Ein Multi-Agenten-System von Entwicklern

Die Entwickler, ihre Teams und Handlungen werden hier mit Begriffen beschrieben, die normalerweise für intelligente Agenten, Multi-Agenten-Systeme oder kooperative Workflows genutzt werden. Damit wird ihre metaphorische Aussagekraft genutzt.

Die Mitglieder eines Teams handeln selbstorganisiert, autonom, unabhängig und kooperativ. Sie haben individuelle Ziele, die zur übergreifenden Vision des entwickelten Systems kulminieren. Wie Agenten in einem Multi-Agenten-System sind die Entwickler in eine Umgebung eingebettet, welche die Kommunikation unterstützt, gewisse Dienste anbietet oder auch die Handlungsmöglichkeiten einschränkt.

Ein besonders wichtiger Aspekt ist die selbstverantwortliche und dynamische Bildung von Teams. Kleine Teams können sich immer wieder neu formieren, so ergibt sich eine natürliche Sicht auf nebenläufiges und verteiltes Arbeiten.

Das Projektmanagement und organisatorische Angelegenheiten werden gemäß des Leitbildes vom Multi-Agenten-System von Entwicklern mit Eigenschaften sozialer Agenten beschrieben: soziale Normen, Konventionen und Motivation werden wichtige Einflussgrößen in den Verhaltensmustern eines Teams.

Zunächst wirkt es befremdlich, das Konzept eines Multi-Agenten-Systems zur Organisation von Software-Systemen nach (sozialen) Organisationen zurück zu übertragen auf die organisatorische Struktur von Menschen. Das Bild eines Multi-Agenten-Systems ist jedoch in den letzten Jahren so stark geworden, dass viele Entwickler mit diesen Begriffen und Schlüsselelementen der Agentenorientierung flüssig hantieren. Dadurch wird das Multi-Agenten-System ein sinnvolles Leitbild. Außerdem wird im Rahmen der agentenorientierten Entwicklung ein zusätzlicher Vorteil erreicht, indem die Struktur des entwickelten Systems direkt mit der organisatorischen Struktur des Entwicklerteams übereinstimmt.

Matrixorganisation eines konkreten Entwicklerteams

Um die Verantwortlichkeiten im Entwicklungsprojekt einzelnen Entwicklern oder Teams zuzuordnen, wird das Leitbild eines Multi-Agenten-Systems von Entwicklern verfeinert angewendet: In einem Multi-Agenten-System sind die wesentlichen Perspektiven die Agentenstruktur, das Verhalten und die Terminologie. Diese Perspektiven sind orthogonal zueinander mit einigen Verbindungspunkten. Die *Agentenstruktur* eines Multi-Agenten-Systems wird durch die Agenten, ihre Rollen, Wissensbasen und Entscheidungskomponenten definiert. Das *Verhalten* eines Multi-Agenten-Systems wird durch die Interaktionen, die kommunikativen Akte und die einer Interaktion zugeordneten agenteninternen Handlungen definiert. Die *Terminologie* eines Multi-Agenten-Systems wird durch eine anwendungsspezifische Ontologie beschrieben, wodurch allen Agenten und Interaktionen dieselben Objekte, Aktionen und Fakten darstellbar sind. Ohne globale Ontologie ist eine Interaktion nicht möglich.

Die schematische zweidimensionale Matrix in Abbildung 3.5 stellt die Unabhängigkeit und die Verbindungspunkte von Agenten und Interaktionen dar. Es gibt weder zwischen zwei beliebigen Agenten noch zwischen zwei Interaktionen eine direkte Verbindung, also werden diese unabhängigen Elemente parallel zueinander dargestellt. Jeder Agent ist in einigen Interaktionen involviert und jede Interaktion hat einige be-

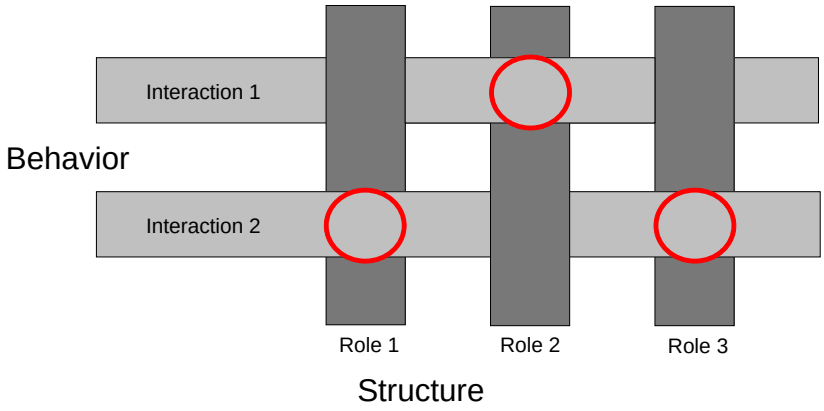


Abbildung 3.5: Zweidimensionale Matrix mit Agenten und Verhalten.
Aus: [CABAC et al. 2007a]

teiligte Agenten, also werden Agenten und Interaktionen orthogonal dargestellt. Mit Kreisen gekennzeichnete Kreuzungspunkte markieren eine Kopplung zwischen Agent und Interaktion. Die Ontologie als dritte Dimension wird im Bild nicht dargestellt.

Da nun diese drei Perspektiven orthogonal zueinander sind und innerhalb einer Perspektive die Elemente unabhängig voneinander sind, kann der Gesamtentwurf auch in unabhängige Entwurfsaufgaben aufgeteilt werden, die dann unabhängigen Teams zugeordnet werden. Nur an den kritischen Stellen (Kreise) müssen die Teams sich koordinieren. Auf diese Weise können die verschiedenen Systemteile unabhängig und nebenläufig entwickelt werden.

Die Erfahrung aus der Praxis hat gezeigt, dass es nicht sinnvoll ist, einem Team Aufgaben aus verschiedenen Perspektiven zu übergeben, weil dann die Verantwortlichkeiten und die Schnittstellen zwischen den Perspektiven unklar werden. Es ist andererseits sehr hilfreich, auf Ähnlichkeiten zwischen unabhängigen Elementen derselben Perspektive zu achten wie z.B. mehrere ähnliche Interaktionen. In dieser Situation kann ein Team die Verantwortung für mehrere parallele Elemente übernehmen und Code und Entwürfe mehrmals verwenden.

Kommunikation und Koordination

Dieser Ansatz wird nun zu einem iterativen und nebenläufigen Entwicklungsprozess verfeinert. Dieser Prozess hat sechs wesentliche Aufgabentypen: (1) Anforderungsanalyse, (2) der (Grob-) Entwurf der Ontologie, der Rollen und Interaktionen, dann die nebenläufige und hochgradig interaktive Implementierung der (3) Ontologie, (4) der Agenten und (5) der Interaktionen und schließlich (6) eine intensive und nebenläufige Integrations- und Testaufgabe. Dieser Prozess wird iteriert, aber auch einzelne Aufgaben können iteriert werden. Eine einzelne Aufgabe besteht z.B. in der Entwicklung einer speziellen Agentenrolle.

Das Vorgehen bei der Anforderungsanalyse wird an dieser Stelle nicht eingehender beschrieben. Der Grobentwurf mündet in einer Reihe von unabhängigen Aufgaben

für Interaktionen, Agenten und Ontologie-Implementierung. Die Synchronisation in Form von Kommunikation zwischen Gruppen wird durch Besprechungen, durch eine Kommunikationsplattform und eine gemeinsame Code-Basis unterstützt. Um jeweils zum Ende eines Entwicklungszyklus einen Meilenstein und ein lauffähiges System zu erhalten, braucht es eine intensive Integrations- und Testphase. Jede dieser Aufgaben umfasst wieder einen strukturierten Prozess.

3.4.3 Modelldiagramme und Werkzeuge

Aus jeder Aufgabe entstehen Modelldiagramme, die in jeder Iteration verfeinert und vervollständigt werden. Die folgenden Abschnitte beschreiben diese Modelldiagramme anhand von Ausschnitten aus Diagrammen, die aus der Entwicklung des agentenbasierten Workflowmanagementsystems für die vorliegende Arbeit stammen. Der inhaltliche Aspekt dieser Diagramme wird erst in Abschnitt 5.3 aufgegriffen.

Grobentwurf

Der Grobentwurf findet hauptsächlich in offenen Diskussionen statt, die Ergebnisse werden in Listen von Systemkomponenten und Interaktionen festgehalten. Diese werden dann in einem Use-Case-Diagramm gesammelt.

Das Use-Case-Diagramm ist hier besonders nützlich, weil die Agenten als Akteure dargestellt werden. Die Benutzer werden durch Proxy-Agenten modelliert. Normalerweise repräsentieren die Akteure im Use-Case-Diagramm Benutzer aus der realen Welt.

Abbildung 3.6 zeigt die Agenten Account Manager (AM), Workflow-Datenbank (WFDB), Workflowmanagementsystem (WFMS) und den Benutzer-Agenten (User) mit einigen Interaktionen, die hier nicht im Einzelnen betrachtet werden. Schon hier wird eine zweidimensionale Matrixstruktur deutlich: Agenten und Agentenrollen ste-

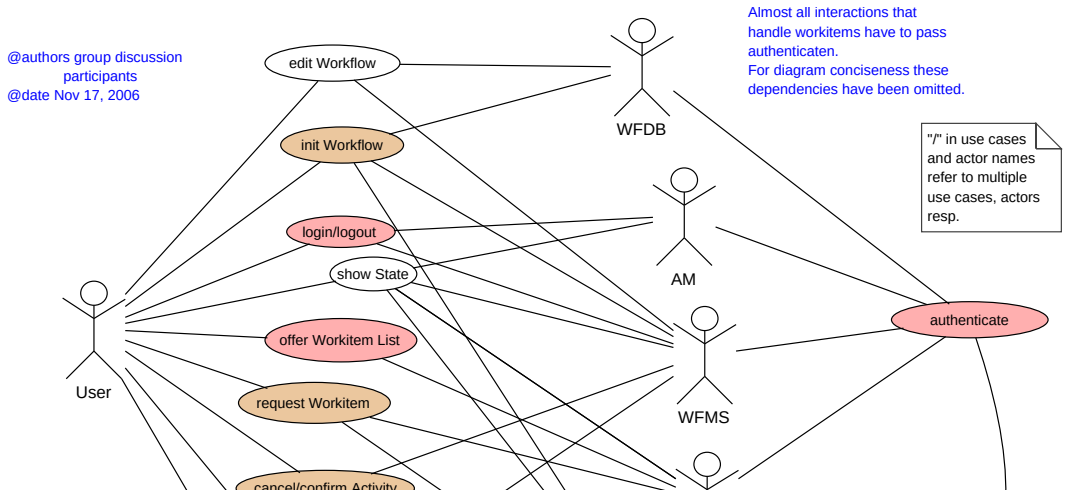


Abbildung 3.6: Fragment eines Use-Case-Diagramms mit dem Grobentwurf.
Aus: [Cabac et al. 2007a]

hen für die Struktur der Anwendung; die Interaktionen stehen für das Verhalten; und die Benutzt-Beziehungen bezeichnen die Verbindungspunkte in der Matrix.

Use-Case-Diagramme werden direkt in RENEW gezeichnet; ein Plugin stellt die nötigen Zeichenwerkzeuge bereit.

Anwendungsstruktur

Die Anwendungsstruktur wird mit einem Multi-Agenten-Diagramm nach und nach während der weiteren Entwicklung verfeinert. Zunächst werden nur einige Agentenrollen eingetragen, später kommen Verfeinerungen und Verallgemeinerungen hinzu sowie die von den Agenten angebotenen Dienste. Für jeden Nachrichtentyp, auf den eine Agentenrolle reagieren soll, wird ein Dienst eingerichtet.

Die Grundlage für das Multi-Agenten-Diagramm ist ein UML-Klassendiagramm, wobei sowohl Agentenrollen als auch Dienste als Knoten modelliert werden. Die Vererbungsbeziehungen werden zwischen Agentenrollen verwendet und Benutztbeziehungen werden zwischen Agentenrollen und Diensten verwendet. Es wird außerdem ein neuer Kantentyp für das Anbieten eines Dienstes eingeführt.

Der Diagramm-Ausschnitt in Abbildung 3.7 zeigt die Elemente, die in einem Multi-Agenten-Diagramm verwendet werden. Zu sehen sind mehrere Rollen, die mit dem Bezeichner `«AgentRole»` gekennzeichnet sind. Gelb hinterlegt sind „Super-Rollen“: von CAPAAgent erben die Rollen AuthenticationNeeder, AccountManager und WFEngi-

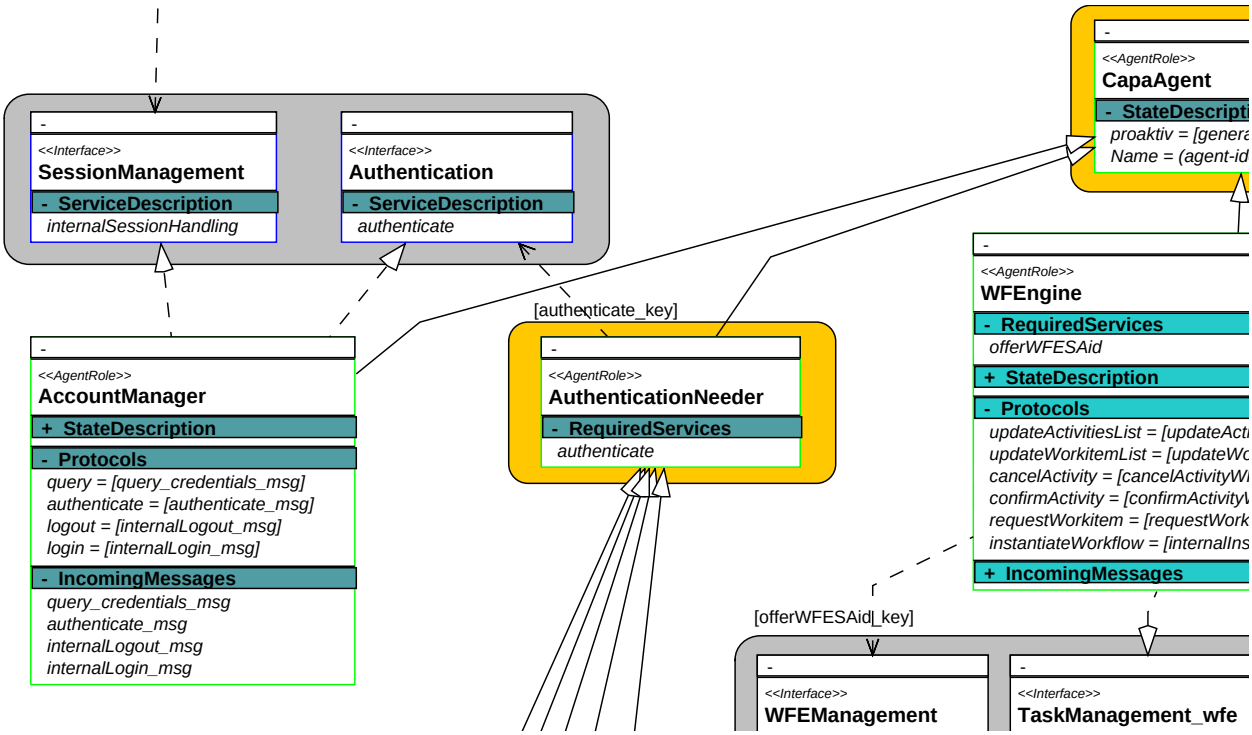


Abbildung 3.7: Fragment eines Multi-Agenten-Diagramms mit Agenten, Rollen und Diensten. Aus: [CABAC et al. 2007a] (Bildausschnitt geändert)

ne. Weiter sind einige Dienst-Schnittstellen dargestellt (mit «Interface» gekennzeichnet): SessionManagement, Authentication sowie die unvollständig dargestellten Dienst-Schnittstellen WFEManagement und TaskManagement_wfe. So wird zum Beispiel die Dienst-Schnittstelle Authentication vom AccountManager angeboten und von jedem Agenten genutzt, der die Rolle AuthenticationNeeder innehat.

Dienst-Schnittstellen haben einen Namen und eine *ServiceDescription* mit einem oder mehreren Einträgen. Rollen haben einen Namen und beliebig viele Einträge in den Kategorien *RequiredServices*, *StateDescription*, *Protocols* und *IncomingMessages*. Die Rolle AuthenticationNeeder nutzt den Dienst *authenticate*, der Teil der Dienst-Schnittstelle Authentication ist. In der StateDescription sind die zur Initialisierung der Wissensbasis nötigen Informationen eingetragen. Aufgrund dieser Informationen werden beim Start eines Agenten auch die Objekte der Wissensbasis angelegt und proaktiv gestartete Protokolle festgelegt. In der Kategorie IncomingMessages sind Nachrichtenmuster hinterlegt, auf die ein Agent mit dieser Rolle sinnvoll reagieren kann. Die Nachrichtenmuster können sehr detailliert sein oder sehr allgemein, wenn der Agent mit dieser Rolle auf sehr viele verschiedene Nachrichten mit einem bestimmten Protokoll reagieren soll. In der Kategorie Protocols wird jedes Nachrichtenmuster einem Protokoll zugeordnet. Später wird jeweils das Protokoll zum detailliertesten passenden Nachrichtenmuster instanziiert, wenn eine Nachricht auf mehrere Nachrichtenmuster passt.

Für die Implementierung der Agentenanwendung werden aus dem Multi-Agenten-Diagramm für jede Agentenrolle Beschreibungen generiert, welche als Bestandteile der initialen Wissensbasis von konkreten Agenten verwendet werden. Die Rollen werden mit Hilfe des Werkzeugs KBE (für *Knowledge Base Editor*) zu Agenten zusammengestellt. Die Agentenanwendung kann kann entweder direkt aus dem Wissensbasis-Editor heraus gestartet werden oder mithilfe eines Startskriptes, aber auch mit einem Startnetz.

Terminologie einer Anwendung

Die Terminologie einer Anwendung wird einerseits in Form einer Ontologiedefinition für Agentenkommunikation verwendet und andererseits zum Austausch im Entwicklerteam.

Zur Definition der Ontologie wird Protégé[8] verwendet, ein frei verfügbares Ontologiewerkzeug. Die Definitionen werden dann mit einem für PAOSE entwickelten Code-Generator in Java-Klassen übersetzt.

Die Hürde, ein externes Programm zur Referenz zu benutzen, ist groß, und so werden im laufenden Projekt meistens die generierten Java-Klassen herangezogen, statt Protégé zu benutzen. Dadurch kann ein Mangel an Überblick entstehen. Durch die Modellierung der Ontologie als Diagramm konnte das teilweise ausgeglichen werden. Dazu wird das Feature-Structure-Plugin von RENEW verwendet. Modelliert wird ein Klassendiagramm mit Vererbungs- und Assoziations-Beziehungen. Das Ergebnis ist ein schneller Überblick über alle Ontologie-Konzepte, allerdings gehen die Kommentare der einzelnen Konzepte und deren Attribute dabei verloren. Zwischen Protégé und dem Ontologiediagramm gibt es (noch) keine automatische Übersetzung.

[8] Protégé: http://protege.stanford.edu/

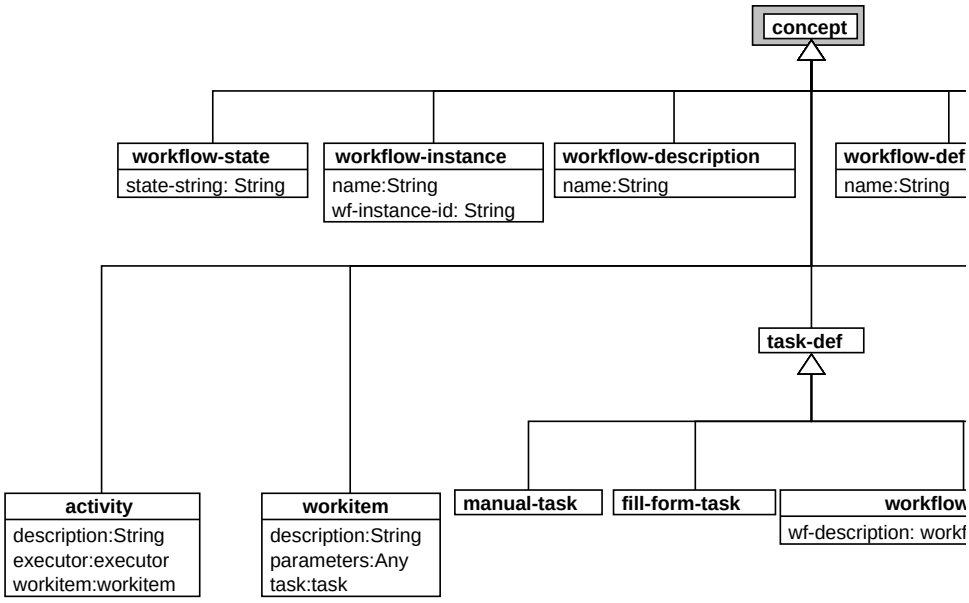


Abbildung 3.8: Fragment der WFMS-Ontologie. Aus: [CABAC et al. 2007a]

Von der Oberklasse `Thing` erben alle Konzepte der Ontologie. Zunächst werden die Gegenstände der Anwendungsdomäne so unabhängig wie möglich voneinander modelliert. Ein Gegenstand hat dabei Attribute, deren Auswahl davon abhängt, in welcher Art von Interaktion das Konzept verwendet werden soll, deshalb können die Feinheiten der Ontologie erst gemeinsam mit den Interaktionen entwickelt werden. Die Attribute werden als Slots modelliert, deren Typen entweder einfache Datentypen sind oder ebenfalls als Konzepte in der Ontologie definiert werden.

Ein spezielles Konzept ist die `Agent-Action`. Für jede Interaktion, die eine Request-Nachricht verwendet, wird ein Unterkonzept von `Agent-Action` in der Ontologie definiert.

Eine weitere Gruppe von Konzepten bilden die Prädikate einer Ontologie, welche bestimmte Konzepte zueinander in Beziehung setzen. Bei der Generierung von Java-Klassen wird also nach drei Gruppen unterschieden: einfache Konzepte, `Agent-Actions` und Prädikate. Jede Gruppe wird mit passenden Hilfsmethoden zur Erzeugung und zum Parsen von Nachrichten versehen.

Entwicklung nach Verhalten

Das Verhalten einer Multi-Agenten-Anwendung wird mit Agenteninteraktionsprotokollen spezifiziert, wie sie im Abschnitt 2.1.2 eingeführt wurden. Abbildung 3.9 zeigt einen Ausschnitt mit zwei Rollen in der Authentifikations-Interaktion. Die Interaktionsprotokolle bilden einen wesentlichen Schritt beim Entwurf der Agentenanwendung und liefern Input für die Verfeinerung der Ontologie und des Multi-Agenten-Diagramms. Deshalb ist besonders an dieser Stelle eine ausführliche Dokumentation der Ideen und Entscheidungen der Entwickler in Form von Kommentaren nötig. Die

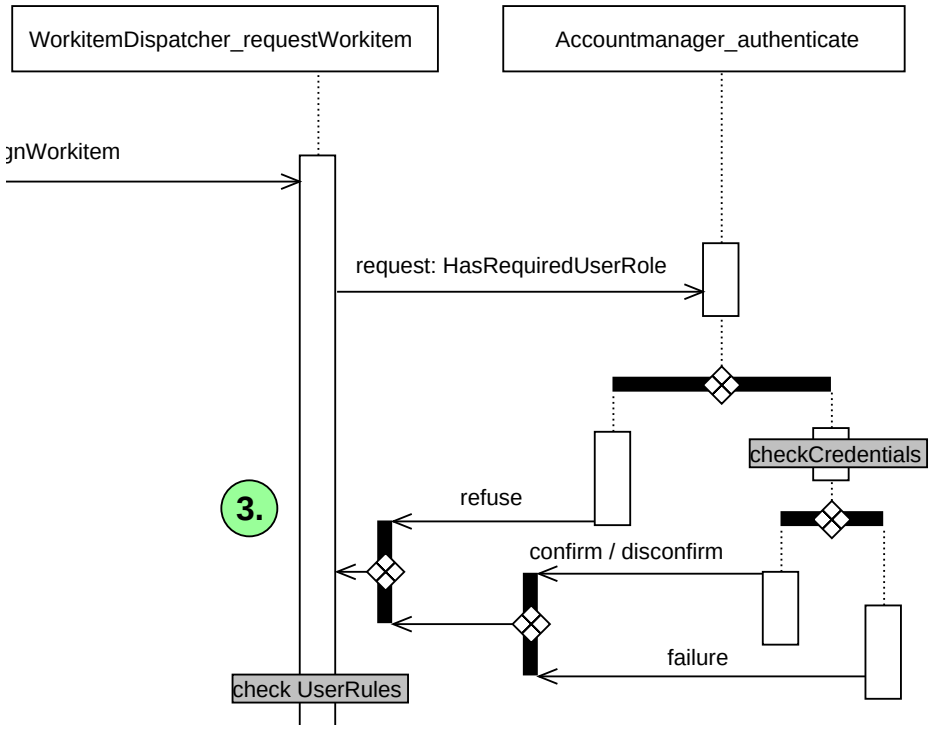


Abbildung 3.9: Fragment eines Interaktionsdiagramms. Aus: [Cabac et al. 2007a]

Kommentare werden direkt in das Diagramm notiert oder als Netzdokumentation im Latex-Format mit dem Werkzeug netdoc an die Datei angehängt.

Mit Hilfe eines Plugins werden die Diagramme in Renew erstellt und bilden dann den Ausgangspunkt für die automatische Generierung von lauffähigen Netzstrukturen für Protokollnetze unter Verwendung von Netzkomponenten. Diese Netzstrukturen werden während der Implementation mit Anschriften vervollständigt. Ein fertiges Protokollnetz ist in Abbildung 3.10 dargestellt. Es ist so skaliert, dass nur noch die Netzkomponenten erkennbar sind, wodurch deren Wert für die Strukturierung großer Netze deutlich wird. In dem Protokollnetz werden nach dem Empfang einer Request-Nachricht mehrere Fälle unterschieden, um schließlich die passende Antwort zurück zu schicken.

Entwicklung nach Agenten

Das Faktenwissen eines Agenten ist in seiner Wissensbasis gespeichert. Das Anfangswissen wird durch die initiale Wissensbasis in einer Datei gepeichert, welche aus mehreren Rollenbeschreibungen zusammengesetzt ist. Diese XML-Datei kann zwar zusätzlich angepasst werden, es ist aber vorzuziehen, diese Einträge in Form einer Rollenbeschreibung in das Multi-Agenten-Diagramm einzutragen, damit die Code-Generierung funktioniert. Die generierte Wissensbasis wird schließlich zur Initialisierung des Agenten verwendet. Es gibt in Capa auch noch ein älteres einfaches Dateiformat mit Schlüssel-Wert-Paaren.

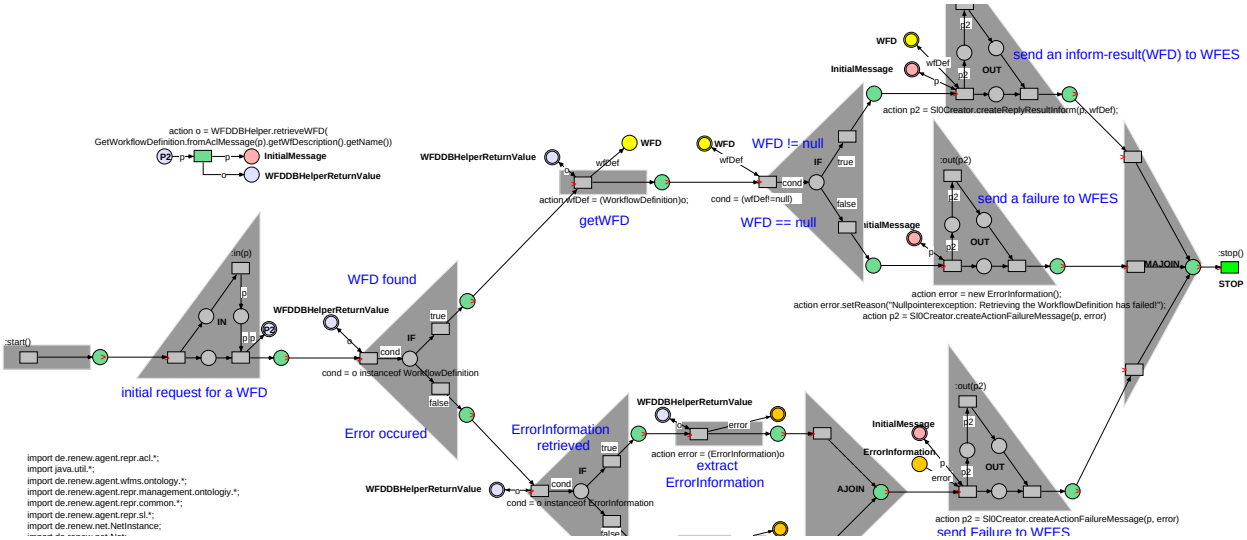


Abbildung 3.10: Fragment eines Protokollnetzes, welches mit Netzkomponenten aus einem Interaktionsprotokolldiagramm erzeugt wurde und dann manuell bearbeitet wurde. Aus: [CABAC et al. 2007a].

Die Entscheidungskomponenten werden als Referenznetze implementiert. Es stehen Netzkomponenten zur Verfügung sowie eine allgemein verwendbare Entscheidungskomponente für den Standard-Agenten in CAPA, über die eine GUI angebunden werden kann.

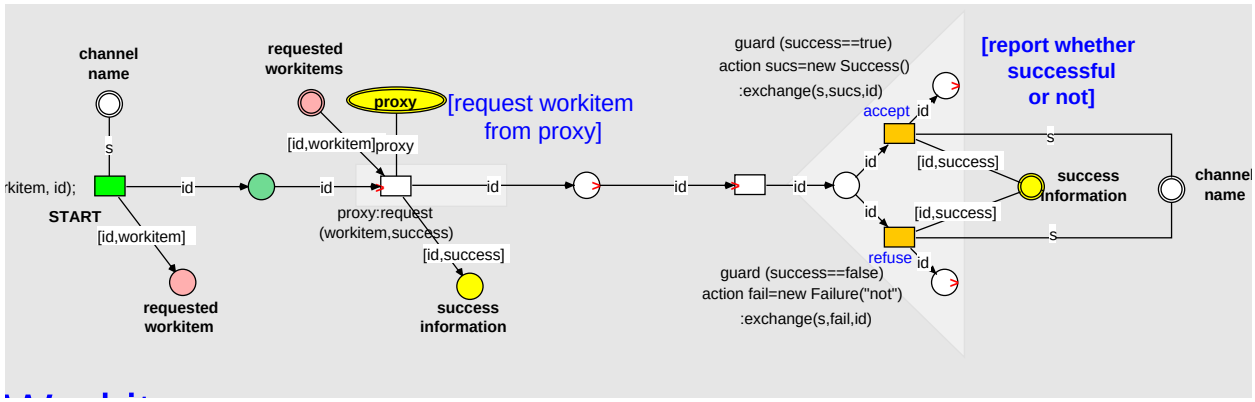


Abbildung 3.11: Fragment einer Entscheidungskomponente: RequestWorkitemHandling. Aus: [CABAC et al. 2007a]

Abbildung 3.11 zeigt einen Ausschnitt einer Entscheidungskomponente, in der die internen Details bei der Anforderung eines Workitems behandelt werden. Das Netz kapselt hier dauerhaft einen Proxy, der die Schnittstelle zur Workflowengine implementiert. Ein Request wird dem Proxy übergeben und die Antwort wird dem aufrufenden Protokollnetz zurück gegeben, welches wiederum die passende Antwort an den Benutzer schickt. Die genauen Code-Bestandteile werden hier nicht betrachtet, wie schon in der vorigen Abbildung.

3.5 Ein Prozess in Mulan

Im vorigen Abschnitt wurde der PAOSE-Entwicklungsansatz beschrieben. Ein besonders wichtiger Punkt ist dabei, dass die agentenübergreifenden Prozesse in Form von Interaktionsprotokollen entwickelt werden. So steht zur Entwurfszeit die prozessorientierte Anwendungsentwicklung in Form von Interaktionen auf gleicher Stufe mit der strukturorientierten Anwendungsentwicklung in Form von Agenten. Im Gegensatz zu dieser Entwurfszeit-Perspektive betrachtet dieser Abschnitt einen Anwendungsprozess in MULAN zur Laufzeit.

Zunächst wird an einem Beispiel verdeutlicht, wie man sich einen Prozess einer Agentenanwendung in MULAN vorstellen kann („Gesamtprozess“ gemäß Definition 17 auf Seite 73). Dazu wird hier ein stark vereinfachtes, aber ausführbares MULAN vorgestellt und daraus dann ein Prozess abgewickelt und in einem Kausalnetz dargestellt.

Das Beispiel ist in drei Schritten aufgebaut: Zuerst werden die Referenznetze des vereinfachten MULAN vorgestellt, als zweites wird eine „Anwendung“ mit zwei Agenten und einer Interaktion vorgestellt und im dritten Schritt wird ein möglicher daraus entstehender Prozess als Kausalnetz betrachtet.

Danach werden die vorhandenen Prozessbeobachtungswerkzeuge vorgestellt und schließlich stellt der letzte Teil einige Möglichkeiten zur Gestaltung eines Workflows für ein gegebenes Agententinteraktionsprotokoll vor.

3.5.1 Das vereinfachte Mulan

Um einen (nahezu) vollständigen Prozess einer Agentenanwendung in MULAN in einem darstellbaren und erläuterbaren Umfang zu halten, ist es notwendig, MULAN zu vereinfachen, so dass weniger Transitionen in den Netzen vorkommen. Es für diese Demonstration wichtig, dass auch die Startkonfiguration in den Netzen enthalten ist, damit die Anwendung ganz in RENEW definiert und ausgeführt werden kann. Andernfalls liegt ein Teil der anwendungsspezifischen Startprozedur auswärtig vor, z.B. in einem Shellskript und ist damit nicht mehr als schaltende Transition beobachtbar.

Weitere Vereinfachungen entstehen, weil es für dieses Beispiel nicht wichtig ist, dass ansonsten wesentliche Eigenschaften eines Agentensystems erfüllt sind. Die wesentlichen Einschränkungen sind: (a) beim Nachrichtenversand fehlt die Addressierung, (b) es ist kein Systemnetz für mehrere Plattformen vorhanden und es ist daher auch kein plattformübergreifender Nachrichtenversand vorgesehen, (c) der Zugriff aus den Protokollnetzen auf die Wissensbasis ist nur angedeutet und (d) der proaktive Protokollstart wird nicht durch die Wissensbasis gesteuert. Die so entstehenden Mehrdeutigkeiten müssen bei der Ausführung manuell aufgelöst werden, so dass der gewünschte von mehreren möglichen Prozessen zur Ausführung kommt. Insofern sind die hier angegebenen Netze nicht für allgemeine Zwecke geeignet, sondern sollen dem Prozessverständnis in MULAN dienen.

Das vereinfachte MULAN besteht aus einem Plattform-Referenznetz, einem Agenten-Referenznetz und einem Referenznetz für die Protokollfabrik (Abbildung 3.12c). Die Plattform hat eine zentrale Stelle für die instanziierten Agenten-Referenznetze, daran je eine Transition zum Starten und Beenden von Agenten sowie eine Transition für Plattform-internen Nachrichtenaustausch. Außerdem gibt es eine Stelle, in

der zwei Strings zum Initialisieren der Agenten „A“ und „B“ liegen. An der Transition **start** werden die Agenten-Referenznetze instanziiert.

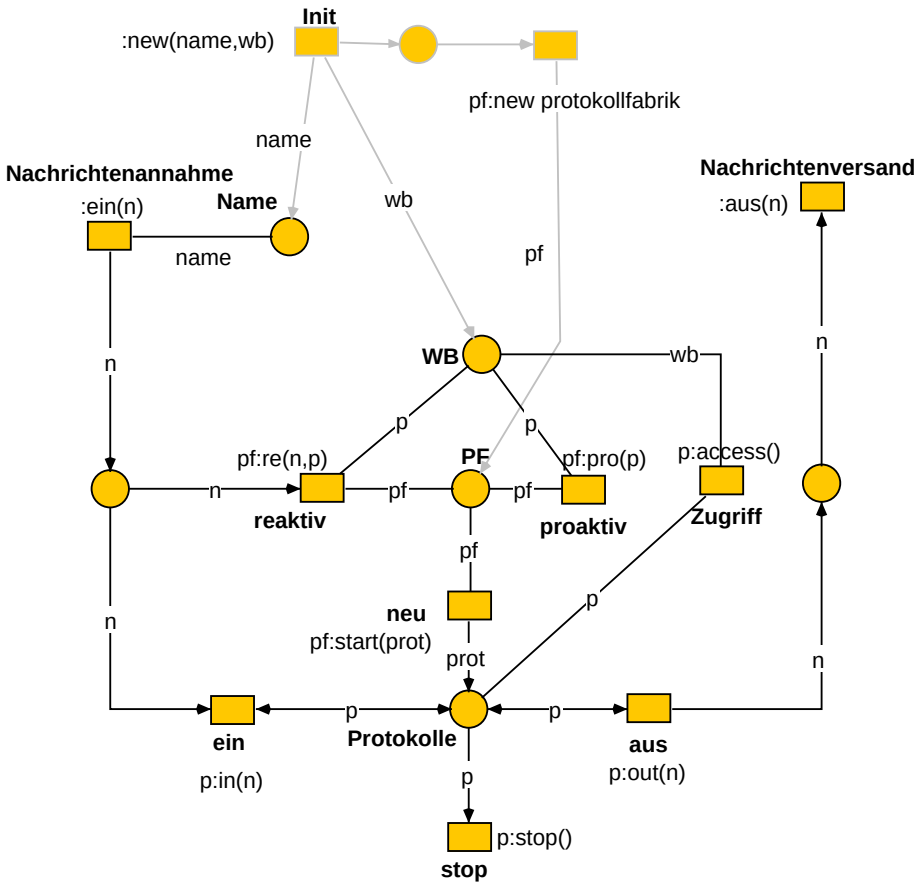
Das Agenten-Referenznetz hat je eine Stelle für die Wissensbasis, für die Protokollfabrik, für den Namen des Agenten und für instanziierte Protokolle. Außerdem hat das Agentennetz zwei Stellen zur Pufferung von eingehenden bzw. ausgehenden Nachrichten-Marken. Die Initialisierung des Agenten besteht aus zwei Transitionen: eine Transition trägt den **start**-Kanal, über den die Informationen zum Namen und zur Wissensbasis bei der Instanziierung des Netzes übergeben werden müssen. Die zweite Transition instanziiert die Protokollfabrik. Die übrigen Transitionen im Agenten-Referenznetz implementieren die Handlungsmöglichkeiten des Agenten. **Nachrichtenannahme** und **Nachrichtenversand** bilden die Schnittstelle zur Transition **Nachrichtenaustausch** im Plattform-Netz. Über die Protokollfabrik kann der Agent entweder **reaktiv** oder **proaktiv** ein Protokollnetz instanziieren und mit der Transition **neu** in die Stelle **Protokolle** ablegen. Die Transitionen **ein** und **aus** bilden die Schnittstelle zur Nachrichtenübermittlung in Protokollnetze. Protokollnetze können über die Transition **Zugriff** auf die Wissensbasis zugreifen.

Die Protokollfabrik schließlich hat je einen Zweig für die reaktive und für die proaktive Instanziierung von Protokollnetzen. Im reaktiven Fall wird dem Protokollnetz die Nachricht, auf die reagiert wird, übergeben. Die Transition **start** bildet die Schnittstelle zur Transition **neu** im Agentennetz, so dass die frische Protokollnetzinstanz in die Stelle **Protokolle** im Agentennetz abgelegt werden kann.

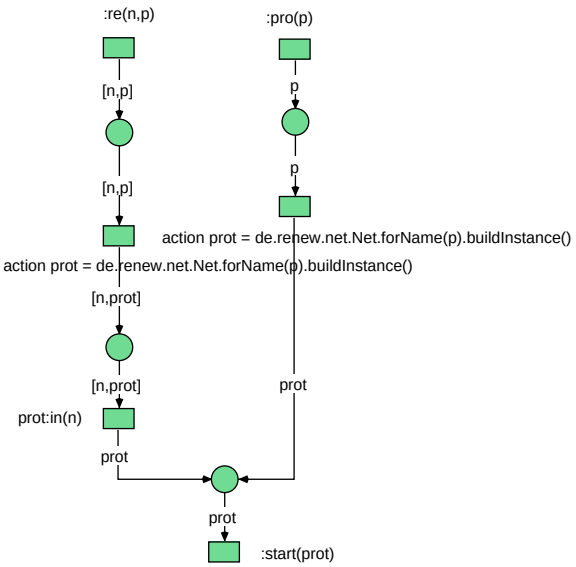
3.5.2 Eine Anwendung für das vereinfachte Mulan

In diesem einfachen MULAN sollen die Agenten A und B eine kleine Interaktion wie in Abbildung 3.13 ausführen. Die internen Aktionen sind normalerweise nicht Gegenstand eines *Inter*aktionsprotokolls. An dieser Stelle sind sie aus zwei Gründen einbezogen. Erstens kann solche Information die Intention des Entwicklers verdeutlichen. Annotationen für interne Aktionen werden im Rahmen des PAOSE-Ansatzes von dem Code-Generator berücksichtigt, d.h. es ist üblich und sinnvoll, sie zu modellieren. Zweitens ist für eine klare Darstellung in diesem Abschnitt eine Alternative im Ablauf notwendig, ansonsten ist eine Prozessdefinition für mehrere erlaubte Prozesse nicht von einer Prozessdarstellung zu unterscheiden. Zusätzlich alternativ versendete Nachrichten würden hier aber zu viel Aufwand und Umfang bei der Prozessabwicklung erzeugen.

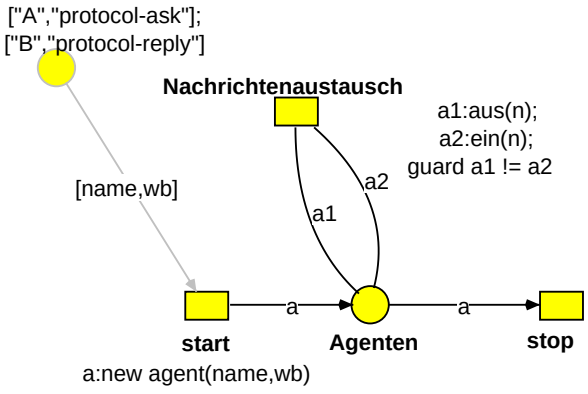
In der Umsetzung in Protokollnetze in Abbildung 3.14 ist der Nachrichtenversand und -empfang zu erkennen sowie je ein Zugriff auf die Wissensbasis (jeweils ohne dass der Nachrichteninhalt oder der Inhalt der Wissensbasis als Marke dargestellt wird).



(a) Agent



(b) Protokollfabrik



(c) Plattform

Abbildung 3.12: Stark vereinfachtes MULAN mit drei Ebenen: Plattform, Agent und Protokollfabrik

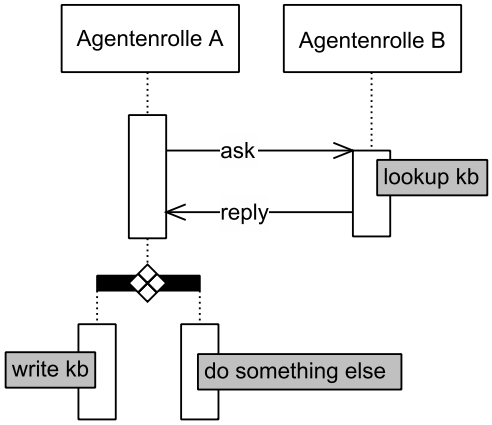
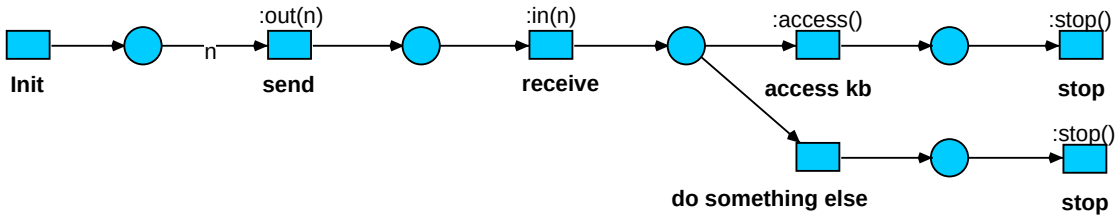
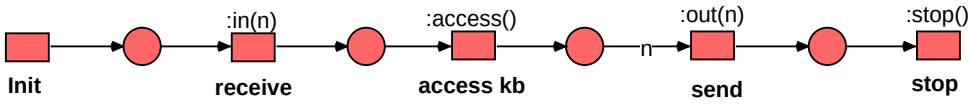


Abbildung 3.13: Interaktionsprotokoll für das einfache MULAN



(a) Anfrage-Protokoll (`protocol-ask`)



(b) Antwort-Protokoll (`protocol-reply`)

Abbildung 3.14: Protokollnetze zum Interaktionsprotokoll

3.5.3 Ein Prozess im vereinfachten Mulan

Im Folgenden wird ein möglicher Anwendungsprozess mit Hilfe von Kausalnetzen dargestellt. Das Kausalnetz zu einem möglichen Prozess der Netze in Abbildung 3.12 und Abbildung 3.14 ist in Abbildung 3.15 auf der nächsten Doppelseite dargestellt.

Kausalnetze sind für einfache Petrinetze definiert, die Originalnetze sind hier jedoch Referenznetze und enthalten getypte Marken. Die Definition von gefärbten Kausalnetzen für die Darstellung von Prozessen über mehrere Referenznetze hinweg muss auf eine spätere Arbeit verschoben werden. Hier wurde eine Darstellung gewählt, die sinngemäß einem Kausalnetz entspricht, d.h. es ist ein zyklenfreies Netz ohne Konflikte. In einem Kausalnetz gibt es keine Markierung. Die Marken aus dem Original-Netz sind hier deshalb als Annotationen zu verstehen. Ein synchroner Kanal wird durch zwei Transitionen dargestellt, die mit einer gestrichelten Linie verbunden sind (in Anlehnung an die Darstellung in [Jessen und Valk 1987, S. 205]). Diese Transitionen schalten gemeinsam, so dass die Semantik die einer einzigen Transition ist.

Die Farben der Netzelemente werden beibehalten. Das Netz „Agent“ und das Netz „Protokollfabrik“ werden je zweimal instanziiert. Jede Netzinstanz wird in einer eigenen Bahn („Swimlane“) dargestellt. Swimlanes sind eine allgemeine Technik für eine nach strukturellen Gesichtspunkten geordnete Prozessdarstellung, die insb. für die Activity-Diagramme der UML verbreitet ist[9]. Aus Platzgründen ist an vier Stellen im Kausalnetz eine vergröberte Darstellung gewählt worden, wo ansonsten nichtsynchronisierte Transitionen oder Stellen dargestellt würden (bei der Initialisierung der beiden Agentennetze und bei der Instanziierung der beiden Protokollnetze in der jeweiligen Protokollfabrik). Die Vergröberung ist durch ein kleineres Rechteck innerhalb einer Transition bzw. durch einen kleineren Kreis innerhalb einer Stelle dargestellt. Im Folgenden wird der Ablauf des Kausalnetzes erläutert.

Das Plattform-Netz (Abbildung 3.12c) wird instanziiert. Es enthält in einer Initialisierungsstelle zwei einfache Agentenkonfigurationen aus einem Namen und einer Wissensbasis (welche aus dem Namen eines Protokollnetzes besteht). Die Transition **start** kann zweimal feuern: es wird je eine Instanz des Agentennetzes (Abbildung 3.12a) erzeugt und mit dem Kanal **:new** werden die Wissensbasis und der Agentenname übermittelt. In der Stelle **Agenten** liegen nun die frisch erzeugten Netzreferenzen. Bei der Instanziierung wird auch der Kanal **:new** im Agentennetz gefeuert, der an der Transition **Init** im Agentennetz vorhanden ist. So werden bei der Initialisierung der beiden Agentennetz-Instanzen die Stellen **Name**, **WB** und **PF** (für *Wissensbasis* und *Protokollfabrik*) mit Marken bestückt. In der Darstellung im Kausalnetz ist dieser Vorgang vergröbert: die Transition zur Erzeugung der Protokollfabrik wird nicht mit dargestellt. Agent **A** schaltet nun die Transition **proaktiv** synchron mit **pro(p)** in der Protokollfabrik. Aus der **WB** wird das Protokoll **p** gebunden, so „weiß“ die Protokollfabrik, welches Netz zu instanziieren ist. Im Kausalnetz ist der Schritt zur Netzinstanziierung in der Protokollfabrik vergröbert dargestellt. Der Agent holt über die Transistion **neu** die Protokollnetzinstanz aus der Protokollfabrik ab und legt sie in **Protokolle** ab. Das Protokollnetz selbst kann bereits **Init** feuern. Die Anfragenachricht aus dem Protokollnetz wird über **send** (im Protokollnetz) und **aus**

[9] Eine kleine Übersicht über Herkunft und Verwendung von Swimlanes in der Prozessmodellierung gibt [ViCon GmbH 2009]

(im Agentennetz) übermittelt. Nun können die Transitionen **Nachrichtenversand** in Agent **A**, **Nachrichtenaustausch** und **Nachrichtenempfang** in Agent **B** synchron schalten: das Plattform-Netz übermittelt die Nachricht von Agent **A** zu Agent **B**. Im Agent **B** wird nun über **reaktiv** das Protokollnetz aus **WB** instanziiert, dann wird die Nachricht an das Protokollnetz übergeben (Transitionen **receive** im Protokollnetz und **prot:in(n)** in der Protokollfabrik), bevor das Protokollnetz mit **neu** in die Stelle **Protokolle** abgelegt wird. Das Protokollnetz kann über **access** weitere Informationen aus der **WB** abfragen (es steht nichts weiter drin, die Koordination sieht aber wie dargestellt aus) und eine Antwortnachricht **n** erzeugen. Diese wird über **aus** und **send** an den Agenten übermittelt und das Protokollnetz im Agent **B** wird gestoppt (**stop** im Agentennetz und im Protokollnetz). Nebenläufig zum Stoppen findet wieder die Nachrichtenübertragung durch das Plattform-Netz mit den drei Transitionen **Nachrichtenversand**, **Nachrichtenaustausch** und **Nachrichtenempfang** statt. Mit den Transitionen **receive** und **ein** erhält das Protokollnetz im Agent **A** die Nachricht und greift daraufhin per **access** auf die **WB** des Agenten zu. Nun kann auch dieses Protokollnetz gestoppt werden.

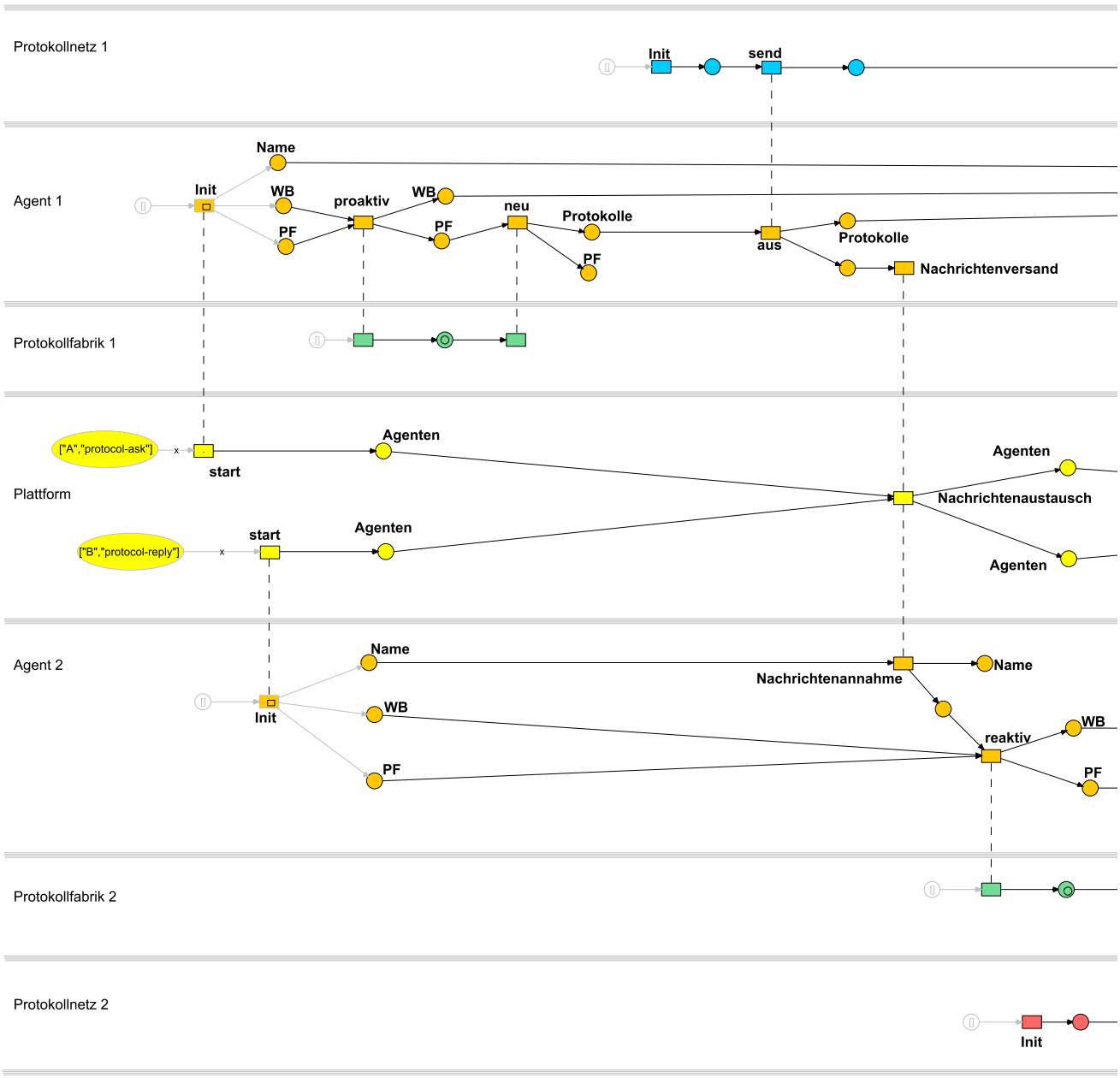


Abbildung 3.15: Prozessdarstellung als Kausalnetz, Teil 1

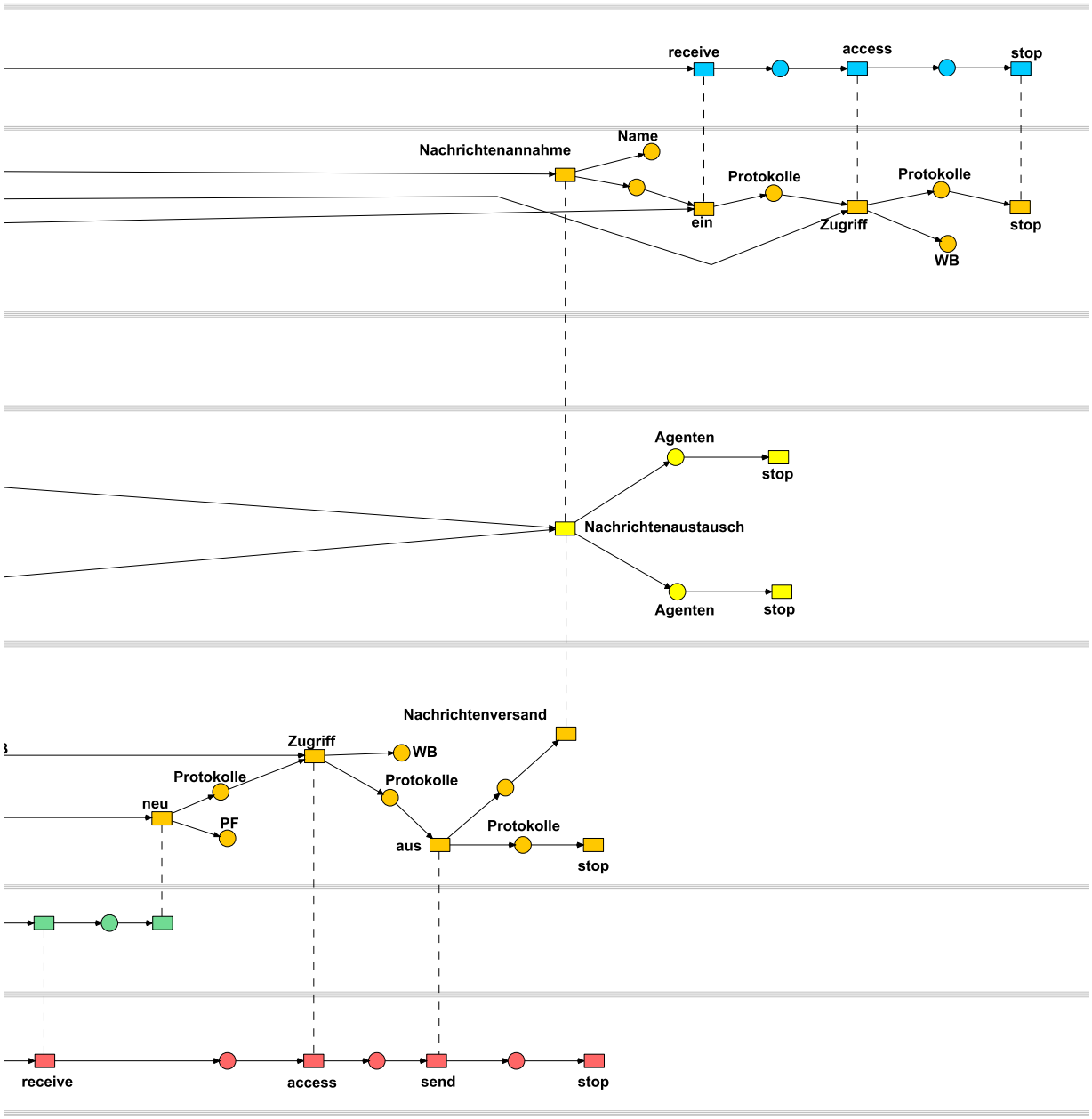


Abbildung 3.16: Prozessdarstellung als Kausalnetz, Teil 2

3.5.4 Prozessbeobachtung in Mulan

Für den praktischen Umgang mit Prozessen zur Laufzeit in MULAN gibt es Werkzeuge, von denen im Folgenden eine Auswahl vorgestellt wird.

Sniffer Der MULAN-Sniffer ist ein spezialisiertes Werkzeug zur Prozessbeobachtung auf Nachrichtenebene. Im Transportdienst der Plattform werden alle Nachrichten gesammelt und nummeriert. Es gibt eine Listendarstellung der Nachrichten, eine Darstellung für den Inhalt einzelner Nachrichten und eine graphische Darstellung als Interaktionsdiagramm[10]. Die Darstellung kann gefiltert werden, z.B. nach Agentennamen oder Nachrichtennummern.

Der Sniffer nutzt damit die prinzipielle Möglichkeit, für eine einzelne ausgewählte Stelle oder Transition einen Prozess darzustellen. Dabei muss dann der jeweilige Inhalt der Marke oder die Bindung der Variablen mit dargestellt werden, um eine nützliche Information zu erhalten. Übertragen auf die Darstellung des Kausalnetzes beobachtet der Sniffer die Transition **Nachrichtenaustausch** im Plattform-Netz und stellt eine Reihe von verschiedenen Darstellungs- und Filteroptionen bereit.

Mulan-Viewer Der MULAN-Viewer ist eher ein umfassendes Werkzeug zur Inspektion und Steuerung des aktuellen Zustands. Der Anwendungsprozess wird also in Echtzeit abgebildet. Dargestellt werden Plattformen und Agenten sowie deren Wissensbasis-Inhalt, laufende Protokolle und Entscheidungskomponenten.

Der MULAN-Viewer beobachtet also eine Auswahl von Stellen und Transitionen, bezogen auf das hier dargestellte Kausalnetz. Diese Beobachtung wird wie beim Sniffer stark aufbereitet und auf den Zweck der Beobachtung und Steuerung angepasst. Anders als beim Sniffer wird aber der aktuelle Zustand dargestellt und es existieren eine ganze Reihe von Möglichkeiten zur Steuerung seitens des Beobachters.

Renew RENEW bietet die Möglichkeit, eine schaltende Netzinstanz zu beobachten. Wie beim MULAN-Viewer wird der jeweils aktuelle Zustand abgebildet. Es ist insbesondere im Schrittmodus möglich, einzelne Schaltvorgänge gezielt auszulösen und die Bindung zu steuern.

Wenn der Prozess innerhalb einer ausgewählten Netzinstanz (z.B. ein Agentennetz) beobachtet werden soll, so kann man mit RENEW die Netzinstanz anzeigen und die Schaltvorgänge direkt beobachten. Dazu existieren die Werkzeuge Breakpoint setzen, simulationsweiter Schrittmodus und netzweiter Schrittmodus sowie manuelles Auslösen von Schaltvorgängen. Für Agentennetz A im oben vorgestellen Kausalnetz kann also beobachtet werden, dass Initialisierung, proaktiver Protokollstart, Nachricht aus Protokoll versenden, Nachricht empfangen und in Protokoll leitet, Zugriff eines Pro-

[10] An dieser Stelle sind wieder die Interaktionsdiagramme im Gegensatz zu den Protokolldiagrammen oder Agenteninteraktionsprotokolldiagrammen (AIP) zu verstehen: Es wird genau ein zeitlicher Ablauf dargestellt.

tokolls auf die Wissensbasis und stop eines Protokolls nacheinander stattfinden. Dies entspricht der Darstellung einer einzelnen Bahn im Kausalnetz[11].

Als nicht-flüchtige Prozessdarstellung bietet RENEW einen Simulation-Log an, das ist eine globale Liste der geschalteten Transitionen mit ihren jeweiligen Bindungen.

3.5.5 Ein Workflow zur Agenteninteraktion

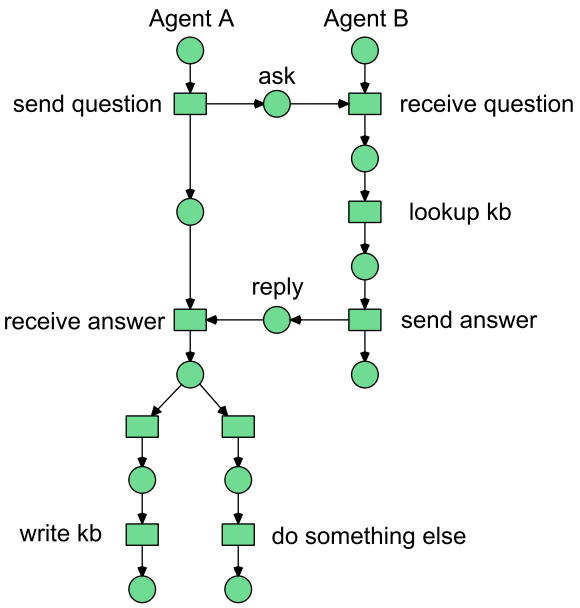
Im vorigen Text dieses Abschnitts wurde anschaulich deutlich, wie ein Interaktionsprotokoll bei der Implementation in zwei agentenbezogene Vorgänge zerfällt. Dieser Unterabschnitt zeigt einige Möglichkeiten auf, um ein Interaktionsprotokoll in einen Workflow zu übersetzen. Sobald die für Workflows typischen Modellierungsschwerpunkte (nämlich Inhalt und Wirkungen von Tätigkeiten, vgl. Abschnitt 3.2 auf Seite 80) einbezogen werden, ändert sich die Semantik des beschriebenen Prozesses.

In Abbildung 3.17 sind vier aufeinander folgende Möglichkeiten dargestellt, um einen Workflow zu der Interaktion aus dem einfachen Beispiel im vorigen Abschnitt zu gestalten. Abbildung 3.17a zeigt eine Übersetzung des Protokolldiagramms in Abbildung 3.13 in ein Petrinetz. Schon dieses Petrinetz könnte als Workflow verwendet werden (vorausgesetzt, die Schnittstelle zum WFMS wird eingehalten: Start- und Stop-Transitionen sowie Anschriften an den (Task-) Transitionen). Abbildung 3.17b stellt eine üblichere Form desselben Workflows dar, in dem die für den Kontrollfluss des Workflows überflüssigen Stellen weggelassen wurden. Die Akteure sind nun als Anschriften (Parameter) an den Transitionen vermerkt und der Ablauf wurde um eine Start- und eine Stop-Transition für den Workflow versehen. Diese Sorte Workflow könnte noch automatisch aus dem Interaktionsprotokoll gewonnen werden.

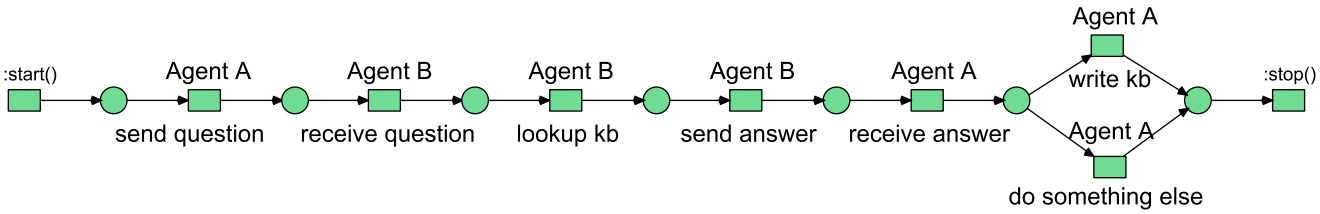
Nun wird i.A. nicht das Senden und Empfangen von Nachrichten in Workflows modelliert, sondern die Tätigkeiten, die *dazwischen* zu erledigen sind. Dies wird in Abbildung 3.17c angedeutet: Agent A und Agent B können sich unabhängig voneinander auf ihre jeweilige Aufgabe vorbereiten und wenn beide fertig sind, sorgt das WFMS dafür, dass Agent B alle Informationen bekommt, um den nächsten Schritt auszuführen, danach bekommt Agent A die Aufgabe, die Antwort zweckmäßig zu verarbeiten. Die Semantik des Interaktionsprotokolls ist hier bereits verändert.

In der Variante in Abbildung 3.17d wird nicht mehr explizit gefordert, dass die Agenten sich auf das Formulieren und Empfangen von Nachrichten vorbereiten müssen. Statt dessen bekommt der Workflow als Startparameter die Anfrage (`question`) vom Initiator des Workflows übergeben. Agent B bekommt dann die Aufgabe, der Frage eine eine Antwort hinzuzufügen. Schließlich muss noch „jemand“ etwas mit der Antwort anfangen. Hier wird die Verwertung von Frage und Antwort dem Agent A zugewiesen. An dieser Stelle wird das Rollenkonzept von Workflows besonders deutlich: Im Interaktionsprotokoll (Abbildung 3.13) sind Agenten-*Rollen* angegeben. Ebenso sind die Anschriften an allen Tasktransitionen der Workflow-Varianten *Rollenbezeichner*. In diesem Workflow kann also der Initiator jemand anders sein als derjenige Agent, der die Antwort verwertet. Dies verlagert wiederum den Schwerpunkt der Mo-

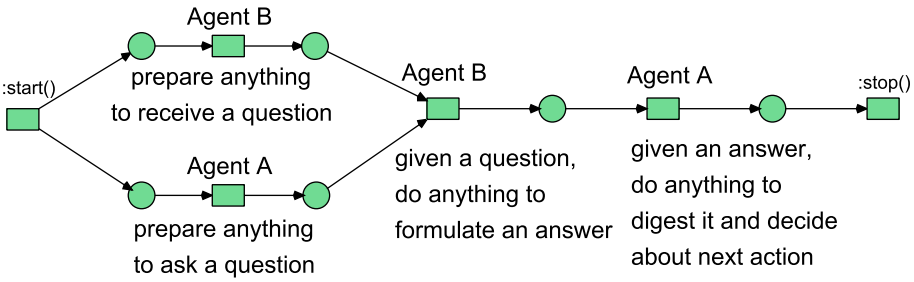
[11] Für die Erzeugung eines Kausalnetzes aus mehreren instanziierten Referenznetzen gibt es keine Werkzeugunterstützung in RENEW, deshalb wurde im Abschnitt 2.1.6 das für diese Arbeit entwickelte und verwendete Vorgehen beschrieben.



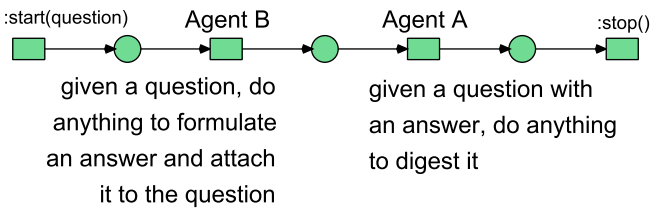
(a) Interaktionsprotokoll als Petrinetz



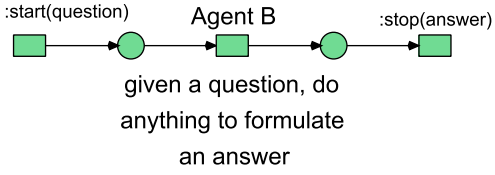
(b) Workflow 1



(c) Workflow 2



(d) Workflow 3



(e) Workflow 4

Abbildung 3.17: Verschiedene mögliche Workflow-Varianten zum Interaktionsprotokoll

dellierung gegenüber dem Interaktionsprotokoll, so dass die Semantik wieder verändert wird.

In Abbildung 3.17e ist die Antwort als Ergebnisparameter der Stop-Transition hinzugefügt. Es gibt also einen Initiator des Workflows, der eine Frage als Parameter an das WFMS übermittelt. Der eigentliche Task ist dann die Erzeugung einer Antwort. Die Antwort wird dem Initiator als Ergebnis des Workflows mitgeteilt. Hier ist nun die Information verloren gegangen, dass der Initiator die Rolle A innehaben muss. Ob dieses eine relevante Information ist, kann wieder nur aus dem Anwendungszusammenhang entschieden werden. Für sehr komplexe Interaktionen kann diese knappe Darstellungsform von großem Vorteil sein.

Diese Aufzählung von vier Workflow-Varianten zu einem gegebenen Interaktionsprotokoll erhebt keinen Anspruch auf Vollständigkeit, sondern soll deutlich machen, dass der Entwurf von Workflows *nicht* eindeutig aus einem Interaktionsprotokoll hervorgeht. Vielmehr kommt es auf den gewünschten Effekt und auf die gewünschte Information an.

3.6 Anforderungen und Konzeption für PIA

Zuerst werden die Anforderung benannt, um diese dann im zweiten Abschnitt mit den vorhandenen Möglichkeiten zu vergleichen und Lösungskonzepte zu formulieren.

3.6.1 Anforderungen an PIA

In einer verteilten Anwendung sollen sowohl die Anwendungsstruktur als auch die Prozesse der Anwendung explizit und jeweils zusammenhängend repräsentierbar sein, und zwar sowohl zur Entwurfs- als auch zur Laufzeit. Dadurch sollen sowohl die Struktur- als auch die Prozesskomponenten der Anwendung zum Entwurf, zur Spezifikation, zur Beobachtung und zur Änderbarkeit zur Verfügung stehen. Der Entwurf einer solchen Anwendung soll mit einem bestehenden Entwurfsansatz möglich sein (d.h. die Schnittstelle des Rahmenwerkes soll bewährt sein).

Die Prozess-Infrastruktur für Agentenanwendungen (PIA) soll es ermöglichen, sämtliche agentenübergreifenden Prozesse der Anwendung wie Workflows zu beobachten und zu steuern. Dazu gehören die explizite Instanziierung der agentenübergreifenden Prozesse und die Darstellung von Informationen zur Laufzeit: der aktuelle Zustand, der Fortschrittsverlauf und aktuelle Parameter und beteiligte Agenten (strukturelle Komponenten). Für jede strukturelle Komponente soll ersichtlich sein, an welchen Prozessen sie aktuell beteiligt ist. Eine Anwendung soll dabei gegenüber dem Rahmenwerk als Agentenanwendung spezifiziert werden, nicht als Misch- oder Workflowsystem.

Eine Architektur für PIA (also APIA) zeigt die Komponenten und Schichten auf, aus denen jeweils ein lokales System zusammengesetzt ist. Die benötigte grundlegende Funktionalität für die eben beschriebenen Anforderungen bekommt auf diese Weise einen konzeptionellen Implementationsort zugewiesen. Die Architektur kann auch als Richtschnur für die Umsetzung von PIA verwendet werden.

3.6.2 Ausgangslage

Aus den Darstellungen im Abschnitt 3.5 lässt sich schließen, dass die nötigen Informationen für das Ziel der Arbeit in den vorhandenen Möglichkeiten zur Prozessbeobachtung prinzipiell nicht verfügbar sind. Dies wird im Folgenden ausgeführt.

Die Rekonstruktion einer expliziten Darstellung aus implizit vorliegenden Informationen ist auch ohne PIA möglich: Es können Werkzeuge entwickelt werden, die die Nachrichten inspizieren und aus den darin enthaltenen Identifikationsnummern für Konversationen (cid für conversation id) und evtl. aus einem vorhandenen Interaktionsprotokoll die Nachrichten wieder sortieren und sie zur Interaktionen zusammenfassen und so eine Prozessdarstellung ermöglichen. Das wäre auch durchaus gut. Es fehlen dabei wesentliche Eigenschaften zur Prozessverwaltung, da lediglich für die Darstellung die Prozess-Instanzen simuliert werden.

Die Darstellungsmöglichkeiten von RENEW können die geforderten prozessbezogenen Informationen nicht liefern: die Schaltfolgenliste bezieht sich auf einen Ort im real verteilten System und bildet keine entfernten Prozesse ab. Die Beobachtung der Marken ist keine explizite Prozessdarstellung.

Insgesamt gibt es eine gute Unterstützung für die Beobachtung und Steuerung von Prozessen entlang von Strukturkomponenten. Der Sniffer bietet eine strukturorientierte Darstellung des Anwendungsprozesses: neben der zeitlichen Reihenfolge wird der Agentenzusammenhang dargestellt, d.h. es ist sichtbar, zu welchem Agenten eine Nachricht gehört. Der Interaktionszusammenhang geht jedoch verloren: es ist nicht erkennbar, zu welcher Interaktion eine Nachricht gehört. Für eine prozessorientierte Prozessdarstellung müsste der Interaktionszusammenhang gewahrt bleiben.

Auch im Kausalnetz ist der Interaktionszusammenhang nicht adäquat dargestellt. Sobald mehrere Interaktionen nebenläufig ausgeführt werden, ist nicht erkennbar, welche Prozessschritte zu einer Interaktion dazugehören. Insbesondere ist die Wissensbasis der Agenten dabei ein kritischer Punkt: sie enthält Schlüssel für verschiedene Interaktionen, die sich so auch gegenseitig beeinflussen können. Ob dies in einem konkreten Beispiel der Fall ist, kann nur aus dem Anwendungszusammenhang, aus der Semantik der Interaktion bestimmt werden. Diese Information fehlt im Kausalnetz.

Ein *Workflow* fasst genau notwendigen Aspekte für eine prozessorientierte Prozessdarstellung zusammen. Im letzten Abschnitt wurde deutlich, dass es bei der Gestaltung eines Workflows für eine Interaktion großen Spielraum bei der Ausgestaltung gibt. Für die hier beschriebenen Zwecke der Beobachtung kann es reichen, schon die erste Variante, die einfache Übersetzung eines Interaktionsprotokolls zu verwenden.

3.6.3 Konzeption für PIA

PIA setzt auf CAPA auf und bedient sich der Workflowtechnologie für die Funktionalität der Prozessunterstützung.

PIA ist eine verteilte Infrastruktur-Schicht für Agentenanwendungen mit der Aufgabe, anwendungsbezogene Prozesse zu verwalten. Die Prozessverwaltung wird unter Verwendung der Workflow-Technologie realisiert. Der Entwurfsansatz für Anwendungen ist ein im Werkzeugaspekt leicht modifizierter PAOSE-Ansatz. Die verteilte Infra-

struktur wird mit Hilfe von Konzepten aus Agentcities, openNet und dem Subscription Manager realisiert.

Die Art der Prozessunterstützung, die Workflowmanagementsysteme für Workflowanwendungen bereitstellen, soll die hier entwickelte Prozess-Infrastruktur für Agentenanwendungen leisten. Ein WFMS für Agenten bereitzustellen ist nur ein Schritt in diese Richtung, weil das WFMS nur diejenigen Prozesse der Agentenanwendung verwalten kann, die explizit als Workflows übergeben werden. Die Prozess-Infrastruktur für Agentenanwendungen (Pia) soll jedoch alle Prozesse dergestalt verwalten, dass sie nicht in gesonderter Form angegeben werden müssen. Diese Arbeit hat die Architektur einer Prozess-Infrastruktur für Agentenanwendungen (Apia) zum Ziel.

Meine darüber hinausgehende Vision ist, dass die Prozessverwaltung so *automatisch* und *ubiquitär* ist wie der Nachrichtentransport für Agentenanwendungen. Das ist wie folgt zu verstehen: Zum Nachrichtentransport gehören nicht nur eine Netzwerkverbindung und ein gemeinsames Nachrichtenformat, sondern auch die Fähigkeit jedes einzelnen Agenten, Nachrichten zu versenden, zu empfangen und zu interpretieren. Diese Fähigkeit wird vom Rahmenwerk unterstützt, aber ein Agent, der auf eingehende Nachrichten nicht reagiert, kann davon nicht profitieren. In diesem Sinne bildet die zusammengenommene Fähigkeit zum Umgang mit Nachrichten von allen beteiligten Agenten plus die Netzwerk-Fähigkeiten der Plattformen und eine gemeinsame Basis-Ontologie das Nachrichtentransportsystem im weitesten Sinne.

In einem Multi-Agenten-System trägt jeder Agent seinen Teil zum Gesamtprozess bei, er trägt also einen Teil der übergreifenden Prozesskontrolle (wie rudimentär diese im Sinne eines Workflowmanagements auch sein mag). Alle Teile zusammen bilden die globale „Prozessverwaltung“. Diese „Prozessverwaltung“ ist insofern bereits *ubiquitär*, als dass sämtliche Vorgänge irgendwo implizit oder explizit definiert sind und ausgeführt werden. Nun sollen die Prozesse aber wie in einem WFMS explizit verwaltet werden, um die Wartbarkeit, Verständlichkeit und Beobachtbarkeit von Workflowsystemen zu erhalten. Wenn nun diese Prozessverwaltung ubiquitär wie in einem WFMS vorhanden sein soll, müssen auch die agenteninternen Prozesse einbezogen werden, da ein Agent selbst wieder komplex aufgebaut sein kann. Und zwar soll dies *automatisch* zur Verfügung stehen, nicht nur, wenn ein Agent eben entsprechend programmiert ist.

Es wird im Folgenden ausgehend von (**I**) Agentensystemen ohne besondere Prozesskontrolle über den Zwischenschritt (**II**) eines agentenbasierten Workflowmanagementsystems ein (**III**) WFMS *für* Agenten entwickelt, um auf den gewonnenen Konzepten (**IV**) Apia vorzustellen. Die (**V**) vollständige, ubiquitäre Prozessverwaltung auch für agenteninterne Prozesse bildet als Vision den Ausblick dieser Arbeit.

In den nächsten beiden Kapiteln folgt die Umsetzung zuerst auf der Architektur-Ebene im Kapitel 4 mit den genannten fünf Stufen über das Ziel Apia hinaus bis zur vollständigen Integration von Workflow- und Agententechnologie zu einem neuartigen Einheiten-Begriff. Die Umsetzung für einen Prototyp, der Pia nicht ganz erreichen kann, folgt in Kapitel 5. — Zum Umfang und zur Komplexität von solchen Systemen sei an dieser Stelle ein zweitesmal auf Dadam et al. verwiesen: „Der [ADEPT-]Prototyp zeigt aber auch, dass solche High-End-WfMS große Software-Systeme sind, die leicht die Code-Komplexität eines High-End-DBMS erreichen werden.“ (vgl. [Dadam und Reichert 2004]).

Kapitel 4

WFMS und Agenten

In diesem Kapitel werden die Workflowtechnologie und die Agententechnologie als gleichwertige Sichtweisen nebeneinandergestellt, um sie dann auf Architekturebene stufenweise zusammenzuführen. Dabei wird eine Schichtenarchitektur entwickelt, die auf jeder Stufe ergänzt wird. Die fünf Stufen werden im Text mehrmals durchlaufen, um jeweils mit unterschiedlichem Detailgrad verschiedene Aspekte zu beleuchten. Abschnitt 4.1 gibt einen Überblick über das Kapitel. Abschnitt 4.2 bereitet die weitere Integration vor und gibt auch schon einen Überblick über die einzelnen Stufen. In den Abschnitten 4.3 bis 4.7 wird jeder Stufe ein eigener ausführlicher Abschnitt gewidmet. Schließlich werden in Abschnitt 4.8 einige angrenzende Fragestellungen zur Umsetzbarkeit einer solchen Architektur und zum Einsatz eines entsprechenden Rahmenwerkes innerhalb von Paose aufgegriffen, gefolgt von einer Zusammenfassung mit Diskussion, welche die vorgestellten Integrationsstufen auch noch einmal in Bezug auf das Ziel dieser Arbeit, die Architektur einer Prozess-Infrastruktur für Agentenanwendungen (Apia) beleuchtet.

Das weitergesteckte Ziel der vollständigen Integration von Workflow- und Agententechnologie wird in diesem Kapitel als Orientierungspunkt hinzugenommen.

4.1 Überblick

Sowohl Workflowmanagementsysteme als auch Agenten wurden in Kapitel 2 als Möglichkeiten für die Strukturierung komplexer Software vorgestellt. Die Möglichkeiten dieser „reinen Formen“ stoßen bei zunehmender Komplexität an ihre Grenzen.

Jede Anwendung hat Struktur und Verhalten. Diese sind für größere Systeme nur als Sichten erkennbar. Das ganze System in seinen strukturellen und Verhaltensaspekten kann nicht auf einen Blick erfasst werden. Deshalb braucht man Sichten, die jeweils einen Teil der Struktur oder einen Teil des Verhaltens darstellen. Optimalerweise ist für alle Systembestandteile je eine Sicht für deren Struktur und Verhalten vorhanden. Für die Beziehungen zwischen zwei Sichten werden ebenfalls Darstellungsweisen (also wieder Sichten) benötigt. Das weitergesteckte Ziel ist, sowohl die Verhaltens- als auch die Strukturansicht zu jedem Zeitpunkt verfügbar zu machen, also sowohl zur Entwurfs- als auch zur Laufzeit einer Anwendung.

Es ist ein wichtiges Prinzip, wesentliche Teile einer Anwendung mit einer Identität zu versehen (Kapselung und Abstraktion). Agentenbasierte Anwendungen sind anhand ihrer Struktur in Teile (Agenten) zerlegt. Workflowbasierte Anwendungen sind anhand ihres Verhaltens in Teile (Workflows) zerlegt. Zur Änderung eines übergreifenden Verhaltensaspekts in einer agentenbasierten Anwendung müssen mehrere Agenten angepasst werden. Zur Änderung eines übergreifenden strukturellen Aspekts in einer workflowbasierten Anwendung müssen mehrere Workflows angepasst werden.

Das weitergesteckte Ziel ist, für jede Änderung die passende Abstraktion zur Verfügung zu haben. Für dieses Ziel werden Workflowtechnologie und Agententechnologie in diesem Kapitel konzeptuell integriert. Die Integration soll für das weitergesteckte Ziel symmetrisch sein, so dass zu jedem Zeitpunkt wahlweise eine der beiden Perspektiven eingenommen werden kann. Grundlegend dafür sind die Gemeinsamkeiten von Agententechnologie und Workflowtechnologie: Beide zerlegen eine Anwendung in Teile, die zur Laufzeit instanziiert werden und eine Identität erhalten. Beide bieten eine Umgebung zum Entwurf, zur Implementierung und zur Ausführung der Teile. Die Gemeinsamkeiten werden im Abschnitt 4.2.1 durch Begriffsklärung deutlich herausgearbeitet. Die Verbindung speziell von Agenten- und Workflowtechnologie aus der Menge aller Software-Entwicklungsansätze ist sinnvoll, weil dadurch die orthogonalen Aspekte Struktur und Verhalten abgedeckt werden können.

Bei der Integration entsteht letztendlich eine komplexe Architektur für ein Anwendungsrahmenwerk, das sowohl ein Workflowsystem als auch ein Agentensystem ist und das sowohl die Funktionen beider Technologien nutzt als auch anbietet. Die Architektur wird in fünf charakteristischen Stufen aufgebaut. Jede Stufe beschreibt die Architektur eines Anwendungsrahmenwerkes mit zunehmender Integration der Agenten- und Workflowtechnologie. Zur Verdeutlichung der charakteristischen Eigenschaften der Stufen werden auch Beispiele aus Paose hinzugezogen.

Die erste Stufe beschreibt die Vorteile, die durch einen klar strukturierten Vordergrund bei der Software-Entwicklung erreicht werden. Dabei rücken andere Aspekte in den Hintergrund der Aufmerksamkeit, die dann in den weiteren Stufen in den Vordergrund integriert werden. In der fünften Stufe stehen die Prozesssicht und die strukturelle Sicht beide zur Verfügung. Die Charakterisierung der Stufen ist durch eine Arbeit aus der Psychologie inspiriert [Frambach 1994]. Sie betrachtet die Entwicklung von persönlicher Identität durch Integration einer emotionalen dualen Polarität in fünf Stufen, was hier inhaltlich nicht weiter von Interesse ist, aber wegen der einprägsamen Charakterisierung Verwendung findet.

Abschnitt 4.2.2 erläutert in einem ersten Durchgang die fünf Stufen unter Verwendung von einprägsamen Begriffen aus [Frambach 1994] und anhand von Beispielen aus dem Kontext der Workflow- und Agententechnologie. Abschnitt 4.2.3 fasst die fünf Stufen auf einer Text- und einer Diagrammseite zusammen, so dass der Aufbau des Anwendungsrahmenwerkes auf jeder Stufe deutlich wird. Im Abschnitt 4.2.4 wird die detaillierte Betrachtung der Stufen syntaktisch vorbereitet, indem eine Farbcodierung zur Einordnung in die vorliegende Arbeit und eine Symbolcodierung für Schnittstellentypen eingeführt wird.

In den Abschnitten 4.3 bis 4.7 wird jede Stufe ausführlich in einem eigenen Abschnitt dargestellt. Es wird jeweils die Architektur eines konkreten Systems vorgestellt. Beispiele zur Laufzeitstruktur ergänzen die Darstellung. Das Kapitel 4 schließt mit einer Zusammenfassung und Diskussion in Abschnitt 4.8, worin wird der Begriff „Prozess-Infrastruktur“ auf jede Stufe bezogen dargestellt wird.

4.2 Vorbereitung

Wir betrachten zunächst die Gemeinsamkeiten von Workflows und Agenten im Abschnitt 4.2.1, danach das verwendete Integrationsschema als solches (anhand von hier

interessanten Inhalten in Abschnitt 4.2.2), anschließend einen Überblick über die Architektur auf den fünf Stufen (Abschnitt 4.2.3) und schließlich im Abschnitt 4.2.4 einige syntaktische Hilfsmittel zur detaillierten Darstellung der fünf Stufen in den folgenden Abschnitten 4.3 bis 4.7.

4.2.1 Analogie und Gemeinsamkeiten von Workflows und Agenten

Um Workflow- und Agententechnologie integrieren zu können, brauchen sie eine gemeinsame Basis. Dieser Abschnitt dient der Feststellung einer solchen Basis. Sowohl die Workflowtechnologie als auch die Agententechnologie bieten Mittel an, um den wesentlichen Teilen einer Anwendung im Verlauf des Entwurfs, der Umsetzung und der Ausführung eine Identität zuzuweisen. Das verwendete Rahmenwerk definiert die technischen Möglichkeiten, wiederkehrende Strukturen und Funktionen und den Lebenszyklus von Systemteilen. Die verwendete Abstraktionsrichtung (Agenten oder Workflows) ist eine charakteristische Eigenschaft einer Anwendung. Workflowtechnologie und Agententechnologie haben folgende Gemeinsamkeiten:

- Es gibt eine Hauptabstraktionsrichtung, die die Eigenschaften von instanziierbaren Einheiten festlegt.
- Es gibt ein Rahmenwerk (und häufig einen ganzen Software-Entwicklungsansatz) für die Implementierung.
- Es gibt eine spezielle Laufzeitumgebung.
- Die Hauptabstraktionsrichtung ist ein wichtiges Merkmal einer fertigen Anwendung.

Für Workflows fallen alle diese Aspekte unter dem Begriff Workflowmanagementsystem zusammen. Zur Verdeutlichung und Betonung der Analogie zwischen Agententechnologie und Workflowtechnologie werden hier möglichst analog gebildete Begriffe angestrebt, deshalb wird im Folgenden der Begriff „Agentenmanagementsystem“ genauso umfangreich verwendet, obwohl der entsprechende Begriff Agentenmanagementsystem im allgemeinen nicht so umfassend benutzt wird. Insbesondere ist die Abkürzung AMS von der FIPA für einen Standard-Agenten in Plattformen vergeben[1], deshalb wird im Folgenden abgrenzend die Abkürzung AgMS verwendet.

Begriffsbestimmung 20 (Agentenmanagementsystem (AgMS))
Ein AgMS ist ein Managementsystem entsprechend der Begriffsbestimmung 3 auf Seite 61, bei dem die instanziierbaren Einheiten Agenten sind. ◇

In der folgenden Tabelle sind die Begriffe gegenübergestellt.

[1] Schaut man genauer in die Spezifikation [FIPA 23 2005], zeigt sich, dass genau das gemeint ist: ein Agentenmanagementsystem, das selbst wie ein Agent ansprechbar sein muss. Die Agenten sprechen also ihr eigenes Managementsystem an. Dieses Konzept kann noch zu spannenden Ideen führen, wird aber hier erstmal nicht weiter betrachtet.

Workflows	*Agenten*	*Gemeinsamer System-Anteil*
Workflow	Agent	**Hauptabstraktionsrichtung** einer Anwendung; Identitätsbildende Einheit per Instanziierung.
WFMS Workflow-management-system	AgMS Agenten-management-system	Ein **Managementsystem** stellt Management und Rahmenwerk (Definition, Verwaltung, Laufzeitumgebung, technische Möglichkeiten, wiederkehrende Strukturen und Funktionen, Lebenszyklus) für Anwendungen bereit.
WFES Workflow-enactment-service	APF Agenten-plattform	Eine **Laufzeitumgebung** für Anwendungen. Hier als Teil von Managementsystemen betrachtet.
WFA Workflowan-wendung	MAA (Multi-) Agenten-Anwendung	Eine **Anwendung** wird anhand einer Hauptabstraktionsrichtung entworfen. Ein Managementsystem bildet die Infrastruktur für ein solches System.
WFS Workflowsystem	MAS (Multi-) Agenten-System	Wie Anwendung, evtl. jedoch ohne einen Anwendungszusammenhang (z.B. bilden alle Agenten in einem Agentennetz ein MAS, aber nicht unbedingt eine MAA)

Auf dieser Grundlage kann PIA wie folgt charakterisiert werden: PIA ist Teil eines *Managementsystems*, das eine *Hauptabstraktionsrichtung* vorgibt, mit dem *Anwendungen* mit charakteristischen Eigenschaften entwickelt und in einer spezifischen *Laufzeitumgebung* ausgeführt werden können.

4.2.2 Integrationsschema

Das angestrebte Managementsystem soll alle Eigenschaften von Workflow- und Agententechnologie zusammenführen. In diesem Abschnitt wird ein Integrationsschema vorgestellt, bevor (nach weiteren Vorbereitungen) jeder Stufe ein eigener Abschnitt gewidmet ist. Das Integrationsschema hat fünf Stufen, deren Beschreibung durch eine Arbeit aus der Psychologie [FRAMBACH 1994] bereichert und durch die dort verwendeten charakteristischen Begriffe inspiriert ist. Frambach betrachtet in seiner Arbeit die Entwicklung von persönlicher *Identität* durch *Integration der Pole* einer dualen Polarität. Zu Beginn des Prozesses ist einer der Pole im *Vordergrund* (im Bewusstsein), der andere im *Hintergrund* (unbewusst). Der Prozess wird in fünf charakteristische *Stufen* eingeteilt, beginnend mit der Fixierung der Pole in der ersten Stufe bis hin zur Integration der Pole in der fünften Stufe. Frambach fasst die Stufen und ihre Eigenschaften aus einer Vielzahl von Quellen zusammen, so dass er letztlich ein sehr allgemeines Schema zur Integration einer beliebigen dualen Polarität vorstellt. In meiner Arbeit

geht es um die Integration der *Pole* Struktur und Verhalten einer Anwendung unter Ausbildung der *Identität von Programmteilen*. Bei großen Anwendungen ist es nicht mehr möglich, beide Pole gleichzeitig zu betrachten (d.h. ein Pol ist im *Vordergrund*, der andere im *Hintergrund*). *Integration* bedeutet hier, dass beide Sichtweisen eingenommen werden können, je nachdem, was zum Zeitpunkt der Betrachtung angemessen ist. Von Frambach werden für den folgenden Text einige einprägsame und unmittelbar hilfreiche Begriffe zur Charakterisierung von *Integrationsstufen* verwendet. Die Übertragung als solche ist ausdrücklich nicht Thema dieser Arbeit.

Zunächst sollen die Pole der Dualität noch einmal explizit benannt werden: Der eine Pol beschreibt die Struktur, den statischen Aufbau einer Anwendung und ermöglicht die Bildung einer Identität für strukturelle Komponenten, in diesem Falle Agenten. Der andere Pol beschreibt das Verhalten, die dynamischen Aspekte einer Anwendung und ermöglicht die Bildung von Identität für Prozesse, die hier als Workflows gehandhabt werden. Jede Anwendung vereint zwangsläufig beide Seiten, d.h. jede Anwendung hat sowohl eine Struktur als auch ein Verhalten. An dieser Stelle wird betrachtet, inwiefern diese Pole beim Entwurf und bei der Ausführung einer Anwendung explizit und vordergründig berücksichtigt werden. Wenn gar kein Rahmenwerk verwendet wird (d.h. es wird weder der eine noch der andere Pol explizit unterstützt), herrscht eine Beliebigkeit bei der Umsetzung und Ausführung und die Identität der Bausteine ist nicht definiert. Dann muss bei der Anwendungsentwicklung eine eigene Art der Abstraktion entwickelt werden, sonst erscheint das ganze System als ein unstrukturierter Block. Wenn beim Entwurf einer dieser Pole explizit dargestellt wird, rückt der andere damit aus der Betrachtung heraus: Ein Pol ist im Vordergrund, der andere im Hintergrund. Dieses ist die erste Stufe des Integrationsschemas. Das weitergesteckte Ziel der Integration ist es, beide Pole, also beide Sichten auf eine Anwendung zur Verfügung zu haben und einen Identitätsbegriff zu erfinden, der beide Abstraktionsrichtungen zusammenfasst. Im folgenden Text geht es um die Darstellung des Integrationsschemas mit seinen fünf Stufen. Die Ausprägung der Stufen in Form von Systemarchitekturen und die konkreten Beispiele für jede Stufe folgen dann in eigenen Abschnitten.

Stufe I Die erste Stufe besteht in der „Fixierung der Identität“ ([Frambach 1994]) von Systemteilen, z.B. in Form von Agenten. Damit steht die Struktur als identitätsbildendes Kriterium im Vordergrund und die Prozesssicht im Hintergrund. Die Struktur wird explizit dargestellt und wird beim Entwurf und bei der Ausführung als abgrenzendes Kriterium für die Identität der Systemteile verwendet. Die Prozesse einer Agentenanwendung sind nach wie vor ein wichtiger Aspekt, haben aber keine Identität. Je weniger der Hintergrundaspekt explizit vertreten ist, also in einer Agentenanwendung das Verhalten und die Prozesse, desto stärker ist die Fixierung auf den Vordergrund, die Anwendungsstruktur festgelegt. Je mehr der Hintergrundaspekt sichtbar gemacht und im Vordergrund einbezogen wird, desto näher kommt man der Grenze zur zweiten Stufe.

Stufe II In der zweiten Stufe wird ein gewisser Teil aus den vorhandenen, aber nicht explizit betrachteten Eigenschaften des Hintergrundes in die explizite Betrachtung

einbezogen. Das bedeutet z.B., dass die Verhaltens- und Prozesseigenschaften einer Agentenanwendung bereichernd in den Vordergrund rücken.

Ein Beispiel für die Bereicherung beim Entwurf sind Entwicklungsansätze aus dem Agentenumfeld, wie z.B. PAOSE (siehe Abschnitt 3.4), wo die Prozesse einer Anwendung explizit in Form von Interaktionsprotokollen in den Entwurf einbezogen werden. Diese weitere Arbeit hat die Integration zur Laufzeit zum Ziel. Ein möglicher Ansatz dazu ist die unterliegende Schicht, also das WFMS bzw. das AgMS zu betrachten: ein monolithisches WFMS auf Stufe I wird zu einem agentenbasierten WFMS auf Stufe II. Im Prinzip sollte dieser Ansatz auch umgekehrt möglich sein: ein herkömmliches AgMS auf Stufe I wird zu einem workflowbasierten AgMS auf Stufe II. Selbst wenn ein agentenbasiertes WFMS funktional äquivalent zu einem gegebenen monolithischen WFMS ist, kann die Anwendung vom flexibleren Rahmenwerk profitieren, z.B. über austauschbare Komponenten.

Gegenüber Stufe I wird die Identität der Systemteile differenzierter: Ein Agent ist nicht einfach ein Agent, sondern ein Agent mit Bezug zu bestimmten Abläufen, die benennbar sind und somit Teil des bewussten Vordergrundes sind. Es handelt sich um Agenten, die im Entwurf oder in der Ausführung von Workfloweigenschaften profitieren. In Stufe II wird also der Vordergrund (hier die Struktur) durch Berücksichtigung des Hintergrundes (das Verhalten) bereichert. Frambach benennt diese Stufe mit „Differenzierung der Identität“ ([FRAMBACH 1994]).

Stufe III In der dritten Stufe steht für die Anwendungsentwicklung sowohl ein WFMS als auch ein Agentenrahmenwerk mit voller Funktionalität zur Verfügung. So kommt es zu einer Vermischung von Workflow- und Agenteneigenschaften („Diffusion“ nach Frambach [FRAMBACH 1994]). Jedes Element der Anwendung kann entweder als Workflow oder als Agent implementiert werden. Die Stufe III bietet damit einen erheblichen Zuwachs an Möglichkeiten für die Anwendungsentwicklung, was durch eine ebenfalls erheblich erhöhte Komplexität der Anwendungsentwicklung erreicht wird, da keine Struktur zur Benutzung in der Anwendungsentwicklung durch das Rahmenwerk vorgegeben wird. In Frambachs Integrationsschema wird die Stufe III mit der gesteigerten Komplexität mit dem prägnanten Begriff „Chaos“ beschrieben. Um bei der Entwicklung einer konkreten Anwendung drohendes Chaos zu vermeiden, muss das Entwicklerteam die Verwendung der Workflows und Agenten strukturieren. Diese Strukturierung ist genau deshalb notwendig, weil sie im Rahmenwerk fehlt.

Der Begriff der Identität von Systemteilen bekommt gegenüber Stufe II noch stärkere Bedeutung, weil alle Anteile einer Anwendung explizit und vollständig in Form von Workflows oder Agenten sichtbar sind und eine eigenständige Identität erhalten.

Stufe IV In der vierten Stufe werden die Möglichkeiten der dritten Stufe kreativ für ein Rahmenwerk mit einer strukturierten Schnittstelle genutzt. Bei einer erfolgreichen Anwendungsentwicklung auf Stufe III werden vom Entwicklerteam Regeln erstellt, wann ein Workflow und wann ein Agent als Modellierungseinheit verwendet wird. Wenn diese Regeln umfassend und ausgereift genug sind, so dass z.B. sämtliche Workflows einer Anwendung anhand dieser Regeln erstellt werden, erreicht die Anwendung eine hohe Transparenz und Klarheit. Diese Regeln werden für Stufe IV

automatisiert, so dass sie vom Rahmenwerk realisiert werden und damit ihre Einhaltung durchgängig sichergestellt ist.

Auf Stufe IV werden die Elemente der einen Abstraktionsrichtung komplett vom Rahmenwerk erzeugt und ausgeführt und für die Anwendungsentwicklung wird nur noch die andere Abstraktionsrichtung bereitgestellt. Damit stehen für die Anwendungsentwicklung (wie in Stufe I oder II) nur *entweder* Workflows *oder* Agenten zur Verfügung. Zur Laufzeit hat die Anwendung jedoch dieselbe Funktionsweise wie auf Stufe III. Die Anwendungsentwicklung ist auf Stufe IV aber einfacher, weil die Elemente der einen Abstraktionsrichtung automatisch vom Rahmenwerk erstellt und verwaltet werden. Die Freiheiten bei der Anwendungsentwicklung werden also gegenüber Stufe III eingeschränkt, die Ausdrucksstärke wird mit dem Entwurf des Rahmenwerkes konkretisiert und festgelegt.

In der Gestaltung der Programmierschnittstelle besteht vollständige Freiheit: ein Rahmenwerk benutzt die Funktionen eines WFMS und eines Agentenrahmenwerkes, um entweder für Agenten oder für Workflows beliebige Funktionen verfügbar zu machen. Diese „Beliebigkeit“ besteht natürlich nur für die Gesamtheit der Systeme, die Stufe IV entsprechen und resultiert aus den enormen Möglichkeiten, die Stufe III bereits bietet.[2]

Der Identitätsbegriff wird durch die Einfachheit und Klarheit der Schnittstelle gegenüber Stufe III deutlich klarer. In Stufe III hat zwar jedes Systemteil eine eigene Identität, diese ist aber durch die Verfügbarkeit von Workflow- und Agentenidentitäten nicht einheitlich, was durch die Vereinfachung der Schnittstelle behoben ist. Die Stufen I und II haben ebenfalls einen einheitlichen Identitätsbegriff, jedoch ist die Ausdrucksmächtigkeit in Stufe IV erheblich größer.

Stufe V Stufe V bildet die Synthese der beiden Pole zu einer flexiblen Polarität („Integration“ nach [Frambach 1994]), zu einer neu gebildeten Einheit. In dieser Einheit sind beide Pole explizit sichtbar und stehen beide vollständig zur Verfügung. Die Identität ist einheitlich und durchgängig verfügbar.

Durch die auf den vorigen Stufen gewonnene Klarheit und durchgehende Identität von Systemteilen ist es auf Stufe V möglich, einen Identitätsbegriff für eine flexible Polarität aus Workflow und Agent zu schaffen: Jeder Systemteil kann in jedem Stadium sowohl als Agent als auch als Workflow betrachtet werden. Beide Abstraktionsrichtungen stehen für die Anwendungsentwicklung vollständig und wahlfrei zur Verfügung (anders als in den Stufe I, II und IV), dabei ist die Schnittstelle einfach (anders als in Stufe III). Dieses weitergesteckte Ziel, die Integration der Polarität von Verhalten und Struktur, wird durch einen neuen Identitätsbegriff „Agent/Workflow“ erreicht.

Auf der folgenden Doppelseite wird die Charakterisierung der Stufen in Schichtenarchitekturen übersetzt.

[2] Frambachs Bezeichnung „Identitätsvakuum“ [Frambach 1994] für die Stufe IV ist hier nicht hilfreich, da die Übertragung auf Eigenschaften einer Software-Architektur nicht gelingt. Mir kommt es hier aber nicht auf die Übertragung der Begriffe an, sondern auf die erwünschten charakteristischen Eigenschaften der Stufe IV.

4.2.3 Architekturüberblick

Im Folgenden werden die Stufen auf konkrete Schichten-Architekturen übertragen. Die Schichten der jeweiligen Architektur werden für den Überblick nur benannt und in Abbildung 4.1 dargestellt, bevor sie in den Abschnitten 4.3 – 4.7 einzeln betrachtet werden und ihre konkrete Ausprägung dargestellt wird.

Stufe I In der ersten Stufe wird eine Anwendung durch die Verwendung eines Rahmenwerkes strukturiert: Eine Workflowanwendung baut auf einem WFMS auf und eine Agentenanwendung baut auf einem AgMS (im Sinne obiger Definition 4 auf Seite 61) auf und es entsteht eine zweischichtige Architektur.

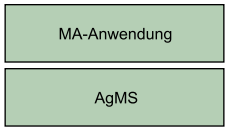
Stufe II In der zweiten Stufe kommt eine Schicht zur Architektur hinzu, indem das Rahmenwerk differenziert wird (als werkzeugorientierte Umsetzung der zweiten Stufe). Das wäre für Workflows ein agentenbasiertes Workflowmanagementsystem. Für Agenten wäre das ein Agentenrahmenwerk, das auf Workflowbasis entworfen wurde (eher unüblich).

Stufe III In der dritten Stufe werden beide Technologien direkt für die aufsetzende Anwendung verfügbar gemacht. Statt dem Agentenrahmenwerk ein WFMS beziehungslos an die Seite zu stellen, erlauben wir ausgehend von Stufe II den Durchgriff auf die unterste Schicht. Betrachtet wird nur die verbreitete Variante mit Agenten als unterster Schicht.

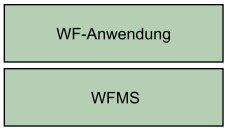
Stufe IV In der vierten Stufe kommt eine weitere Schicht zur Architektur hinzu, indem die Anwendung in zwei Schichten geteilt wird. Die obere dieser Schichten enthält alle anwendungsspezifischen Teile. Der andere Teil koordiniert die generelle Verwendung von Agenten und Workflows und wird in einer eigenen Integrationsschicht ausgegliedert, so dass zur Anwendungsentwicklung wieder eine einfache Schnittstelle angeboten wird. Die neue Schnittstelle zur Anwendungsentwicklung kann entweder Agenten oder Workflows unterstützen, da beide Grundformen innerhalb des Rahmenwerkes vertreten sind. Die übernommenen Funktionen, die nicht neu gestaltet werden, können von der Integrationsschicht einfach durchgereicht und nach oben angeboten werden.

Stufe V In der fünften Stufe ist die Integrationsschicht in zwei voneinander abhängige Teile geteilt. Der untere Teil leistet die Integration des WFMS und des AgMS wie in Stufe IV, bietet aber *beide* erweiterten Schnittstellen in geeigneter und passender Weise nach oben an. Der obere Teil bietet eine einheitliche Schnittstelle zur Anwendungsentwicklung an, so dass für jede Einheit einer Anwendung jederzeit beide Perspektiven gleichberechtigt eingenommen werden können.

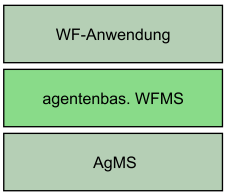
Die detaillierte Betrachtung der Stufen wird im Folgenden durch die Beschreibung der Farbgebung und durch die Einführung von Schnittstellensymbolen vorbereitet.



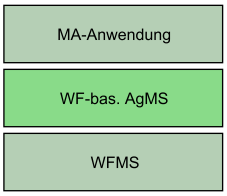
Stufe I: Agentenmanagementsystem



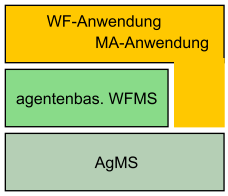
Stufe I: Workflowmanagementsystem



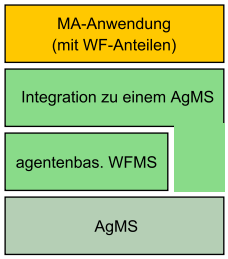
Stufe II: Agentenbasiertes WFMS



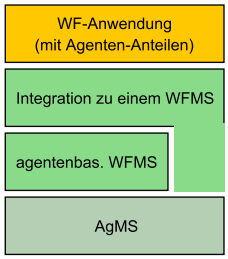
Stufe II: WF-basiertes AgMS



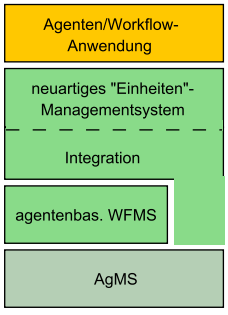
Stufe III: Mischform für Agenten und Workflows



Stufe IV: WF-Agenten



Stufe IV: Agenten-WFs



Stufe V: Neuartiges Managementsystem

Abbildung 4.1: Fünf Stufen der Architektur im Überblick.

4.2.4 Syntax

Die detaillierte Darstellung der Stufen wird durch wiederkehrende syntaktische Elemente in Graphiken unterstützt, und zwar werden Schnittstellentypen mit Symbolen codiert, Farben werden in gleicher Bedeutung verwendet und für die Architektur-Blockbilder und Laufzeit-Diagramme werden wiederkehrende Bildtypen verwendet.

Schnittstellencodierung

Die Architekturen bestehen aus mehreren Schichten. Jede Schicht nutzt die Schnittstelle der darunterliegenden Schicht und bietet selbst eine Schnittstelle zur Nutzung an (die Schichten können in diesem Sinne als Subsysteme wie in [Lilienthal 2007] betrachtet werden). Dabei gibt es Typen von ähnlichen Schnittstellen, z.B. bietet ein WFMS eine grundsätzlich andere Schnittstelle an als ein AgMS. Die Schnittstellen zweier verschiedener Workflowmanagementsysteme können sich dabei stark unterscheiden, der Teil jedoch, der quasi allen Workflowmanagementsystemen gemeinsam ist, wird hier zu einem *Schnittstellentyp* zusammengefasst. Dasselbe gilt für Agentenmanagementsysteme: Ein Agent wird in Capa ganz anders programmiert als in Jade, dennoch haben Capa und Jade denselben Schnittstellentyp.

Die Symbole für die Schnittstellentypen sind im Folgenden wie folgt verwendet (bei jeder Verwendung wird eine Legende mit allen Symbolen mit angezeigt):

Eine Programmiersprache bietet eine allgemeine Schnittstelle für Programme aller Art. Hier sind höhere Programmiersprachen gemeint. An der Stelle der Programmiersprache in der Hierarchie der Infrastruktur beginnen die Betrachtungen dieser Arbeit, die Hardware und das Betriebssystem werden nicht betrachtet. In dieser Arbeit wird Java verwendet, andere Programmiersprachen kommen für konkrete Systeme jedoch genauso in Frage.

Eine spezialisierte Programmiersprache kann einem darauf implementierten System eine gewisse Betonung von Aspekten verleihen. In dieser Arbeit werden die Referenznetze als spezialisierte Programmiersprache bezeichnet. Es ist mit Referenznetzen möglich, beliebige Programme zu schreiben, sie betonen jedoch die Abläufe und Unabhängigkeit leichter als eine allgemeine Programmiersprache wie Java. Dieses Symbol wird außerdem für die Programmierschnittstelle (API) eines zu erweiternden Systems verwendet.

Ein Workflowmanagementsystem bietet als Gesamtschnittstelle die Zusammenfassung der fünf von der WfMC definierten Schnittstellen (WAPI). Jede auf einem Workflowmanagementsystem basierende Anwendung nutzt diesen Schnittstellentyp. Die Anwendung ist entlang von Prozessen (Workflows) entwickelt, das ist die *Hauptabstraktionsrichtung* beim Entwurf und bei der Ausführung der Anwendung.

Die gesamte Funktionalität zur Spezifikation und Ausführung von Multi-Agenten-Systemen wird als ein Schnittstellentyp aufgefasst. Da ein Multi-Agenten-System auch auf Agenten von verschiedenen Agentenrahmenwerken basieren

kann, ist dieser Schnittstellentyp sehr weit gefasst. Eine Anwendung ist entlang autonomer Komponenten (den Agenten) entwickelt, das ist die *Hauptabstraktionsrichtung* beim Entwurf und bei der Ausführung der Anwendung. In dieser Arbeit handelt es sich um FIPA-kompatible Agenten, d.h. dass Teile dieser Schnittstelle (wie die Kommunikationsarchitektur) von der FIPA spezifiziert sind, wie im Abschnitt 2.2.2 vorgestellt. Andere Teile wie die Entscheidungsarchitektur sind von AgMS zu AgMS verschieden.

Farbcodierung

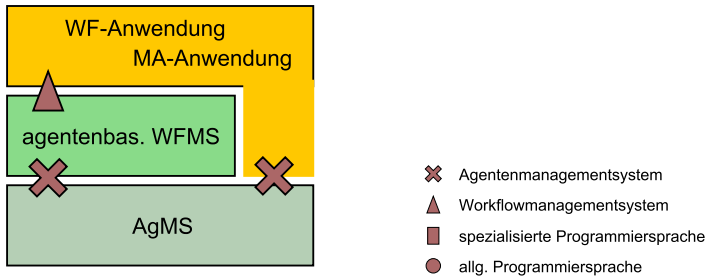
Die Farben der Graphiken sind wie folgt verwendet:

- grau in Blockbildern für bereits vorhandene Bestandteile, die nicht Gegenstand der Betrachtung sind
- grün in Blockbildern für die „interessanten“ Bestandteile, die z.B. im Laufe der weiteren Arbeit entworfen werden oder durch die hier entworfenen Bestandteile erweitert oder ersetzt werden
- orange in Blockbildern für Anwendungen, die eine eingehende Betrachtung verdienen, aber nicht Hauptgegenstand der Betrachtung sind. Sie können im Kapitel 6.3 wieder aufgegriffen werden, sofern es sich um konkrete Beispiele handelt.
- dunkelgraue schmale Balken in Laufzeitabbildungen für die nicht einzeln dargestellten Schichten auf konkreten Rechnern (z.B. Hardware, Betriebssystem, Java)
- hellgrau in Laufzeitabbildungen für Anwendungen oder Subsysteme, die sich aus mehreren Komponenten zusammensetzen
- braun für das konkret entworfene agentenbasierte WMFS (z.B. Abbildungen 4.11 und 4.12)

Bildtypen

Die Architektur wird auf jeder Stufe mit einem Blockbild dargestellt wie in Abbildung 4.2a auf der nächsten Seite. Jede Schicht kann für sich entwickelt werden unter Verwendung der direkt darunterliegenden Schichten, sowie Java und Referenznetze aus den unterliegenden Schichten. Alle anderen Schichten können nur dann benutzt werden, wenn eine Schicht explizit nach unten durchreicht. Dies bedeutet, dass die obere Schicht von der unteren Schicht charakteristisch und in ihrem Wesen abhängig ist.

Die Legende für die Schnittstellensymbole (Abbildung 4.2b auf der nächsten Seite) wird jedesmal vollständig angegeben. Die Schnittstellensymbole zwischen den Schichten beschreiben wie oben erläutert den *Typ* einer Schnittstelle. Dasselbe Symbol an verschiedenen Stellen bezeichnet nur einen gemeinsamen Typ, die konkreten Schnittstellen können dagegen große qualitative Unterschiede aufweisen.



(a) (b)

Abbildung 4.2: Verwendung von Blockbildern mit Legende

Ein beispielhafter Schnappschuss der Laufzeitkomponenten wird wie in Abbildung 4.3 dargestellt. Zuunterst sind einzelne Computer als Balken dargestellt. Darauf laufen Bestandteile der Anwendung, die vollständig einem Computer zugeordnet werden können und als eigenständig begriffen werden. Diese lokalen Komponenten werden als *konkrete* Komponenten bezeichnet, um sie von den *zusammengesetzten* Komponenten zu unterscheiden, die potentiell verteilt ausgeführt werden (als breiter Balken dargestellt). Jede zusammengesetzte Komponente wird durch eine Anzahl von konkreten Komponenten ausgeführt. Die Wolke „Agentcities“ dient dazu, dem Betrachter die flexible Kombinierbarkeit in Erinnerung zu bringen.

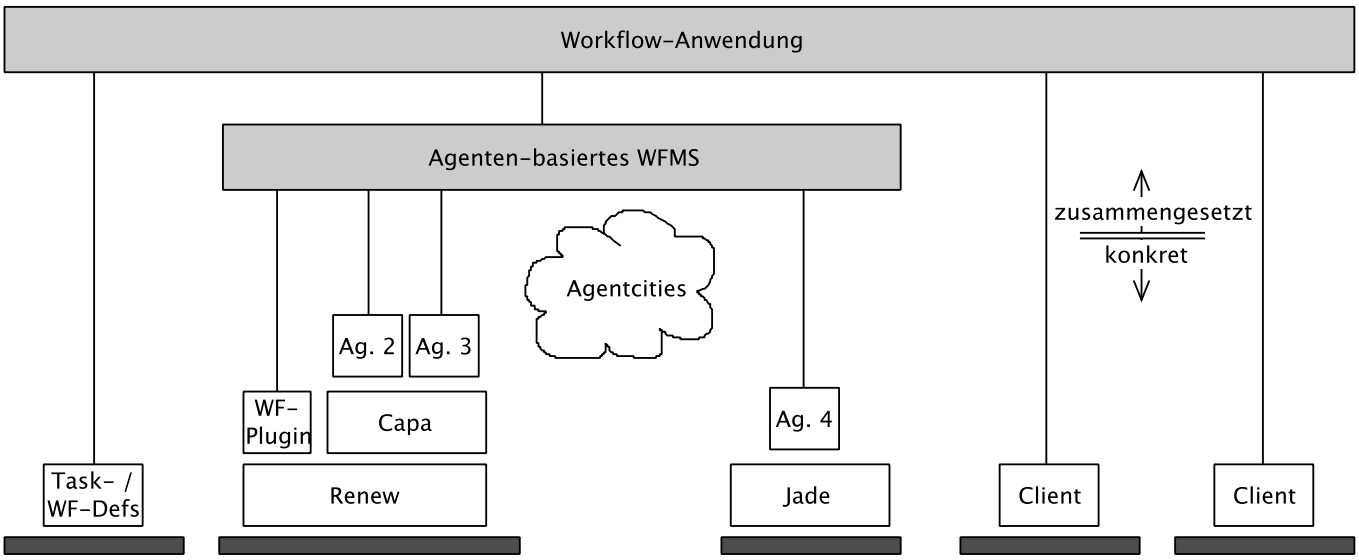


Abbildung 4.3: Verwendung von Laufzeitbildern

Es werden an wenigen Stellen abweichende Abbildungen aus früheren Veröffentlichungen vorgestellt, welche jeweils einen speziellen Aspekt der entsprechenden Stufe gut darstellen.

4.3 Stufe I

In der ersten Stufe geht es gegenüber unstrukturierter Anwendungsentwicklung um die explizite Festlegung von Systemteilen, was grundsätzlich sowohl entlang der Struktur als auch entlang des Verhaltens geschehen kann. Die Systemteile erhalten eine Identität, welche sowohl beim Entwurf als auch bei der Ausführung berücksichtigt werden soll. Objektorientierte Programmierprachen bilden einen wichtigen Schritt Richtung Identität bezogen auf die Struktur einer Anwendung. Die Agentenorientierung führt dies auf einer höheren Abstraktionsebene weiter. Die Workflowmanagementsysteme gehen in die andere Richtung: Vorgänge und Prozesse bekommen eine Identität beim Entwurf und bei der Ausführung zugewiesen.

Mit dieser Anreicherung durch Identität kann auch ein ganzer Software-Entwicklungsansatz einhergehen, wie dies speziell für die Agentenorientierung häufig der Fall ist. Mit zunehmender Verbreitung spielen auch Standardisierungen eine wichtige Rolle.

Die Vorteile sind heute allgemein anerkannt. Für Agentenanwendungen verbessern sich die Änderbarkeit, die Wartbarkeit und die Interoperabilität (durch die Standards der FIPA). Die Kapselung und asynchrone Kommunikation erleichtern die verteilte Ausführung von Anwendungen: prinzipiell kann jeder Agent auf einem eigenen Standort ausgeführt werden. Da die Kommunikation asynchron und standardisiert ist, kann ein Agent auch relativ leicht mit einem anderen Rahmenwerk implementiert sein, eine Anwendung kann also heterogen aufgebaut sein. Für Workflowanwendungen liegen die Vorteile im Monitoring und Controlling der Prozesse: die Prozesse der Anwendung können explizit beobachtet und gesteuert werden, die Einhaltung von Regeln kann sichergestellt werden. Die Geschäftslogik (Definition der Workflows) ist von der Ausführung durch einzelne Tasks und Clientprogramme getrennt. Das Konzept der Autorisierung ist explizit berücksichtigt.

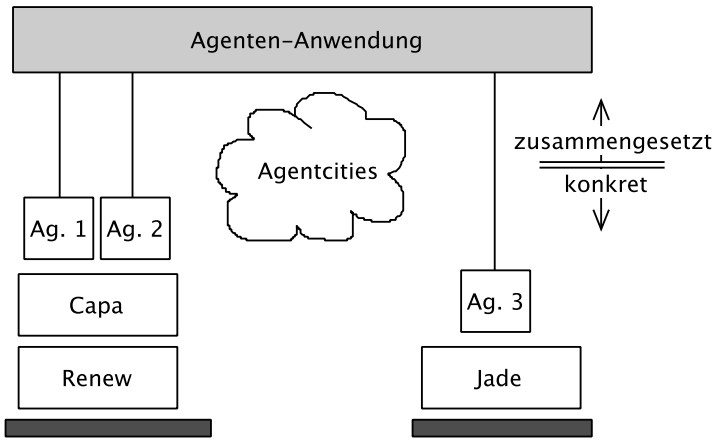


Abbildung 4.4: Stufe I: Beispiel für eine Agentenanwendung

Eine Agentenanwendung besteht wie in Abbildung 4.4 dargestellt aus mehreren Agenten. Die Anwendung selbst ist dabei eine zusammengesetzte Einheit, d.h. sie kann

nicht einem konkreten Rechner oder einem Prozess zugeordnet werden. Die Beziehung zwischen Anwendung und Agentenplattform wird in Abbildung 4.5 verdeutlicht: Eine Agentenplattform muss nicht vollständig zu einer Agentenanwendung dazugehören. Das Besondere an der Standardisierung für Agentenkommunikation ist, dass die

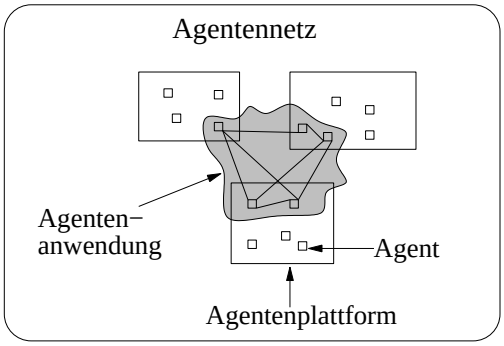


Abbildung 4.5: Stufe I: Agentenanwendung im Agentennetz. (Wdhlg. von Abb. 3.1)

Agenten einer Anwendung nicht nur auf verschiedenen Rechnern laufen können, sondern auch auf verschiedenen Agentenplattformen wie Capa oder Jade. Weiter gibt es offene Agentennetze wie Agentcities, wodurch auch offene Agentenanwendungen realisiert werden können, d.h. es ist nicht von vornherein festgelegt, wer mit welchem Agentenrahmenwerk an der Implementierung einer Anwendung beteiligt ist.

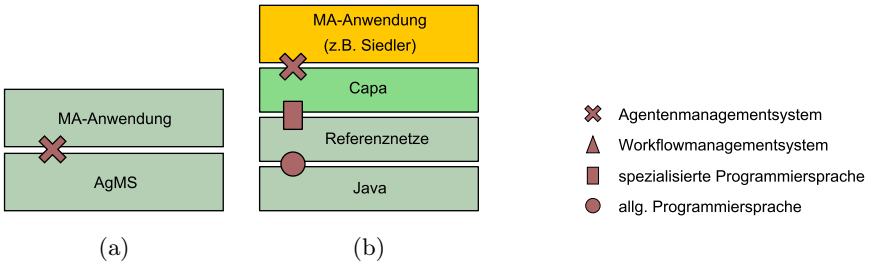


Abbildung 4.6: Stufe I im Detail: Reines Agentensystem

Für Capa-Agenten ist die zweischichtige Architektur aus der Übersicht (Abbildung 4.6a) wie in Abbildung 4.6b verfeinert. Java bietet dabei die allgemeine Programmierschnittstelle samt Laufzeitumgebung für das gesamte System. Renew bietet eine spezialisierte Programmierschnittstelle (unter Verwendung von Java) an: Über Petrinetze können Abläufe, Nebenläufigkeit und (Un-) Abhängigkeit dargestellt werden. Bei den hier verwendeten Referenznetzen kann bei einer entsprechenden Verwendung der synchronen Kanäle auch eine Enthaltenseins-Beziehung zwischen Elementen einer Anwendung modelliert werden. Capa ist schließlich mit Referenznetzen und Java implementiert und bietet als Gesamtheit (also inklusive Java und Referenznetzen) eine Schnittstelle zum Entwurf, zur Spezifikation und zur Ausführung von Agenten an.

Die agentenorientiert umgesetzte Version des Brettspiels Siedler ist eine typische Agentenanwendung: Die Agenten verstehen jeder einen Satz von ACL-Nachrichten

und können alle zusammen auf einem Rechner oder verteilt auf verschiedenen Rechnern ausgeführt werden. Insbesondere die Spieler-Agenten können (und sollen) von anderen Partien implementiert werden, so dass ein heterogenes System entstehen kann. Eine Agentenanwendung in CAPA spezifiziert Agenten über Protokollnetze, Entscheidungskomponenten und Wissensbasen wie in den Abschnitten 2.2.3 (CAPA), 2.2.4 (MULAN) und 3.4 (PAOSE) dargestellt. In einer Agentenanwendung wie Siedler gibt es intuitiv verständliche Einheiten (Spieler, Bank, Insel) mit ihren jeweiligen Interna. Weiter gibt es in jeder Agentenanwendung übergreifende Prozesse (Bauen, Handeln), deren Regeln und Teilschritte auf mehrere Agenten verteilt implementiert sind.

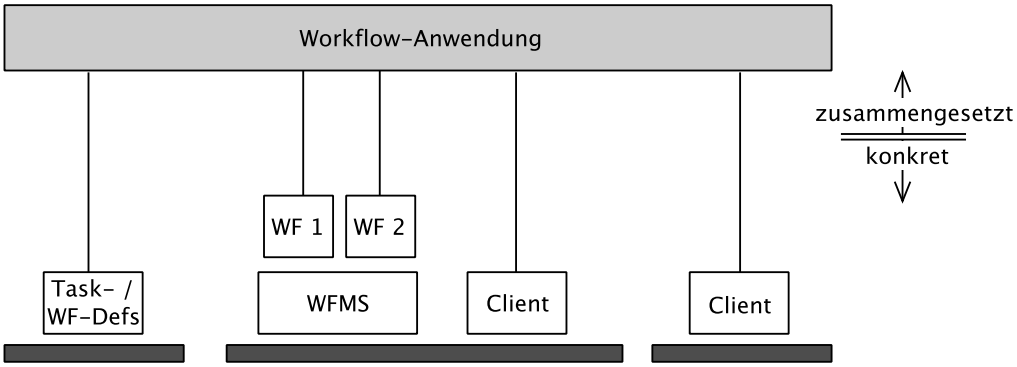


Abbildung 4.7: Stufe I: Beispiel für eine Workflowanwendung

Eine Workflowanwendung besteht wie in Abbildung 4.7 dargestellt aus dem Workflowmanagementsystem, den Spezifikationen für eine konkrete Workflowanwendung (Taskdefinitionen, Rollen, Rechte, Workflowdefinitionen) und den Clients. Die Datenbank mit den anwendungsspezifischen Daten kann dabei auf einem eigenen Rechner liegen. Es könnte auch eine Datenbank mit Falldaten noch ausgegliedert werden, solche Möglichkeiten betreffen den Bereich der Datenbankanwendungen, der hier nicht Gegenstand der Betrachtung ist. Die Workflowanwendung als Ganzes kann hier nicht mehr einem Rechner oder einem Prozess zugeordnet werden und ist also als eine zusammengesetzte Einheit zu betrachten.

Für das Workflowmanagementsystem von Thomas Jacob (vgl. Abschnitt 2.3.4 auf Seite 48; implementiert im „Workflow-Plugin“ für RENEW) ist die zweischichtige Architektur (Abbildung 4.8a) wie in Abbildung 4.8b verfeinert: Die Laufzeitumgebung wird wieder von Java bereitgestellt, RENEW stellt die spezialisierte Programmiersprache der Referenznetze zur Verfügung. Die nächste Schicht nutzt die ersten beiden, um die alle anwendungsunabhängigen Funktionen eines WFMS zur Verfügung zu stellen. In diesem Falle bedeutet das, dass eine Datenbankanbindung für Task- und Workflowdefinitionen u.s.w., eine spezielle Task-Transition zur Ausführung von Workflows sowie einfache Standard-Clients vom Workflow-Plugin zur Verfügung gestellt werden. Die Workflowanwendungsschicht implementiert dann alle anwendungsspezifischen Komponenten, evtl. auch spezialisierte Clients.

Auf diesem WFMS wurde die Anwendung „PhoneShop“ implementiert. PhoneShop modelliert einen Telefonversand. In der Datenbank werden Subrollen für Mitarbei-

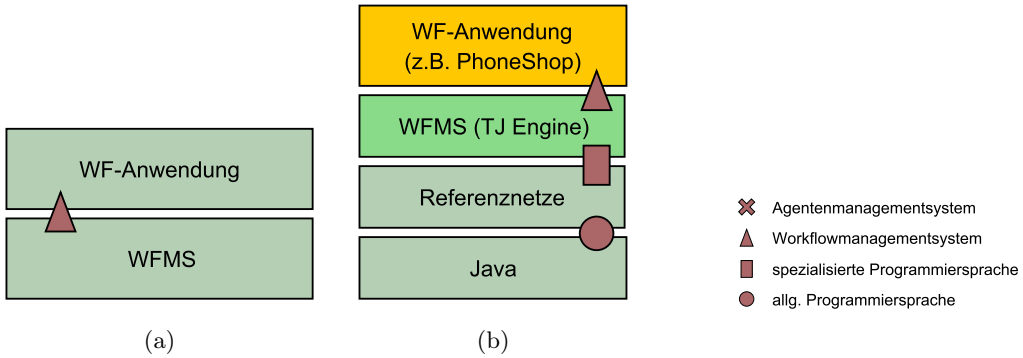


Abbildung 4.8: Stufe I: Reines Workflowmanagementsystem

ter wie Verpacker, Versender, Prüfer definiert, es wird ein Workflow mit zwei Sub-Workflows für den Ablauf einer Telefonbestellung definiert sowie einige Tasks und Datenstrukturen (z.B. „Auftrag“). Während der Ausführung können z.B. weitere Mitarbeiter eine Zugangsberechtigung erhalten oder ihre Rollen können sich ändern. Zu jedem Zeitpunkt in der Ausführung kann man unter anderem beobachten, wie viele Bestellungen gerade bei einem bestimmten Task bearbeitet werden. Es gibt also in einer Workflowanwendung explizit modellierte Prozesse (hier z.B. Zahlungsdaten prüfen, Versand) mit ihren Interna. Daneben gibt es Einheiten (Sekretariat), deren Aufgaben in verschiedenen Prozessen definiert sind.

Die Nachteile der beiden Ausprägungen der Stufe I sind insbesondere durch die fehlenden Vorteile der jeweils anderen Ausprägung deutlich: Die Verteilung von Workflowanwendungen bezieht sich zunächst auf die entfernte Anbindung von Clients, dann allgemein auf die Ausführung von Tasks. Grundlage ist jedoch meistens eine proprietäre Schnittstelle, so dass Verteilung im Sinne heterogener Implementierung viel schwieriger ist als bei Agentensystemen. Weiter ist es in Agentenanwendungen so, dass durch das Konzept der Agenten als flexible, autonome Einheiten die Flexibilisierungspunkte einer Anwendung von vornherein einbezogen sind, die Flexibilisierung hat gewissermaßen einen „natürlichen“ Platz in der Anwendung. Die Prozessunterstützung in Agentensystemen verblasst allerdings im Vergleich zu den Möglichkeiten eines Workflowsystems: die übergreifenden Prozesse sind ein wesentlicher Teil einer Agentenanwendung, diese werden jedoch zur Laufzeit nicht ausreichend durch explizite Repräsentation und Beobachtbarkeit unterstützt (vgl. Abschnitt 3.6).

Einige der jeweiligen Vorteile können über die Schichtenarchitektur integriert werden, indem z.B. das WFMS auf Agentenbasis entwickelt wird. Dies ist Gegenstand der Stufe II.

4.4 Stufe II

In der zweiten Stufe geht es gegenüber der starren Vordergrund- / Hintergrundtrennung aus Stufe I darum, Aspekte aus dem Hintergrund bereichernd einzubeziehen. Wenn

eine Architektur wie in Stufe I gegeben ist, kann dies immerhin noch methodisch über einen Entwicklungsansatz gehen:

Der im Abschnitt 3.4 vorgestellte Entwicklungsansatz PAOSE ist ein Beispiel für eine weitreichende Berücksichtigung der Prozesssicht für Agentenanwendungen: Der Entwurf der Interaktionsprotokolle ist ein wesentlicher Schritt beim Vorgehen. Aus den Diagrammen werden Gerüste für die Protokollnetze automatisch erzeugt, die Interaktionsprotokolle werden also explizit für die Codegenerierung verwendet. Durch Ansätze zum Roundtrip-Engineering, bei denen die Zusammengehörigkeit von Protokollnetzen und Interaktionsprotokollen erhalten bleibt, können strukturelle Änderungen in den Protokollnetzen sogar automatisch in das Interaktionsprotokoll übernommen werden. So werden die übergreifenden Prozesse einer Agentenanwendung beim Systementwurf sehr weitreichend explizit unterstützt.

In dieser Arbeit geht es aber nicht um die Entwicklung einer Methode, sondern um die Entwicklung einer Architektur. Für die Berücksichtigung der Hintergrundsicht durch die *Architektur* des Rahmenwerkes werden zunächst die charakteristischen Eigenschaften von Agentenanwendungen und Workflowanwendungen betrachtet. Agentenanwendungen sind aus autonomen Einheiten zusammengesetzt, die asynchron miteinander kommunizieren und damit in gewissem Maße lose gekoppelt sind. Agentenanwendungen können leichter verteilt ausgeführt werden und die Implementierung einzelner Agenten kann sich ändern. Insbesondere sind auch heterogene Agenten konzeptionell leicht zu berücksichtigen, da von dem Kommunikationsstandard der FIPA ausgegangen werden kann. Eine Workflowanwendung hat andere Eigenschaften: Die Prozesse der Anwendung werden explizit beschrieben und die Ausführung kann kontrolliert, gesteuert und protokolliert werden. Wenn nun ein WFMS selbst mit Agenten entworfen wurde, dann handelt es sich um ein agentenbasiertes WFMS. Das WFMS hat damit die oben aufgezählen Vorteile einer Agentenanwendung. Die Workflowanwendung, die auf dem WFMS aufsetzt, ist weiterhin „nur“ eine Workflowanwendung, kann aber ebenfalls von den Vorteilen aus der Agentenorientierung (z.B. leichtere Verteilbarkeit) profitieren.

Abbildung 4.9 zeigt diese Kombination von Agententechnologie und Workflowtechnologie durch ein agentenbasiertes Workflowmanagementsystem. Das WFMS besteht als zusammengesetzte Einheit aus mehreren Agenten, im Gegensatz zu Stufe I, wo das WFMS als eine konkrete Einheit dargestellt wurde. Natürlich hat jedes konkrete WFMS eine interne Struktur, die evtl. zur verteilten Ausführung des WFMS geeignet ist; in dem Moment jedoch, wo das WFMS als agentenbasiertes WFMS implementiert ist, ist das Konzept der verteilten Ausführung verfügbar, ohne weitere Details des WFMS benennen zu müssen.

Die konkrete Schnittstelle zwischen WFMS und Workflowanwendung kann für ein agententenbasiertes WFMS genauso aussehen wie für ein WFMS auf Stufe I. Auf jeden Fall bleibt der Schnittstellentyp zwischen WFMS und Anwendung erhalten, wie Abbildung 4.10 darstellt. Das WFMS nutzt seinerseits die Schnittstelle eines Agentenmanagementsystems, so dass im Überblick eine dreischichtige Architektur entsteht.

Wie wird ein agentenbasiertes WFMS aufgebaut? Für CAPA-Agenten steht das MULAN-Modell zur Verfügung, so dass das Konzept der hierarchischen Schachtelung zur Verfügung steht (vgl. Abschnitte 3.2 auf Seite 79 für geschachtelte Strukturen, 2.2.4 für die Einführung in MULAN und 5.1.1 für die Umsetzung des agentenbasierten

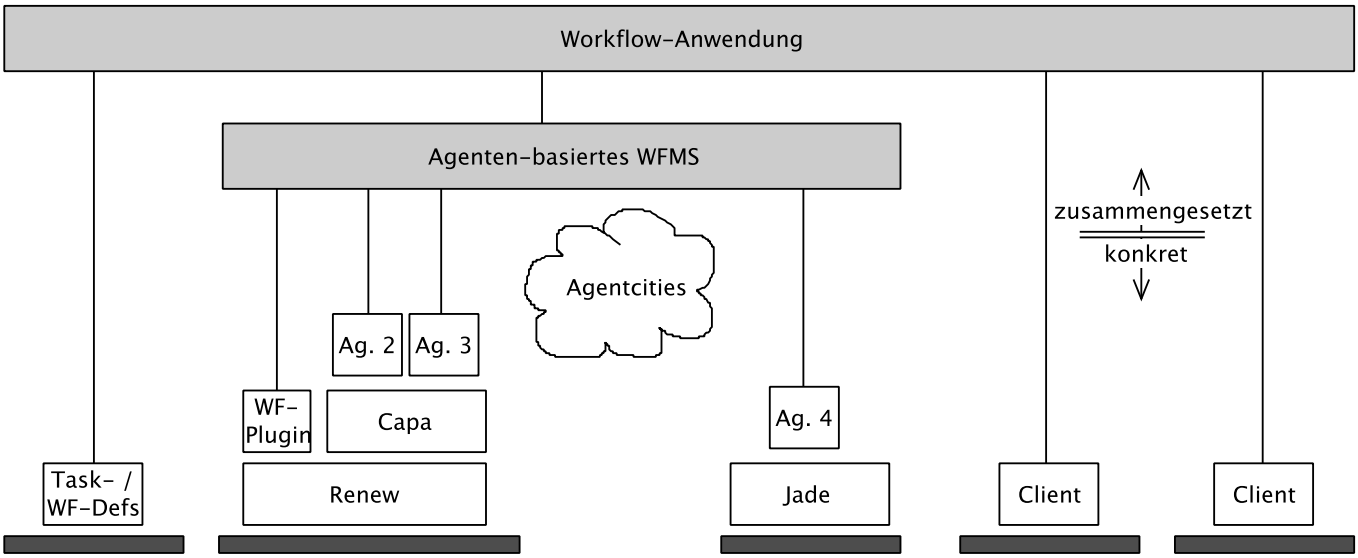


Abbildung 4.9: Stufe II: Beispiel für ein agentenbasiertes WFMS

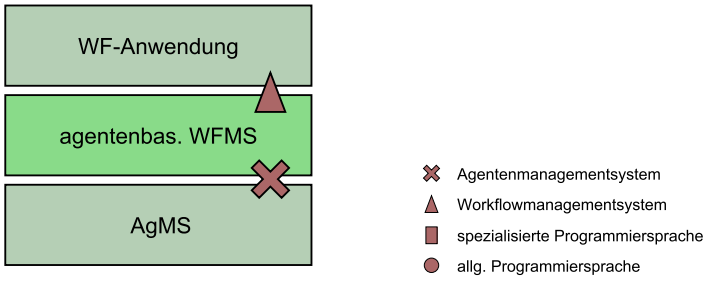


Abbildung 4.10: Stufe II: Ein agentenbasiertes WFMS mit Schnittstellentypen

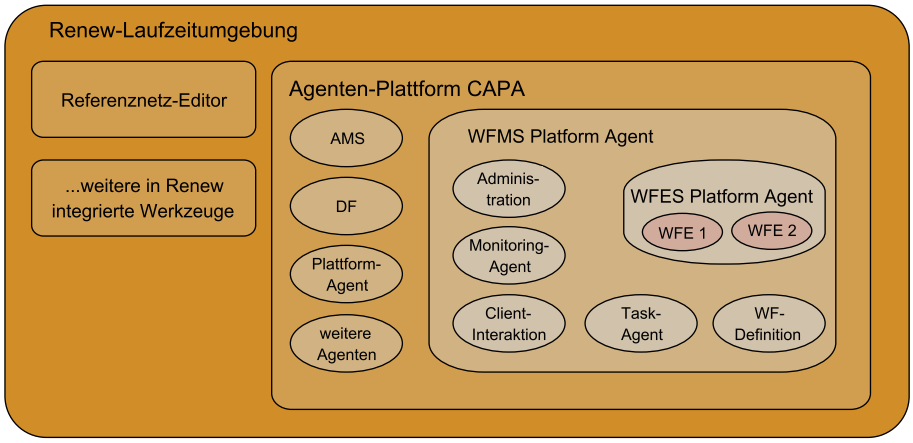
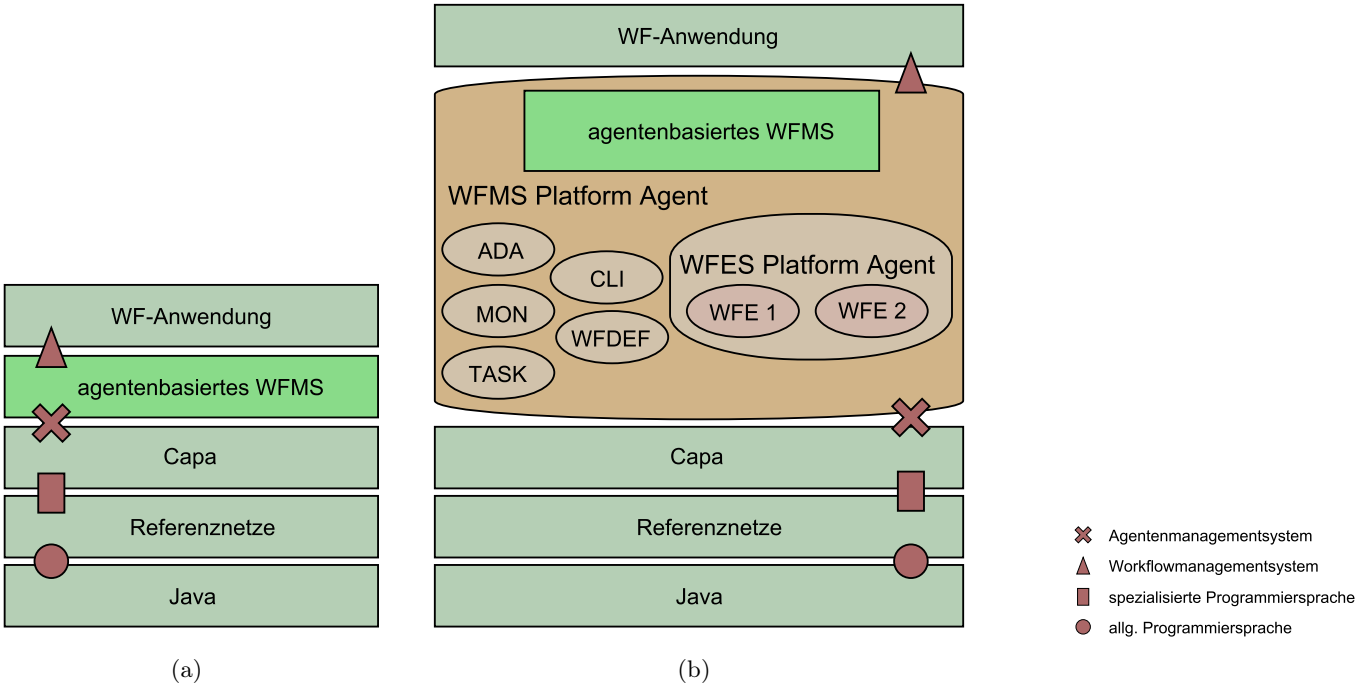


Abbildung 4.11: Stufe II: Struktur eines agentenbasierten WFMS

WFMS). Es liegt also nahe, einen einzelnen WFMS-Agenten zu entwerfen, der durch mehrere Agenten implementiert wird. Abbildung 4.11 zeigt sechs Agenten für die verschiedenen funktionalen Teile eines WFMS. Der Enactment-Service-Agent (WFES) beherbergt die einzelnen Workflowengine (WFE-) Agenten. Der WFMS-Agent ist in Renew wie folgt eingebettet: Capa läuft (neben anderen Plugins) in der Laufzeitumgebung von Renew. In Capa laufen die Standard-Agenten (AMS und DF) sowie der Plattform-Agent und weitere mögliche Agenten neben dem eigentlichen WFMS-Agent. Abbildung 4.12a zeigt das entsprechende Architekturbild: das agentenbasierte WFMS nutzt die Schnittstelle von Capa zur Definition und Ausführung von Agenten. Zur Einordnung ist im Bildteil 4.12b der WFMS-Agent aus Abbildung 4.11 mit seinen Sub-Agenten in das Architekturbild anstelle des agentenbasierten WFMS eingefügt.



(a) (b)

Abbildung 4.12: Stufe II: Architektur eines agentenbasierten WFMS. Die Agentennamen in Bildteil (b) sind Abkürzungen der entsprechenden Agenten in Abbildung 4.11.

Im Prinzip können dieselben Überlegungen, die hier für agentenbasierte WFMS dargestellt wurden, auch für workflowbasierte Agentenmanagementsysteme durchgeführt werden. Das wäre ein Agentenmanagementsystem zur Definition und Ausführung von Agenten, welches aus Workflowdefinitionen zusammengesetzt ist. Auf Ebene der Agentenplattform könnten dann die einzelnen Vorgänge (Erzeugen eines Agenten, Empfang einer Nachricht) besonders einfach mit den Mitteln des WFMS protokolliert werden oder für bestimmte Agenten könnten Rechte über das Rollenmodell des WFMS eingeschränkt werden. So etwas gibt es in der Literatur nicht. Es sind zwar etliche Arbeiten zur Verbindung von Agenten und Workflows vorhanden, jedoch keine Arbeit zu einem workflowbasierten Agentenrahmenwerk. Dies liegt daran, dass die

Workflowtechnologie für Geschäftsprozesse und nicht für die allgemeine prozessorientierte Programmierung verwendet wird. Die konkrete Gestaltung hängt von einem konkret verwendeten WFMS ab. Für die vorliegende Arbeit steht kein solches generisches und skalierbares WFMS zur Verfügung, dafür aber ein generisches und auf allen Ebenen anwendbares Agenten-Modell: MULAN. Aus diesem Grunde wird diese unübliche Variante nicht eingehender betrachtet. Statt dessen wird angenommen, dass es möglich ist, ein Agentenrahmenwerk mit Hilfe eines WFMS zu implementieren.

In diesem Abschnitt wurden mit einfachen Mitteln einige Hintergrundaspekte einbezogen ohne von der gewählten Hauptabstraktionsrichtung bei der Anwendungsentwicklung abzuweichen. Es gibt dazu grundsätzlich die Möglichkeit, zunächst eine „funktional äquivalente“ Implementierung eines WFMS mit Agenten zu erzeugen. Diese Herangehensweise wurde z.B. von Buhler (siehe Abschnitt 2.3.5) für die Ausführung von BPEL4WS-Beschreibungen genutzt. In der agentenorientierten Implementierung sind dann durch die Agenten die möglichen Verteilungs- und Flexibilisierungspunkte explizit benannt. Stück für Stück kann auf dieser Grundlage das WFMS flexibilisiert werden.

Neben dem agentenorientierten WFMS wurde die prinzipielle Möglichkeit eines workflowbasierten Agentenmanagementsystems aufgezeigt.

Was kann mit der Stufe II erreicht werden? Im Falle des agentenbasierten WFMS hängen die tatsächlich gewonnen Vorteile von der Art und Umsetzung der Flexibilisierungsmöglichkeiten ab: bei einer funktional äquivalenten Umsetzung sind die Vorteile eher allgemeiner Natur. Im Falle der Verwendung von explizit prozessorientierten Sichten im Rahmen von PAOSE sind die Vorteile handfester, stehen aber nur zur Entwurfszeit zur Verfügung. Bei der Ausführung und bei der Verwaltung von Rollen und Rechten kommt auch eine mit PAOSE entwickelte Agentenanwendung nicht entfernt an die Möglichkeiten einer Workflowanwendung: der Zusammenhang einer Interaktion als Prozess geht bei der Ausführung verloren, es werden lediglich lokale Prozesse in Form von Protokollnetzen ausgeführt.

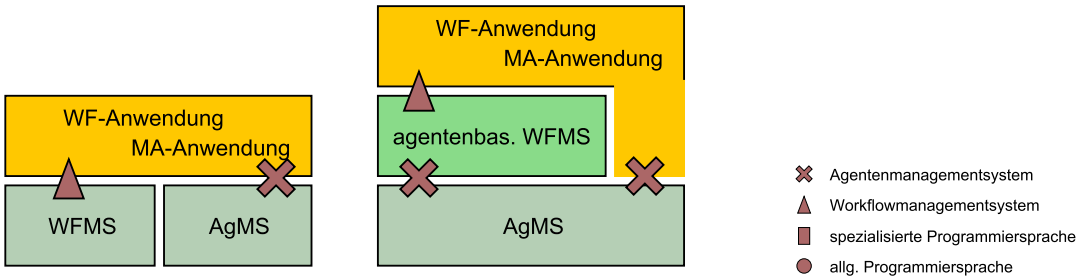
In der nächsten Stufe werden für die Anwendungsentwicklung beide Schnittstellen zur Verfügung gestellt. So sollen in einer Anwendung alle Vorteile gemeinsam verfügbar werden.

4.5 Stufe III

In der dritten Stufe geht es darum, alle Aspekte vollständig zur Verfügung zu stellen, statt wie in Stufe I und II in Vorder- und Hintergrund zu trennen. Dies wird erreicht, wenn zur Anwendungsentwicklung sowohl ein Agentenmanagementsystem als auch ein Workflowmanagementsystem verfügbar ist. So können die wesentlichen strukturellen Bestandteile der Anwendung als Agenten modelliert, implementiert und ausgeführt werden und die wesentlichen übergreifenden Prozesse können als Workflows modelliert, implementiert und ausgeführt werden. Die Bausteine einer Anwendung erhalten also eine klare Identität und im Gegensatz zur zweiten Stufe können nun sowohl die strukturellen Aspekte als auch die Prozessaspekte jeweils adäquat abgebildet werden.

Dabei entstehen wichtige Fragen, z.B. nach der Art, wie sich die Agenten und Workflows in der Anwendung beeinflussen, wie das Verhalten der Agenten in Beziehung zu

den Prozessen der Anwendung steht, oder wie die Struktur der Workflows in Beziehung zu der Struktur steht, die von den Agenten gebildet wird. Diese Fragen müssen bei der Anwendungsentwicklung am besten von vornherein und grundsätzlich geklärt sein, sonst kommt es leicht zu unübersichtlichen Entwicklungsprozessen und auch die resultierende Anwendung wird dann schwer zu überblicken.



(a) WFMS neben AgMS

(b) Agentenbasiertes WFMS neben AgMS

Abbildung 4.13: Architektur der Stufe III

Der einfachste Entwurf aus der Architekturperspektive ist es, ein WFMS und ein Agentenmanagementsystem (AgMS) nebeneinander zu benutzen, wie in Abbildung 4.13a dargestellt. In diesem Falle sind die eben gestellten Fragen besonders schwierig zu beantworten, weil WFMS und AgMS beziehungslos nebeneinander stehen. Es ist besser, ein agentenbasiertes WFMS wie aus Stufe II zu benutzen und der Anwendung explizit den Durchgriff auf die untere Architekturschicht, also auf das AgMS zu erlauben. Der Unterschied zu einem agentenbasierten WFMS aus Stufe II ist die Schnittstelle des agentenbasierten WFMS zur Anwendung: Für Stufe III sollte die WMFS-Schnittstelle in eine Agentenkommunikationssprache „verpackt“ sein, so dass die Anwendungsagenten direkt mit dem WFMS interagieren können. Wie in Abbildung 4.13b dargestellt, bleibt der Schnittstellentyp des WFMS dabei erhalten. Bei der Anwendungsentwicklung wird also auf ein AgMS zugegriffen und auf ein WFMS, das eine WFMS-Schnittstelle in ACL-Syntax (FIPA-ACL ist die *Agent Communication Language* der FIPA) bereitstellt.

Eine Architektur, die ein agentenbasiertes WFMS für Agenten anbietet, bietet als Grundlage viele Gestaltungsmöglichkeiten für Anwendungen. Die Anwendung kann neben den Client-Agenten für das WFMS auch weitere Agenten implementieren. Die Interaktionen der Anwendungsagenten sind frei gestaltbar. Die Agenten können je nach Anforderung und Schwerpunkt mit verschiedenen Agentenmanagementsystemen implementiert sein, solange diese über Standardisierung kompatibel sind. Abbildung 4.14 zeigt ein Beispiel für die Aufteilung einer solchen Anwendung zur Laufzeit. Alle Clients sind hier Agenten. Ein Client-Agent wird mit Capa ausgeführt. Dieser Agent wird Tasks aus einem Workflow ausführen, der im agentenbasierten WFMS ausgeführt wird. Ein anderer Agent der Anwendung ist mit Jade implementiert. Es könnte sein, dass dieser Agent die Stärken von Jade für seine besondere Aufgabe

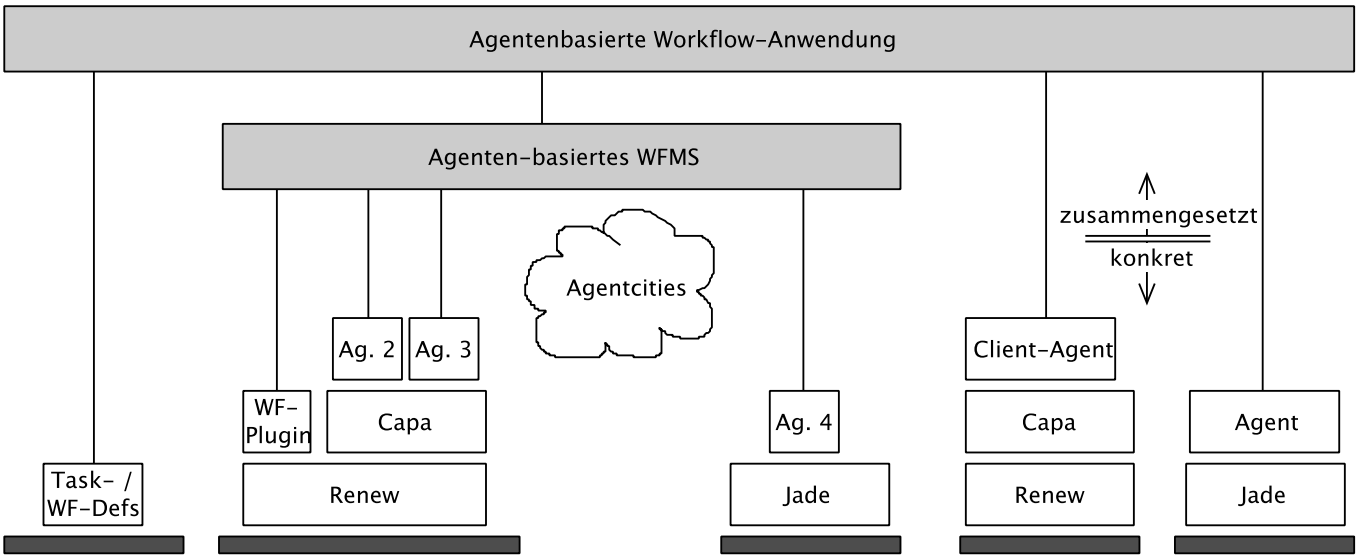


Abbildung 4.14: Stufe III: Beispiel für die Nutzung eines agentenbasierten WFMS in einer Agentenanwendung

nutzt. Auch das WFMS selbst kann heterogene Komponenten haben, wie dies bereits in Stufe II der Fall war.

Im Folgenden wird die Umsetzung des agentenbasierten WFMS noch genauer betrachtet, als dies im Abschnitt zur Stufe II geschehen ist. Die vorgestellten Varianten folgen den Erfahrungen während der prototypischen Umsetzung im Lehreprojekt. Im Abschnitt zur Stufe II wurde vorgeschlagen, das gesamte WFMS mit Agenten zu implementieren. Dies ist sicherlich mit fast jedem Agentenmanagementsystem möglich. Nun setzt aber das konkret verwendete AgMS – Capa – auf den Referenznetzen auf, und diese eignen sich im Prinzip hervorragend für die Spezifikation von Workflows. Im Abschnitt 2.3.4 auf Seite 48 wurde das petrinetzbasierte WFMS von Thomas Jacob vorgestellt, das als wesentliche Erweiterung der Notation die Task-Transition einführt. Für die Capa-Agenten fehlt jedoch der Mechanismus der Task-Transition, deshalb wird die Implementierung eines WFMS mit Capa-Agenten die Vorteile der unterliegenden Referenznetze nicht direkt nutzen können. Die Agenten müssten einen Mechanismus mit der Funktion der Task-Transition nachbauen. Dies ist umständlich und ineffizient und verschenkt die Vorteile der unterliegenden Schicht.

Im Verlauf der prototypischen Umsetzung wurde daher zunächst ein anderer Weg gewählt: Das vorhandene WFMS von Thomas Jacob (im Folgenden wieder als „Workflow-Plugin“ bezeichnet) wird von Agenten gekapselt. Der Vorteil dieser Vorgehensweise liegt darin, dass die Eigenschaften der Referenznetze sowohl für Agenten als auch für Workflowspezifikation gut genutzt werden können. Im Sinne der Verteilbarkeit kommt die Möglichkeit einer entfernten Workflow-Instanziierung zu den Möglichkeiten der Stufe I hinzu, nämlich dadurch, dass das WFMS über ACL asynchron ansprechbar ist. Die entfernte Administration und der entfernte Zugriff für Clients war vorher auch möglich, allerdings ist dieses nun ebenfalls asynchron entgegen den

synchronen RMI-Aufrufen des Workflow-Plugins. Der Nachteil dieser Vorgehensweise ist, dass das WFMS gemäß Stufe I monolithisch implementiert ist und damit Vorteile der Stufe II bezüglich Verteilbarkeit und Flexibilisierbarkeit des WFMS wieder verloren gehen.

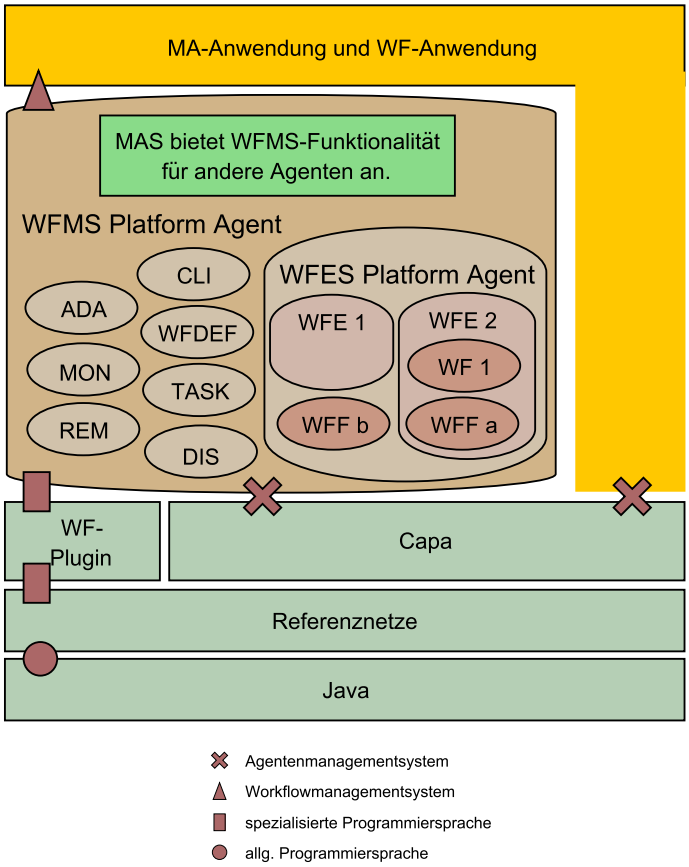


Abbildung 4.15: Stufe III: Agentenbasiertes WFMS mit den zusätzlichen Agenten REM („Remote Agent“), DIS („Distribution Agent“), Workflow- und Workflow-Fragmentagenten (WF und WFF) für erweiterte Flexibilität

Nachdem nun weder die vollständige Neu-Implementierung eines WFMS mit Capa-Agenten, noch die vollständige Kapselung des vorhandenen referenznetzbasierten WFMS dauerhaft akzeptable Lösungen bieten, gibt es einen Mittelweg: Die Basisfunktionalität für Workflownetze und deren Ausführung wird in einer Variante des Referenznetzformalismus als eigenes Plugin aus dem Workflow-Plugin extrahiert (von Michael Duvigneau). Das so entstandene Plugin „WF-Net“ stellt die Tasktransition und eine Schnittstelle zur lokalen Beobachtung und zum Schalten von Tasktransitionen bereit. Das WF-Net-Plugin wird den Capa-Agenten zur Implementierung eines agentenbasierten WFMS zur Verfügung gestellt. Wir erhalten die Möglichkeit der flexiblen Implementierung eines agentenbasierten WFMS auf Basis der Referenznetze.

Im Folgenden werden einige Möglichkeiten betrachtet, wie eine zusätzliche Flexibilität im agentenbasierten WFMS umgesetzt werden kann. Einige der dabei zusätzlich vorgesehenen Agenten werden in Abbildung 4.15 dargestellt.

Externe Task-Agenten: Zusätzlich zu dem bereits in Stufe II vorhandenen Task-Agenten können externe Task-Agenten eingeführt werden. Ein interner Task-Agent kann direkt für die Ausführung eines Standard-Tasks zuständig sein, der dann zum Funktionsumfang des WFM-Systems gehört. Externe Task-Agenten können anwendungsspezifische Tasks ausführen. In einem Workflow wird dann ein Task dem Typ nach referenziert, und erst wenn der Task aktiviert ist, wird (z.B. im offenen Agentennetz) nach einem passenden Task-Agenten gesucht. Denkbar ist hier ein Szenario mit verhandelnden Agenten. Ein Task-Agent kann auch einen Subworkflow an anderer Stelle starten und auf diese Weise mehrere Workflowmanagementsysteme koppeln.

Fragmentierte Workflows: Ein Workflow kann beim Entwurf mit Fragmentierungs-Stellen versehen werden, wie es im Abschnitt 3.3 vorgestellt wurde. Diese werden dann auf mehrere Workflowmanagementsysteme verteilt und ausgeführt. Ein spezieller WFMS-Agent („Distribution Agent"; in Abbildung 4.15 abgekürzt: DIS) ist für die Verteilung und Koordination der Fragmente zuständig.

Kooperierende Workflowmanagementsysteme: Mehrere Workflowmanagementsysteme können miteinander über ihre aktuellen Workflows kommunizieren und eigenständig über den geeigneten Ort der Ausführung von (Teil-) Workflows entscheiden. Ein spezieller Agent („Remote Agent"; REM im Bild) sorgt für diese Kommunikation. Hier kann es um Last-Ausgleich gehen oder um Zuständigkeiten, wenn je ein WFMS eine Abteilung oder eine Firma in einem kooperierenden Szenario repräsentiert. Eine Schnittstelle zur Kopplung von WFM-Systemen ist von der *WfMC* vorgesehen (Interface 4 in Abbildung 2.9 auf Seite 45).

Hierarchische Workflowmanagementsysteme: Eine Variante der kooperierenden WFM-Systeme bildet die Möglichkeit, mehrere Workflowmanagementsysteme zu einem verteilten WFMS mit voll funktionsfähigen lokalen Komponenten zusammenzuschließen. Dieser Aufbau entspricht dann wortgenau den Kriterien für ein *agentenbasiertes* WFMS im Gegensatz zu einem *agent-enhanced* WFMS nach Yan et al. [Yan et al. 2001]. Die Möglichkeiten eines solchen Aufbaus sind in Abbildung 4.16 angedeutet. Auf jeder konkreten Plattform (Rechner mit Java, Renew, Capa) laufen in einer eigenen Schicht zuerst die Standardagenten AMS und DF sowie der Capa-Plattform-Agent. In der nächsten Schicht laufen lokale WFMS-Agenten (mit ihren inneren Agenten wie in Abbildung 4.11). Diese WFMS-Agenten vertreten z.B. je eine Geschäftsstelle einer Firma. Sie kommunizieren untereinander und bilden dadurch ein verteiltes WFMS, welches z.B. eine ganze Firma abbildet. Dieses verteilte WFMS bekommt eine explizite Repräsentation in Form eines WFMS-Agenten auf der dritten Schicht. Auch diese WFMS-Agenten der dritten Schicht können miteinander kommunizieren und dadurch ein interorganisationales verteiltes Workflowmanagementsystem bilden („Multilevel-WFMS"). Eine unternehmensübergreifende Anwendung kann darauf aufbauen.

Workflow-Agent: Ein Workflow wird nicht in Form einer Datenstruktur an eine Workflow-Engine (an den WFE-Agenten) zur Ausführung übergeben, sondern wird als

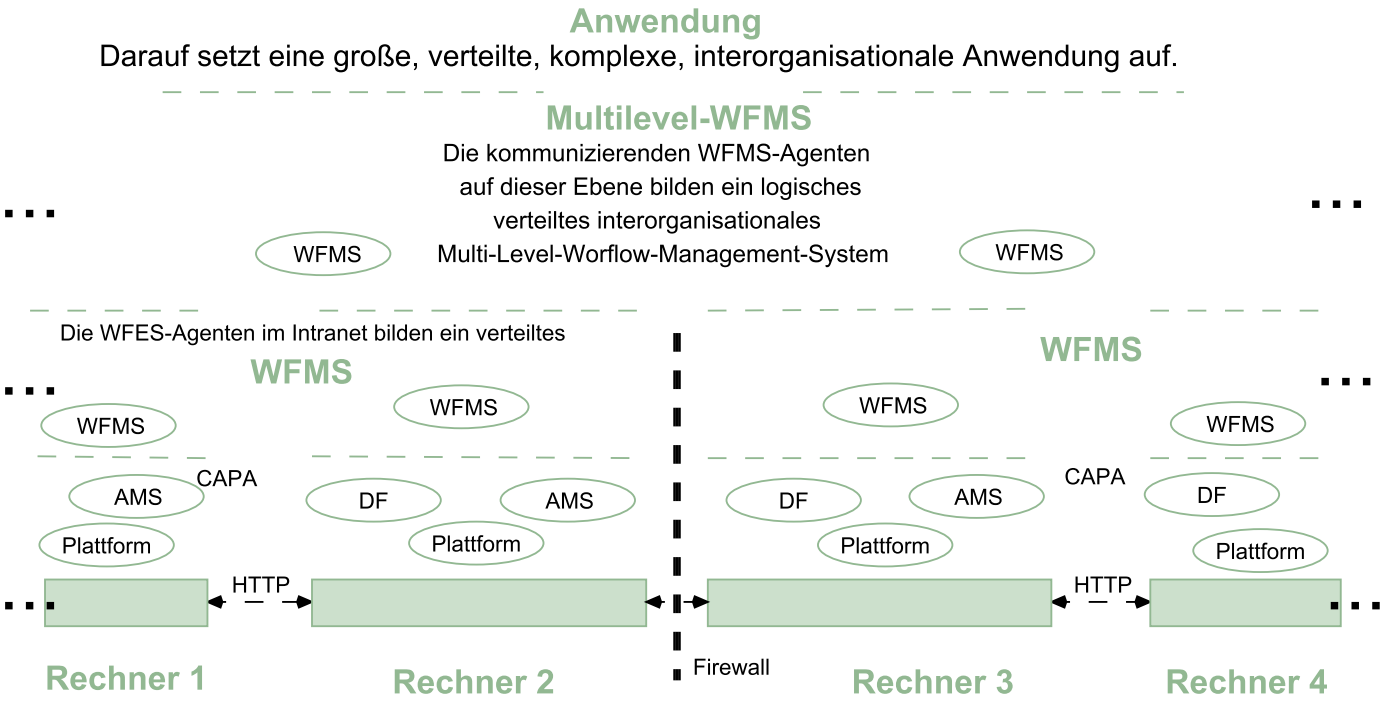


Abbildung 4.16: Stufe III: Hierarchisches verteiltes WFMS durch kooperierende lokale Workflowmanangementsysteme. Aus: [Reese et al. 2006b]

Agent gekapselt. Szenarien zur Verteilung sind erstens: Der Workflow-Agent kapselt einen Workflow mit Fragmentierungs-Stellen und Workflow-Fragment-Agenten kapseln je ein Fragment. Zweitens: Der Workflow-Agent ist mobil (das schränkt möglicherweise die verteilte und parallele Ausführung unabhängiger Teilworkflows ein). Szenarien für die vom Workflow-Agent gekapselten Daten sind erstens: Der Workflow-Agent kapselt den Zustand einer Workflow-Instanz (Falldaten). Ein solcher Workflow-Agent ändert die Aufgaben der WF-Engine-Agenten. Zweitens: Der Workflow-Agent wird außerhalb des WFMS definiert und gestartet (z.B. in einem Szenario, wo das WFMS ein allgemein verfügbarer Dienst im offenen Agentennetz ist) und trägt dann auch die Workflow-Definition mit sich. Ein solcher Workflow-Agent ändert die Aufgaben des Workflow-Definitions-Agenten im WFMS. Ganz allgemein können im Workflow-Agenten die Workflow- und die Agenten-Identität zusammenfallen. Diese Idee wird in Stufe IV und V weiter verfolgt als eine Möglichkeit der weitreichenden Integration. Im Abschnitt 5.1.2 wird der Workflow-Agent ausführlich diskutiert.

In diesem Abschnitt wurde dargestellt, wie sich durch die Verfügbarkeit von WFMS und Agentenmanagementsystem eine beachtliche Gestaltungsfreiheit bei der Anwendungsentwicklung erreichen lässt. Die wesentlichen Bestandteile einer Anwendung können mit der passenden Abstraktion modelliert werden: Agenten für strukturelle Bestandteile, Workflows für wesentliche Prozesse der Anwendung. Die Art und Weise, wie diese unterschiedlich geprägten Bestandteile in Beziehung zueinander stehen,

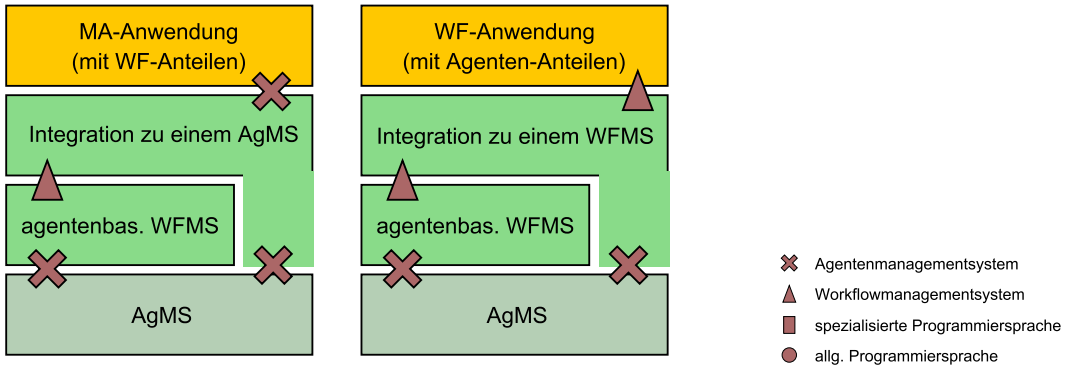
wird nicht von der Architektur beantwortet oder auch nur nahegelegt. Bei der Anwendungsentwicklung muss deshalb zunächst ein allgemeines und verbindliches Konzept erstellt werden.

Es wurde argumentiert, dass es ungünstig ist, ein WFMS und ein AgMS beziehungslos nebeneinander zu benutzen, statt dessen wird vorgeschlagen, ein agentenbasiertes WFMS mit einer ACL-Schnittstelle zur Verfügung zu stellen, so dass die Strukturkomponenten der Anwendung von der gleichen Art sind wie die Strukturkomponenten des WFMS (nämlich in beiden Fällen Agenten).

Für die Realisierung des agentenbasierten WFMS wurden drei Varianten vorgestellt: (1) Die gesamte WFMS-Funktionalität wird auf Agenten-Ebene realisiert. (2) Ein vorhandenes WFMS wird durch Agenten gekapselt und so für Agenten ansprechbar gemacht. (3) Es wird eine spezialisierte Schnittstelle mit einer workflowspezifischen Erweiterung der Referenznetze bereitgestellt und darauf mit Agenten das WFMS implementiert. Nur die dritte Variante vereint tatsächlich die Vorteile aus Stufe I und Stufe II mit der Forderung der Stufe III, nämlich sowohl WFMS- als auch AgMS-Funktionalität vollständig zur Verfügung zu stellen. Weiterhin wurden vier naheliegende Möglichkeiten zur Flexibilisierung des agentenbasierten WFMS angedeutet, die jeweils zu weitreichenden Ideen und Gestaltungsspielräumen führen. Für die Entwicklung einer konkreten Anwendung ist es dringend notwendig, aus diesen großen Spielräumen eine angemessene Auswahl zu treffen und diese verbindlich und vor allem einschränkend festzulegen. Am sichersten gelingt dieses, wenn die getroffenen Entscheidungen als Teil des Anwendungsrahmenwerkes, als Infrastruktur für die Anwendung umgesetzt werden. Dies ist Gegenstand der vierten Stufe: Im nächsten Abschnitt wird ein konkreter Vorschlag für eine vereinfachte Schnittstelle gemacht, die auf Basis der Möglichkeiten der Stufe III weitreichende Vorteile der Workflow- und Agentenorientierung vereint.

4.6 Stufe IV

In der vierten Stufe sollen die Vorteile und Möglichkeiten von Workflow- und Agententechnologie gemeinsam verfügbar sein, allerdings gegenüber Stufe III mit einer vereinfachten Schnittstelle für die Anwendungsentwicklung. In Stufe III ist das schwerwiegendste Problem, dass bei der Anwendungsentwicklung die Art des Zusammenspiels von Workflow- und Agententechnologie quasi erfunden werden muss und dem Rahmenwerk ergänzend zur Seite gestellt werden muss, um sinnvoll strukturierte Anwendungen entwickeln zu können. Dieses Problem der Strukturierung wird mit Stufe IV von der Anwendungsentwicklung in die Entwicklung des Rahmenwerkes verlegt, indem eine zusätzliche Schicht in die Architektur eingeführt wird (siehe Abbildung 4.17). Die Anwendung, die in Stufe III noch direkt auf dem WFMS und AgMS aufsetzt, ist in Stufe IV in zwei Schichten aufgeteilt: das Rahmenwerk (Architekturschicht „Integration“) enthält als strukturierende Schicht die Entscheidung für die Workflow- oder Agentenausrichtung und die Art, wie die jeweils andere Ausrichtung integriert oder verwendet wird. Diese Entscheidung geht nun der Anwendungsentwicklung voraus, so dass während der Anwendungsentwicklung darin kein Spielraum mehr besteht. Dies wird dadurch explizit, dass nur noch *entweder* Agenten *oder* Workflows als Schnittstelle zur Anwendungsentwicklung bereitgestellt werden. Die Anwendungsschicht enthält



(a) Stufe IV für Agenten (b) Stufe IV für Workflows

Abbildung 4.17: Architektur der Stufe IV

gegenüber Stufe III weniger strukturierende Elemente, so dass die Anwendungsentwicklung gegenüber Stufe III vereinfacht ist. Die Möglichkeiten sind gegenüber Stufe III eingeschränkt, allerdings sollen ja gerade die konkret verwendeten Vorteile von Agenten- und Workflowtechnologie fest im Rahmenwerk verankert sein.

Der Schnittstellentyp für die Anwendungsentwicklung ist wieder einfach wie in den Stufen I und II. Es besteht jedoch ein bedeutender Unterschied in der Leistungsfähigkeit, indem die jeweils im Rahmenwerk integrierte (transparent vorhandene) Technologie unterstützt wird. Das Rahmenwerk der Stufe IV ermöglicht neben neuen Funktionen immer noch dieselben Anwendungen wie in Stufe I und II. Die neuen Funktionen stammen aus zusätzlichen Sichten, die in Stufe I und II nicht verfügbar waren oder von außen (z.B. durch ausführliche Dokumentation) hinzugefügt werden mussten.

Zunächst soll das Rahmenwerk näher betrachtet werden für den Fall, dass Agenten die Zieltechnologie sind (Abbildung 4.17a, detailliert in Abbildung 4.18). Die Integrationsschicht benutzt ein AgMS und ein WFMS und bietet Funktionen eines AgMS für die Anwendungsentwicklung an, die über die übliche Funktionsweise eines AgMS hinausgehen, indem sie gewisse Eigenschaften von Workflows bereitstellt. In der detaillierteren Architektur baut Capa nach wie vor auf der allgemeinen Programmiersprache Java auf und auf der speziellen Programmiersprache der Referenznetze. Neben Capa steht das Workflow-Plugin zur Verfügung, welches zum Mechanismus der Referenznetze die Tasktransition und den Beobachtungsmechanismus hinzufügt. Auf diesen beiden Architekturbausteinen setzt das agentenbasierte WFMS wie in Stufe III auf (die Darstellung einzelner Agenten wurde der Übersichtlichkeit halber weggelassen). Das WFMS hat die speziellen Programmierschnittstellen des WF-Plugins und der Referenznetze sowie Java zur Verfügung und außerdem die Schnittstelle von Capa zum Entwurf, zur Definition und Ausführung von Agenten. Die neue Integrationsschicht der Stufe IV erbringt einen Teil ihrer Funktion, indem sie Agenten definiert und ausführt. Diese Schicht soll darüber hinaus selbst die Definition von Anwendungsagenten ermöglichen und ist also als eine Erweiterung von Capa zu betrachten. Die Schicht greift zur Realisierung dieser Erweiterung auf Capa-interne Vorgänge und Elemente zu. In der Architektur wird dies berücksichtigt, indem zur Capa-Anwendungsschnittstelle

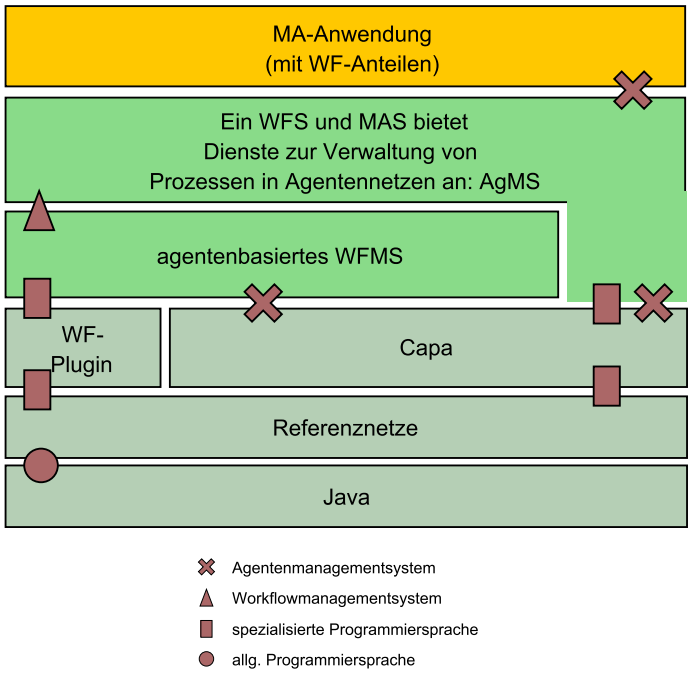


Abbildung 4.18: Stufe IV: Architektur der erweiterten Capa-Plattform mit Agentenanwendung

eine spezielle Capa-Programmierschnittstelle hinzugefügt ist (in Abbildung 4.18 als Rechteck dargestellt). Das Rahmenwerk erweitert also die Funktionen von Capa und ermöglicht den Entwurf, die Implementierung und die Ausführung von Agenten einer Anwendung. Diejenigen Agentenfunktionen, die unverändert gegenüber Capa sind, werden von der Integrationsschicht „durchgereicht“, so dass nicht Capa neu implementiert wird. Das Ergebnis ist ein spezielles Agentenmanagementsystem, mit dem spezielle Workflow-unterstützte Agentenanwendungen entwickelt und ausgeführt werden können.

Eine Ausprägung kann z.B. sein, dass vom Rahmenwerk alle Interaktionen zwischen den Agenten einer Agentenanwendung in Form von Workflows verwaltet und ausgeführt werden. Die Interaktionen werden wie beim Paose-Ansatz mit Interaktionsprotokollen entworfen, die dann direkt vom Rahmenwerk in Workflows umgesetzt werden. Die Agenten der Anwendung werden also über Workflows untereinander gekoppelt. Diese Kopplung wird von der Integrationsschicht realisiert, d.h. die Anwendungsentwickler definieren Agenten, Wissensbasen, Agenteninteraktionsprotokolle und Ontologie, und das Rahmenwerk erzeugt intern Workflowdefinitionen. Zur Laufzeit werden Agenten und Workflows in ihren jeweiligen Laufzeitkomponenten ausgeführt.

Zur Laufzeit (vergleiche Abbildung 4.19) ist das agentenbasierte WFMS wie in Stufe III aufgebaut. Das darauf aufsetzende AgMS benutzt das WFMS und eigene Agenten sowie auch direkte Capa-Komponenten, um seine Funktionen zu realisieren. Das neue AgMS ist im Bild als Capa+ bezeichnet. Die Anwendung benutzt Capa+. Die Abbildung der anwendungsspezifischen Agenten auf konkrete Laufzeitkomponen-

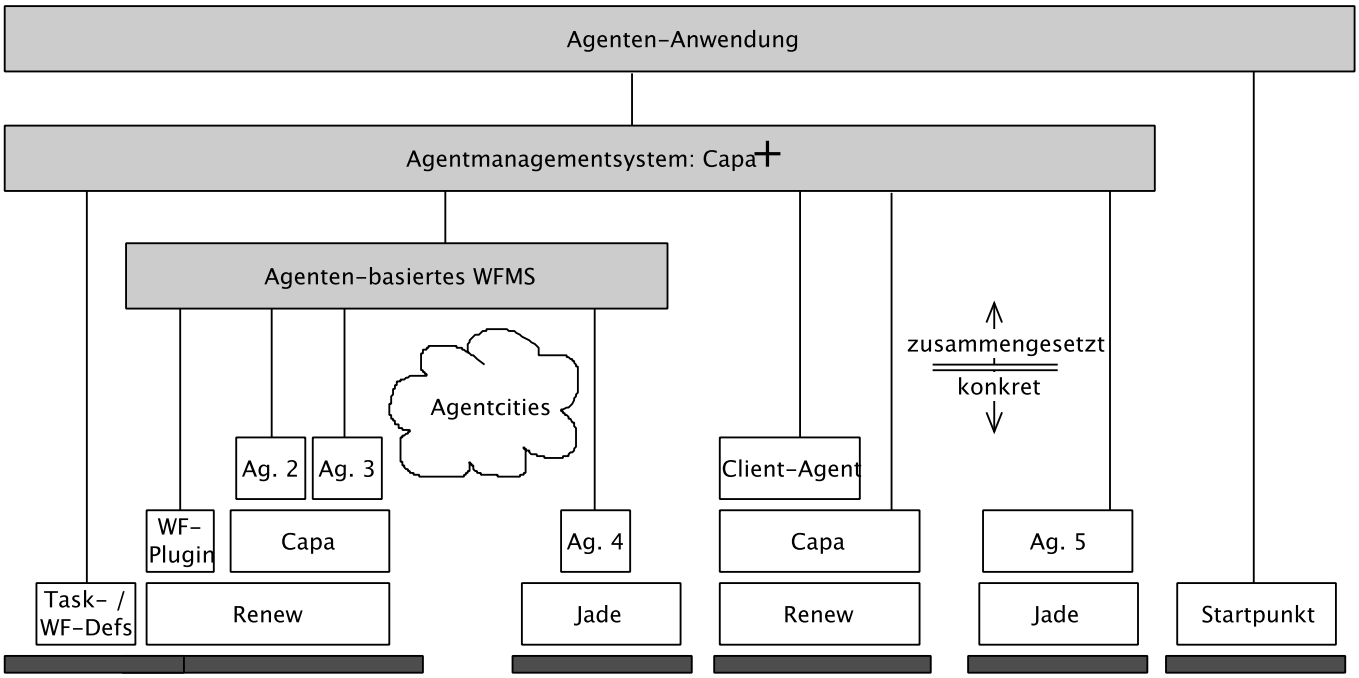


Abbildung 4.19: Stufe IV: Beispiel für die Nutzung der erweiterten Capa-Plattform

ten übernimmt Capa+. Wahrscheinlich wird innerhalb von Capa+ für jeden Anwendungsagenten ein Capa-Agent gestartet. Da dies jedoch nicht von der Architektur so festgelegt ist, sind keine Agenten direkt der Anwendung zugeordnet. Stattdessen ist die einzige nur der Anwendung zugeordnete konkrete Komponente der Startpunkt der Anwendung.

Mit der vorgestellten Architektur ist es nach wie vor möglich, auch sehr einfache Anwendungen zu entwickeln. In diesem Fall wird eine einzige Renew-Instanz mit einer erweitertern Capa-Plattform gestartet. Die Anwendung spezifiziert über die Capa+-Schnittstelle ihre anwendungsspezifischen Agenten und Capa+ benutzt das agentenbasierte WFMS und legt die entsprechenden Workflowdefinitionen in einer lokalen Datenbank ab (siehe Abbildung 4.20).

In diesem Abschnitt wurde gezeigt, wie die Architektur eines speziellen Agentenmanagementsystems aussieht, welches ein agentenbasiertes WFMS benutzt. Als eine mögliche Ausprägung wurde vorgestellt, dass die Agenten einer darauf aufsetzenden Agentenanwendung sich darauf verlassen können, dass ihre Interaktionen (also ihre „Berührungspunkte“) über Workflows organisiert und ausgeführt werden. Vergleicht man dieses mit der Möglichkeit der Anwendungsentwicklung in Stufe III, so könnte dort die gleiche Anwendung entwickelt werden. Wahrscheinlich würden aber nur die als wesentlich erachteten Prozesse als Workflows modelliert und ausgeführt werden. Wo also in Stufe III *einige* Prozesse der Anwendung als Workflows ausgeführt werden, werden in Stufe IV *alle* Prozesse der Anwendung als Interaktionen definiert und dann vom Rahmenwerk als Workflows ausgeführt.

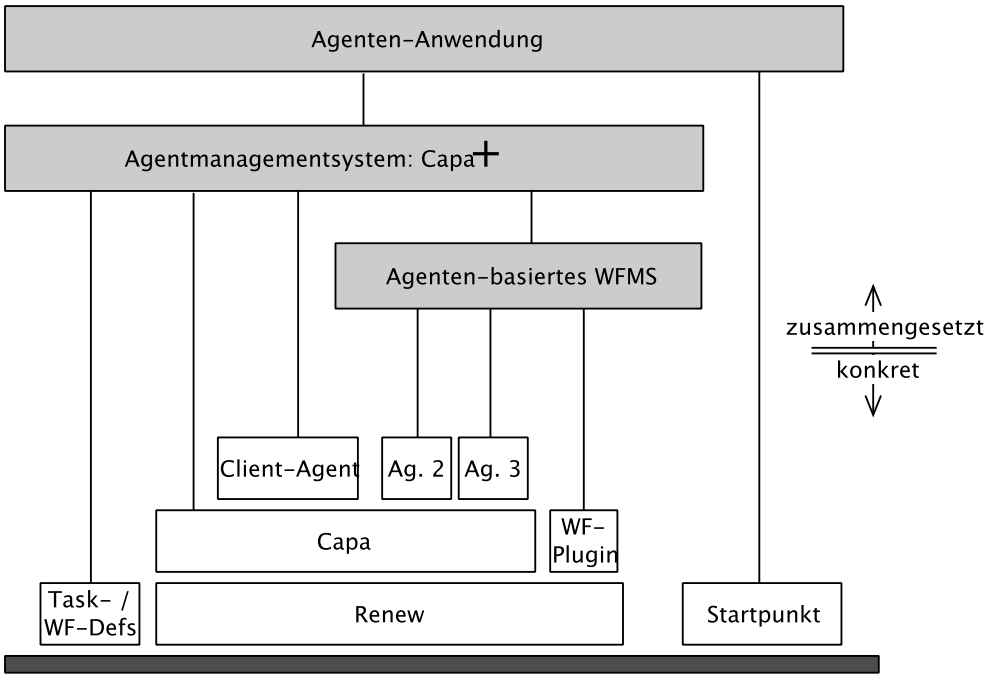


Abbildung 4.20: Stufe IV: Lokale Variante der erweiterten Capa-Plattform

Für die zweite, hier nicht näher diskutierte Variante der Stufe IV, bei der (statt Agenten) Workflowanwendungen unterstützt werden, können symmetrisch entsprechende Überlegungen angestellt werden: Beim Entwurf von Workflows kann man dann davon ausgehen, dass alle ihre Berührungspunkte als Agenten realisiert sind. Diese Denkweise ist in der vierten Stufe im Rahmenwerk festgeschrieben (mit einer Architektur wie in Abbildung 4.17b).

Damit sind in diesem Abschnitt Konzepte für eine sehr mächtige Architektur vorgestellt, die auf weitreichende Weise die Agenten- und Workflowtechnologie verbinden kann und dabei für die Anwendungsentwicklung eine einfache Schnittstelle zur Verfügung stellt. Die Art der Integration von Workflows und Agenten ist im Rahmenwerk lokalisiert und nicht wie in Stufe III in der Anwendung. Durch die Auswahl einer speziellen Vorgehensweise und deren Festschreibung im Rahmenwerk wird sichergestellt, dass die Regeln der Vorgehensweise durchgehend in der Anwendung berücksichtigt werden.

Stufe IV stellt gegenüber dem Ziel der symmetrischen Integration von Workflows und Agenten eine erhebliche Einschränkung im Vergleich zu Stufe III dar und zwar insofern, als dass nur noch ein Teil der Workflow- bzw. Agenten-Funktionalität für die Anwendung nutzbar ist. Im Idealfall können für die wesentlichen Subsysteme einer Anwendung Agenten zur Modellierung verwendet werden und für die wesentlichen Prozesse und Vorgänge einer Anwendung können explizit in Form von Workflows definiert und ausgeführt werden. Dieses soll die fünfte Stufe leisten.

4.7 Stufe V

In der fünften Stufe sollen Agenten und Workflows soweit integriert werden, dass eine Anwendung je nach aktueller Fragestellung sowohl bezüglich ihrer Vorgänge (prozessorientiert) als auch bezüglich ihrer Subsysteme (strukturorientiert) betrachtet werden kann. Ein Wechsel der Perspektive soll zur Laufzeit möglich sein.[3]

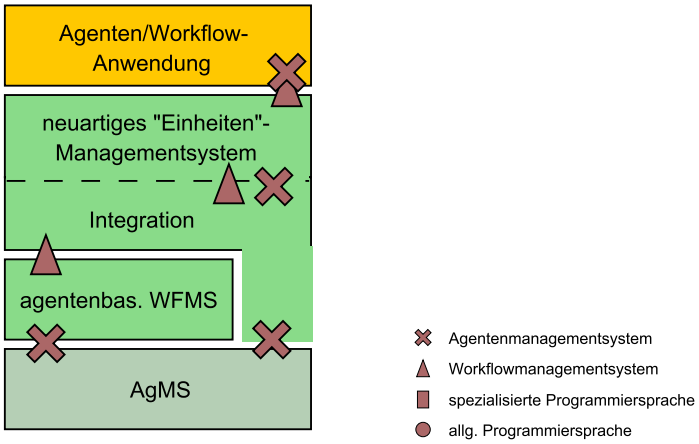


Abbildung 4.21: Architektur auf Stufe V mit Schnittstellen

Beim *Entwurf* werden diese Eigenschaften durch ein Vorgehen und durch entsprechende Techniken, Entwurfs- und Modellierungswerkzeuge bereitgestellt. Dies ist bei der Matrixorganisation des PAOSE-Ansatzes bereits der Fall. Für die *Umsetzungsphase* ist es gut, wenn die Entwurfs- und Modellierungswerkzeuge direkt auch zur Implementierung genutzt werden können. Im PAOSE-Ansatz werden die Interaktionsprotokolle zur Codegenerierung genutzt. Zur *Laufzeit* sollen diese Eigenschaften und Funktionen von einem Managementsystem geliefert werden, das ist das Ziel dieser Architektur. Die Laufzeitumgebung braucht Zugriff auf ein Workflowmanagementsystem und auf ein Agentenmanagementsystem. Es wird dann die abstrakten Spezifikationen der Anwendung geeignet in Agenten und Workflows umsetzen, so dass die genannten Anforderungen erfüllt sind. Die Architektur dazu sieht im Groben aus wie in Abbildung 4.21 dargestellt. Auf dem AgMS und dem agentenbasierten WFMS setzt wie bei Stufe IV eine Integrationsschicht auf, die jetzt nicht entweder die Agentenmanagement- *oder* die Workflowmanagement-Schnittstelle zur Anwendung zur Verfügung stellt, sondern eine neuartige Schnittstelle, die durch ein kombiniertes Kreuz mit Dreieck dargestellt ist. In einer nur schwach abgegrenzten Zwischenschicht (gestrichelte Linie) stehen erweiterte Agenten- und Workflowmanagementschnittstellen zur Verfügung, wie bei Stufe IV zur Anwendung hin.

Eine wesentliche Eigenschaft der Stufe V soll sein, dass diese Einheiten rekursiv schachtelbar sind und auf diese Weise geschachtelte Subsysteme und geschachtelte

[3] Dazu müssen sowohl die Agenten als auch die Workflows abstrakt definiert und modelliert werden, weil sie lediglich Sichten auf eine Anwendung darstellen. Der Entwurf einer abstrakten Agentendefinitionssprache oder einer abstrakten Workflowdefinitionssprache ist nicht Gegenstand dieser Arbeit.

Prozesse abgebildet werden können. Diese Eigenschaft ist nicht im Architekturbild zu erkennen, da sie die Funktionsweise betrifft. Sie wird von der oberen Hälfte der Integrationsschicht bereitgestellt und kann in der konkreten Umsetzung auf geschachtelten Netzinstanzen in Renew beruhen. In Stufe IV wurden alle Interaktionen der Agenten auf Workflowbasis ausgeführt, Stufe V führt nun auch die *agenteninternen* Prozesse auf Workflowbasis aus.

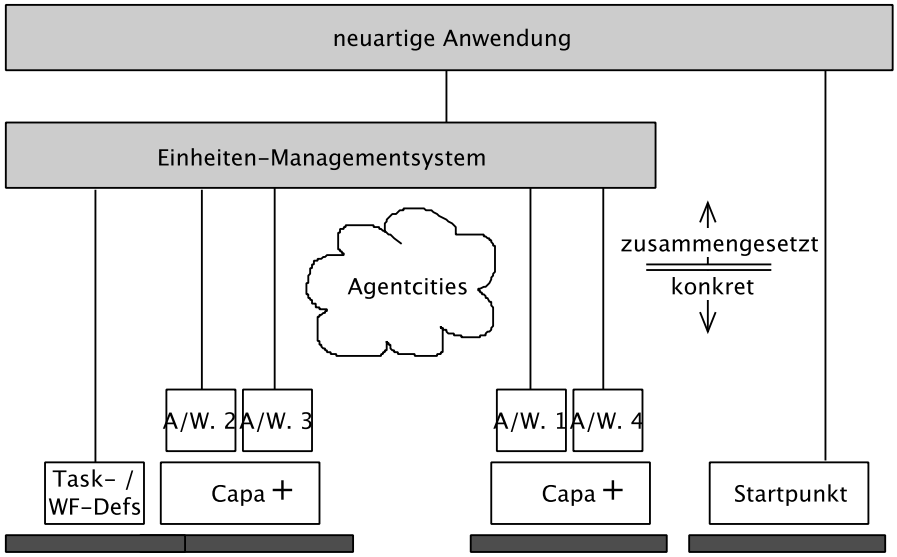


Abbildung 4.22: Stufe V: Einfache Laufzeitvariante

Im einfachsten Fall werden zur Laufzeit lokale Capa+-Plattformen gestartet, auf denen die Agenten und Workflows der Anwendung ausgeführt werden (Abbildung 4.22). Capa+ ist dabei zu lesen wie in Abbildung 4.20 und damit eigentlich ein zusammengesetzter Anwendungsbaustein. Wenn aber alle Komponenten einer Capa+-Plattform lokal auf einem Rechner ausgeführt werden, kann Capa+ genausogut als konkrete Einheit betrachtet werden. Die Möglichkeiten der Stufe V werden allerdings erst im verteilten Fall zur Geltung kommen. Dazu kommen insbesondere zwei der im Abschnitt zur Stufe III erwähnten Gestaltungsmöglichkeiten ins Spiel: Die hierarchischen Workflowmanagementsysteme und der Workflow-Agent. Diese Mechanismen werden für Stufe IV ins Rahmenwerk integriert und auf Stufe V für die flexible Ausführung von Prozess-Einheiten verwendet. Es ist also so, dass die Möglichkeiten der Stufe III in der vierten Stufe eingeschränkt werden auf jeweils einen Spezialfall und in Stufe V wieder zur Verfügung stehen, aber eben im Rahmenwerk.

Eine interorganisationale Anwendung der Stufe V kann als *Agentenanwendung* betrachtet werden. In dieser Perspektive ist die Anwendung in Subsysteme gegliedert, z.B. ein Agent für jede beteiligte Organisation. Jeder dieser Agenten hat eine starke innere Struktur und komplexe innere Prozesse zur Entscheidungsfindung. Die inneren Prozesse werden wie bei einem Agentenmanagementsystem spezifiziert und intern wie Workflows ausgeführt. (Die innere Struktur eines Capa-Agenten ist in Protokollnetze (instanziierte Workflows) und Sub-Agenten (Entscheidungskomponenten) gegliedert.)

Die innere Struktur wird in Form eines Multi-Agenten-Systems modelliert und ausgeführt, wobei jeder Agent z.B. einen Geschäftsbereich der Firma repräsentiert plus Agenten für übergreifende Geschäftsbereiche wie z.B. der Betriebsrat. (Die Agenten sind wie bei der organisationsorientierten Software-Entwicklung mit Positionen, Rollen und Personen modelliert.) Die Agenten für die Geschäftsbereiche werden ebenfalls als Multi-Agenten-System modelliert und ausgeführt, dort sind es dann Abteilungs-Agenten, bis zu der Ebene, wo jeder Mitarbeiter im System durch einen Agenten vertreten ist. (Im Bereich der Notfallmedizin, Telemedizin und für Krankenhausorganisation werden solche feinen Modelle verwendet, wo jeder Arzt und jeder Patient durch einen Agenten im System repräsentiert wird.) Ein solches Modell kann sowohl Top-Down entwickelt werden, so wie hier dargestellt, als auch Bottom-Up, also von bestehenden Systemen z.B. für die Abteilungen ausgehend.

Dieselbe hypothetische Anwendung kann auch als Workflowanwendung betrachtet werden. In dieser Perspektive werden die möglichen Interaktionen mit deren Einzelschritten und alternativen Abläufen in den Fokus gestellt. Eine interorganisationale Zusammenarbeit könnte z.B. Outsourcing sein oder Material-Lieferung. Alle beteiligten Firmen (oder die mächtigste Firma, die auftraggebende Firma) legen die Schritte dieses Ablaufs fest. Alle Taskbearbeiter des Workflows werden wie bei einem Workflowmanagementsystem über Rollen, Rechte und aktive Einheiten modelliert (und intern als Agenten ausgeführt). Dabei sind die Abläufe zunächst auf einer spezifischen Ebene modelliert, haben aber definierte Schnittpunkte zu Abläufen auf den benachbarten Ebenen: Bei einer Lieferanten-Interaktion zwischen zwei Firmen werden in jeder Firma abteilungsinterne Prozesse ausgeführt und letztlich erledigen bestimmte Personen die zugehörigen Aufgaben. Das System kann man sich vorstellen wie in Abbildung 4.16 auf Seite 135 aus Stufe III. Die Modellierung solcher geschachtelter Prozesse ist schwieriger als die hierarchische Modellierung von Subsystemen. Hierarchische Subsysteme entsprechen sehr der menschlichen Fähigkeit und den gebräuchlichen Strategien zur Handhabung von Komplexität, hierarchische Prozesse dagegen brauchen eher ein Gesamtverständnis. (Die Verwobenheit („Komplexität“ im Wortsinne, siehe [Lilienthal 2007, S. 5]) ist bei geschachtelten Prozessen größer, weil sie gerade die Subsysteme, die strukturellen Einheiten miteinander verweben und in Beziehung setzen).

Jeder Agent dieser Anwendung hat sowohl eine innere Struktur (kann also als Multi-Agenten-System betrachtet werden) als auch innere Prozesse, die er koordiniert ausführt (er kann also als Workflowanwendung samt lokalem WFMS betrachtet werden). Jeder Workflow dieser Anwendung stützt sich auf ausführende Einheiten (kann also als Multi-Agenten-System betrachtet werden) und diese Einheiten haben innere Vorgänge, insofern koordiniert der Workflow verschiedene Prozesse und kann deshalb als Workflowanwendung samt innerem WFMS betrachtet werden. Jede Einheit der kompletten Anwendung kann sowohl als Multi-Agenten-System als auch als Workflowanwendung mit eigenem WFMS betrachtet werden, je nachdem, ob der Prozesscharakter oder der Strukturcharakter im Blickwinkel steht. Eine solche Anwendung ist also eine Agenten/Workflowanwendung. Diese Schreibweise ist nach demselben Prinzip gebildet wie der Begriff Praxis/Ordnung bei Köhler, Langer et al. ([Langer 2004]). Langer beschreibt damit eine soziale Organisation, die aus den untrennbaren Aspekten Praxis und Ordnung besteht und erzeugt wird, wobei „Praxis“ sich auf einen

dynamischen Aspekt bezieht und „Ordnung“ auf einen strukturellen Aspekt. In einer Agenten/Workflow-Anwendung existieren in diesem Sinne der strukturelle und der Prozessaspekt untrennbar, aber dabei benennbar und handhabbar.

In diesem Abschnitt wurde eine Vision der fünften Stufe aufgezeigt. Sie addressiert die Anforderungen einer vollständigen Integration von Workflow- und Agententechnologie in einem Rahmenwerk zur Spezifikation und Ausführung von interorganisationalen oder anderweitig komplexen Anwendungen. Vor allem für die Laufzeit wurden in diesem Abschnitt Lösungsvorschläge in Form einer groben Architektur mit Schnittstellentypen vorgestellt. Der einfache Fall mit einigen lokalen Plattformen und darauf laufenden Agenten ist mit dieser Stufe genauso möglich wie in Stufe I, in der Ausführung allerdings gibt es einen beträchtlichen Overhead durch das integrierte WFMS. Ebenso sind die Ideen zur Flexibilisierung auf Stufe III (Externe Task-Agenten, fragmentierte Workflows, hierarchische Workflowmanagementsysteme und mobiler Workflow-Agent) mit dem hier vorgestellten System umsetzbar, und zwar deutlich einfacher zu realisieren (aus der Perspektive der Anwendungsentwicklung), weil die Funktionen im Rahmenwerk „vorbereitet“ sind. Die Betrachtung einer hypothetischen interorganisationalen Anwendung aus Workflow- und Agentenperspektive ermöglicht es, daraufhin ein Gesamtbild einer solchen Anwendung aufzuzeigen und den Begriff Agenten/Workflow-Anwendung zu prägen.

Der nächste Abschnitt fasst alle fünf Stufen und die jeweilige Differenz zu den benachbarten Stufen zusammen.

4.8 Diskussion und Zusammenfassung

In diesem Kapitel wurden zunächst die Gemeinsamkeiten der Agenten und Workflows expliziert, um daraufhin das weitergesteckte Ziel dieses Kapitels im Abschnitt 4.2.1 noch konkreter formulieren zu können: Es soll ein Rahmenwerk zur Anwendungsentwicklung und Anwendungsausführung entworfen werden, so dass eine damit entwickelte Anwendung alle Eigenschaften einer Workflow- wie auch einer Agentenanwendung besitzt. Die Architektur dieses Rahmenwerkes wurde in fünf Stufen aufgebaut.

Zunächst werden die angrenzenden Themen der Umsetzbarkeit und die Beziehung zwischen Paose und der Integration aufgegriffen, danach gibt es eine Zusammenfassung der Stufen und für jede Stufe wird der Bezug zum Ziel der Arbeit festgestellt.

Umsetzbarkeit bezüglich Effizienz

Die Implementierung dieser vielschichtigen Architektur scheint zunächst ineffizient. Wenn aber die Architektur so sauber entworfen ist, kann darüber mit Hilfe von Interfaces auch eine effiziente Implementierung erfolgen. Große Systeme der Wirtschaft haben Querverweise zwischen verschiedenen Komponenten und Prozessen an den Stellen, wo es nötig ist, direkt umgesetzt. Eine Architekturbeschreibung gibt es dazu nicht. Die Implementierung ist wegen des direkten Weges extrem effizient. Eine gute Architekturbeschreibung hilft dann bei der Zuordnung solcher „praktischen“ Wege, steht aber der effizienten Implementierung nicht im Wege. Wichtig ist nur, dass der klare Bezug zur Architekturschicht auch in der Implementierung berücksichtigt wird. Bei

der Implementierung kollabieren die Schichten auf *eine* Laufzeitumgebung und die darin ausgeführten Einheiten.

Die Rolle der PAOSE bei der Integration

Der Ansatz PAOSE – Petrinetzbasierte Agentenorientierte Software-Entwicklung – wird im Abschnitt 3.4 als ein prozessorientierter Ansatz zum Entwurf von Agentenanwendungen vorgestellt. Über die Jahre wird der PAOSE-Ansatz immer weiter verfeinert und konkretisiert. Die Modelle und Vorgehensweisen aus PAOSE werden immer besser durch Entwicklungswerkzeuge unterstützt.

Mit Werkzeugen und sich entwickelnder Automatisierung (wie z.B. Vorschläge für das Roundtrip-Engineering mit Interaktionsprotokollen und Protokollnetzen für CAPA-Agenten) entwickelt sich ein zunehmendes Verständnis darüber, wie die Prozessaspekte in die Architektur einer Anwendung integriert werden können. Diese Entwicklungen sind eine wichtige Triebfeder für die Ideen der vorliegenden Arbeit zur Integration von WFMS und Agenten.

Bezogen auf die Entwicklungszeit befindet sich der PAOSE-Ansatz bereits deutlich im Bereich der Stufe II, denn mittels PAOSE-Techniken kann eine weitreichende Unterstützung für Prozesse zur Entwicklungszeit erreicht werden, indem beim Entwurf und bei der Tätigkeit der Implementierung sowohl die Agenten als auch ihre Interaktionen als eigenständige Entwurfseinheiten betrachtet werden.

Auf Stufe III kann der Implementierungsschritt für Interaktionen in PAOSE angepasst werden, so dass mit einer Vorgehensregel sichergestellt wird, dass jede Interaktion als ein Workflow im verteilten WFMS definiert wird. Die Matrix aus Agenten und Interaktionen hat dann auch zur Laufzeit eine explizite Repräsentation.

Auf Stufe IV kann aus einer Vorgehensregel, dass also Interaktionen als Workflows implementiert werden, ein erweitertes Agentenrahmenwerk entwickelt werden. Dieses bietet eine Schnittstelle zur Definition von Agenteninteraktionen an und speichert diese intern als Workflowdefinitionen. So wird vom Rahmenwerk sichergestellt, dass tatsächlich jede Interaktion in einer Agentenanwendung als Workflow definiert und ausgeführt wird. Ein Teil des PAOSE-Ansatzes wird also in der Entwicklungs- und in der Ausführungsumgebung fest verankert, so dass die Werkzeug- und Methodenvielfalt des Ansatzes wieder vereinfacht wird (durch einen höheren Abstraktionsgrad). Diese Vereinfachung schafft dann Spielraum für die weitere Entwicklung von PAOSE hin zu hierarchischen Systemen. Sobald diese ein erprobtes Vorgehen und Techniken aufweisen, werden wieder Werkzeuge entwickelt und schließlich die Erkenntnisse in eine integrierte Entwicklungs- und Laufzeitumgebung festgeschrieben. Das ist dann die Implementierung der Stufe V.

Bezug der Integrationsstufen zur Prozessinfrastruktur

Wie ist nun der Bezug zwischen den Integrationsstufen und dem Ziel der Arbeit, einer Architektur für eine Prozess-Infrastruktur für Agentenanwendungen (APIA), herzustellen? Am Anfang des Kapitels 3 wird im Abschnitt *Zentrale Begriffe* (Abschnitt 3.1 auf Seite 59) eine *Prozess-Infrastruktur* als eine Ebene in einem komplexen hierarchischen System vorgestellt, welche der übergeordneten Ebene Dienste für die Verwaltung

von Prozessen zur Verfügung stellt. Im Folgenden wird für jede Stufe angegeben, wo eine Prozess-Infrastruktur darin benannt werden kann. Gleichzeitig werden alle Stufen rückblickend zusammengefasst.

In der ersten Stufe wird eine durchgehende Identität für Systemteile (Prozesse / Vorgänge / Workflows oder Struktur / Komponenten / Agenten) durch ein Workflowmanagementsystem oder ein Agentenmanagementsystem ermöglicht (zweischichtige Architektur).

Auf Stufe I für Workflows bildet das WFMS genau diese Ebene der Prozess-Infrastruktur. Alle Aktionen in der Anwendung werden einer Workflowinstanz zugeordnet. Für Agenten gibt es eine solche Prozess-Infrastruktur-Ebene nicht. Die Standard-Interaktionsprotokolle der FIPA legen zwar zur Entwurfszeit eine prozessorientierte Sicht auf eine Anwendung nahe, zur Laufzeit gibt es allerdings nur Agenten als Einheiten mit eigener Laufzeit-Identität, welche einzelne Nachrichten austauschen. Ein konkreter Agent wird dabei natürlich den Zusammenhang mehrerer Nachrichten in einer Interaktion bewahren, aber die Zusammengehörigkeit von Nachrichten im Multi-Agenten-System wird nicht gewahrt.

In der zweiten Stufe wird das Rahmenwerk differenziert: Es wird ein agentenbasiertes Workflowmanagementsystem eingehend betrachtet und die prinzipielle Möglichkeit eines workflowbasierten Agentenmanagementsystems der Symmetrie halber erwähnt. So können auf Werkzeugebene einige Vorteile der jeweils anderen Technologie eingebunden werden. Es entsteht eine dreischichtige Architektur.

Auf Stufe II für Agenten bildet das unterliegende WFMS eine Prozess-Infrastruktur, die der Implementierung und Ausführung der Agentenplattform (also der Laufzeitumgebung für Agenten) zugute kommt. Auf der anwendungsspezifischen Ebene gibt es weiterhin keine Prozess-Infrastruktur für die Anwendungsagenten. Durch Paose kommen zwar prozessorientierte Aspekte in die Anwendungsentwicklung, dies wirkt sich jedoch weder auf die hier verwendete Architekturbetrachtungsebene aus, noch auf das Verhalten oder die Prozess-Beobachtungsmöglichkeiten zur Laufzeit.

In der dritten Stufe werden der Anwendung sowohl ein vollständiges WFMS als auch ein Agentenmanagementsystem zur Verfügung gestellt, indem der Durchgriff auf die unterste Schicht explizit erlaubt wird. Das agentenbasiserte WFMS muss dazu seine Funktionen in einer Agentenkommunikationssprache „verpackt“ anbieten, so dass die Anwendung nicht in zwei unabhängige Teile zerfällt. Damit für die Stufe III die Anwendungsentwicklung nicht chaotisch wird, muss eine Art und Weise der Nutzung von Agenten und Workflows und ihre Berührungspunkte definiert werden. Die wichtigen Prozesse und Vorgänge einer Agentenanwendung können nun als Workflows modelliert und ausgeführt werden.

Auf Stufe III steht den Anwendungsagenten ein WFMS zur Verfügung. So können die wichtigsten Anwendungsprozesse explizit als Workflows dargestellt und ausgeführt werden. Es liegt hier in der Gestaltungsfreiheit der Anwendungsentwickler, inwieweit das WFMS ein bedeutender Teil der Anwendungs-Infrastruktur wird.

In Stufe IV wird diese Entwurfsentscheidung in einer eigenen neuen Integrationsschicht gekapselt, es wird also die Anwendungsschicht der Stufe III in zwei Schichten geteilt (vierschichtige Architektur). Die Integrationsschicht kann entweder Agentenanwendungen oder Workflowanwendungen unterstützen, dabei werden bei einer

Agentenanwendung alle Interaktionen (also Prozesse zwischen Agenten) in Form von Workflows intern abgebildet und ausgeführt.

Auf Stufe IV ist das WFMS in das Rahmenwerk für Agentenentwicklung integriert, so dass bei der Implementierung und bei der Ausführung von Agenten die Prozesse der Anwendung von der Prozess-Infrastruktur unterstützt werden. Das Ergebnis der Stufe IV ist APIA, die Architektur einer Prozess-Infrastruktur für Agentenanwendungen.

In der Stufe V wird die Integrationsschicht der Stufe IV differenziert und in ihrer Funktion erweitert, so dass ausgewählte Möglichkeiten der Stufe III, die Zwecks größerer Klarheit in Stufe IV eingeschränkt wurden, auf strukturierte Weise zur Anwendungsentwicklung verfügbar sind.

Auf Stufe V wird die Prozess-Infrastruktur in ihrer Tragweite umfassender, da sie auch die agenteninternen Prozesse explizit unterstützt. Hier kommt die Prozess-Infrastruktur auch für hierarchische Systeme durchgehend zum Tragen.

Es wird abschließend der Begriff **Agenten/Workflowanwendung** eingeführt, um die neue Qualität der Einheiten und ihrer Identität und Eigenschaften zu verdeutlichen. Die Agent/Workflow-Einheiten sollen rekursiv schachtelbar sein, so dass komplexe Systeme auf der Grundlage von verständlichen und eigenständig lauffähigen lokalen Komponenten entwickelt werden können. Die internen Vorgänge und Prozesse einer Komponente werden vom Rahmenwerk wie Workflows ausgeführt und die interne Struktur einer Komponente wird vom Rahmenwerk wie mit Agenten dargestellt.

Nach dieser konzeptionellen Integration der Workflow- und Agententechnologie wird im nächsten Kapitel der Prototyp vorgestellt, welcher Stufe III erreicht und darüber hinaus einen Workflow-Agenten umsetzt.

Kapitel 5

Prototyp

Nach den konzeptionellen Kapiteln 3 und 4 folgt nun die Umsetzung zu einem Prototypen. Zunächst werden die Bestandteile der Prozess-Infrastruktur in einem Entwurfsabschnitt beschrieben (5.1). Es folgt ein Abschnitt zu den vorbereitenden Umsetzungsarbeiten am Rahmenwerk Capa (Abschnitt 5.2). Der Hauptteil der Realisation ist dann im Abschnitt 5.3 dargestellt: Es wird ein agentenbasiertes WFMS entsprechend der Integrationsstufe III mit Hilfe des Paose-Ansatzes entworfen. Der Abschnitt fasst die Ergebnisse des Lehreprojektes *Agentenorientierte Software-Entwicklung* im Wintersemester 2005 / 06 und z.T. aus dem Wintersemester 2006 / 07 zusammen (einige Vorarbeiten wie die Anbindung von Capa an Agentcities reichen dabei bis zum Wintersemester 2002 / 03 zurück, und aus jedem Jahresdurchlauf sind einige Ergebnisse in diese Arbeit eingeflossen). Der Workflow-Agent (Abschnitt 5.4) ist ein erstes Fragment zur Realisierung eines Rahmenwerkes entsprechend Stufe IV und wurde im Wintersemester 2006 / 07 entwickelt.

5.1 Entwurf des Prototypen

Im Abschnitt 3.6 wurde deutlich, dass ein agentenbasiertes WFMS ein wichtiger Bestandteil für die Umsetzung von Pia ist. Dieses WFMS entspricht dem in der Architektur zu Stufe III (Abbildung 4.13 auf Seite 131) verwendeten WFMS und ist der erste Schritt in der prototypischen Umsetzung. Pia ist als verteilte Schicht konzipiert, deshalb muss das WFMS in der Lage sein, mit anderen WFM-Systemen zu kommunizieren und so ein verteiltes, agentenbasiertes WFMS zu bilden. Aus den verschiedenen Möglichkeiten, die im Abschnitt zu Stufe III (Abschnitt 4.5 auf Seite 130) vorgestellt wurden, wurde für die prototypische Umsetzung der Workflow-Agent gewählt.

Es wird zuerst ein agentenbasiertes WFMS entworfen, dann ein Workflow-Agent für zusätzliche Flexibilität und schließlich wird das Zusammenspiel der Bestandteile für Stufe III und Stufe IV erläutert. In den folgenden Abschnitten dieses Kapitels werden das agentenbasierte WFMS und der Workflow-Agent prototypisch umgesetzt.

5.1.1 Ein agentenbasiertes WFMS

Ein agentenbasiertes Workflowmanagementsystem soll Benutzer, Rollen, Taskdefinitionen, Formulare, Teilworkflows und Regeln (zur Verbindung von Rollen und Tasks zu effektiven Rechten) verwalten. Benutzer (Task-Bearbeiter oder Administratoren) sollen sich einloggen und entweder Einstellungen vornehmen oder Workitems in instanziierten Workflows bearbeiten können. Das heißt, dass das grundlegende Nutzungsszenario für ein WFMS unterstützt wird. Ein agentenbasiertes WFMS ist nun insofern qualitativ anders als andere WFMS-Umsetzungen, als die unterliegenden Konzepte und Techniken besonders flexibel und mächtig sind. Die Funktionen sind zunächst

dieselben wie im WFMS von Jacob (siehe Abschnitt 2.3.4). Damit ist die Funktionalität des agentenbasierten WFMS ausreichend spezifiziert; die weitere Spezifikation der Funktionen in Form von Anwendungsfällen wird im Abschnitt 5.3.1 erstellt.

Im Abschnitt 4.5 wurden auch Funktionen beschrieben, die über die beschriebenen grundlegenden Funktionen eines WFMS hinausgehen und im Prototypen der vorliegenden Arbeit nicht umgesetzt werden. Im Folgenden wird der Aufbau eines *lokalen* WFMS als geschachtelte Struktur aufgegriffen, dann der Aufbau eines *verteilten* WFMS als geschachtelte Struktur und schließlich die Projektion der Kommunikationswege auf lokale Kommunikation.

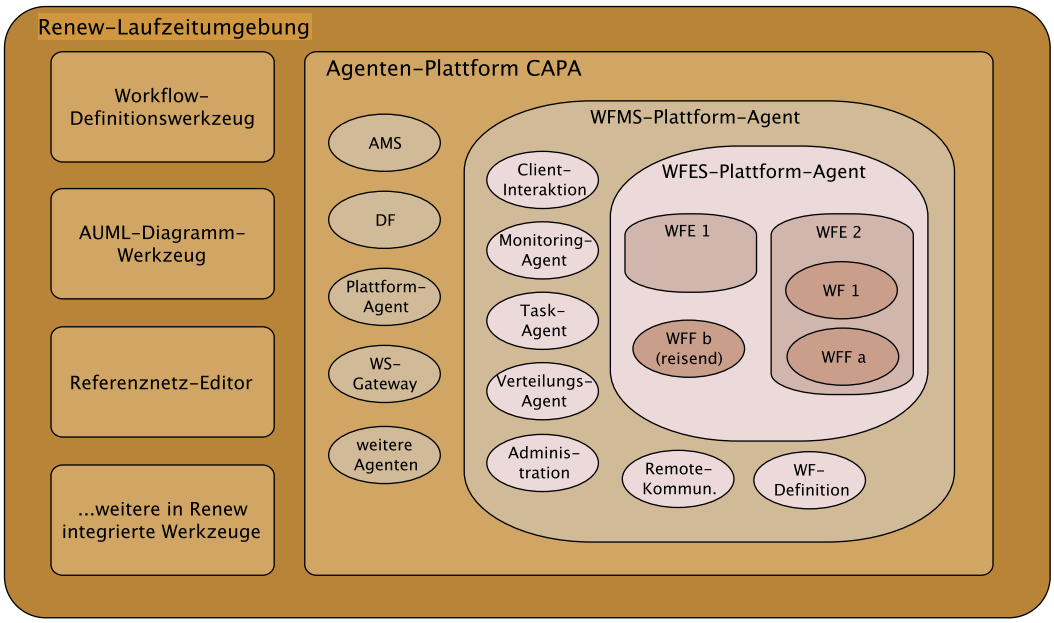


Abbildung 5.1: Geschachtelte WFMS-Agenten. Rechtecke notieren Plugins, Ovale oder ovale Rechtecke notieren Agenten. Aus: [Reese et al. 2006b]

In Stufe II (Seite 126) wurde eine Architektur für ein agentenbasiertes Workflowmanagementsystem (WFMS) vorgestellt und in Stufe III (Seite 130) wurden weitere Agenten zum WFMS hinzugefügt. Alle vorgestellten Agenten sind in Abbildung 5.1 dargestellt. Das Bild repräsentiert den Stand vor Beginn des Lehreprojektes, deshalb weichen auch die Namen der Agenten und die Aufteilung der Funktionen leicht voneinander ab. Grundsätzlich stimmen sie jedoch mit dem entwickelten Prototyp überein. Die Agentenstruktur entspricht der aus Abbildung 4.15, nur dass die Namen der Agenten hier ausgeschrieben sind. Das Bild zeigt Capa als Plugin von Renew sowie einige weitere Plugins, die für das WFMS von Bedeutung sind, so z.B. der Editor zur Definition von Workflownetzen. Der WFMS-Plattform-Agent wird von einem eigenen Plugin zur Verfügung gestellt. Die Abhängigkeiten der Plugins werden im Abschnitt 5.3.1 dargestellt. Die Funktion der umgesetzten Agenten wird in den Abschnitten 5.3.2 und 5.3.3 näher erläutert.

Für den verteilten Fall zeigt Abbildung 4.16 auf Seite 135 die Schichten, in denen eine verteilte Anwendung strukturiert sein kann. Jeder WFMS-Agent ist wie im lokalen Fall aus mehreren Agenten aufgebaut. Zwei solche WFMS-Agenten können über einen zusätzlichen WFMS-Agenten gekoppelt werden, so dass dieser übergreifende WFMS-Agent als ein verteilt implementierter Agent betrachtet werden kann. Diese Art von Schachtelung der WFMS-Agenten ist beliebig häufig möglich. Schließlich nutzt eine verteilte Anwendung das verteilte WFMS.

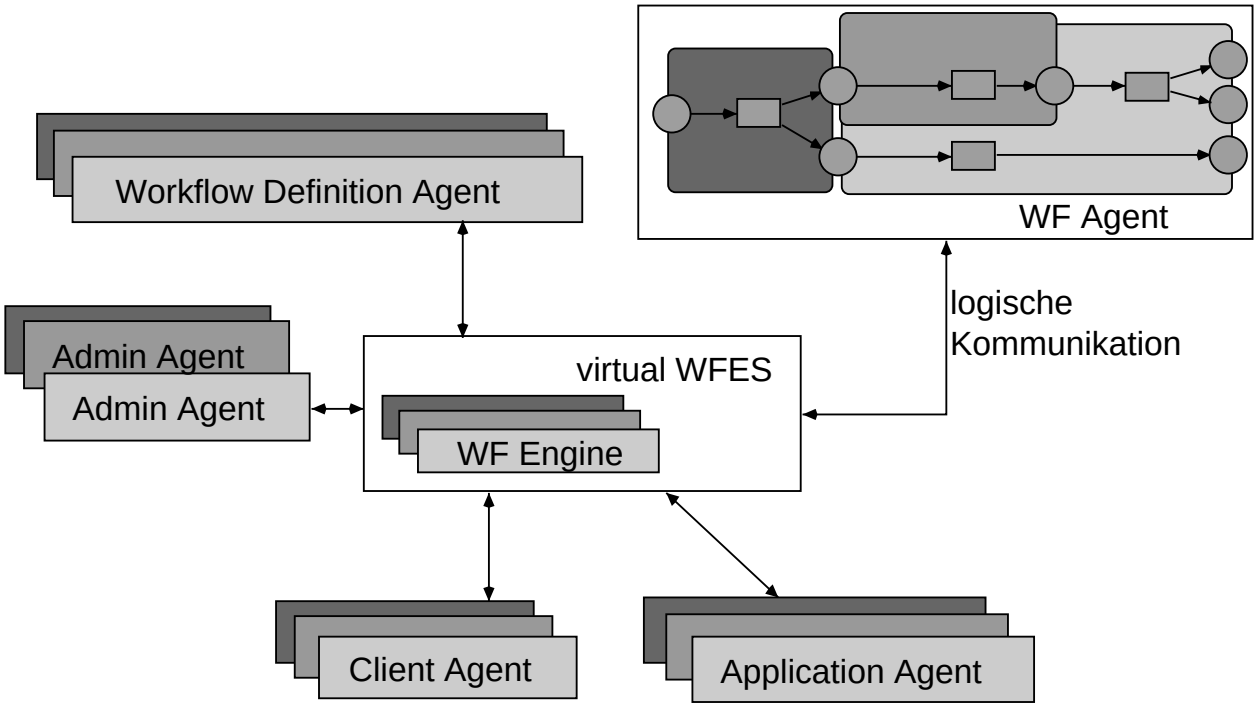


Abbildung 5.2: Virtuelle Agenten. Geändert aus: [Reese et al. 2006a]

Das übergeordnete WFMS besitzt einen Workflow-Enactment-Service (WFES), der auf die jeweiligen lokalen WFES-Agenten zugreift, nach außen jedoch als ein WFES auftritt. Ebenso können Teile des Workflows auf verschiedenen lokalen Plattformen definiert sein. Die in gleichen Graustufen dargestellten Teile eines WFMS in Abbildung 5.2 kennzeichnen Teile aus je einem lokalen WFMS. Die Vorstellung hinter dem Begriff „logische Kommunikation“ ist, dass die verteilt implementierten Agenten nicht unbedingt direkt miteinander kommunizieren, sondern die lokalen Agenten die Kommunikation, die zur Ausführung eines Workflows nötig ist, abwickeln.

5.1.2 Der Workflow-Agent

Durch einen Workflow-Agenten, der einen instanziierten Workflow als solches repräsentiert, wird nun entscheidende zusätzliche Flexibilität bereitgestellt. Dieser Workflow-Agent stellt eine zusätzliche Indirektionsstufe dar und damit eine flexible Verbindung zwischen der Erzeugung (Instanziierung), der Ausführung und evtl.

vorhandenen Änderungen an einer Workflowinstanz: Der Workflow-Agent kennt die Umstände seiner Erzeugung wie Informationen über den Anforderer, das Heimat-Workflowmanagementsystem und die (ursprüngliche) Workflowdefinition. Dem Workflow-Agenten sind auch die Umstände der (verteilten) Ausführung bekannt. Neben der eigentlichen Ausführung können dem Workflow-Agenten zusätzliche Bedingungen, Ziele und Freiheitsgrade bezüglich Änderungen an der Workflowdefinition mitgegeben werden. Für Abweichungen oder Änderungen und für Entscheidungen über die weitere Ausführung stehen dem Workflow-Agenten all diese Informationen zur Verfügung. Das Ziel des Workflow-Agenten ist grundsätzlich, dass die Workflowdefinition zur Zufriedenheit des Initiators ausgeführt wird, unter beschränkten Möglichkeiten der direkten Umgebung, bestimmte Tasks zu bearbeiten. Jedesmal, wenn der Workflow-Agent eine neue Interaktion in Gang setzt, handelt er proaktiv. Je nach den Gegebenheiten der Umgebung entscheidet der Agent sich für einen Interaktionstyp und seinen nächsten Gesprächspartner.

Um die Ausführung eines Workflows zu verteilen, kann der Workflow-Agent entweder durch mehrere Workflowfragment-Agenten implementiert werden (unter Verwendung der Sterntopologie, wie sie im Abschnitt 3.3 vorgestellt wurde) oder als ein mobiler Agent.

Die Erzeugung des Workflow-Agenten geschieht dynamisch bei der Instanziierung eines Workflows. Dies kann grundsätzlich entweder von außen, von Benutzerseite her geschehen oder von innen, vom WFMS selbst. Im letzteren Falle kann der Workflow-Agent als ein vertrauenswürdiger Teil des Workflowmanagementsystems betrachtet werden.

Durch den Workflow-Agenten wird die Interaktion zur Ausführung eines Workflows zwischen WFES und Workflowengine (WFE) flexibler. Die Möglichkeiten, die Aufgaben bei der Workflowausführung neu zu verteilen umfassen unter anderem:

(a) Der mobile Workflow-Agent transportiert die Workflowdefinition und übergibt sie der lokalen WFE zur Ausführung (dies ist die Wahl für den ersten Prototypen) (b) Der Workflow-Agent transportiert die Workflowdefinition und führt sie auch aus, dabei interagiert er mit dem lokalen WFES als Schnittstelle zur lokalen Umgebung (also das lokale WFMS mit Benutzern und deren Fähigkeiten).

Dabei sind verschiedene Formen von Zwischenpositionen möglich, z.B. könnte der Workflow-Agent auch jeden aktivierten Task einzeln an die lokale WFE zur Ausführung übergeben.

Bei seiner Erzeugung wird der Workflow-Agent mit Initialisierungsinformationen versorgt, die dann nicht mehr von außen geändert werden können (z.B. die Workflowdefinition und Informationen über den Erzeuger). Der für diese Funktion entworfene Mechanismus wird in Abschnitt 5.2.4 genauer dargestellt. Von den Fähigkeiten des Workflow-Agenten hängt ab, ob die eigentliche Workflowdefinition zur Laufzeit geändert werden kann: Entweder er bietet eine Schnittstelle nach außen an, über die Änderungen beantragt werden können, oder er entscheidet aufgrund von Gegebenheiten der Umgebung, dass eine Änderung notwendig ist, um den Workflow zum Ende zu bringen. Auch der Status der Workflowausführung kann vom Workflow-Agenten verwaltet werden: Er kann die Ergebnismeldung einer lokalen WFE begutachten und entscheiden, ob er die entsprechenden Tasks intern als bearbeitet betrachten will oder ob sie noch einmal unter anderen Randbedingungen ausgeführt werden müssen.

Die verschiedenen Unterziele des Workflow-Agenten (das nächste passende WFMS für den Workflow finden, migrieren, den Workflow ausführen und beenden und schließlich zur Heimatplattform migrieren) sind in einem Lebenszyklus koordiniert.

Damit ist die Funktionalität des Workflow-Agenten zunächst ausreichend spezifiziert; die weitere Spezifikation wird im Abschnitt 5.4 vorgestellt.

5.1.3 Workflowmanagement für Agentennetze

Zunächst ist ein Agentennetz eine Infrastruktur, die Verzeichnisdienste, Identitätsmanagement und Nachrichtentransport kombiniert zur Verfügung stellt.

Ein agentenbasiertes Workflowmanagementsystem (WFMS) wie es in Kapitel 4 als Stufe III vorgestellt wurde, kann in einem Agentennetz als Dienst zur Verfügung gestellt werden. Agenten können mit diesem Dienst Prozesse initiieren, beobachten, steuern, ändern oder stoppen, wobei die Rechte und Rollen ein Mittel sind, um die Informationen zu strukturieren. So kann eine Agentenanwendung entwickelt werden, in der einige Prozesse von dem WFMS organisiert werden. Ein Agent, der das WFMS nutzen möchte, braucht also Zugang zum Agentennetz und muss die Interaktionsprotokolle des WFMS kennen.

Eine Prozess-Infrastruktur für Agentenanwendungen (Pia), wie sie als Stufe IV benannt wurde, kann wie folgt ausbuchstabiert werden: Ein Agentenrahmenwerk bietet eine Schnittstelle zur Definition von Agentenanwendungen an, welche Interaktionsprotokolle (AIPs) entgegennimmt. Das Rahmenwerk setzt diese Interaktionsprotokolle intern in Workflowdefinitionen um. Wenn bei der Ausführung ein Agent die betreffende Interaktion startet, wird vom Rahmenwerk in Form eines verteilten WFMS ein Workflow instanziiert, in dem festgelegt ist, welcher Rolle welche Teilaufgabe zugeordnet ist. In den Werkzeugen zur Beobachtung einer Anwendung zur Laufzeit können nun einerseits einzelne Agenten mit ihrem internen Zustand betrachtet werden und andererseits einzelne Interaktionen in Form von instanziierten Workflows mit ihrem aktuellen Zustand. Es könnte auch Werkzeuge zur Ansicht geplanter und abgeschlossener Interaktionen geben.

Die IT-Infrastruktur einer Anwendung, die auf Pia aufbaut, hat die folgenden Bestandteile:

- Ein Agentennetz mit Verzeichnisdiensten und festen Dienstmerkmalen wie Erreichbarkeit unter festen IP-Adressen (dies umfasst wiederum die Infrastruktur für dieses Agentennetz und je nach Anwendungszusammenhang eine ausreichende Anzahl Nutzer, welche die Verzeichnisdienste füllen);
- Eine IT-Infrastruktur für die anwendungsspezifischen Bestandteile (Gebäude und Rechner als Server und als Plattformen für Agentenplattformen, ein Agentenrahmenwerk u.s.w.);
- Damit die beiden vorigen Punkte zusammen passen, braucht man die Spezifikationen zur Zusammenarbeit, in diesem Falle z.B. die Standards der FIPA und die Vorgaben des Agentennetzes (z.B. das Ping-Protokoll für Agentcities).
- Die Prozess-Infrastruktur selbst (ebenfalls mit Vorgaben und Protokollen zur Benutzung), welche ihrerseits die vorigen Punkte als Infrastruktur nutzt.

Wenn die Prozess-Infrastruktur wie in Stufe V beschrieben eingesetzt wird, kann ein Agent auch seine internen Prozesse organisieren, insbesondere wenn er seinerseits verteilt implementiert ist.

Dieser Teil wird im Rahmen der vorliegenden Arbeit nicht weiter ausbuchstabiert.

5.2 Arbeiten am Rahmenwerk Capa

In diesem Abschnitt sind einige Arbeiten zusammengefasst, die im Umfeld des vorgestellten Projektes durchgeführt wurden und notwendig für den Prototypen sind. Die konzeptionelle Motivation und eine Zusammenfassung der Ergebnisse sind bereits im Abschnitt 3.2 vorgestellt worden.

5.2.1 Agenteninterne Kommunikation

Bis zur Implementierung des agentenbasierten WFMS gab es innerhalb von Capa-Agenten Kommunikationskanäle für Protokollnetze, für die Protokollfabrik und für die Wissensbasis. Kommunikation zwischen Protokollnetzen war entweder über Einträge in der Wissensbasis möglich oder über Nachrichten, die der Agent sich selbst schickt. Weiter gab es eine Planer-Schnittstelle, welche für Prolog- oder Java-Planungskomponenten verwendet wurde. Diese Schnittstelle war Teil des Wissensbasis-Netzes im Capa-Standard-Agenten. Die Schnittstelle war so konzpiert, dass Protokollnetze dort Informationen hineingaben und dann über die Wissensbasis proaktive Protokolle entsprechend des Planungsergebnisses gestartet wurden. Dies reicht für das WFMS nicht aus, weil für die Workitems und Activities von verschiedenen Protokollnetzen aus Informationen in beide Richtungen ausgetauscht werden müssen. Außerdem soll das Agentennetz (das Referenznetz, welches einen Capa-Agenten repräsentiert) die Kommunikationswege seiner Bestandteile explizit machen. So wurde also für das WFMS die Planer-Schnittstelle zu einer Entscheidungskomponente (engl. Decision Component, kurz DC) verallgemeinert und aus der Wissensbasis in den Agenten verlegt.

Nach der bisherigen Konsolidierung sieht nun das vergröberte Agentennetz aus wie in Abbildung 5.3 dargestellt. Entscheidungskomponenten werden ähnlich wie Protokollnetze von der Fabrik erzeugt und liegen dann in einer Stelle des Agentennetzes, wo eine Schnittstelle bereitgestellt wird. Diese Schnittstelle umfasst den Zugriff auf die Wissensbasis, den bidirektionalen Informationsaustausch mit Protokollnetzen sowie unter den Entscheidungskomponenten. Es ist denkbar, diese stark verallgemeinerte Schnittstelle wieder einzuschränken.

Für den Informationsaustausch ist sowohl eine synchrone als auch eine quasi-asynchrone Variante vorgesehen. Die Verfeinerung der beiden Exchange-Transitionen zeigt Abbildung 5.4. Die Kanäle werden auf beiden Seiten mit Strings gekennzeichnet, welche als Teil der Ontologie einer Anwendung definiert werden. Für die asynchrone Kommunikation wird eine Nummer vergeben, unter der mehrere Kommunikationsfälle untereinander in Beziehung gesetzt werden können, solange sich beide Kommunikationspartner die Nummer merken. Die Vergabe einer eindeutigen Nummer wird im Agentennetz in der Verfeinerung der dargestellten Exchange-Transistionen realisiert, ebenso die Unterscheidung in synchrone und asynchrone Kommunikation (vgl. Abbildung 5.4). Protokollnetze untereinander können diese Kanäle nicht benutzen und

Entscheidungskomponenten können keine Nachrichten von Außen empfangen oder nach Außen versenden.

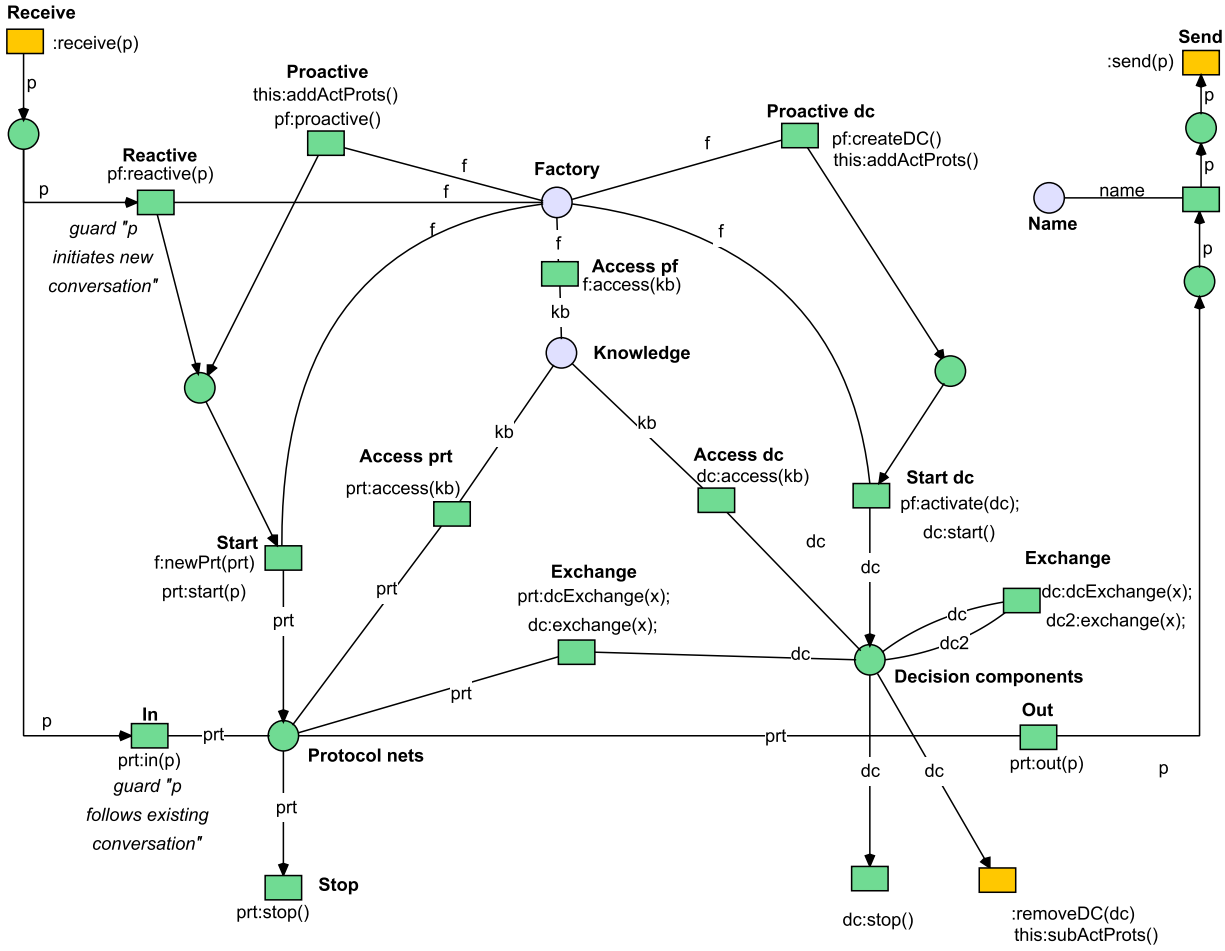


Abbildung 5.3: Erweitertes Agentennetz in Capa. Die beiden Exchange-Transitionen sind in Abbildung 5.4 in ihrer Verfeinerung dargestellt.

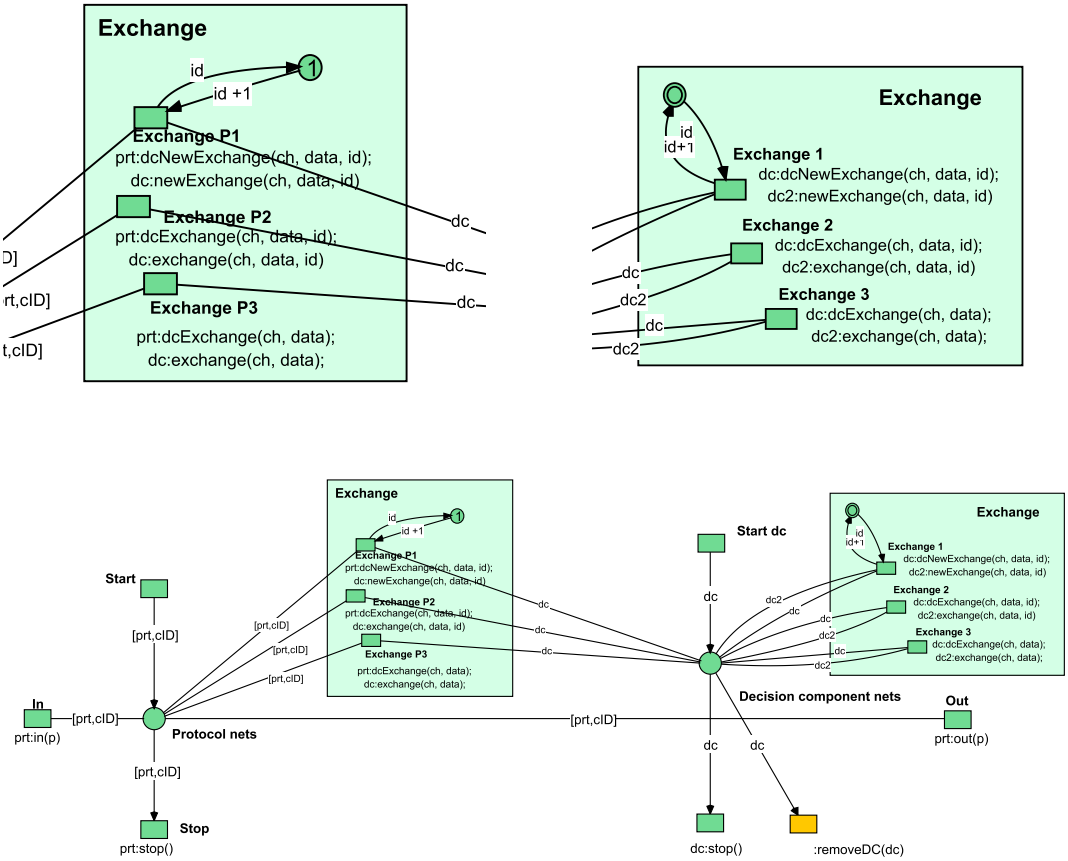


Abbildung 5.4: Ausschnitt aus dem Agentennetz mit Verfeinerung der beiden Exchange-Transitionen in Abbildung 5.3. Unten mit angrenzenden Netzelementen dargestellt, darüber dieselben Transitionen in lesbarer Schriftgröße.

5.2.2 Netzwerkanbindung

Es gibt zwei Konzepte für die Netzwerkanbindung von Capa: Das Plugin Ace realisiert die Anbindung an Agentcities und openNet und der zentrale DF (ZDF) realisiert ein an die Bedürfnisse im lokalen Subnetz angepasstes Netzwerk-Konzept.

ACE

In Capa wird die Netzwerkanbindung durch das Plugin Ace bereitgestellt und über Properties beim Start von Renew konfiguriert.

Ace wurde im Rahmen meiner Diplomarbeit [Reese 2003] erstellt, und in der Diplomarbeit von Wojciech Laka [Laka 2007] für openNet erweitert. Die Konzepte und Umsetzung zum zentralen DF habe ich im AOSE-Projekt 2004 / 2005 in Zusammenarbeit mit Wojciech Laka und Teilnehmern des Projektes entwickelt.

ACE stellt einen Ping-Agenten bereit sowie für openNet einen Platform-Service-(PS-) Agenten, welcher die Plattform im Netz vertritt.

ACE stellt außerdem Proxy-Agenten bereit, um mit CAPA einen eigenen openNet-Knoten betreiben zu können. Hinter den Proxy-Agenten laufen die original-openNet-Agenten, so dass neue Implementationen sofort übernommen werden können.

ZDF

Da sowohl Agentcities als auch openNet nicht mehr als offene und belebte Plattformen im Netz zur Verfügung stehen, gibt es in CAPA eine Umsetzung für ein lokales Netz von CAPA-Plattformen. Dazu wird über eine Property des Platform-Plugins die Adresse eines zentralen DF konfiguriert („ZDF“). Der DF einer jeden CAPA-Plattform, die zur Benutzung des zentralen DF konfiguriert wurde, sendet alle DF-Ereignisse (Änderungen im Dienste-Verzeichnis) an den zentralen DF. Eine Suchanfrage wird entsprechend ihrer einschränkenden Parameter ebenfalls an den zentralen DF weitergeleitet, so dass im Netz alle Anbieter eines gesuchten Dienstes aufgefunden werden können. Das Konzept unterscheidet sich grundlegend von der Funktionsweise im Agentcities-Verzeichnisdienst, weil die Anwendung verschieden ist: Agentcities verbindet verschiedene Forschungsgruppen, die jeweils eine oder mehrere Plattformen angemeldet haben. Feste Internet-Adressen für die Plattformen sind eine Voraussetzung für die Anmeldung. Ob und wie lange die Plattformen tatsächlich laufen und erreichbar sind, wird mit Hilfe der Ping-Agenten laufend geprüft und als Information jeweils aktuell zur Verfügung gestellt.

Innerhalb der Arbeitsgruppe rund um die vorliegende Arbeit ist das Konzept der Identität einer Plattform ein anderes: alle Arbeitsgruppen arbeiten auf den beiden Unix-Servern der Universität, haben also meistens dieselbe Internetadresse bis auf den Port. Der Port wiederum kann sich bei jedem Start einer Plattform ändern, sofern er nicht explizit konfiguriert wird. In der Implementierungs- und Testphase werden außerdem immer dieselben Agententypen gestartet, deren Identität jedoch außer im Falle der mobilen Agenten weder über Plattformen hinweg noch über mehrere Starts einer Plattform hinweg erhalten bleiben soll. Die Lebensdauer eines Agenten ist damit auf die tatsächliche Laufzeit einer Plattform beschränkt. Aus diesem Grund werden bei der Anmeldung eines Agenten und seiner Dienste beim zentralen DF alle dort verzeichneten Einträge zu dieser Adresse gelöscht: Der zentrale DF geht davon aus, dass die Agenten neu gestartet wurden und damit eine neue Identität erhalten haben.

Die konzeptionelle Grundlage für den zentralen DF sind Verbindungen von DF-Agenten, wie sie von der FIPA vorgestellt wurden. Diese Verbindung wird, da es wesentlich leichter konfigurierbar ist, automatisch etabliert: Implizit meldet sich der zentrale DF bei jeder gestarteten CAPA-Plattform an und abonniert den oben beschriebenen Informationsdienst und wird infolge dessen über die DF-Ereignisse informiert.

5.2.3 Verzeichnisdienst und Subscription Manager

Das Ergebnis einer Suche im verteilten Verzeichnisdienst wird durch drei Parameter bestimmt: die Suchstrategie (policies), explizite Beschränkung (constraints) und Ein-

stellungen (preferences). Der DF von CAPA wurde in diesem Sinne im Rahmen dieser Arbeit erweitert: Der DF kann eine Suchanfrage an registrierte DFs weiterleiten (entsprechend der Suchstrategie). Um Schleifen zu verhindern, muss der DF feststellen können, ob eine Suchanfrage bereits bearbeitet wurde. Eine herkömmliche Suchanfrage enthält nur Einstellungen für eine Suche (nämlich das Suchmuster). Explizite Beschränkung wird von der FIPA in Form eines optionalen Feldes `search-constraints` spezifiziert, welches die maximale Suchtiefe und die maximale Anzahl an Ergebnissen enthalten kann sowie eine identifizierende Markierung einer Suchanfrage.

Für offene, dynamische Netze von Dienst-Anbietern und Dienst-Konsumenten gibt es nach [MBALA et al. 2005] vier typische Anforderungen, die hinsichtlich der Anzahl und Größe der erforderlichen Nachrichten am Besten von einem Verzeichnisdienst erfüllt werden können:

(1) Einfache Suche. Ein Suchender möchte entsprechend der Strategie und unter expliziten Beschränkungen für ein bestimmtes Suchmuster den aktuellen Stand im Netz erfahren.
(2) Persistente Suche. Ein Suchender möchte dauerhaft aktuelle Informationen über den Stand im Netz erhalten, wobei Suchstrategie, Beschränkungen und Suchmuster wie bei der einfachen Suche berücksichtigt werden. Sobald der Suchende dies nicht mehr wünscht, meldet er sich ab.
(3) Subscribe-One. Ein Dienstkonsument möchte einen (Informations-) Dienst bei genau einem Dienstanbieter abonnieren. Falls der aktuell verwendete Anbieter ausfällt, möchte der Dienstkonsument irgendeinen anderen Anbieter beauftragen.
(4) Subscribe-All. Ein Dienstkonsument möchte einen (Informations-) Dienst von allen passenden Anbietern abonnieren. Jeder neu angemeldete Anbieter soll möglichst bald beauftragt werden.

Aufbauend auf dem vorhandenen Mechanismus der lokalen Suche hat Wojciech Laka im Rahmen seiner Diplomarbeit [LAKA 2007] unter Verwendung der FIPA-Spezifikationen und der Ausarbeitung von MBala, Padgham und Winikoff zum Thema Subscription Management ([MBALA et al. 2005]) unter meiner Anleitung den DF von CAPA um die oben beschriebenen Fähigkeiten erweitert. Dem DF wird eine zusätzliche Rolle SubscriptionManager zugewiesen. Zur Umsetzung der neuen Funktionen nutzt der DF nun seine eigenen Funktionen: alle Funktionen benötigen die Überwachung gewisser DF-Ereignisse (Registrierung, Abmeldung usw.). Diese Überwachung ist als interne persistene Suche zu verstehen und auch so implementiert. Damit agiert der DF (1.) als klassischer Verzeichnisdienst, (2.) als Subscription Manager und eine Ebene darunter sowohl (3.) als Anbieter eines abonnierbaren Dienstes (nämlich Monitoring von DF-Ereignissen) als auch (4.) als Nutzer (desselben) abonnierbaren Dienstes. Die Abbildungen 5.5, 5.6, 5.7 und 5.8 zeigen beispielhaft, wie einige der neuen Funktionen des DF genutzt werden können. Die Abbildungen stammen aus der Dokumentation während der Umsetzung des Prototyps. Bei der Beschriftung der Nachrichtenpfeile ist zu beachten, dass `subscribe` ein eigenes Performativ[1] ist, welches z.B. bei der persistenten Suche das Performativ `request` ersetzt, ansonsten bleibt die Nachricht die gleiche wie für eine einfache Suche.

[1] Performativ = Sprechakt, vgl. Abschnitt 2.2.2 auf Seite 34

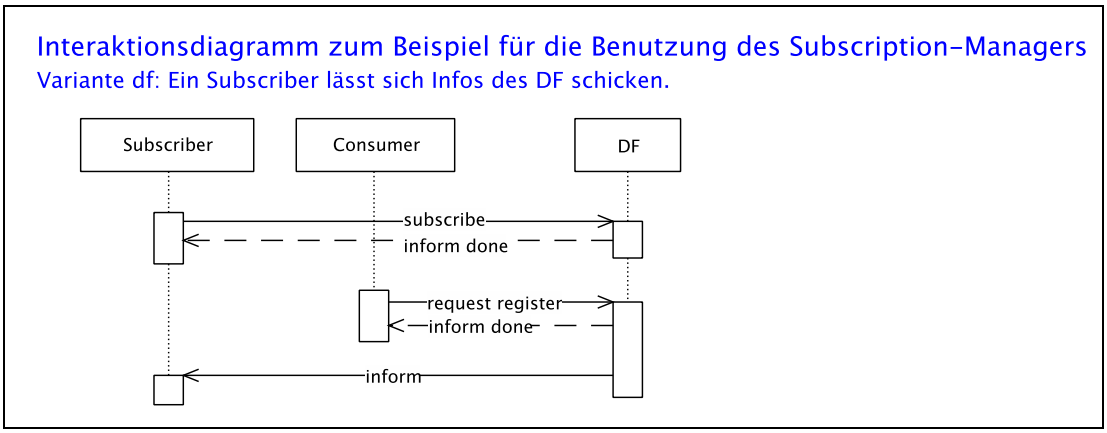


Abbildung 5.5: Ein Subscriber kann sich dauerhaft über bestimmte DF-Ereignisse informieren lassen.

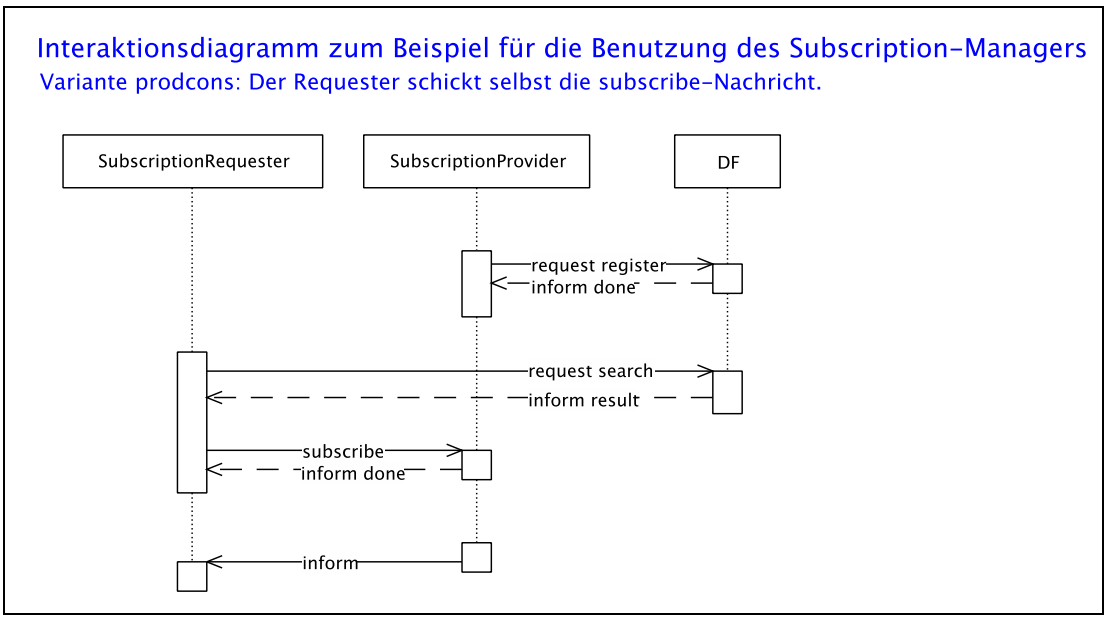


Abbildung 5.6: Ein beliebiger Agent kann dauerhaft bestimmte Informationen versenden.

5.2.4 Interaktion für Erzeuger und erzeugten Agent

Wenn ein Agent einen anderen Agenten erzeugt, schickt er dem lokalen AMS einen Namen und das initiale Wissen des neuen Agenten in String-Form. Aus konzeptioneller Sicht müsste der Erzeuger dieses Wissen einfach anpassen, z.B. indem er einen Schlüssel mit dem eigenen Agent-Identifier hinzufügt und einen Eintrag für ein reaktives Protokoll, bei dem im Nachrichtenmuster der Agent-Identifier des Erzeugers einbezogen ist. In der Implementierung wird jedoch das initiale Wissen von Agenten in Dateien gespeichert, aus denen es mit Hilfsmethoden als String extrahiert wird. Dieser String wird dem AMS übergeben. Die typischen Zugriffs- und Manipulati-

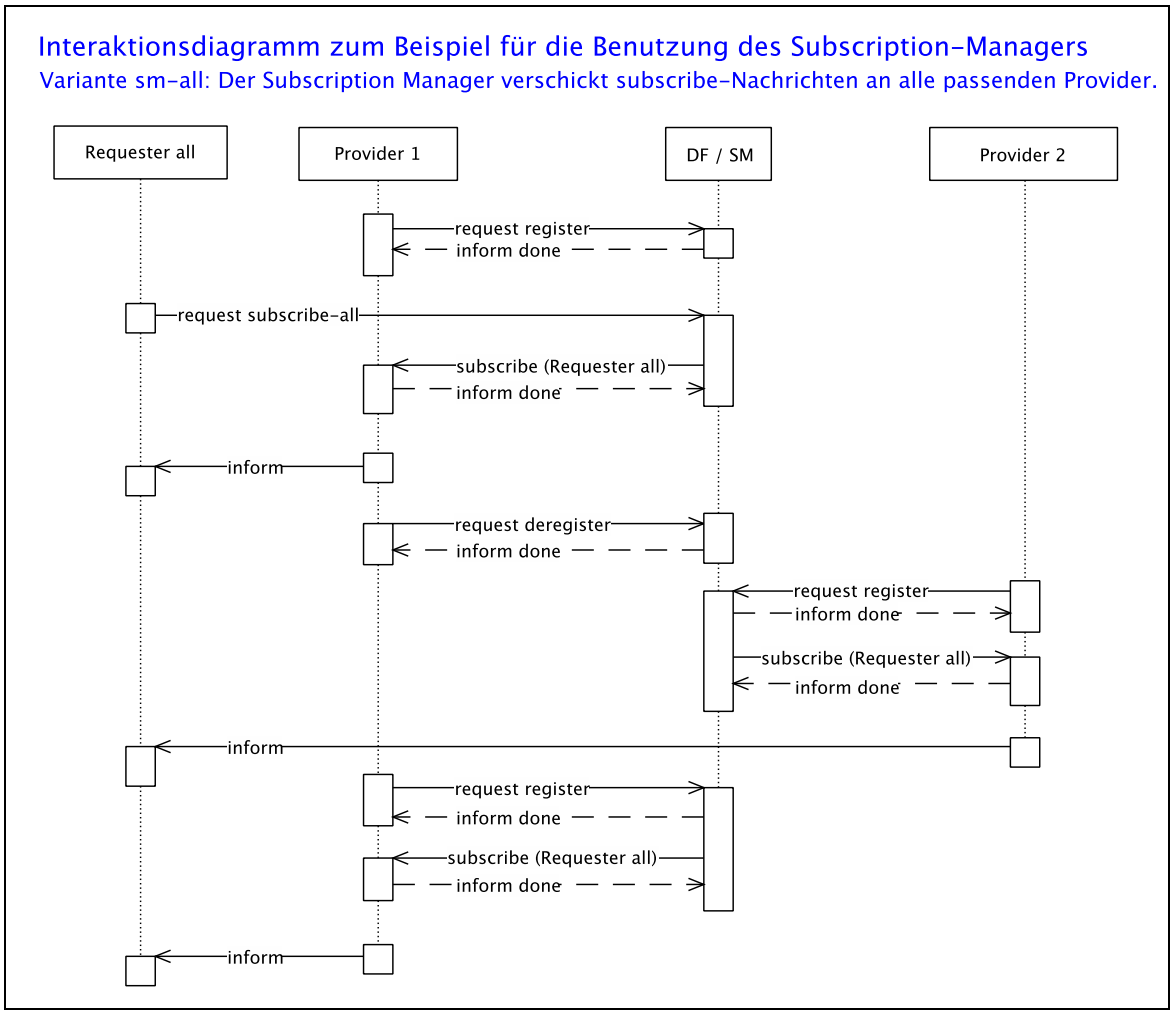


Abbildung 5.7: Der Subscription-Manager kann für einen Abonnenten automatisch alle passenden dauerhaften Informationsdienste abonnieren.

onsmöglichkeiten, die über die spätere Schnittstelle der Wissensbasis bereitgestellt werden, stehen dem Erzeuger nicht zur Verfügung. Um nun zu vermeiden, dass über „blinde“ Stringmanipulation das Wissen angepasst wird, wird hier ein anderer Weg eingeschlagen: Der erzeugte Agent hat in seiner Wissensbasis mindestens einen Eintrag für ein proaktives Protokoll, das GeneralAgentSetup, und einen Eintrag für ein reaktives Protokoll. Dieses reaktive Protokoll reagiert auf eine inform-Nachricht mit einem bestimmten Prädikat. Das Protokoll ändert dann als erstes die Protkolleinträge der Wissensbasis, so dass der reaktive Protkolleintrag aus der Wissensbasis entfernt wird. Im Prädikat im Nachrichtenkörper sind dann alle notwendigen Informationen für den neuen Agenten in Bezug auf seinen Erzeuger vorhanden und der Agent nimmt die notwendigen Eintragungen in seine Wissensbasis selbst vor.

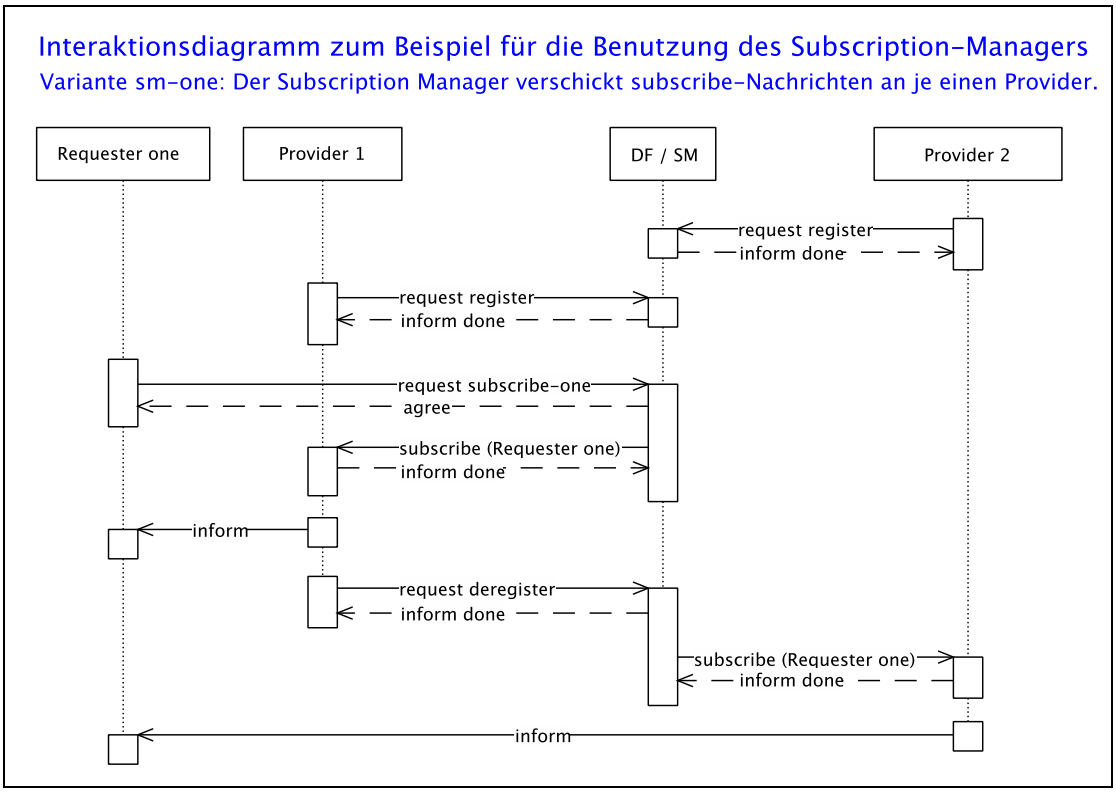


Abbildung 5.8: Der Subscription-Manager kann für einen Abonnenten sicherstellen, dass immer genau einer der verfügbaren passenden Informationsdienste abonniert ist.

5.3 Ein agentenbasiertes Workflowmanagementsystem

Die Anforderungen, prinzipielle Nutzungsszenarien und Komponenten sind nach den Ausführungen der *WfMC* gegeben und für die vorliegende Arbeit in Abschnitt 5.1.3 festgelegt. Das Ergebnis des Entwurfsprozesses enthält in diesem Falle also eine Auswahl der Nutzungsszenarien, die von der *WfMC* spezifiziert wurden. Die Umsetzung mit Paose erfordert weiter den Entwurf von Rollen, Interaktionen und einer Ontologie.

Drei verschiedene Typen von **Rollen** werden für das Workflowmanagementsystem (WFMS) definiert: (1) Agentenrollen wie z.B. WFMS, Benutzer, Workflowengine (WFE), Workflow-Enactment Service (WFES), Zugangsverwaltung und Datenbank-Agenten für Tasks und Workflowdefinitionen; (2) Nutzerrollen verfeinern die Rolle „Benutzer“, z.B. Administrator oder Task-Bearbeiter; und (3) anwendungsspezifische Rollen verfeinern die Rolle „Task-Bearbeiter“.

Der WFMS-Agent ist dabei für die funktionalen Agenten WFE, WFES usw. konzeptionell wie eine Plattform zu verstehen. Die einzelnen Agentenrollen für den Prototypen werden im Abschnitt 5.3.3 vorgestellt.

Der Entwurfprozess der **Interaktionen** führte zu (1) Interaktionen zur Verwaltung (Workflowinstanziierung, Datenbanken, Sitzungsverwaltung und Abonnements

von jeweils aktuellen Informationen zu verfügbaren Workitems und Activities), (2) Interaktionen zur Handhabung der Workitems und Activities und (3) interne Interaktionen, an denen der Benutzer nicht beteiligt ist. Die Interaktionen des Prototypen werden im Abschnitt 5.3.4 vorgestellt.

Den Überblick über alle Rollen und Interaktionen bietet das Use-Case-Diagramm in Abschnitt 5.3.1.

Die **Ontologie** enthält grundlegende Konzepte wie „Workitem“ und Prädikate für Aussagen wie „currentWorkitemsOf“. Aktionen wie „instantiateWorkflow“ sind durch spezielle Konzepte in der Ontologie modelliert (*agent-action*s). Insgesamt enthält die Ontologie acht Prädikate sowie je 24 Konzepte und Agent-Action-Konzepte.

Die resultierende Anwendungsstruktur umfasst neben den identifizierten Agentenrollen noch weitere Rollen, die als mehrfach genutzte Schnittstellen mehreren konkreten Agenten dienen: z.B. müssen sowohl der Workitem Dispatcher als auch der WFES eine Authentifizierung durchführen und benutzen dazu dasselbe Protokoll – also ist Authentication Needer eine zusätzlich eingeführte Agentenrolle. Die Abhängigkeiten der Agentenrollen sind in einem Multi-Agenten-Diagramm in Abschnitt 5.3.2 dargestellt.

5.3.1 Grobentwurf

Abbildung 5.9 zeigt das Use-Case-Diagramm zum agentenorientierten Workflowmanagementsystem. Wie im Abschnitt 3.4.3 erläutert, stellen die Männchen-Figuren keine menschlichen Nutzer des System dar, sondern die beteiligten Agenten (-Rollen). Die leeren Anwendungsfälle wurden zunächst nicht implementiert. Die umgesetzten Rollen und Interaktionen werden weiter unten in den Abschnitten 5.3.3 und 5.3.4 ausgeführt.

Die Laufzeitumgebung Renew ist vollständig Plugin-basiert. Das bedeutet, dass der Prototyp ebenfalls als Plugin umgesetzt wird. Die Abhängigkeiten zu anderen Plugins sind in Abbildung 5.10 dargestellt.

Die Laufzeitumgebung des vorgeschlagenen Systems wird durch das Simulator-Plugin bereitgestellt. Das Workflow-Agenten-Plugin, das hier entworfen wird, baut auf dem Capa-Plugin und auf dem Workflow-Plugin auf. Die direkte Abhängigkeit zum Simulator von Renew resultiert aus der Bereitstellung der Tasktransition mitsamt dem Mechanismus zum Beobachten der Aktivierung von Tasktransitionen.

Das Renew-Pluginsystem ermöglicht die Definition von *bedingten Abhängigkeiten*, also Plugins, die genutzt werden können, obwohl das Grundsystem auch ohne sie läuft. Dieses Prinzip wird für die GUI-Anbindung benutzt.

5.3.2 Anwendungsstruktur

Das Multi-Agenten-Diagramm muss als eine Zusammenfassung des gesamten Projekt-Ergebnisses betrachtet werden, weil darin alle zentralen Bestandteile des Programms eindeutig und bindend benannt sind: Rollen und Super-Rollen der Agenten, zu startende Protokollnetze und sämtliche Nachrichtentypen. Die Protokollnetze und Nachrichten sind hier allerdings nur benannt, was jedoch als Referenz auf die Implementierung der Interaktionen und der Ontologie gelten soll. Die grobe Struktur des Diagramms

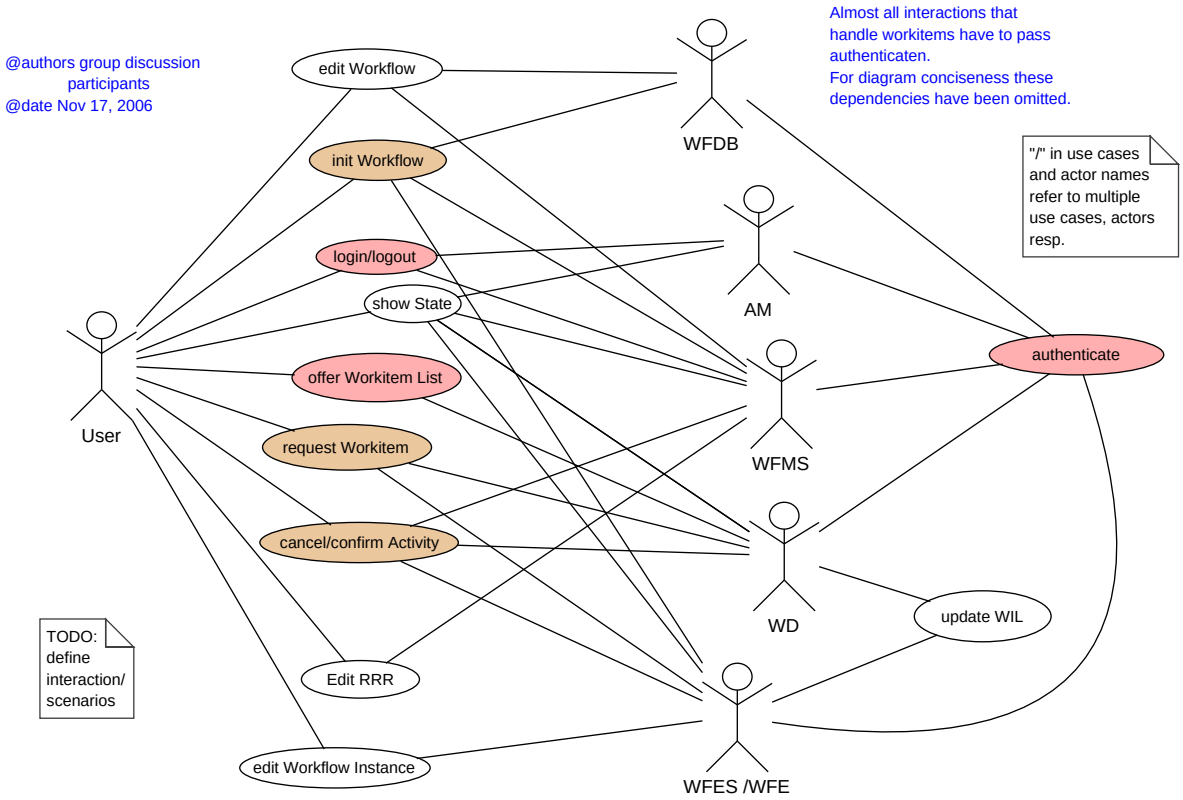


Abbildung 5.9: Grobentwurf: Use-Case-Diagramm für das WFMS

wird bereits am Projektbeginn erstellt, die Einzelheiten über empfangene Nachrichten, zu startende Protokolle und Dienste wurden dem Diagramm im Verlauf des Projektes auf Antrag der einzelnen Gruppen hinzugefügt. Das Diagramm muss als Teil des Codes betrachtet werden, weil mit den Wissensbasen der beteiligten Agenten die Knotenpunkte der Anwendung generiert werden. Wie jeder Code ist das Diagramm stets aktuell. Es ist nicht das Ziel dieses Abschnitts, jeden Eintrag im Diagramm zu verdeutlichen, das würde den Lesefluss erheblich behindern. Abbildung 5.11 zeigt die Diagrammstruktur mit eingeklappten Knoten (das „+“ an Elementen zeigt an, dass die Details ausgeblendet sind). Zur besseren Übersicht wurden in der Mitte des Bildes die Vererbungspfeile übereinandergelegt, so dass ein Pfeilkopf für alle einmündenden Kanten die Vererbung anzeigt. Die Funktionen der dargestellten Rollen, Interaktionsbeziehungen und die Ontologie werden in den nächsten drei Abschnitten vorgestellt.

In jedem Interface sind die zugehörigen `request`-Nachrichten unter Verwendung der Ontologieobjekte angegeben. Eine `implements`-Beziehung (gestrichelte Linie mit geschlossener Pfeilspitze) bedeutet in der Regel, dass die Agentenrolle eine entsprechende Nachricht empfangen und verarbeiten kann, eine `requires`-Beziehung (gestrichtelte Linie mit offener Pfeilspitze) bedeutet, dass der Agent, der die Rolle innehat, eine entsprechende Nachricht versenden kann.

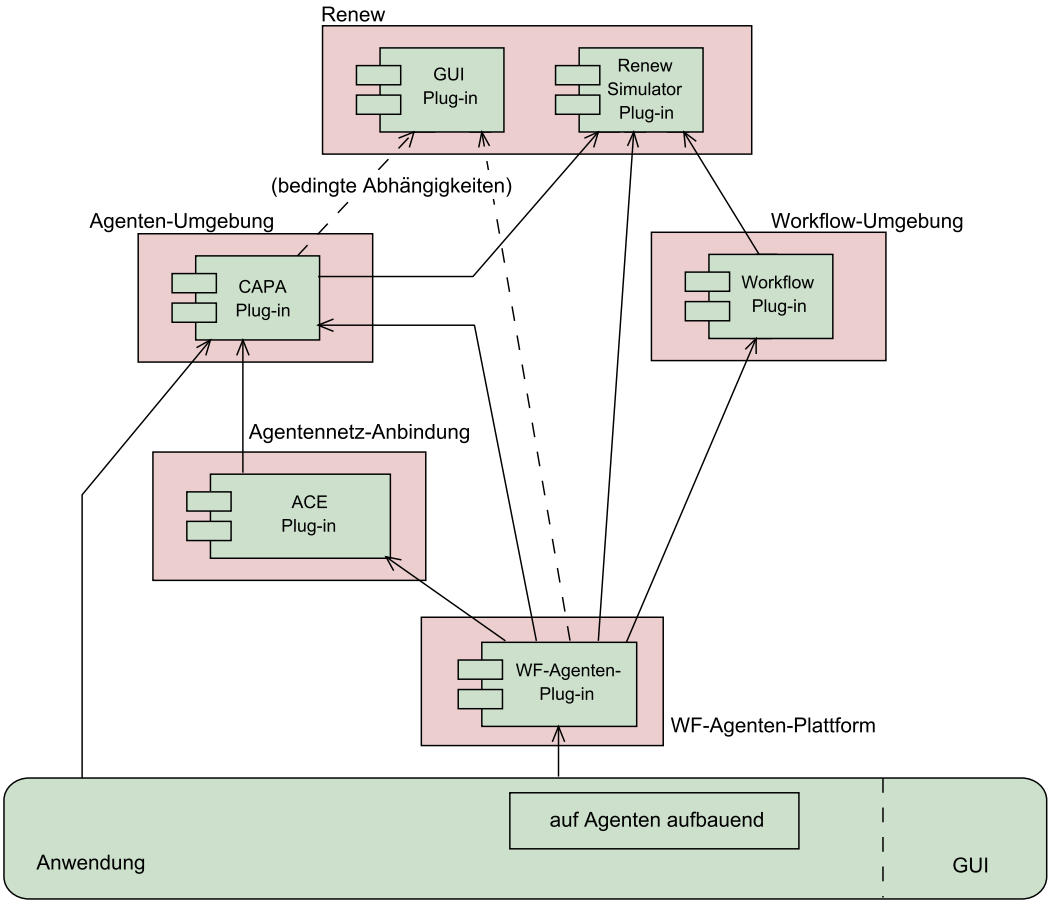


Abbildung 5.10: Abhängigkeiten der Plugins

5.3.3 Rollen

Die hier definierten Agententypen stellen Teile der WFMS-Plattform dar und implementieren deren Dienste (vergleiche Abbildung 5.11 auf der nächsten Seite mit folgendem Text). Die zentralen Funktionen der Workflow-Engine werden in einem eigenen Abschnitt detaillierter dargestellt.

Überblick über alle Rollen

(1) Rollen im WFMS:

- WFMS-Agent (Workflowmanagementsystem)
- WD-Agent (Workitem-Dispatcher)
- WFES-Agent (Workflow-Enactment-Service)
- WFE-Agent (Workflow-Engine)
- AccountManager
- RRR-Manager (Rules-Roles-Rights)
- TaskDefDB (Task-Definition-DB)
- WFDefDB (Workflow-Definition-DB)

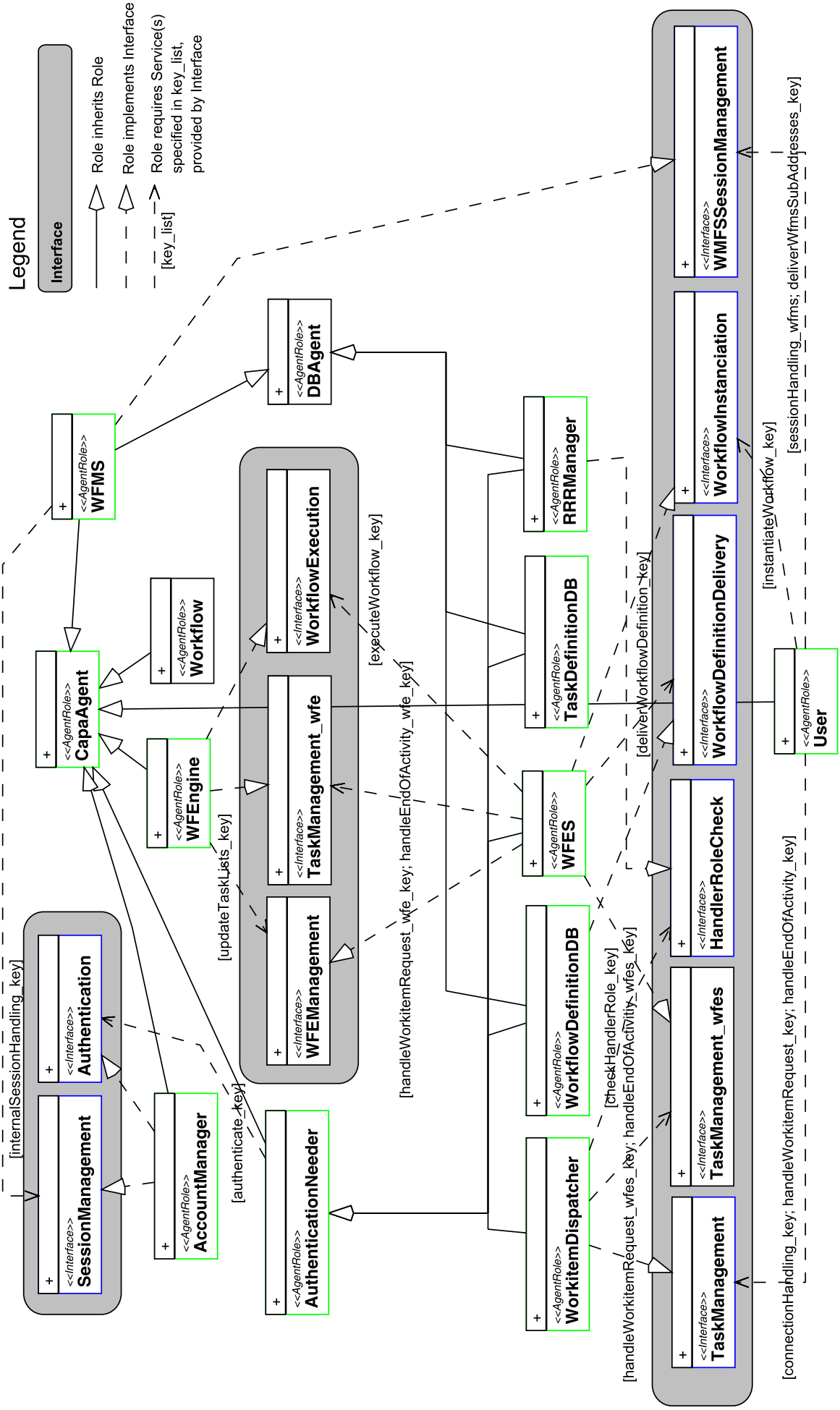


Abbildung 5.11: Anwendungsstruktur

(2) Rollen, die dynamisch instanziiert werden:
- User (nicht vom WFMS instanziiert)
- Workflow (instantiated Workflow)

(3) Allgemeine Rollen, die von mehreren Sub-Rollen geerbt werden (können):
- CAPA-Agent (alle Rollen erben von CAPA-Agent)
- Authentication-Needer
- DB-Agent (Basis-Datenbank-Agent)

WFMS-Agent Der *Workflow-Management*-Agent kapselt das WFMS auf einem Rechner. Er bietet nach außen die Funktionalität eines lokalen WFMS an. Dieser Agent funktioniert als Plattform für alle anderen Agenten des Workflow-Agenten-Plugins, indem er sie erzeugt. Für User-Agenten bietet er Login und Logout an (WFMSSessionHandling) und gibt nach erfolreichem Login die Adressen der Subagenten an den User-Agent (deliverWFMSSubroleAddresses). Für das Einloggen nutzt er einen internen Dienst (SessionManagement). Der WFMS-Agent erbt außerdem von der Super-Rolle DB-Agent, um auf diese Weise alle Datenbank-Anfragen für nicht-eingeloggte Benutzer zur Verfügung stellen.

WD-Agent Der *Workitem-Dispatcher*-Agent stellt die Verbindung zwischen Benutzern und der Menge der Workitems und Activities her, unter Kenntnis der Benutzer, Gruppen, Rollen und Rechte für das WFMS, indem er von der Super-Rolle Authentication-Needer erbt. Er implementiert das Interface TaskManagement, d.h. er stellt einerseits die aktuellen Tasks je nach Rolle für den Benutzer dar, nachdem der Benutzer diesen Dienst abonniert hat und andererseits leitet er Anfragen wie z.B. request Workitem von autorisierten Benutzern an den WFES-Agenten weiter (vgl. auch Abbildung 5.12).

WFES-Agent Der *Workflow Enactment Service*-Agent stellt die Verbindung zwischen den Workflow-Engines und der Menge der Workitems und Activities her. Alle verfügbaren Workitems und Activities werden dem WD-Agenten mitgeteilt (Interface WFEManagement). Vom WD-Agenten kommende Anfragen werden an den passenden WFE-Agenten weitergeleitet (Interface TaskManagement_wfes). Der WFES-Agent nimmt außerdem die User-Anfrage zum Instanziieren eines Workflows entgegen (Interface WorkflowInstantiation).

WFE-Agent Eine *Workflow Engine* führt einen gegebenen Workflow oder ein Workflow-Fragment aus. Im Prototyp ist zunächst nur der einfache confirmedTask implementiert: Sobald die Task-Transition aktiviert ist, wird ein Workitem erzeugt und zur Verfügung gestellt; sobald ein autorisierter Benutzer dieses anfordert, wird eine Activity erzeugt. Diese kann vom Benutzer abgeschlossen oder unterbrochen werden. Der WFE-Agent kommuniziert nur mit dem WFES (später auch mit dem Workflow-Agent). Er implementiert die Interfaces TaskManagement_wfe und WorkflowExecution. Die Umsetzung dieses zentralen Mechanismus wird am Ende dieser Liste auf Seite 168 gesondert aufgegriffen.

Capa-Agent Ein Agent, der von dieser Super-Rolle erbt, meldet sich beim lokalen AMS und DF an und sucht beim lokalen DF entsprechend seiner Konfiguration

in der Wissensbasis nach Diensten, die er benötigt, bevor er mit weiteren Aktionen beginnt. Sein vollständiger `agent-identifier` wird in der Wissensbasis abgelegt. Alle Agenten des WFMS erben von dieser Super-Rolle.

Authentication-Needer Alle Agenten, die Anfragen von Benutzern entgegennehmen, erben von dieser Super-Rolle. Zusammen mit der Super-Rolle CAPA-Agent bewirkt dieses, dass beim Start eines Agenten die vorhandenen Anbieter des Interfaces Authentication in die Wissensbasis eingetragen werden.

AccountManager Hier ist das Interface Authentication implementiert. Der Account-Manger realisiert das Login von Benutzern.

DB-Agent Ein Agent, der von dieser Super-Rolle erbt, empfängt und verarbeitet Nachrichten zum Hinzufügen, Ändern, Löschen und Abfragen von Datenbank-Elementen. Im Multi-Agenten-Diagramm (Abbildung 5.11) gibt es kein Interface für diesen Dienst, weil im Diagramm nur die Dienste explizit aufgeführt werden, die für mindestens eine Rolle zwingend notwendig sind. Die Datenbank-Manipulation und -Anfragen gehen von der Rolle User aus, sind aber für dessen Funktionieren nicht notwendig, solange wie im Prototyp bereits (sinnvolle) Einträge in den Datenbanken vorliegen. Für die anfragende Seite gibt es jeweils Dienste der spezialisierten DB-Agenten, so wird z.B. das Interface WorkflowInstantiation vom WFDefDB-Agenten implementiert.

RRRManager Mit dem *RulesRolesRights-Manager*-Agenten können Rollen, Rechte und Benutzer verwaltet werden. Er implementiert das Interface HandlerRoleCheck, welches vom WD-Agent benutzt wird um zu bestimmen, welche Workitems für einen bestimmten Benutzer zugänglich sind.

TaskDefDB Ein *Task-Definition-DB*-Agent erbt von der Super-Rolle DB-Agent und verwaltet Task-Definitionen.

WFDefDB Der *Workflow-Definition-DB*-Agent verwaltet die dem WFMS bekannten Workflow-Definitionen. Neue Defintionen werden von autorisierten Benutzern hinzugefügt und bekannte Workflows können von autorisierten Benutzern instanziiert werden.

User Mit der Agentenrolle *User* wird die Schnittstelle für einen Benutzer spezifiziert, in diesem Sinne gehört diese Rolle und ihre Implementierung ebenfalls zum WFMS, wird jedoch keinesfalls vom WFMS-Agenten gestartet.

Workflow Die Rolle des Workflow-Agenten wird erst im Abschnitt 5.4 vorgestellt.

Es sind nun alle Rollen kurz vorgestellt worden. An zwei Stellen lohnt sich eine eingehendere Betrachtung: erstens der Weg von einer aktivierten Task-Transition zu einer Nachricht im Format FIPA-ACL und zweitens das Zusammenspiel der zentralen Agentenrollen im WFMS.

Im Detail: Von der Task-Transition zur ACL-Nachricht

Der Workflownetz-Simulator, der im Abschnitt 2.3.4 erläutert wurde, stellt die nötigen Komponenten zur Verfügung, um von einer aktivierten Task-Transition zu weiteren spezifizierten Aktionen zu gelangen: die Task-Transition hat einen inneren Aufbau, so dass Aktionen ausgeführt werden können, die allein auf der Aktivierung einer Tasktransition beruhen. Dieses Prinzip der verfeinerten Transition wird für das WFMS beibehalten. Die nächste Komponente ist der Listener-Mechanismus, mit dem festgelegt wird, was genau geschieht, wenn eine Task-Transition aktiviert ist. Das Workflow-Plugin von Renew stellt ein Interface bereit, mit dem sämtliche Workflownetze einer laufenden Simulation beobachtet werden können. Das hat den Vorteil, dass ein Workflow-Netz an irgendeiner Stelle in der Simulation instanziiert werden muss, um an der Implementierung des Listeners die Workitems zu bearbeiten und sie an Benutzer zu verteilen. Die Benutzer melden sich mit einem Client-Programm an, woran Rollen und Berechtigungen geknüpft sind und sehen dann nur die für sie relevanten Informationen.

Für das agentenbasierte WFMS wird zunächst die Nutzung dieses Interfaces geändert: Statt der Clients aus der Implementierung von Thomas Jacob, die von jedem Benutzer selbst gestartet wurden, wird nun ein Proxy-Netz eingesetzt, welches das Interface für Clients implementiert. Für jeden Benutzer, der über für ihn zutreffende Workitems informiert werden möchte, wird in einer Decision Component (DC) ein solches Proxy-Netz instanziiert und mit username und password initialisiert. Die Implementierung eines Interfaces mit einem Referenznetz sieht allgemein so aus, dass das Netz für die Methoden des Interfaces Transitionen ohne Vorbedingungen mit Uplinks bereitstellt, deren Signatur den Methoden des Interfaces entspricht, in diesem Falle :notifyAgendaChange(event). Im Proxy wird nach dem Schalten der Transition zwischen den Objekten des Workflow-Plugins und den Ontologie-Objekten des WFMS konvertiert und die verfügbaren Workitems werden über Uplinks zur Verfügung gestellt. In der DC wird nun mittels ständig aktivierter Downlinks „gelauscht“. Mittels der Wissensbasis wird nun die Verbindung zwischen dem WFMS-Usernamen und einem Agent-Identifier hergestellt, an den die Informationen gesendet werden. Dann werden die verfügbaren Workitems in ein Prädikat der Ontologie verpackt und diese Information wird dem passenden Protokollnetz übergeben, das die entsprechende ACL-Nachricht verschickt.

Im umgekehrten Fall geht eine Anfrage vom Benutzer aus, der z.B. ein Workitem annehmen möchte. Ein reaktiv gestartetes Protokollnetz im WFMS extrahiert das Workitem aus der Nachricht und gibt die Information an die DC weiter. Entsprechend der Rückmeldung aus der DC wird dann eine Erfolgs- oder Fehlernachricht zurückgeschickt. In der DC wird die Information über den passenden Kanal an das passende Proxy-Netz weitergeleitet. Der Informationsfluss ist in diesem Fall bidirektional: Das Workitem bindet in der DC, ein Erfolgsindikator wird im Proxy gebunden und der username wird von beiden Seiten gebunden, um die richtige Zuordnung zu gewährleisten. Das Proxy-Netz ruft an der entsprechenden Transition eine Methode auf, die das Workitem beim Workflow-Simulator anfordert und einen Boolean als Erfolgsindikator zurückgibt.

Soweit die Implementierung im ersten Projektdurchlauf (als Teil des Workitem Dispatcher (WD)-Agenten). Diese Wiederverwendung der bestehenden Workflow-Engine in Renew mittels Kapselung wurde gewählt, um zunächst die Interaktionen und Aufgabenverteilung im agentenorientierten WFMS zu entwickeln.

Mit dieser Methode ist es allerdings nach wie vor möglich, den Client aus dem Workflow-Plugin zu starten und so an vorhandene Workitems zu gelangen, ohne sich an die WFMS-Agenten zu wenden. Um zu verhindern, dass die Workitems auch außerhalb des agentenorientierten WFMS zugreifbar sind, wird ein anderes Listener-Interface benötigt. Dieser Listener meldet sich direkt bei einer Instanz eines Workflow-Netzes an. Um sich anzumelden, muss eine Referenz auf die Instanz vorliegen, und dies ist nur bei dem WF-Engine-Agenten des WFMS der Fall. Dieser hat also in seiner Wissensbasis alle verfügbaren Workitems vorliegen und steuert über das Interface TaskOccurrenceListener über TaskOccurrence-Objekte direkt das Schalten der aktivierten Task-Transitionen. Die Workitems werden zunächst an den WD-Agent geschickt, der sie dann dem richtigen Benutzer zusendet.

Zusammenspiel von Workflow-Enactment-Service und Workflow-Engine

Der Informationsfluss im laufenden Workflowmanagementsystem geht einerseits von den instanziierten Workflows aus, welche über aktivierte Tasktransitionen bewirken, dass Benutzer informiert werden. Andererseits geht ein Informationsfluss von den Benutzern aus, die Workitems annehmen und Activities fertigstellen oder abbrechen und damit letztendlich Schaltungen in Workflowinstanzen auslösen. An mehreren Stellen im Informationsfluss muss die Menge der Benutzer der Menge der Aktivierungen im Workflownetz und umgekehrt zugeordnet werden, wobei beide Mengen dynamisch sind. Dabei ist ein Agent für die Kenntnis der Benutzer und ein Agent für die Kenntnis der Aktivierungen im Workflownetz zuständig.

Abbildung 5.12 zeigt das Zusammenspiel der Komponenten für den aufwärtsgerichteten Informationsfluss.

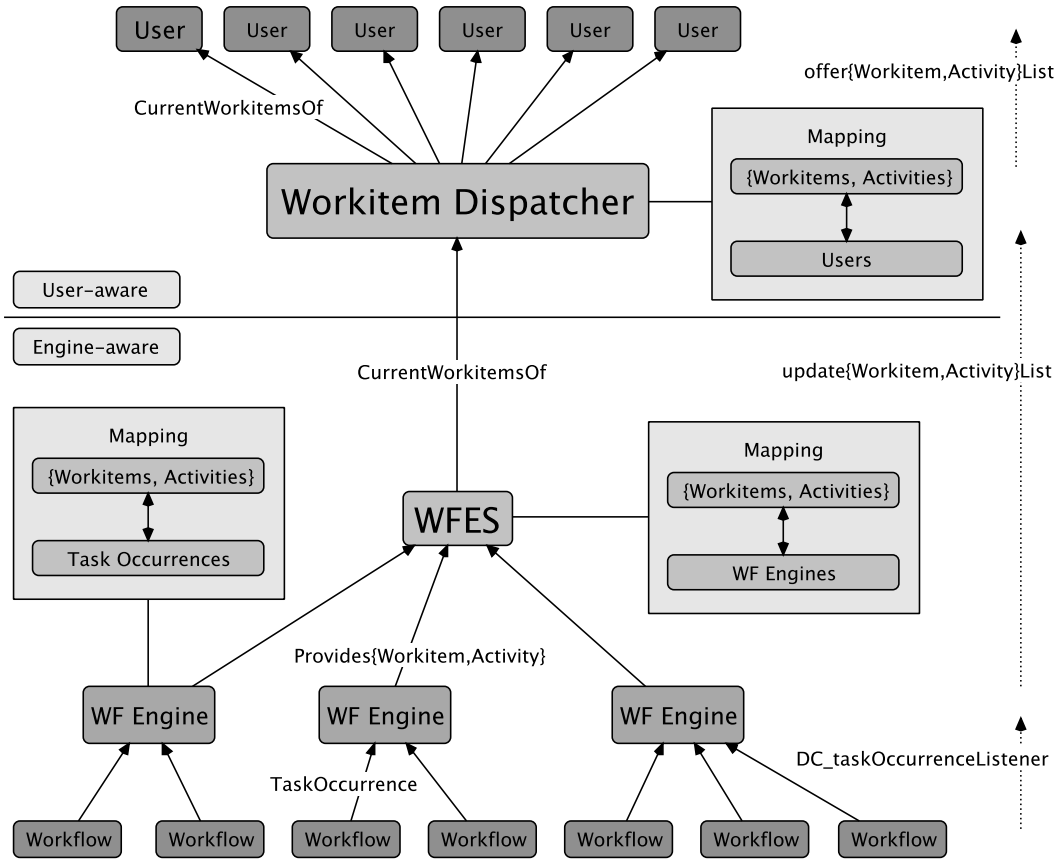


Abbildung 5.12: Zuständigkeiten von WFES und WFE. Nach: AOSE-Projekt 2006/2007, Gruppe WFES (Mathias Baggendorf und Kai Jander)

Hier schließt nun der dritte von fünf Teilen zur Umsetzung des agentenbasierten Workflowmanagementsystems nach dem Paose-Ansatz: Grobentwurf, Anwendungsstruktur, Rollen, Verhalten und Ontologie. Im nächsten Abschnitt geht es um das Verhalten, d.h. die Interaktionen der Agenten(-rollen) werden beschrieben.

5.3.4 Verhalten

Für die Implementierung des Verhaltens wird jeder Use-Case aus Abschnitt 5.3.1 über ein Agenten-Interaktions-Protokolldiagramm (AIP) ausformuliert. Allerdings geschieht dies nicht automatisch, so dass das Diagramm am Ende nicht aktuell gehalten ist. Die folgende Liste beschreibt alle implementierten Interaktionen bewusst kurz und informell, da die Ergebnisse des AOSE-Projektes sehr umfangreich sind und nicht von mir selbst implementiert wurden.

Überblick über alle Interaktionen

(1) Interaktionen zur Verwaltung:
- Login, Logout
- Connect, Disconnect
- Get WFMS Subrole Addresses
- Instantiate Workflow, Workflow End Reached
- DB-Edit (User and Workflow Data)

(2) Interaktionen für die Handhabung von Work-Items und Activities:
- Offer Workitems bzw. Activities
- Accept Workitem
- Cancel Activity, Confirm Activity

(3) Interaktionen, an denen der Benutzer nicht beteiligt ist:
- Authenticate, Get Satisfied Rules und Query
- Update Activity List und update Workitem List
- Setup Agent

Es folgt eine Beschreibung für jede Interaktion:

login Der Benutzer schickt eine Login-Nachricht an den WFMS-Agenten, welcher sie dem Account-Manager-Agenten weiterleitet. Dieser prüft mit Hilfe der internen query-Interaktion die angegebenen Credentials und führt dann das Login aus.

logout Der Benutzer schickt eine Logout-Nachricht an den WFMS-Agenten, welcher sie dem Account-Manager-Agenten weiterleitet. Dieser prüft mit Hilfe der internen query-Interaktion die angegebenen Credentials. (In einer erweiterten Variante fragt er dann den Workitem-Dispatcher-Agenten nach offenen Activities. Falls da welche sind, fragt er beim Benutzer, ob diese abgebrochen werden können und bricht dann die noch laufenden Activities ab.) Dann wird das Logout ausgeführt.

getWFMSSubroleAddress Der Benutzer kann nach dem Login die Adressen der WFMS-Subagenten für direkte Interaktionen beim WFMS-Agenten erfragen. Nach einer Authentifizierung werden sie zugeschickt.

connectToDispatcher Ein Benutzer kann mit dieser Interaktion einen Dienst abonnieren, um regelmäßig über die ihm verfügbaren Workitems und seine laufenden Activities informiert zu werden. Dazu schickt er dem WD-Agenten eine Nachricht, dieser prüft die Credentials mit Hilfe der internen authenticate-Interaktion

und sendet eine Agree-Nachricht. Dann fügt der WD-Agent den Benutzer in eine interne Liste der zu benachrichtigenden Benutzer ein. Danach fragt er den RRR-Manager nach den Regeln für den Benutzer, sucht die passenden Workitems aus seiner Wissensbasis und sendet diese Liste dem Benutzer.

disconnectFromDispatcher Ein Benutzer schickt eine Nachricht an den WD-Agenten. Wenn die Credentials des Benutzers gültig sind und er bereits beim WD-Agenten angemeldet ist, wird er aus der Liste der zu benachrichtigenden Benutzer entfernt.

instantiateWorkflow Der WFES prüft die Credentials und fragt dann den Wf-DB-Agenten nach der Workflowdefinition. Dann entscheidet er, welcher WFE-Agent den Workflow ausführen soll und beauftragt diesen. Der WFE-Agent instanziiert das Workflow-Netz, instanziiert eine Entscheidungskomponente, die auf aktivierte Tasktransitionen lauscht und schaltet den :startwf()-Kanal. Der Benutzer wird über die erfolgreiche Aktion mit einer Workflow-ID informiert.

workflowEndReached Wenn eine Workflow-Instanz den Kanal :stopwf() aktiviert, stößt der WFE-Agent diese Interaktion an, indem er eine Nachricht an den WFES-Agenten schickt. Dieser holt aus seiner Wissensbasis den Initiator und schickt ihm eine Nachricht über den beendeten Workflow.

update / offerWorkitemList Wenn im WFE-Agent die Entscheidungskomponente, die auf aktivierte Tasktransitionen lauscht, eine Änderung meldet, wird zunächst die interne Interaktion updateWorkitemList gestartet: Der WFE-Agent schickt eine Nachricht mit aktuellen Informationen an den WFES-Agenten. Die Interaktion offerWorkitemList wird dann reaktiv im WD-Agenten gestartet: Der Benutzer erhält eine Nachricht mit einer aktuellen und vollständigen Liste der für ihn verfügbaren Workitems.

Diese Zweiteilung der Interaktion ist ein Relikt aus dem ersten Projektdurchlauf, wo der WD-Agent einen eigenen Listener in Form des Proxy-Netzes betrieben hat (siehe Abschnitt 5.3.3). Dies gilt auch für die nächste Interaktion.

update / offerActivityList Auch diese Interaktion wird wie die verwandte Interaktion offerWorkitemList letztlich von der Workflow-Netzinstanz angestoßen. Aus der internen Interaktion updateActivityList erhält der WD-Agent die Liste und schickt dann dem Benutzer eine Nachricht mit der aktuellen Liste von Activities.

requestWorkitem Diese Interaktion wird wieder vom Benutzer angestoßen: er wählt aus den ihm angebotenen Workitems eines aus und schickt eine Nachricht an den WD-Agenten, dieser prüft die Credentials und leitet den Request an den WFES-Agenten weiter, welcher den zugehörigen WFE-Agenten sucht und wiederum die Nachricht weiterleitet. Der WFE-Agent schaltet die entsprechende Task-Transition und sendet eine Antwortnachricht, welche über WFES und WD-Agent an den Benutzer geleitet wird.

confirmActivity Für eine fertig bearbeitete Activity schickt der Benutzer eine Confirm-Nachricht an den WD-Agenten, dieser prüft die Credentials und leitet die Anfrage an den WFES-Agenten weiter, von dort wird sie an den WFE-Agenten

geleitet. Dieser schaltet die entsprechende Task-Transition und sendet eine Antwortnachricht, die über WFES und WD-Agent an den Benutzer geleitet wird.

cancelActivity Für eine Activity, die er nicht weiter bearbeiten möchte (oder kann), schickt der Benutzer eine Cancel-Nachricht an den WD-Agenten, dieser prüft die Credentials und leitet die Anfrage an den WFES-Agenten weiter, von dort wird sie an den WFE-Agenten geleitet. Dieser schaltet die entsprechende Task-Transition und sendet eine Antwortnachricht, die über WFES und WD-Agent an den Benutzer geleitet wird.

dbEdit User, Rechte, Workflows und Tasks werden von Datenbank-Agenten verwaltet. Diese Interaktion fasst den immer gleichen Ablauf für den schreibenden Zugriff auf eine solche Datenbank zusammen: Berechtigung prüfen (Subinteraktion: authenticate), die Änderungsanfrage an den passenden DB-Agenten weiterleiten und schließlich die Rückmeldung an den Benutzer senden.

authenticate Immer, wenn die Credentials eines Benutzers für eine Aktion geprüft werden, schickt der AuthenticationNeeder eine Nachricht an den Accountmanager. Der prüft, ob usename und password gültig sind, ob der Benutzer eingeloggt ist und ob er für die Aktion die passende Rolle innehat. Dann schickt er eine Antwortnachricht.

getSatisfiedRules Wenn der WD-Agent aus den verfügbaren Workitems diejenigen auswählt, die für einen Benutzer zutreffen, fragt er beim RRR-Manager nach den für diesen Benutzer gültigen Regeln. Die Regeln aus der Antwortnachricht vergleicht er dann mit den Regeln, die für einen Task definiert sind und entscheidet daraufhin, ob ein Workitem ausgeteilt werden kann.

query Eine Interaktion, um beim Accountmanager die Gültigkeit von username und password nachzufragen. Es handelt sich also um eine geschmälerte Variante der authenticate-Interaktion.

setupAgent Diese Interaktion startet der WFMS-Agent. Er liest aus seiner Wissensbasis die Pfade zu den initialen Wissensbasen der WFMS-Subagenten und sendet Start-Aufforderungen an den lokalen AMS (Agent Management System). Für jeden Dienst des WFMS legt er einen Wissensbasis-Eintrag an mit dem entsprechenden AgentIdentifier.

5.3.5 Ontologie

Wie im Abschnitt 3.4.3 beschrieben, hat die Ontologie drei Kategorien: Dinge (Oberklasse „Thing“), AgentActions und Prädikate. Bis auf die mit „z.B.“ und „u.s.w.“ gekennzeichneten Stellen sind alle Ontologieklassen aufgeführt.

Dinge

Zunächst sind in der Ontologie die „Dinge“, mit denen in einem WFMS hantiert wird, als Konzepte eingetragen: Task, Workitem und Activity; User, Executor, Credentials sowie Role und Rule; schließlich verschiedene Aspekte von Workflows: Workflow-

Description (enthält einen Workflow-Namen), WorkflowDefinition (mit ausführbarer Definition), WorkflowInstance und WorkflowState.

Tasks werden mit einer TaskDefinition beschrieben, welche spezifiziert, um welche Art von Task es sich handelt. Es stehen zur Verfügung: ManualTask (d.h. nachdem die Arbeit fertiggestellt ist, meldet der Benutzer das und es gibt keine weitergehende automatische Unterstützung für diesen Task) und WorkflowTask (ein Subworkflow wird gestartet). Zur Bearbeitung von Tasks zur Datenerhebung mit Formularen gibt es außerdem EditFormTask und FillFormTask samt zugehörigen Konzepten für Form-Elemente wie z.B. CheckBoxElement.

Zum Formulieren von Rechten gibt es Unterkonzepte von Role: ApplicationRole und UserRole, letztere untergliedert sich in AdministratorRole, ExecutorRole und InitiatorRole.

Agent Actions

Die große Gruppe der AgentActions definiert für jede Request-Nachricht ein Ontologie-Konzept, welches das Erstellen, Auswerten und Beantworten einer solchen Nachricht erleichtert. Es gibt es zwei Untergruppen: WFMS-interne Interaktionen und die Unterkonzepte von WorkflowAction, worin alle Requests des Benutzers zusammengefasst sind. Das sind im Einzelnen: Login und Logout, Connect und Disconnect sowie GetAllCurrentItems (falls der Benutzer aktuelle Informationen über seine Workitems und Activities nicht mittels Connect abonnieren möchte, kann er sich hiermit gezielt informieren). Zur Handhabung der Workitems und Activities gibt es AssignWorkitem, SetActivityDone und SetActivityCanceled. Weiter gibt es zum Starten von Workflows InstantiateWorkflow und, um nach dem Login direkt mit den WFMS-Subagenten kommunizieren zu können, SearchWorkflowService.

Schließlich gibt es für den Benutzer eine Reihe von AgentActions, die als DatabaseAction zusammengefasst werden: CreateDbEntry, DeleteDbEntry u.s.w.

Die AgentActions für WFMS-interne Vorgänge sind z.T. intern verwendete Formen für Benutzer-Aktionen: InternalLogin, InternalLogout und InternalInstantiateWorkflow. Die weiteren internen Aktionen finden bei der Instanziierung von Workflows Verwendung: GetWorkflowDefinition, ChooseWorkflowEngine und EnactWorkflow. Bei der Zuordnung von Tasks zu Benutzern wird ComputeSatisfiedRules verwendet.

Prädikate

Die Prädikate aus der Ontologie werden für Inform-Nachrichten verwendet (wie CurrentActivitiesOf, CurrentWorkitemsOf und Finished (Ende eines Workflows)) oder dienen als Antwort auf eine Anfrage, wie HasRequiredUserRole, IsValid (für Credentials) und bei Fehlernachrichten auch ErrorInformation. Für die interne Zuordnung von WFE-Agenten und Workflows bzw. Activities im WFES-Agenten gibt es die Prädikate ProvidesActivity und ProvidesWorkitem.

Hiermit schließt die Beschreibung des agentenbasierten WFMS. Die einzelnen Elemente zum Grobentwurf, Anwendungsstruktur, zu den Rollen, zum Verhalten und zur Ontologie wurden für diese Darstellung direkt aus den Modelldiagramme der Implementierung entnommen. Das Ergebnis ist ein agentenbasiertes WFMS, welches in

Form einer Schnittstelle in ACL-Nachrichten die geforderten Anforderungen erfüllt, nämlich die grundsätzlichen Anwendungfälle für ein WFMS, wie im Abschnitt 5.1.1 auf Seite 149 vorgestellt.

5.4 Der Workflow-Agent

Dieser Abschnitt beschreibt die im Prototyp realisierten Konzepte aus dem Abschnitt 5.1.2. Der Workflow-Agent ist eine Erweiterung des agentenbasierten WFMS, wie es im vorigen Abschnitt 5.3 vorgestellt wurde. Die wesentlichen Komponenten in der Umsetzung des Worklfow-Agenten sind:

- ein Lebenszyklus
- einige Ontologiekonzepte
- eine Entscheidungskomponente
- und einige Interaktionen

Im folgenden Text werden diese Bestandteile vorgestellt.

5.4.1 Der Lebenszyklus

Der Lebenszyklus des Workflow-Agenten ist in Abbildung 5.13 dargestellt. Der Workflow-Agent wird vom Workflow-Enactmentservice im Rahmen der Workflowinstanziierung erzeugt. Die fünf wesentlichen Zustände im Lebenszyklus des Workflow-Agenten sind:

- frisch erzeugt (created),
- angekommen (noch nicht voll funktionstüchtig – arrived),
- untätig (aktionsfähig – idle),
- in der Ausführung (workflow execution) und
- mit der Arbeit abgeschlossen (done).

Der Zustand idle ist der eigentlich interessante Zustand: Die Entscheidungskomponente des Agenten ist aktiv und entscheidet die nächste proaktive Aktion. Dazu werden aktuelle Informationen aus der Wissensbasis verwendet: (a) Informationen zum Stand der Workflowausführung, (b) Informationen über das nächste passende WFMS um (Teile vom) Workflow auszuführen und (c) Informationen über die Heimatplattform. Anhand dieser Informationen stößt die Entscheidungskomponente eine der verfügbaren Interaktionen an.

Die Transitionen zwischen den in Abbildung 5.13 dargestellten Zuständen repräsentieren Interaktionen, die jeweils mehrere Zwischenzustände enthalten. So enthalten z.B. die beiden Migrationsinteraktionen auch einen FIPA-konformen suspendierten Zustand. Innerhalb der Interaktionen wird die Wissensbasis bezüglich Informationen über die Workflowausführung, das nächste passende WFMS und die Heimatplattform genutzt.

Die Zahlen in der Grafik 5.13 stellen eine beispielhafte Folge von Entscheidungen dar. Der Agent kann also proaktiv (3) das nächste passende WFMS suchen, welches Funktionalität für seinen Workflow bereitstellt. Wenn ein passendes WFMS bekannt ist, kann der Agent (4) dorthin migrieren. Nach der Ankunft findet zuerst eine Lokalisation statt, danach ist der Zustand idle auf der aktuellen Plattform erreicht. Jetzt

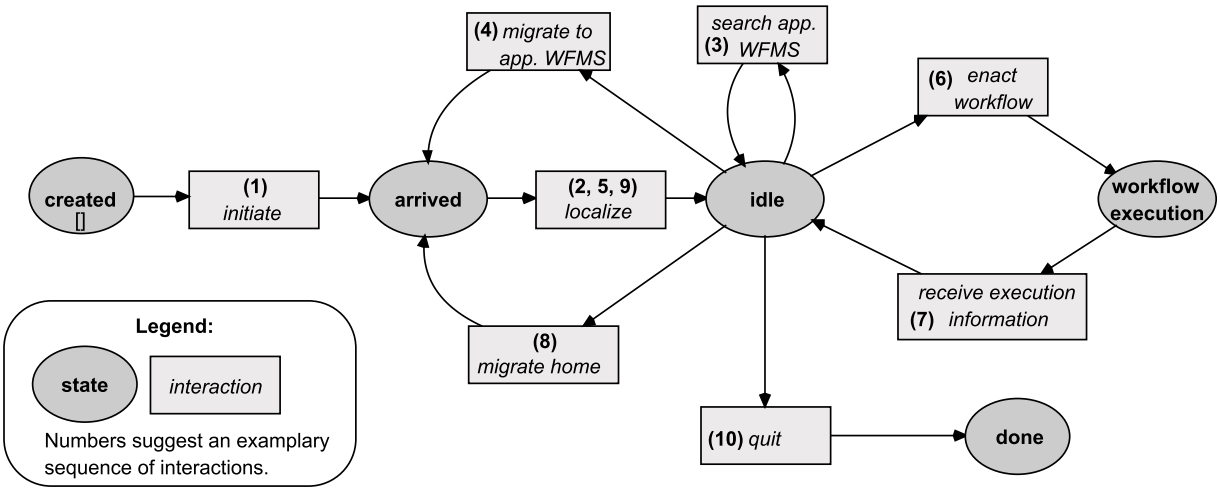


Abbildung 5.13: Vergröberter Lebenszyklus für den Workflow-Agent.
Aus: [Reese et al. 2008].

kann der Workflow oder ein Teil des Workflows (6) vom lokalen Workflow-Enactment-service (WFES) und einer lokalen Workflowengine (WFE) ausgeführt werden. Der Workflow-Agent wartet dann (7) auf Informationen über die Ausführung. Die Schritte drei bis sieben werden bis zur vollständigen Workflow-Ausführung iteriert. Dann kann der Agent (8) zu seiner Heimatplattform migrieren und (10) seine Aufgaben beenden.

Zur Umsetzung des Workflow-Agenten wird die Ontologie mit Konzepten für den Ausführungszustand eines Workflows, für seine interne Wissensrepräsentation und Prädikaten für die Initialisierung des Workflow-Agenten vervollständigt. Die Ontologie-Änderungen werden im Abschnitt 5.4.2 betrachtet.

Die Interaktionstransitionen werden von Protokollnetzen implementiert. Der Lebenszyklus ist an denjenigen Zuständen *statisch*, wo es genau einen Ausgang aus einem Zustand gibt. Der *dynamische* Teil im Lebenszyklus wird durch die Entscheidungskomponente realisiert. Diese wird im Abschnitt 5.4.3 vorgestellt. Die Interaktionen, die durch die Einführung des Workflow-Agenten neu oder geändert sind, werden im Abschnitt 5.4.4 vorgestellt.

5.4.2 Die Ontologie

Die Ontologie wurde im Verlauf der Implementierung für den Workflow-Agenten um vier Konzepte erweitert.

Der Workflow-Enactment-Service-Agent (WFES) nutzt das Prädikat WfesWorkflowDefinitionWithId für seine interne Kommunikation zwischen Protokollnetzen und Entscheidungskomponente. Zur Initialisierung des Workflow-Agenten wird das Prädikat WfesWorkflowInformation benutzt.

Der Workflow-Agent benutzt Ontologie-Objekte für die interne Speicherung von Informationen. Er speichert die Adresse eines WFMS im Agentennetzwerk stets als Kombination einer AMS-Adresse mit einer WFES-Adresse mit Hilfe des Konzepts Wf-

AgentPlatformAddress. Seine Heimatplattform, die nächste geplante Platform sowie die aktuelle Plattform werden zusammen mit Hilfe des Konzepts WfAgentPlatform-Addresses gespeichert.

5.4.3 Die Entscheidungskomponente

Der Teil des Lebenszyklus des WF-Agenten, welcher dynamisch ist, wurde in Form einer Entscheidungskomponente (DC) umgesetzt. Die grobe Form des Lebenszyklusses wurde bereits in Abbildung 5.13 vorgestellt. Der dynamische Teil beschränkt sich auf den Zustand idle. Der Zustand idle hat vier Ausgänge, d.h. die DC kann vier verschiedene Protokolle initiieren. Jede der drei einmündenden Kanten bedeutet, dass die DC im Anschluss an die entsprechenden Interaktionen neu gestartet werden muss. Aufgrund von Wissensbasis-Einträgen entscheidet die Entscheidungskomponente, welche Interaktion als nächstes angestoßen werden soll.

Die Entscheidungsfindung ist in Abbildung 5.14 dargestellt. Zur Darstellung wurde das Format des Interaktionsprotokolldiagramms in abgewandelter Weise benutzt: Horizontale Pfeile stehen hier nicht für ACL-Nachrichten, die zwischen Agentenrollen ausgetauscht werden, sondern für je eine agenteninterne Kommunikation, bei der die Fabrik veranlasst wird, die DC oder ein Protokollnetz zu instanziieren.

Dargestellt sind acht beteiligte kommunizierende Komponenten: vier am oberen Rand und weiter unten im Verlauf noch zweimal zwei weitere Komponenten (aus Platzgründen wurde das Diagramm auf diese Weise gestaucht). Der frisch erzeugte Workflow-Agent erhält von dem Heimat-WFES die Nachricht mit „workflow information“. Workflow information ist ein Konzept der Ontologie. Der Workflow-Agent empfängt diese Nachricht und bearbeitet sie reaktiv durch ein Protokollnetz mit dem Namen „Workflow_instantiateWorkflow_doInstantiateWfAgent“. Der Workflow-Agent initialisiert seinen Lebenszyklus und startet schließlich das Protokollnetz „Workflow_getCurrentPlatform“, welches die Funktion Localize aus Abbildung 5.13 umsetzt. Am Ende startet dieses Protokollnetz die Entscheidungskomponente. Diese prüft nacheinander die Informationen zum aktuellen Zustand und startet eines von vier Protokollnetzen: „Workflow_searchWFMS“, „Workflow_enactWfAgent“, „Workflow_workflowEndReachedPart1“ (für die Funktion receive execution information aus Abbildung 5.13) und „Workflow_workflowEndReachedPart2“ (für die Funktion quit aus Abbildung 5.13).

In Abbildung 5.14 ist der Ablauf in der Entscheidungskomponente vollständig dargestellt, während die Funktionalität in den aufgerufenen Protokollnetzen lediglich durch Action-Anschriften angedeutet ist. Diese sind in eigenen Diagrammen ausführlich entworfen. Nur zwei der Rollenbezeichner tragen kein Ausrufezeichen im Namen. Diese werden vom Code-Generator in Renew verwendet, um je ein Skelett für eine Implementierung als Protokollnetz oder Entscheidungskomponente zu generieren.

Die vollständige Implementierung der Entscheidungskomponente als Referenznetz ist in Abbildung 5.15 stellvertretend für die vielen Protokollnetze und Entscheidungskomponenten des Prototypen angegeben. Zu erkennen ist die Dokumentationskonvention, wo der (per Konvention geregelte) Dateiname, die Autoren und das Erstellungsdatum zusammen mit einem kurzen Kommentar angegeben sind. Weiter ist der

Workflow_lifecycle

@author Christine Reese
@date 15.2.2007

This diagram describes the lifecycle of a workflow agent. It is realized as a DC.

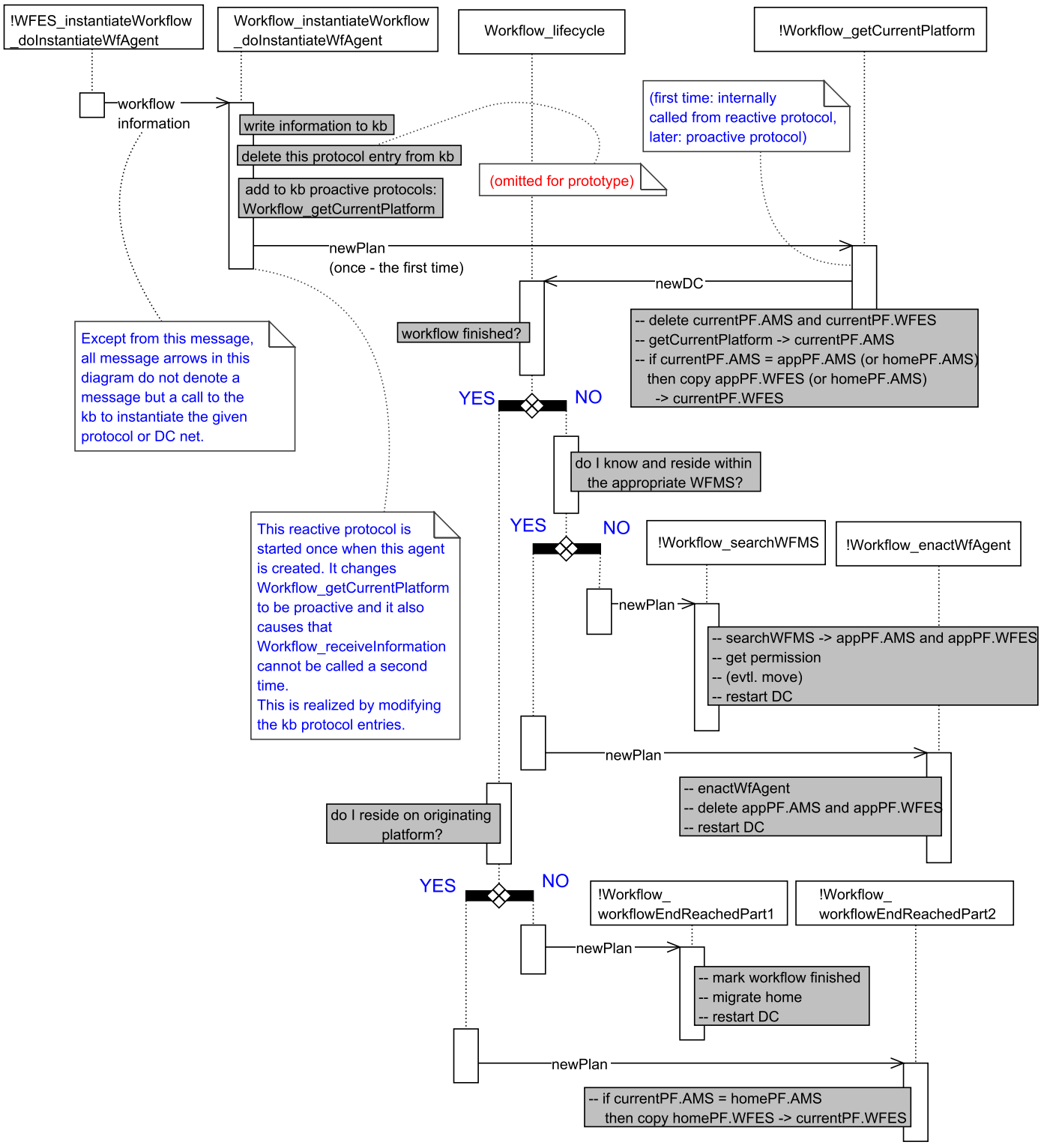


Abbildung 5.14: Details zum Lebenszyklus des WF-Agenten

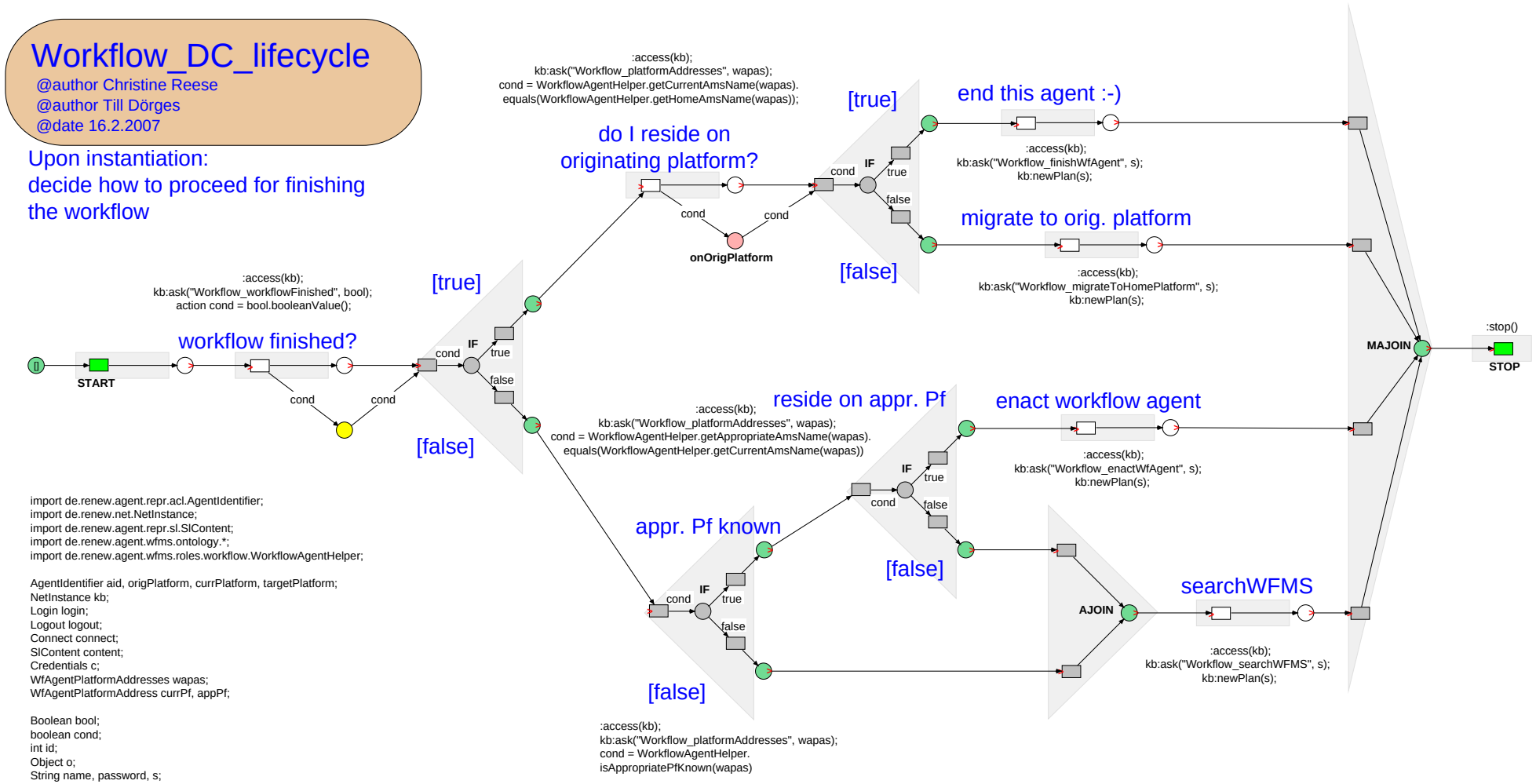


Abbildung 5.15: Implementierung des Lebenszyklus des WF-Agenten. Die Netzkomponenten und die Kommentare in großer Schrift geben einen Überblick über die Funktionsweise.

Deklarationsknoten des Referenznetzes zu erkennen, in dem die notwendigen Java-Klassen importiert werden und Variablen deklariert werden.

Die Funktion der Entscheidungskomponente ist als Netz implementiert. Nach dem Start werden, wie im Konzept in Abbildung 5.14 beschrieben, die Entscheidungen anhand von Wissensbasis-Einträgen getroffen und es wird die jeweils passende Interaktion angestoßen. Am Schluss beendet sich die Entscheidungskomponente. Dieser Schritt ist für eine Entscheidungskomponente im Allgemeinen nicht notwendig (oder auch unerwünscht, wenn Interaktionsübergreifende Zusammenhänge hergestellt werden sollen). Für die Migration ist es jedoch notwendig, dass der Agent keinerlei laufende Entscheidungskomponenten oder Protokollnetze betreibt.

5.4.4 Die Interaktionen

Die Interaktionstypen aus dem Lebenszyklus (Abbildung 5.13) des Workflow-Agenten sind durch Protokolldiagramme spezifiziert:

- initiate: Implementiert durch InstantiateWorkflow
- localize: Implementiert durch getCurrentPlatform
- search app. WFMS: Implementiert durch searchWFMS
- migrate to app. WFMS: Implementiert durch migrate
- enact workflow: Implementiert durch chooseWFE und enactWorkflow
- receive execution information: Implementiert durch WorkflowEndReached (Part 1)
- migrate home: Implementiert durch migrate
- quit: Implementiert durch WorkflowEndReached (Part 2)

InstantiateWorkflow Die Initialisierung eines Workflows wird von einem Benutzer ausgelöst. Nachdem die Rechte geprüft wurden und eine Workflowdefinition verfügbar ist, wird der Workflow-Agent vom WFES-Agenten erzeugt und bekommt Initialisierungs-Informationen zugeschickt. Am Ende dieser Interaktion startet der WF-Agent die Interaktion getCurrentPlatform.

WorkflowEndReached Wenn der Workflow fertig ausgeführt ist, informiert der WFE-Agent den Workflow-Agenten. Dieser migriert zurück an seine Heimatplattform (erster Teil der Interaktion) und informiert den dortigen WFES-Agenten, welcher veranlasst, dass der auftraggebende Benutzer informiert wird (zweiter Teil der Interaktion).

chooseWFE Bei der Workflow-Instanziierung entscheidet der WFES-Agent, welche Workflow-Engine (WFE) den Auftrag bekommt. Wenn ein Workflow-Agent in einem WFMS eintrifft, fragt er den WFES nach der Adresse des passenden WFE-Agenten. Im ursprünglichen WMFS ohne Workflow-Agent war diese Aktion (eben keine *Inter*-aktion) als interne Wissensbasis-Anfrage im WFES-Agenten implementiert.

getCurrentPlatform Beim ersten Start des WF-Agenten wird diese Interaktion nach der Interaktion InstantiateWorkflow angestoßen und im Folgenden nach jeder Migration proaktiv (also ohne eingehende Nachricht) gestartet. Der WF-Agent schickt eine Anfrage an den lokalen AMS und erhält mit der Antwort den voll

spezifizierten Agent-Identifier des AMS. Am Ende dieser Interaktion wird die Lebenszyklus-DC des Agenten angestoßen.

enactWfAgent Beim Starten und Beenden von Workflows übernimmt der WF-Agent einen Teil der Rolle des WFES-Agenten: Er gibt die zur Ausführung notwendigen Informationen an den WFE-Agenten und nimmt auch die Meldung bei der Fertigstellung der Ausführung entgegen (diese Aufteilung der Aufgaben entspricht den Überlegungen aus Abschnitt 5.1.2 auf Seite 151). Diese Interaktion wird aus der DC des Workflow-Agenten angestoßen.

searchWFMS Diese Interaktion stellt sicher, dass die Adresse für die nächste passende Plattform in der Wissensbasis bekannt ist und veranlasst die Migration dorthin, falls der Agent sich nicht bereits dort nicht befindet. Im letzteren Falle wird die DC neu gestartet, so dass die Interaktion enactWfAgent ausgeführt werden kann.

migrate Diese Interaktion wurde bereits von Rölke in [Rölke 2004] vorgestellt. Wegen der Verwendung in einem neuen Zusammenhang wird sie hier genauer vorgestellt (Abbildung 5.16). Der Workflow-Agent („Agent“ im Diagramm) prüft zuerst, ob er bereits auf der Zielplattform ist. Die eigentliche Migration wird durch eine Nachricht mit dem AgentIdentifier des AMS auf der Zielplattform an den lokalen AMS ausgelöst. Der veranlasst den lokalen Plattform-Agenten, den mobilen Agenten einzuschränken, so dass dieser nur mit den lokalen Plattform-Agenten kommunizieren kann. Dazu muss der Agent bestätigen, dass er in keine laufende Interaktion involviert ist, keine DC laufen hat und keine unverarbeiteten Nachrichten (ankommende oder abgehende) warten. Dieses ist als synchrone Kommunikation über einen synchronen Kanal im Agentennetz implementiert. Wegen der Mitwirkung des Agenten ist dieser Schritt als gepunkteter Doppelpfeil dargestellt. Im eingeschränkten Zustand wird nun der Agent aufgefordert, den Inhalt seiner Wissensbasis zum Transport auszuliefern. Der AMS löscht sodann die lokalen Verzeichniseinträge für den Agenten. Schließlich schickt der AMS eine Nachricht mit den erforderlichen Informationen an den AMS der Zielplattform. Dort findet (bis auf die Namensvergabe) derselbe Prozess statt wie bei der Erzeugung eines neuen Agenten. Erst wenn der lokale AMS eine Bestätigungsnachricht mit dem neuen voll spezifizierten AgentIdentifier erhalten hat, veranlasst er den Plattform-Agenten, den mobilen Agenten lokal zu beenden. Der mobile Agent wird wahrscheinlich auf der neuen Plattform wieder das GeneralAgentSetup starten, um sich beim AMS und DF zu registrieren und um nach benötigten Diensten zu suchen. Im Falle des Workflow-Agenten meldet er sich beim AMS und DF an, hat aber die Adresse des lokalen WFES-Agenten bereits in seiner Wissensbasis. Um sicherzustellen, dass der Plan korrekt ausgeführt wurde, startet der Workflow-Agent die Interaktion getCurrentPlatform, an deren Ende wieder die DC gestartet wird.

Abbildung 5.17 zeigt nun zum Überblick ein Interaktionsdiagramm mit einer beispielhaften Sequenz von Interaktionen des Workflow-Agenten. Dabei sind an den

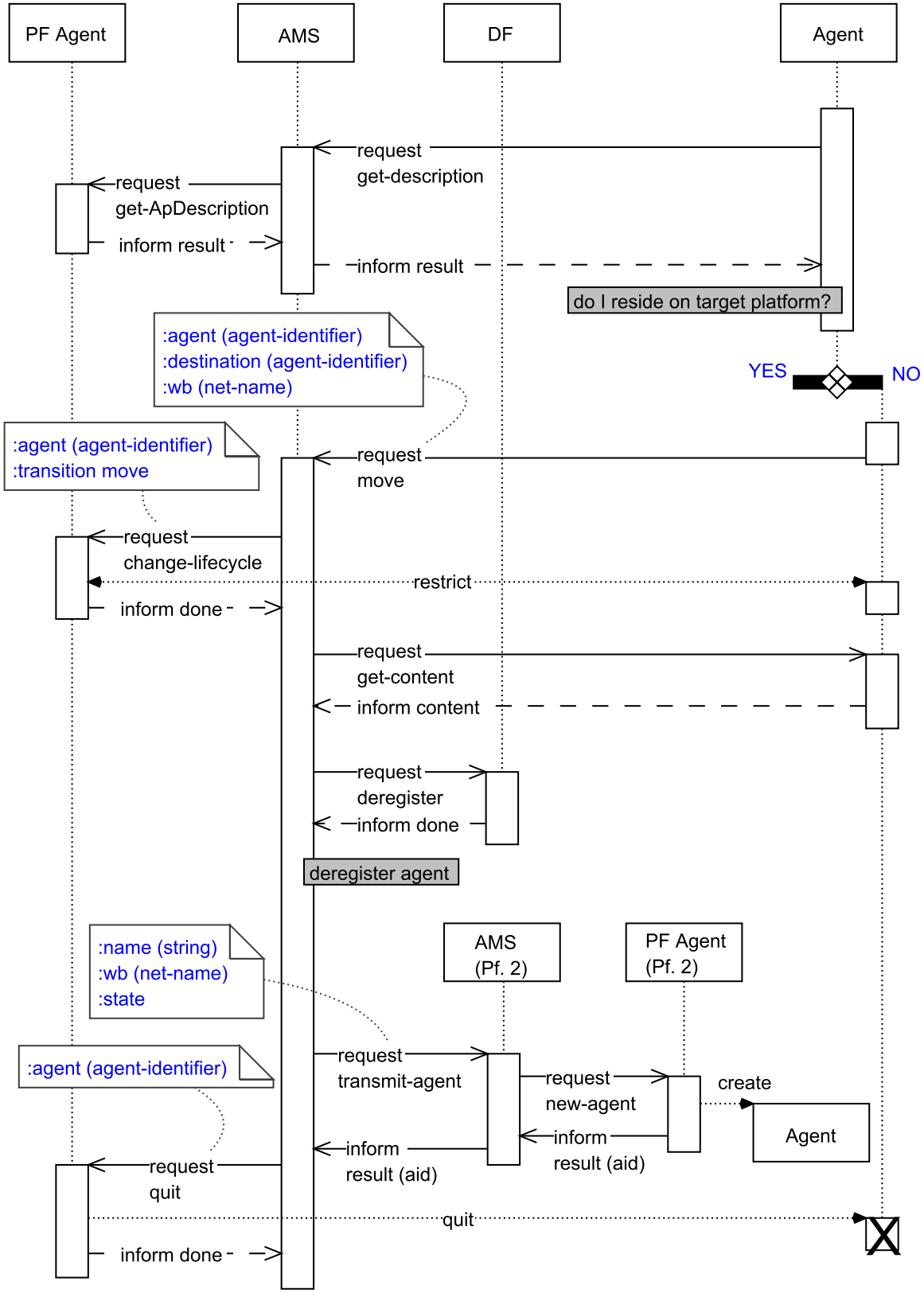


Abbildung 5.16: Interaktionsdiagramm für eine Migration

Nachrichtenpfeilen jeweils das Performativ und die benannte Agent-Action angegeben. Auf der Startplattform (Platform 1) sind der Workflow-Enactmentservice (WFES) und die Workflowdefinitionsdatenbank involviert (neben den Plattform-Agenten Agentenmanagementservice (AMS) und Verzeichnisdienst (DF)). Der AMS erzeugt den Workflow-Agenten, welcher dann seine Initialisierungsinformationen erhält (dies ist nur einmal möglich). Die Zahlen entsprechen den Zahlen aus Abbildung 5.13. Jede Aktivierung in Abbildung 5.17 (außer Nummer sieben) auf der Lebenslinie des Workflow-Agenten entspricht einer proaktiv gestarteten Interaktion. Die Abschnitte der Lebenslinie zwischen den Aktivierungen repräsentieren den idle-Zustand des Agenten (außer am Anfang und zwischen Aktivierung sechs und sieben). Auf der zweiten Plattform repräsentiert der Workflowmanagementagent (WFMS) das ganze System für nicht-eingeloggte Agenten und wird deshalb zunächst kontaktiert. Die Antwort enthält dann die Adresse des Sub-Agenten Workflow-Enactmentservice (WFES) auf der zweiten Plattform, so dass der Workflow-Agent nach der Migration sich direkt an den WFES wenden kann, um Kontaktinformationen für die passende Workflowengine (WFE) für seinen Workflow zu erhalten. Die Ausführung durch den WFE-Agenten erfordert weitere Interaktionen mit anderen WFMS-Subagenten und eingeloggten Benutzern, die hier nicht dargestellt sind, weil der Workflow-Agent nicht daran beteiligt ist.

5.5 Zusammenfassung

In diesem Kapitel wurde die prototypische Umsetzung der im vorigen Kapitel ausgearbeiteten Architektur dargestellt. Dazu wurde im Abschnitt 5.1 der konzeptuelle Entwurf vorgestellt: erstens für das WFMS im Allgemeinen und als geschachteltes System, zweitens für den Workflow-Agenten und drittens für die Prozessverwaltung in Agentennetzen. Diese Anforderungen sind stark konzeptueller Natur und entsprechen (für das WFMS) den Anforderungen zum Beginn der prototypischen Umsetzung im Lehreprojekt. Die detaillierteren Anforderungen wurden erst im Rahmen der Umsetzung mit dem Paose-Ansatz erarbeitet und wurden weiter unten in Form des Grobentwurfs in diese Arbeit eingeordnet.

Bevor im Anschluss die Durchführung nach dem Paose-Ansatz beschrieben wird, beschreibt der Abschnitt 5.2 wichtige Arbeiten, die im Laufe der Umsetzung am Rahmenwerk Capa durchgeführt wurden. Hier werden folgende Ergebnisse vorgestellt: Ein Subscription Manager zur Verwaltung von Abonnements wie z.B. zeitlich ausgedehnte Suchfunktionen, die Anbindung an openNet mittels Proxy-Agenten, ein an die Anforderungen der Universität angepasster lokaler Netzknoten für einen zentralen DF („ZDF“), die Erweiterung des Capa-Standard-Agenten um eine Schnittstelle für Entscheidungskomponenten („DC“ – Decision Component) und als letzten Punkt die Nutzung einer bis hierher ungenutzten Möglichkeit, dass ein Agent seine Wissensbasis bezüglich der zu startenden Protokolle ändern kann (für eine nur einmal auszuführende reaktive Interaktion eines Agenten bei seiner Erzeugung).

Es folgt im Abschnitt 5.3 der Hauptteil der Umsetzung, in dem alle Schritte des Paose-Ansatzes vorgestellt werden. Dabei können aus Platzgründen nicht alle Details dargestellt werden (insbesondere nicht die Interaktionsprotokolle). Das Ergebnis ist ein lauffähiges WFMS, welches Nachrichten in ACL entgegennimmt und dement-

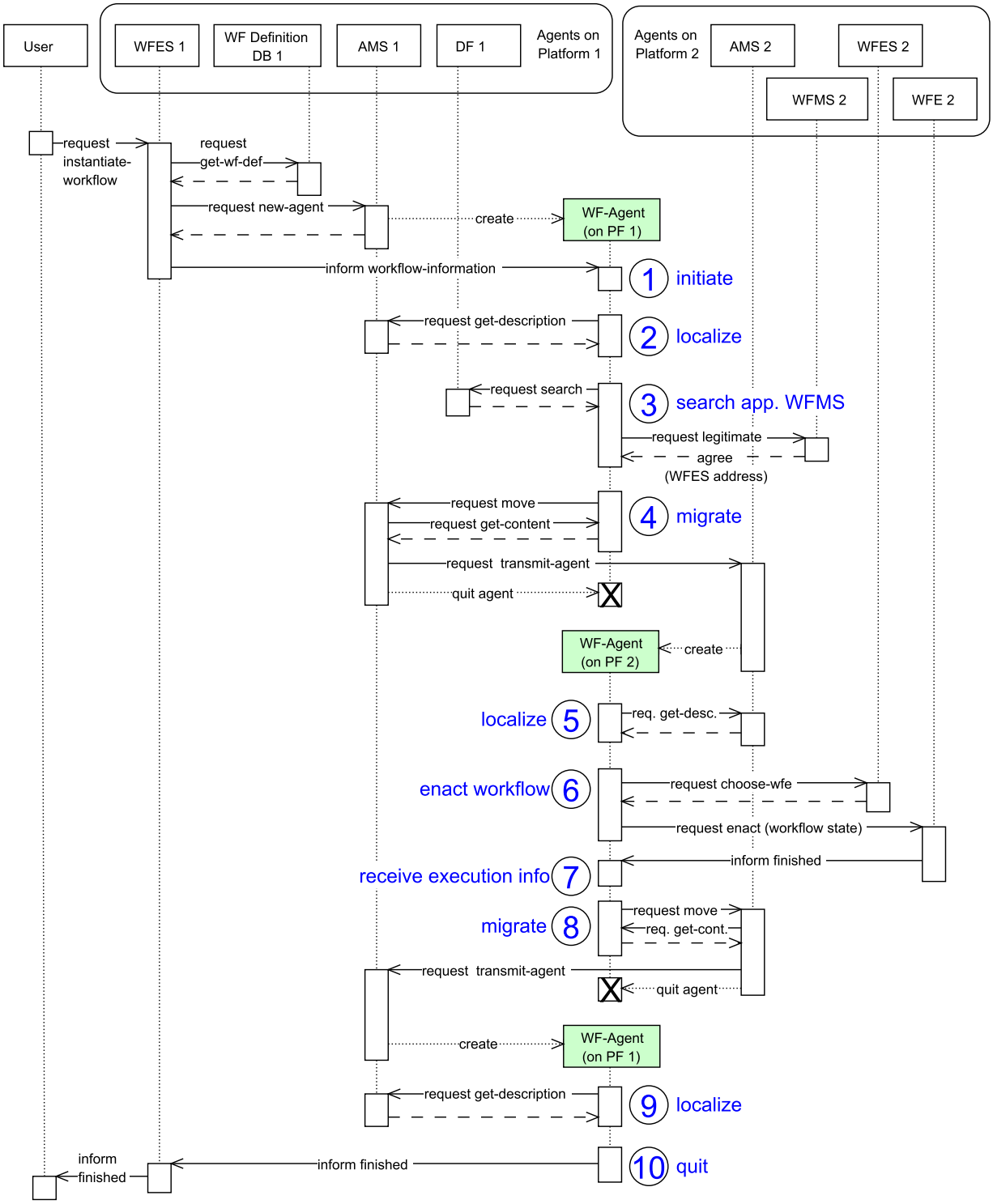


Abbildung 5.17: Beispiel: Interaktionsdiagramm für den Workflow-Agenten. Aus: [Reese et al. 2008].

sprechend Workflows instanziiert und den als Benutzern angemeldeten Agenten eine jeweils aktuelle Liste von Workitems anbietet. Der Workflow-Agent wird dann im Abschnitt 5.4 ausgearbeitet. Insofern ist das Ziel eines flexiblen und verteilten agentenbasierten WFMS für Agenten erreicht.

Im nächsten Kapitel werden alle Ergebnisse integrierend diskutiert und zusammengefasst. Ergänzend werden Weiterentwicklungsmöglichkeiten aufgezeigt und Beispiele vorgestellt.

Kapitel 6

Diskussion und Integration

In diesem letzten inhaltlichen Kapitel sind die Ergebnisse der Arbeit in ihrem Zusammenhang dargestellt. Hier ist es möglich, die Funktion und den Nutzen der einzelnen Abschnitte in den Kontext des gesamten Textes einzuordnen und die Querbezüge herauszustellen. Auch die Veröffentlichungen, die im Rahmen dieser Arbeit entstanden sind, werden in den gesamten Kontext eingeordnet. An den Stellen, wo es angebracht ist, wird ein Ausblick auf weiterführende konkrete Arbeiten oder auf größer angelegte, weiterführende Ideen gegeben.

Zu vier der beschriebenen Integrationsstufen gibt es Beispiele aus der Lehre oder prototypische Beispiele, die am Ende dieses Kapitels vorgestellt werden: zum verwendeten petrinetzbasierten WFMS „gehört“ die Beispielanwendung eines Telefonversand („Phoneshop“); das in vielen Jahren verwendete Beispiel für die Lehre der agentenorientierten Software-Entwicklung ist die Anwendung Siedler nach dem Brettspiel „Die Siedler von Catan“; den Rahmen für eine Testanwendung des agentenbasierten WFMS bildet eine Reiseagentur; und schließlich ist die Vision einer Anwendung, deren Anforderungen so komplex sind, dass die Verwendung der hier beschriebenen Infrastruktur gemäß Stufe V sinnvoll ist, eine Plattform für verteilte Software-Entwicklung. Die Beispiele sind dabei unterschiedlich detailliert und unterschiedlich funktionsfähig: Siedler ist mit Abstand die umfangreichste Anwendung mit vielen Einzelfunktionen und Grundlage diverser Diplom- und anderer Studienarbeiten, der Phoneshop ist ebenfalls funktionsfähig mit verteilten Benutzern ausführbar und demonstriert einige Funktionen des petrinetzbasierten WMFS. Die Reiseagentur bildet eher einen begrifflichen Hintergrund für die Testanwendung des agentenbasierten WFMS, während die Software-Entwicklungsumgebung im vorliegenden Stadium als Denkrahmen und als Projektionsfläche zu verstehen ist (auch wenn Einzelfunktionen durchaus prototypisch umgesetzt wurden).

6.1 Ergebnisse der Arbeit

In dieser Arbeit wird eine Architektur für den Aufbau einer Prozess-Infrastruktur für Agentenanwendungen entwickelt. Petrinetze dienen hier als „Denkzeug“ und als Modellierungswerkzeug zur Definition von Verhalten und nicht zuletzt als Programmiersprache zur Implementation des Prototyps.

Auf Grundlage der von mir herausgestellten Entsprechungen zwischen Workflow- und Agentenkonzepten wird die Integration in fünf Stufen vorgestellt. Stufe I beschreibt die reinen Architekturen, Stufe II die in den unteren Architekturschichten angereicherten Architekturen. Folgend den weiteren Stufen, sind die wesentlichen erreichten Ergebnisse der Arbeit:

- Konzepte, Architektur und Prototyp für ein agentenbasiertes WFMS, das in einem Agentennetz wie openNet eingesetzt werden kann (Stufe III der Integration).
- Konzepte, Architektur und konkrete Ideen für eine Prozess-Infrastruktur in Agentenanwendungen (Stufe IV der Integration).
- Konzepte und Architektur für die umfassende und vollständige Integration von Struktur- und Prozesskomponenten von Anwendungen zu *Einheiten*, indem Agenten und Workflows verschmolzen werden. Damit wird eine verbesserte Software-Architektur für Agenten/Workflowanwendungen erreicht (Stufe V der Integration).

Das Ziel der Arbeit ist die Architektur einer Prozess-Infrastruktur für Agentenanwendungen (Apia), welches mit Stufe IV der Integration erreicht wird. Das weitergesteckte Ziel einer vollständigen Integration von Workflow- und Agententechnologie für eine Form von *Einheiten*, die eine Agenten-/Workflow-Anwendung bildet, wird in Form einer Vision behandelt.

Als technische Ausgangspunkte werden für diese Arbeit das petrinetzbasierte Agenten-Rahmenwerk Capa von Michael Duvigneau (Abschnitt 2.2.3) und das petrinetzbasierte Workflowmanagementsystem von Thomas Jacob (Abschnitt 2.3.4) herangezogen. Die Umsetzung des Prototyps erreicht Stufe III. Der Prototyp wird um dem Workflow-Agenten erweitert, so dass ein nach Yan et al. *agentenbasiertes* WFMS resultiert. Damit geht der Prototyp für das agentenbasierte WFMS über eine funktional äquivalente Umsetzung des Workflowmanagementsystems von Thomas Jacob hinaus.

Die grundlegende Idee zur Entwicklung einer Architektur für eine Prozess-Infrastruktur ist die Integration von Workflowtechnologie in die Agententechnologie. Im Paose-Ansatz werden die Entwicklung von Strukturkomponenten (Agenten) und von Prozesskomponenten (Interaktionen) gleichberechtigt als Teamaufgaben gehandhabt. Dieses Konzept wird in dieser Arbeit weitergeführt und in Form einer Architektur umgesetzt: Agenten und Workflows werden als zwei gleichberechtigte Blickwinkel auf eine komplexe Anwendung betrachtet. Aus der daraus abgeleiteten Systematik kommt die Arbeit sogar über das ursprüngliche Ziel hinaus und es können Ideen und eine Architektur für die *vollständige* Integration von Agenten und Workflows zu neuartigen Einheiten aus der Systematik gewonnen werden.

Im Laufe dieser Arbeit gab es einige zeitlich klar begrenzte Phasen, durch welche die Arbeit jedesmal besonders stark geprägt wurde. Jeweils im Wintersemester fanden die Lehre-Projekte zur agentenorientierten Software-Entwicklung statt, in denen der Paose-Ansatz praktisch gelehrt wird. In diesem Rahmen fand von der Machbarkeitsstudie bis zur fertigen Umsetzung des Prototyps ein Großteil der Realisierungsarbeiten statt. Neben aktuellen Fragestellungen aus Workshops und Konferenzen habe ich auch viel Inspiration und Rückmeldung aus der praktischen Lehre gezogen.

Die Gliederung und wesentliche Elemente der gesamten Arbeit sind in mehreren Veröffentlichungen vorbereitet worden. In [Reese et al. 2006a] wird ein erster thematischer Überblick mit Literaturgrundlage aufgebaut. Die weitgehend vollständige

Argumentationslinie ist in [Reese et al. 2007]) vorgestellt worden und ist in überarbeiteter Fassung in den Postproceedings [Reese et al. 2008] erschienen.

Es folgt eine übergreifende Diskussion, bei der auch eine Zusammenfassung der Inhalte der Arbeit geleistet wird.

6.2 Ergebnisse der Kapitel und Abschnitte

Kapitel 1

In der **Einleitung** wird das Gebiet der komplexen Systeme mit der grundlegenden Problematik der Strukturierung nach Strukturkomponenten oder nach Prozessbestandteilen für diese Arbeit benannt. Ergebnis ist eine systematische Übersicht über die drei wesentlichen beteiligten Gebiete: Petrinetze, Agententechnologie und Workflowtechnologie. Das Ziel der Arbeit – die Prozess-Infrastruktur für Agentenanwendungen – und die notwendigen Arbeiten dazu werden dargelegt.

Kapitel 2

Im Kapitel **Prozesse, Agenten und Workflows** werden für die verwendeten Technologien die hier wesentlichen Begriffe und Bestandteile in ihrem Zusammenhang dargestellt. Das formale Gebäude aus der Standardliteratur darzustellen würde keinen wesentlichen Beitrag zum konzeptionellen Charakter dieser Arbeit liefern, vertiefende Literatur ist jedoch stets mit angegeben.

Im Abschnitt über **Prozessmodellierung und Petrinetze** werden aus der AUML die Sequenzdiagramme als in dieser Arbeit und für den Paose-Ansatz wesentliche UML-Elemente vorgestellt. Der Unterschied zwischen *Interaktionsdiagrammen* für einen bestimmten Ablauf und *Protokolldiagrammen* für die Spezifikation von mehreren alternativen Abläufen ist dabei wichtig. Die Petrinetze und die Referenznetze werden anhand von Prinzipien vorgestellt.

Bei den Petrinetzen steht dabei die Eignung zur Modellierung verteilter Systeme im Vordergrund (Prinzipien der Dualität, Lokalität, Nebenläufigkeit, des Zustandes, der graphischen und formaltextuellen Darstellung). Mit der Schaltregel und der Unterklasse der Kausalnetze wird der Begriff *Prozess* präzise erfassbar. Die Darstellung von Konfliktsituationen und die formalen Relationen *li* und *co* ermöglichen dabei die formale Fassung von Abhängigkeit und Unabhängigkeit in einem Petrinetz (und damit auch in dem Modell eines verteilten Systems).

Bei den Referenznetzen steht die Eignung zur Programmierung komplexer Systeme im Vordergrund. Die Prinzipien der Instanziierung, der Wert- und Referenzsemantik und der synchronen Kommunikation ermöglichen zusammen das Konzept der hierarchischen Schachtelung, welches insbesondere die Modellierung von Orten mit mobilen Einheiten in verteilten Systemen intuitiv ermöglicht.

Für die Programmierung wird das Werkzeug Renew verwendet. Die für die Programmierung wesentlichen Elemente von Renew sind neben dem Editor-Kern und dem Simulator-Kern insbesondere die enge Java-Anbindung, die real nebenläufige Ausführung sowie die Verfügbarkeit von Netzkomponenten und Code-Generatoren. Ein Beispiel für einen Code-Generator ist das AIP-Plugin aus dem Paose-Ansatz,

welches mit Hilfe von automatisch zusammengefügten Netzkomponenten die Grundstruktur der zugehörigen Protokollnetze erzeugt. CAPA ist ebenfalls ein Plugin für RENEW und wird später im Text vorgestellt, im Zusammenhang mit den Konzepten aus der Agententechnologie.

Das AIP-Plugin generiert aus Protokolldiagrammen Petrinetz-Code, so dass die erzeugten Petrinetze zusammen die Semantik des Protokolldiagramms aufweisen. Für jeden Rollenbezeichner wird ein passendes Protokollnetz[1] generiert. Im PAOSE-Ansatz werden so die Protokollnetze der einzelnen Agenten in ihrer Struktur erstellt und dann für die Implementierung um Code-Anschriften ergänzt. Es wird dabei die Fähigkeit der Rollen abgebildet, Nachrichten zu senden und zu empfangen.

Aus diesem Sachverhalt kann jetzt im Rückblick das Ziel der Arbeit auch so formuliert werden: Der Zusammenhang der Rollen durch Interaktionen zerfällt bisher bei der Implementierung und Ausführung in Einzelteile und soll nun durch die Prozess-Infrastruktur auch bei der Ausführung von Agentenanwendungen erhalten bleiben. Yan et al. [YAN et al. 2001] geben passend dazu für verteilte agentenbasierte WFM-Systeme an, dass die Prozessdefinition in jeweils lokale Bestandteile auseinanderfällt. Genau dies ist der Fall im bisherigen PAOSE-Ansatz: Die Interaktionen werden modelliert und zerfallen bei der Implementation und Ausführung in lokale Bestandeile. Mit der hier vorgestellten Lösung des verteilten WFMS und mit dem mobilen Workflow-Agent wird dieser Nachteil umgangen. Um dabei nicht die nebenläufige Ausführung von unabhängigen Tasks an verschiedenen Orten im Netz zu verlieren, muss in einer weiterführenden Arbeit die vorgestellte Fragmentierung von Workflows in den Prototypen integriert werden.

Im Abschnitt über **Agenten** werden aus dem weiten Feld der Agententechnologie gerade die Begriffe und Konzepte eingeführt, die in dieser Arbeit Verwendung finden und für das weitere Verständnis notwendig sind. Agenten bilden eine ausdrucksstarke Metapher für die Softwaretechnik. Unter der Agententechnologie finden die Künstliche Intelligenz, die Sozialwissenschaften, das Gebiet der verteilten Systeme, die Spieltheorie und eben die Softwaretechnik und theoretische Informatik einen Rahmen, in dem die jeweiligen Schwerpunkte integriert werden können. Das wesentliche Merkmal für Agenten ist die Autonomie. Die FIPA spezifiziert als Standardisierungsgremium ein Referenzmodell für Agenten, Plattformen und deren Kommunikationsstandards. Das verwendete Agentenrahmenwerk CAPA basiert auf der Modellierung der Basisabstraktionen aus der Agententechnologie mit Referenznetzen (in Form von MULAN mit den vier Ebenen Umgebung – Plattform – Agent – Protokoll[2]).

Zur Softwaretechnik gehört immer auch ein Software-Entwicklungsansatz; in diesem Fall ist das der PAOSE-Ansatz mit spezifischen Ausgestaltungen der fünf Facetten eines Entwicklungsansatzes: Werkzeuge, Vorgehen, Techniken, Organisation und geeignete Anwendungsfelder sowie ein Rahmen aus Prinzipien und Paradigmen, hier aus der Agenten- und Prozessorientierung. PAOSE dient in dieser Arbeit jedoch nicht nur als Ansatz für die Entwicklung des Prototyps: die Matrixorganisation wird später

[1] Protokollnetz = Referenznetz zur Implementierung von Verhalten für einen CAPA-Agenten

[2] MULAN mit seinen vier charakteristischen Ebenen ist nebenbei auch ein Ausgangspunkt für die Modellierung, Formulierung und Formalisierung von Referenzmodellen für verschiedene komplexe Sachverhalte, siehe z.B. [MOLDT et al. 2005, KÖHLER und WESTER-EBBINGHAUS 2007]

in der Arbeit als Vorbild für die gleichberechtigte Verbindung von Struktur und Verhalten eingesetzt und die Werkzeugfacette kann (im Sinne des Ausblicks) durch eine Umsetzung von Pia weiterentwickelt werden.

Mit den beiden offenen Agentennetzen Agentcities und openNet schließt das Einführungskapitel für Agenten. Die Agentennetze betonen die konzeptionelle Möglichkeit eines *Service-Grids*, in dem heterogene Dienste auf skalierbare Weise in stets aktueller Form veröffentlicht werden und so für die dynamische Dienstkomposition zur Verfügung stehen. Die konkrete Anbindung an diese Agentennetze hat den Grad der Standardkonformität von Capa deutlich vorangebracht und die Tauglichkeit im internationalen Umfeld gezeigt. Nicht zuletzt fand diese Arbeit im Umfeld der Agentennetze eine zu der Zeit sehr rege Community (Veröffentlichungen: [Reese et al. 2003, Willmott et al. 2005]). Für später auftauchende andere Agentennetze, die auf FIPA und Jade beruhen, kann das Plugin Ace um die dann notwendigen spezifischen Komponenten erweitert werden.

Der Abschnitt **Workflows** führt die Begriffe aus dem Bereich der Workflowtechnologie ein. Besonders wichtig für diese Arbeit ist dabei, dass in dieser Arbeit die Workflows nicht lediglich als Formalisierung von Geschäftsprozessen zu verstehen sind, sondern allgemeiner als Explizierung von Prozessen innerhalb von Anwendungen.

Für die Workflows ist die *WfMC* das Standardisierungsgremium. Das Referenzmodell für Workflows besteht insbesondere aus dem Workflow-Enactment-Service (WFES) mit den Workflowengines, die den Kern eines Workflowmanagementsystems bilden. Ein WFES hat fünf Schnittstellen: Administrations-, Definitions-, Benutzer- und Anwendungsschnittstelle sowie eine Schnittstelle zu weiteren Workflow-Enactment-Services. Diese grundlegende Architektur wird für den Prototypen in dieser Arbeit verwendet.

Der Bezug zwischen Workflows und Petrinetzen wird auf besonders elegante Art durch die Workflownetze von van der Aalst hergestellt. Dabei wird als Ergebnis deutlich, dass das Vokabular und der Methodenreichtum aus dem Bereich der Petrinetze auch für Workflownetze zur Verfügung steht. Insbesondere ist der Begriff des korrekten Workflows durch die Soundness-Eigenschaft formalisiert. Soundness ist zunächst eine Charakterisierung des *Verhaltens* von Workflownetzen, müsste also über eine Analyse des Zustandsraumes nachgewiesen werden. Eine solche Analyse stößt schon bei einfachen Beispielen an Realisierungsgrenzen. Im realen Fall werden bevorzugt *strukturelle Eigenschaften* von Workflows untersucht, die in polynomieller Zeit nachgewiesen werden können. Dazu werden drei weitere Eigenschaften verwendet: die Free-Choice-Netze, wohlstrukturierte Netze und S-Coverable-Netze. Die Vorteile dieser Eigenschaften lassen sich nicht nutzen, wenn z.B. Ressourcenplanung zur Workflowdefinition hinzugenommen wird. Die Netzstruktur passt dann nicht mehr zu den definierten strukturellen Eigenschaften und die Analyse hat exponentiellen Aufwand. Zur Analyse von Workflows wird auch aus diesem Grunde von den fünf vorgestellten Perspektiven auf einen Workflow (Kontrollfluss-, Task-, Ressourcen-, Funktions- und Datenperspektive) nur die Kontrollflussperspektive verwendet. Damit kann für die vorliegende Arbeit die Analyse von Workflows im Prinzip angewendet werden. Die konkrete Einbindung von Analyse-Werkzeugen in den Werkzeugkasten des Paose-Ansatzes ist eine weiterführende Arbeit, die auch auf der Arbeit von Marquardt (geb. Lehmann) aufsetzen kann

([Lehmann 2003]). Die Analyse von Workflows gewinnt mit zunehmender Verwendung des in dieser Arbeit entwickelten Workflowmanagementsystems an Bedeutung. Solche weiterführenden Ansätze sind z.B. in der Arbeit von Kolja Markwardt zu einer verteilten Entwicklungsumgebung ([Markwardt et al. 2006]) enthalten. Diesbezügliche Ziele wurden in [Reese et al. 2008] und [Reese et al. 2006a] formuliert.

Für die Verwendung von Petrinetzen zur Workflowdefinition sind zwei Besonderheiten zu beachten, in denen Tasks in Workflows sich von dem üblichen Verhalten von Transitionen in Petrinetzen unterscheiden: erstens können Tasks in Workflows erfolglos abgebrochen werden und zweitens werden schon aufgrund der Aktivierung eines Tasks wesentliche und nach außen sichtbare Aktionen im WFMS durchgeführt (nämlich die Benachrichtigung der Benutzer über einen verfügbaren Task). Der erste Punkt wird durch die Einführung einer verfeinerten Task-Transition mit drei Transitionen `start`, `commit` und `rollback` geklärt. Diese Verfeinerung wird für den Prototypen übernommen und bildet einen wesentlichen Baustein der hier verwendeten Workflownetze. Der zweite Punkt wird durch den Listener-Mechanismus gelöst, welcher in dem WFMS von Thomas Jacob vorgestellt wird. Dazu werden die Subtransitionen einer Task-Transition mit Uplinks versehen, um die Kopplung zum Listener und zur Steuerung des WFMS zu realisieren. Dieser Listener wird für den Prototyp verwendet, allerdings in angepasster Form, um später dem WFE-Agenten die alleinige Kontrolle über den Informations- und Steuerungsfluss zum Workflownetz zu erlauben.

Der Bezug zwischen **Agenten und Workflows** wird anhand von diversen Literaturstellen hergestellt, insbesondere werden die Arbeitsgruppen um Dadam (mit ADEPT), Purvis (mit JBees) und Arbeiten von Buhler (funktional äquivalente Umsetzung eines WFMS mit Agenten) vorgestellt.

Die Unterscheidung zwischen *agentenbasierten* vs. agenten-*enhanced* WFM-Systemen nach Yan et al. wird für den weiteren Verlauf der Arbeit beibehalten. Insofern ist das im Prototyp realisierte WFMS in seiner lokalen Variante *agent-enhanced*, da der WFMS-Agent beim Start die anderen WFMS-Subagenten erzeugt und zu einem System verbindet. Selbst der realisierte Workflow-Agent gehört im lokalen Fall zu einem Agent-enhanced WFMS. Für den verteilten Fall bewirkt der Workflow-Agent die Verbindung zwischen mehreren WFM-Systemen, die auch erst zur Laufzeit gekoppelt werden. In diesem Falle bilden die beteiligten WFMS-Agenten zusammen ein einziges *agentenbasiertes* WFMS (sie existieren unabhängig voneinander). Im Abschnitt zur Stufe III in Kapitel 4 werden weitere Möglichkeiten der Verbindung von WFMS-Agenten aufgezeigt, die dann auch in einem agentenbasierten WFMS resultieren: Die Fragmentierung von Workflows, frei kooperierende WFMS-Agenten und hierarchisch organisierte WFMS-Agenten.

Aus der Arbeit von Buhler wird der Begriff der funktionalen Äquivalenz übernommen und bezeichnet ein System mit gleicher Funktionalität, aber erhöhter Laufzeitkomplexität. Die Vorteile liegen in mächtigeren Konzepten und in der mächtigeren Architektur, welche gegenüber dem Originalsystem eine zusätzliche Indirektionsschicht aus Agenten aufweist. Buhler setzt die so gewonnene Flexibilität nicht selbst in konkrete Verbesserungen um, sondern zeigt in einer Übersicht, wie im Allgemeinen die funktional äquivalenten Systeme auf dem Weg zur adaptiven Workflowausführung einzuordnen sind. In dieser Arbeit wird der Begriff der funktionalen Äquivalenz in

Bezug auf das WFMS von Thomas Jacob verwendet, allerdings in einer schwächeren Form als Buhler dies für die BPEL4WS-Workflows durchführt: Bei Buhler bleiben die Schnittstellen des WFMS zur Definition und Ausführung von Workflows tatsächlich gleich, während in dem hier realisierten WFMS bewusst eine geänderte Schnittstelle angestrebt wird (nämlich für Agenten und asynchrone Kommunikation im Gegensatz zu der vorher verwendeten rmi-basierten Kommunikation).

JBees ist von den verwendeten Technologien her eng verwandt, bis dahin, dass zu den Anfängen von JBees der hier verwendete Petrinetzsimulator Renew verwendet wurde ([Purvis et al. 2001, Purvis et al. 2000]). Purvis et al. zielen stärker als die vorliegende Arbeit auf die Simulation und Analyse von Wartezeiten und Skalierbarkeit von Workflows sowie auf die Adaptivität in der Ausführung. Purvis et al. haben nicht die Entwicklung einer Architektur für beliebige komplexe Anwendungen im Fokus, sondern solche Workflows, welche Teile von Geschäftsprozessen abbilden.

Mit ADEPT wird ein technologisch weniger verwandtes Projekt vorgestellt, in dem auch die Zielsetzung abweicht: Weder Agenten noch Petrinetze spielen eine herausragende Rolle in den Veröffentlichungen zu ADEPT, noch geht es um eine Architektur für Anwendungen. Mit ADEPT können Workflowanwendungen spezifiziert und ausgeführt werden. ADEPT ist aber in Benutzbarkeit und umgesetzter Adaptivität ein besonders fortgeschrittenes Projekt und dabei durch diverse Veröffentlichungen der Forschungsgemeinde zugänglich. Bei einer Weiterentwicklung meiner Arbeit können aus dem ADEPT-Projekt wichtige Anregungen und Erkenntnisse gezogen werden. Nicht zuletzt liefern Dadam et al. die auf ihrer Erfahrung beruhende Aussage, dass die Komplexität von umfangreichen WFM-Systemen und umfangreichen Datenbanksystemen vergleichbar sei. Dadurch wird deutlich, warum in einer Arbeit wie der vorliegenden kein vollständig adaptives WFMS und damit auch die Prozess-Infrastruktur nicht fertig entwickelt abgeliefert werden kann.

Das Kapitel wird dann im letzten Abschnitt über die verwandten Technologien zusammengefasst und abgerundet. Die WebServices sind sowohl im Agenten- als auch im Workflowumfeld eine zentrale Technologie, die breit verfügbar und bekannt ist. WebServices und Agenten können relativ leicht über Gateways miteinander verbunden werden, da die verwendeten Nachrichtenstrukturen sich in ihren Grundzügen aufeinander abbilden lassen. Für Capa wurde der WebService-Gateway-Agent 2005 im Rahmen der openNet-Demo vorgeführt ([Willmott et al. 2005]). Bei den Workflows werden WebServices für die flexible Abarbeitung von Tasks verwendet und insbesondere von der *WfMC* in den jährlich erscheinenden Workflow-Handbooks (z.B. [Wheeler und Bortolotty 2005, Bezancon 2005, Lienhard 2003]) betont. Die WebServices können allerdings in dieser Arbeit weder die Agenten noch die Workflows ersetzen, da sie die nötige konzeptionelle Prägnanz nicht aufweisen. Die WebServices sind allerdings für eine ausgereifte Umsetzung der Prozess-Infrastruktur auf der Basis der hier erarbeiteten Konzepte und Architektur durchaus in Betracht zu ziehen, gerade wegen der genannten Vorteile und weil der Technologiebruch zwischen Agenten und Workflows noch weiter verringert wird. Die Grid-Technologie dagegen verbindet Konzepte von verteilten Anwendungen mit dem Ziel der Gebrauchstauglichkeit und Skalierbarkeit. Auch hier muss für eine reifere Umsetzung das vorhandene Know-How berücksichtigt werden. Im Rahmen der vorliegenden Arbeit gehören diese Punkte deutlich in den Bereich des Ausblicks auf weiterführende Arbeiten.

Abschließend ist noch zu betonen, dass die Verwendung von Workflows für die prozessorientierte Strukturierung von Anwendungen neuerdings verstärkt vorgeschlagen wird. Ein prominentes Beispiel dafür ist das Paket „WF“ im .NET-Framework 3.0 von Microsoft und die zugehörige Philosophie. Westphal et al. fassen dieses sehr prägnant und treffend zusammen [Westphal und Weyer 2007, S. 51–56]: „Workflow ist überall.“

Kapitel 3

Das Kapitel **Grundlagen für den Entwurf der Prozess-Infrastruktur PIA** stellt die konzeptionellen Grundlagen zusammen, die für das Ziel dieser Arbeit notwendig sind und die ich im Rahmen dieser Arbeit erstellt oder an denen ich mitgewirkt habe.

Das Ergebnis des ersten Abschnitts ist die klare Fassung der **zentralen Begriffe** für diese Arbeit. Die Begriffsbestimmungen sind informell gehalten, da diese Arbeit nicht auf formale Beweise unter Bezug auf die definierten Eigenschaften der Begriffe abzielt. Die allgemeinen Begriffe wie System, Architektur, Rahmenwerk sind so bestimmt, dass sie frei kombinierbar sind. Dies bildet eine wesentliche Grundlage für die spätere Bestimmung der Gemeinsamkeiten zwischen Workflow- und Agententechnologie und damit für ihre Integration. Ein wesentliches Ergebnis dieses Abschnitts, wie auch der Arbeit als Ganzes, ist die Prägung des Begriffes *Prozess-Infrastruktur*. Dieser Begriff wurde in der obengenannten Veröffentlichung [Reese et al. 2007] zuerst eingeführt, wird jedoch in der vorliegenden Arbeit erheblich genauer ausgearbeitet. Als wesentliche Eigenschaften eines Prozesses im Allgemeinen werden dabei die Mehrschichtigkeit von Prozessen und die (implizit oder explizit) vorhandene Prozessdefinition genannt. Der Begriff Infrastruktur wird ebenfalls bezüglich der Schichten eines Systems bestimmt. Auf dieser Grundlage kann dann erfasst werden, was unter einer Prozess-Infrastruktur für Agentenanwendungen zu verstehen ist: eine verteilte Schicht in einer Architektur mit der Aufgabe, die Prozesse der darüberliegenden Schicht zu verwalten. Ein WFMS ist eine Prozess-Infrastruktur für eine Workflowanwendung. Die weiteren Begriffe: System, Managementsystem, Rahmenwerk, Laufzeitumgebung, Anwendung und Architektur werden sowohl auf Referenznetze als auch auf Agentenanwendungen einzeln angewendet, um erstens das Verständnis zu vertiefen und zweitens um die allgemeine Verwendbarkeit sicherzustellen. Die später in dieser Arbeit entwickelte Prozess-Infrastruktur für Agentenanwendungen (Pia) ist nach dieser Begriffsbestimmung ein Teil eines *Agenten-Managementsystems*; in der fünften Integrationsstufe wird später ein verallgemeinertes *Managementsystem für Einheiten* vorgestellt.

Im zweiten Abschnitt aus Kapitel 3 werden einige **zentrale Entwurfskonzepte in Bezug auf die technische Umsetzung** betrachtet. Dies geschieht bewusst an dieser prominenten Stelle des Textes, da die gewählte technische Umgebung einen erheblichen Einfluss auf die Art der Lösungsidee hat. Das Ergebnis dieses Abschnitts ist ein Überblick über spezielle Themen, die sonst unter dem Gewicht der eigentlich in dieser Arbeit entwickelten Konzepte zu weit zurückstehen würden: (1) Die Rolle der Petrinetze zusammengefasst, (2) die Notwendigkeit für geordnete agenteninterne Kommunikation, (3) die Verwendung von geschachtelten Strukturen, sowie die eher technischen Punkte der (4) Netzwerkanbindung, (5) die abonnierbaren Dienste und (6) die Beziehung zwischen Erzeuger und erzeugtem Agent, (7) die verschiedenen

Modellierungsschwerpunkte in der Prozessbeschreibung mit Interaktionsdiagrammen bzw. mit Workflows und schließlich (8) die Fragmentierung von Workflownetzen. Die einzelnen Punkte werden an geeigneter Stelle aufgegriffen und dort genutzt bzw. detailliert.

Die **Fragmentierung** erhält gleich im Anschluss eine ausführliche Darstellung: Die Vorarbeiten zur Darstellung und Realisierung von verteilten Workflows werden in einem eigenen Abschnitt dargestellt. Workflows werden dazu mit Referenznetzen dargestellt und nach einem Mechanismus mit Synchronisations-Stellen in Workflow-Fragmente geteilt. Die Fragmentierung wurde z.T. von Timo Carl implementiert [Carl 2004]. Die Ergebnisse sind in [Reese et al. 2005] veröffentlicht. Der Beitrag erschien überarbeitet und erweitert in [Reese et al. 2006c]. Die Workflow-Fragmente werden durch benutzerdefinierte Randstellen automatisch und konsistent erzeugt, so dass sie die gestellten Anforderungen an eine Fragmentierung erfüllen, nämlich eine frei gestaltbare Menge von unabhängigen Fragmenten ohne zusätzliche Semantik der Netzelemente. Für die weitere Arbeit sind damit die konzeptionellen und technischen Voraussetzungen für das verteilte Management[3] von Workflows erfüllt. Im Prototyp werden diese Ergebnisse nicht eingebaut. Die Integration der Workflow-Fragmentierung ist ein möglicher Punkt für die weiterführende Realisation. Dadurch würden auch solche unabhängigen Tasks, die von *verschiedenen* WFMS-Agenten verwaltet werden müssen, verteilt und nebenläufig ausgeführt werden können. Der mobile Workflow-Agent reist von WFMS zu WFMS und kann so nur innerhalb eines Ortes unabhängige Tasks auch nebenläufig ausführen. Die Workflowfragmente können hier also gewinnbringend integriert werden.

Der **Paose-Ansatz** ist der Software-Entwicklungsansatz, der im Kapitel 5 dieser Arbeit für die Entwicklung des Prototyps verwendet wird. Hier im Abschnitt über den Paose-Ansatz werden die wesentlichen Elemente des Ansatzes vorgestellt. Um mit Capa eine Anwendung zu implementieren, sind für jeden Agenten eine initiale Wissensbasis sowie Protokollnetze und Entscheidungskomponenten zu implementieren. Die Organsiation des Entwicklerteams ist der zweite wichtige Aspekt von Paose und schließlich werden die Diagrammtypen von Paose und die darin verwendeten Ausdrucksmittel vorgestellt.

Die Beispiele für die Diagrammtypen sind direkt aus der Entwicklung des Prototyps übernommen. Da die Diagramme eine wesentliche Rolle bei der Implementierung spielen, sind sie am Ende der Implementierung äußerst zahlreich und umfangreich und können nicht vollständig hier abgebildet werden. Vielmehr werden im Kapitel 5 wesentliche Inhalte zusammmengefasst und einige spezielle Inhalte ausführlich dargestellt.

Ein wichtiger Punkt in diesem Abschnitt ist die Ausgestaltung der Funktion und Eingliederung der Entscheidungskomponente für Capa-Agenten. Entscheidungskomponenten werden ähnlich wie Protokollnetze als Subnetze in einen Agenten eingebettet und werden bei Bedarf instanziiert. Während Protokollnetze jedoch grundsätzlich

[3] Hier ist der Unterschied zwischen verteilter Ausführung und verteiltem Management zu beachten: Die Task-Bearbeiter können bei den meisten WFM-Systemen verteilt sein, während das Management zunächst lokal auf einen Ort beschränkt ist und erst beim verteilten Workflowmanagement mehrere Orte involviert sind.

einen Bezug zu einer konkreten Interaktion haben, sind Entscheidungskomponenten auch für andere Zusammenhänge verwendbar. Aus einer Entscheidungskomponente können die gleichen agenteninternen Aktionen wie aus einem Protokollnetz heraus verwendet werden, ausgenommen den Empfang und Versand von Nachrichten. Entscheidungskomponenten werden z.B. für graphische Benutzungsoberflächen, für die Kapselung von Altsystemen oder für komplexe Wissensbasis-Funktionalitäten verwendet, für die die Schlüssel-Wert-Struktur der Wissensbasis nicht ausreicht.

Die Matrixorganisation des Entwicklerteams aus dem PAOSE-Ansatz (vorgestellt z.B. in [CABAC 2007]) spielt eine wichtige Rolle für das Verständnis des Ziels dieser Arbeit. Es wird in diesem Abschnitt gezeigt, wie im PAOSE-Ansatz die Prozess- und Strukturkomponenten weitgehend unabhängig voneinander entwickelt werden können und dass die Koordiniation an den definierten Schnittpunkten stattfindet.

Der Abschnitt zum Entwurfsansatz PAOSE bildet also einen Angelpunkt für drei thematische Fäden der vorliegenden Arbeit: Die Entscheidungskomponente für CAPA-Agenten, die Matrixorganisation des Entwicklerteams als Vorbild oder methodische Entsprechung für das Ziel der Arbeit (die Integration von Prozess- und Struktursicht) und nicht zuletzt das Vorgehen bei der Entwicklung des Prototyps. Die Darstellung im Text richtet sich teilweise nach der Veröffentlichung [CABAC et al. 2007a], worin die Anwendungsentwicklung nach dem PAOSE-Ansatz zusammengefasst wird. Eine Kurz-Zusammenfassung ist in [CABAC et al. 2007b] veröffentlicht worden und eine überarbeitete und erweiterte Version ist in [CABAC et al. 2008] präsentiert worden.

Ein konkreter **Prozess in MULAN** wird im nächsten Abschnitt an einem Beispiel in einem stark vereinfachten, aber ausführbaren MULAN vorgestellt. An diesem Beispiel wird der *Anwendungsprozess* einer Agenten-„Anwendung“ in Form eines Kausalnetzes angegeben. Dazu wird die im Abschnitt 2.1.6 vorgestellte Methode verwendet, mit der aus Referenznetzen ein Kausalnetz entwickelt werden kann. Die Netzdefinitionen von MULAN und von der Beispielanwendung werden hier zu einem Anwendungsprozess abgewickelt.

Das Kausalnetz ist im Bereich der Agentenorientierung eine eher unübliche Form der Prozessdarstellung auf dieser Detailebene und wird in PAOSE nicht verwendet. Die vorhandenen und genutzten Werkzeuge und Möglichkeiten zur Prozessdarstellung sind der *Sniffer*, der MULAN-*Viewer* und die grundlegenden Möglichkeiten, die RENEW als Ausführungsumgebung bietet. Keines der vorhandenen Werkzeuge und auch nicht die Darstellung als Kausalnetz erhält den Zusammenhang mehrerer Nachrichten zu einer Interaktion. Vielmehr wird in jeder Darstellungsform der Zusammenhang einer Strukturkomponente gewahrt: Im Kausalnetz sind dies die Bahnen (*Swimlanes*), die je ein beteiligtes Referenznetz repräsentieren; im Interaktionsdiagramm aus dem Sniffer sind dies die Rollenbezeichner, die je einen Agenten repräsentieren; im MULAN-Viewer sind die Informationen zum aktuellen Zustand von Plattformen und Agenten sichtbar; RENEW stellt den aktuellen Zustand für je eine Referenznetz-Instanz in einem eigenen Fenster dar. Insofern sind alle vorhandenen Prozessbeobachtungswerkzeuge *strukturorientiert*. Es fehlen Werkzeuge für eine *prozessorientierte* Prozessbeobachtung. Bevor diese Werkzeuge existieren können, muss die Information über den Prozesszusammenhang zur Laufzeit erhalten bleiben. Das Interaktionsprotokoll aus der Entwurfsphase muss deshalb selbst in eine strukturelle Komponente zur Laufzeit eingebunden werden.

Im nächsten Abschnitt werden einige Varianten vorgestellt, wie zu einem gegebenen Interaktionsprotokolldiagramm ein Workflownetz entworfen werden kann. Die Gestaltung des Workflownetzes ist stark von den konkreten Anforderungen abhängig, was anhand von vier möglichen Ausgestaltungen eines Workflows erläutert wird. Da Workflows einen anderen Modellierungsschwerpunkt haben als Interaktionsprotokolle, kann der resultierende Workflow die gleiche Semantik und Aussage haben wie das gegebene Interaktionsprotokoll, muss es aber nicht.

Damit sind in diesem Abschnitt die Problemstellung und der Lösungsansatz für die vorliegende Arbeit verdeutlicht worden: der Zusammenhalt von Nachrichten in Interaktionen soll erhalten bleiben und dies wird erreicht, indem ein Workflow-Netz instanziiert wird, welches der Semantik des Interaktionsprotokolls entspricht.

Die Prozess-Infrastruktur für Agentenanwendungen (Pia) und die Architektur dazu (Apia) sind die Form, in der die Prozess- und Struktursicht auf Agentenanwendungen zur Entwurfs- und zur Ausführungszeit realisiert werden sollen. In einem eigenen Abschnitt werden die Anforderungen an Pia und die Konzeption für Pia zusammengefasst. Der weitere Weg zu diesem Ziel wird mit fünf Stufen skizziert. Diese Stufen werden im nächsten Kapitel ausführlich behandelt.

Kapitel 4

Das Kapitel **WFMS und Agenten** bildet den Hauptteil der Arbeit mit den wichtigsten Ergebnissen. Hier wird die konzeptionelle Integration von Workflows und Agenten durchgeführt. Dabei können die Workflows als Stellvertreter für die Prozesssicht auf eine Anwendung gelten und die Agenten als Vertreter für die Struktursicht auf eine Anwendung. Das Ergebnis der Stufe I ist die Gegenüberstellung von Workflow- und Agententechnologie (Verhalten und Struktur) als Vordergrund bzw. Hintergrund bei der Software-Entwicklung. Die Stufe II beschreibt, wie sich diese beiden Sichtweisen ergänzen können. Die Stufe III stellt eine Architektur vor, in der sowohl ein WFMS als auch ein Agentenrahmenwerk zur Anwendungsentwicklung zur Verfügung stehen. Der Vorteil der vorgestellten Architektur ist, dass die Flexibilisierung der Workflowausführung leicht in die Architektur eingeordnet werden kann. Das Ergebnis der Stufe IV ist eine Architektur für die Prozess-Infrastruktur für Agentenanwendungen (Apia), wie es das Ziel dieser Arbeit ist. Diese Stufe IV umfasst aber viel mehr als das Ziel dieser Arbeit. Die hier vorgeschlagene Ausprägung für Pia ist nur eine von vielen möglichen Ausprägungen der Stufe IV. Die fünfte Stufe beschreibt eine Architektur für eine darüber hinausgehende Integration von Verhalten und Struktur, die es ermöglicht, die jeweils passende Perspektive auf eine Anwendung einzunehmen: je nach Fragestellung kann die Anwendung sowohl als prozessorientierte Anwendung als auch als strukturorientierte Anwendung betrachtet werden.

Am Ende von Kapitel 4 gibt es eine Diskussion mit Zusammenfassung (daher fällt die zusammenfassende Darstellung der fünf Stufen hier vergleichsweise kurz aus). Drei Diskussionspunkte werden behandelt: die Umsetzung der Architektur, der Bezug der Stufen zur Paose und zusammenfassend zum Abschluss von Kapitel 4 der Bezug jeder Stufe zur Pia, die im Folgenden zusammengefasst werden.

Zur effizienten Umsetzung und damit zur Benutzbarkeit der vorgestellten Architektur wird vorgeschlagen, die Architektur mit Hilfe von Schnittstellen (Interfaces) auf eine Realisierung abzubilden. Als Ergebnis soll deutlich werden, dass die Architektur

trotz der vielen Schichten und beteiligten Systeme auf eine effiziente Weise umgesetzt werden kann, was jedoch nicht im Mittelpunkt dieser Arbeit steht.

Der Abschnitt zu der Rolle der PAOSE stellt den PAOSE-Ansatz in eine langfristige Perspektive, bei der die Werkzeugunterstützung durch immer ausgereiftere Methoden und Organisationsformen stetig erweitert wird.

Der Bezug der vorgestellten Stufen zu dem Begriff der Prozess-Infrastruktur wird zum Abschluss von Kapitel 4 hergestellt. Auf Stufe I ist ein WFMS eine Prozess-Infrastruktur für Workflowanwendungen. Eine Prozess-Infrastruktur für Agentenanwendungen fehlt. Auf Stufe II (mit der prinzipiellen Möglichkeit eines workflowbasierten Agentenmanagementsystems) dient das integrierte WFMS als Prozess-Infrastruktur für die Agentenplattform, aber nicht für die Anwendungsagenten. Die Verfügbarkeit für die Anwendungsagenten wird in Stufe III erreicht. Dabei ist die Verwendung der Prozess-Infrastruktur (in Form eines WFMS *für* Agenten) optional bei der Anwendungsentwicklung, d.h. die Entscheidung über die Verwendung der Prozess-Infrastruktur für Agentenanwendungen liegt beim Anwendungsentwickler. Auf Stufe IV wird APIA vorgestellt, die Architektur einer Prozess-Infrastruktur für Agentenanwendungen. Über die Verwendung der Prozess-Infrastruktur entscheidet dann nicht mehr der Anwendungsentwickler, sondern sie ist fest im Rahmenwerk verankert. Als konkrete Ausgestaltung wird ein Agentenrahmenwerk beschrieben, welches Protokolldiagramme einliest und intern in Workflowdefinitionen umsetzt. Bei der Ausführung der so spezifizierten Agentenanwendung wird für jede Interaktion ein Workflow instanziiert, der eine Prozessunterstützung wie bei einem WFMS für die Agentenanwendung verfügbar macht. Auf Stufe V wird die Prozess-Infrastruktur auch für agenteninterne Prozesse integriert, indem Agent / Workflow-Einheiten benannt werden. Solche Einheiten sind rekursiv schachtelbar und können sowohl als Agent als auch als Workflow betrachtet werden.

KAPITEL 5

Im Kapitel zur Umsetzung des **Prototyps** wird mit dem PAOSE-Ansatz ein agentenbasiertes WFMS umgesetzt als Prototyp für die Stufe III. Ferner wird ein Workflow-Agent als ein wichtiger Schritt in Richtung eines Prototyps für Stufe IV umgesetzt. Einige ergänzende Entwicklungen werden zunächst dargestellt. Die Ergebnisse daraus sind für den Prototypen wichtig, können aber auch für sich genommen verwendet werden:

(1) Die Entscheidungskomponente für CAPA-Agenten (DC für *decision component*) ist inzwischen ein fester Bestandteil von PAOSE und wird im jährlichen Lehreprojekt eingesetzt. Neben den Protokollnetzen (Referenznetze zur Implementation von Verhalten in Agenteninteraktionen) gibt es nun im CAPA-Agenten die Entscheidungskomponenten (Referenznetze zur Implementation von agenteninternen Vorgängen). Entscheidungskomponenten werden für komplexe agenteninterne Handlungen verwendet: zur Kapselung von „quasi-internen“ Handlungen aus Legacy-Code und GUI-Anbindung und für solche internen Handlungen, die mehrere Interaktionen untereinander verknüpfen und in Beziehung setzen.

(2) Die Anbindung an openNet hat im Nachhinein einen eher indirekten Nutzen, da openNet nicht mehr betrieben wird. Der bleibende Nutzen ist, dass CAPA auf den

aktuellen Stand in der Kommunikation mit dem Agentenrahmenwerk Jade als defacto-Standard gebracht ist und damit zukünftige Verknüpfungen in offenen und heterogenen Agentenvernetzungen erleichtert sind. Konzeptionell ist mit der Anbindung an Agentcities und openNet gegeben, dass man bei der Entwicklung von Agentenanwendungen davon ausgehen kann, dass eine aktuelle und skalierende Sicht auf das Agentenumfeld zur Verfügung steht.

(3) Der zentrale Verzeichnisdienst (ZDF für *zentraler DF*) hingegen ist als maßgeschneiderte Variante der Plattformvernetzung von direkterem Nutzen. Das Prinzip basiert auf den Spezifikationen der FIPA zu förderierten DFs, so dass auch hier durch die konkrete Umsetzung die Standardkonformität von Capa verbessert wurde. Der ZDF ist konfigurierbar und wird immer dann eingesetzt, wenn eine Agentenanwendung verteilt ausgeführt werden soll. Der ZDF wurde erstmalig im Wintersemester 2004/2005 im Rahmen des Paose-Lehreprojektes für den Startmechanismus der verteilten Agentenanwendung Siedler eingesetzt.

(4) Die Erweiterung des Capa-DFs als Subscription Manager wird ebenfalls erfolgreich eingesetzt. Dadurch werden neben der normalen Suchanfrage an den DF drei weitere Funktionen zur Verfügung gestellt: die persistente Suche und die automatische Vermittlung von Abonnements für wahlweise einen beliebigen oder alle vorhandenen Anbieter des entsprechenden abonnierbaren Dienstes (Subscribe-One und Subscribe-All). Die persistente Suche wird genutzt, um von der Startreihenfolge unabhängig zu sein, wenn mehrere voneinander abhängige Agenten gestartet werden sollen. Die persistente Suche ist von der FIPA spezifiziert, also ist auch hiermit die Standardkonformität von Capa weiter verbessert worden. Die Funktion Subscribe-One und Subscribe-All werden erst dann breit genutzt, wenn Agenten abonnierbare Dienste anbieten. Die persistente Suche selbst ist das erste Beispiel für einen abonnierbaren Dienst, die Information über die aktuelle Liste von Workitems in einem WFMS ist ebenfalls ein abonnierbarer Dienst.

Jeder dieser Punkte bildet technisch eine wichtige Weiterentwicklung von Capa und in allen Punkten bis auf Punkt (1) wurde die Standardkonformität von Capa erweitert. Konzeptionell wurden diese Punkte bereits im Kapitel 3 eingeführt. Bei anderen Details aus dem Prototyp, wie z.B. beim Mechanismus für die Beziehung zwischen Erzeuger und erzeugtem Agenten wird der konzeptionelle Gehalt im Rahmen dieser Arbeit nur angedeutet. Erreicht wurde jeweils eine Lösung für ein konkretes Problem, z.B. das Problem der Initialisierung eines Capa-Agenten mit dynamischen Informationen.

Im Abschnitt zum **Workflowmanagementsystem** wird schließlich der Prototyp entsprechend dem Paose-Ansatz entwickelt und vorgestellt. Dazu werden die bis hierhin ausgearbeiteten Konzepte zu einem verteilten, agentenbasierten Workflowmanagementsystem verwendet. Die erste Umsetzung fand im Rahmen der Lehrveranstaltung „Agentenorientierte Software-Entwicklung“ im Wintersemester 2005 / 2006 statt. Daraus hervorgehende konkretisierte Konzepte wurden in [Reese et al. 2006b] dargestellt. Die Aufbereitung des agentenbasierten WFMS im Rahmen der Baccalaureatsarbeit von Thomas Wagner stellt nun die aktuell verfügbare Version für Weiterentwicklungen dar ([Wagner 2009]).

Das gesetzte Ziel wurde erreicht: Ein agentenbasiertes WFMS, welches für Agenten die Basisfunktionalitäten eines WFMS zur Verfügung stellt. Das WFMS kommuniziert mit ACL-Nachrichten, Workflows werden instanziiert und ausgeführt, die Workitems werden entsprechend der Rolle von Benutzern richtig zugeordnet und können angenommen, bearbeitet, abgebrochen und wieder zur Bearbeitung angenommen werden. Die Verwendung eines solchen WFMS wurde im Kapitel 4 zur Stufe III diskutiert, konkret wurde der Prototyp bereits für das Change Management verwendet ([Markwardt et al. 2006]).

Im Text sind einige Weiterentwicklungsmöglichkeiten angegeben, die in weiterführenden Arbeiten aufgegriffen werden können: Ein Distribution-Agent, der für die Verteilung von Workflow-Fragmenten zuständig ist, ein Remote-Agent, der mit einem benachbarten WFMS kommuniziert. Dieser Remote-Agent kann z.B. für die Umsetzung der hierarchischen WFMS-Schachtelung verwendet werden.

Im Prototyp wurde exemplarisch für die verschiedenen Möglichkeiten zur Erweiterung und Flexibilisierung des WFMS der **Workflow-Agent** ausformuliert und umgesetzt.

Der Workflow-Agent stellt in dem agentenbasierten WFMS entscheidende zusätzliche Flexibilität zur Verfügung, indem zwischen dem WFMS und der Ausführung eines Workflows eine zusätzliche Indirektion eingeführt wird. Gegenüber dem einfachen WFMS wird nun ein Workflow nicht mehr in Form einer Datenstruktur an die lokale Workflow-Engine (an den WFE-Agenten) zur Ausführung übergeben, sondern wird als Agent gekapselt. Im Workflow-Agenten fallen die Workflow- und die Agenten-Identität zusammen.

Der mobile Workflow-Agent verknüpft bei der Ausführung mehrere WFM-Systeme. Yan et al. unterscheiden zwischen *agent-enhanced* und *agent-based* anhand der Frage, ob die beteiligten Agenten eines WFM-Systems unabhängig voneinander existieren oder ob ein übergeordneter Agent die übrigen erzeugt, so dass diese abhängig von dem übergeordneten Agent existieren. Der Workflow-Agent ist abhängig von seinem Heimat-WFMS, ebenso wie die Teil-Agenten WFE, WFES u.s.w.. Ein zweites WFMS, welches für den mobilen Workflow-Agent Tasks ausführt, existiert dagegen unhabhängig von dem Heimat-WFMS des Workflow-Agenten. In diesem Sinne wird das hier entwickelte WFMS ein nach Yan et al. *agentenbasiertes* WFMS. Eine Umsetzung des nur im Konzept vorgestellten Agenten *Distribution* (für die Verteilung von Workflow-Fragment-Agenten auf mehrere passende WFM-Systeme) oder *Remote* (für die Kommunikation mehrerer WFMS-Agenten) hätte denselben Effekt: Das WFMS ist dann *agentenbasiert* statt *agent-enhanced*.

Durch den Workflow-Agenten kann ein Workflow instanziiert und teilweise ausgeführt werden, selbst wenn nicht alle Tasks im lokalen WFMS definiert sind.

Der Workflow-Agent erhält zur Erreichung seiner Aufgaben einen speziellen Lebenszyklus. Durch den aktuellen Zustand im Lebenszyklus sind die nächsten möglichen Handlungen des Agenten geregelt. Die fünf wichtigsten Zustände im Lebenszyklus des Workflow-Agenten sind *created*, *arrived*, *idle*, *workflow execution* und *done*. Dies sind die Zustände, wo der Workflow-Agent an keiner laufenden Interaktion beteiligt ist. Jede Interaktion hat Zwischenzustände. Die Interaktion zur Migration wird ausführ-

lich beschrieben, insbesondere muss der Workflow-Agent für die Migration in einen suspendierten Zwischenzustand wechseln.

Der Lebenszyklus ist in Form einer Entscheidungskomponente für den Workflow-Agenten implementiert. Hierin werden die proaktiven Handlungen des Agenten koordiniert, indem aufgrund der aktuellen Wissensbasis-Einträge die nächste Interaktion angestoßen wird.

Von den in Abschnitt 3.2 ausgearbeiteten Konzepten kommen mehrere beim Workflow-Agenten zum Einsatz: Petrinetze zur Implementation der Interaktionen und für die Workflow-Definition; die Entscheidungskomponente als wesentlicher Bestandteil des Lebenszyklus; Die Netzwerkanbindung (zentraler DF) für die Suche nach einem passenden WFMS für die nächsten Schritte in der Workflow-Ausführung; Die Beziehung zwischen Erzeuger und erzeugtem Agent wurde extra für diesen Fall notwendig (der Workflow-Agent des Prototyps wird vom WFES-Agenten erzeugt).

6.3 Anwendungsbeispiele

Nachdem im letzten Abschnitt diskutierend, integrierend und ausblickend alle Kapitel und Abschnitte der Arbeit aufgegriffen worden sind, wird im Folgenden den verschiedenen Entwicklungsstufen je ein Beispiel zugeordnet. Diese Beispiele dienen eher der Einordnung und als Anker für weiterführende Arbeiten, als zur detaillierten Veranschaulichung der geleisteten Arbeit. Eine umfassende Auswertung der Verbesserung durch die vorliegende Arbeit wird nicht gegeben. Der Grund dafür ist, dass das vorgeschlagene System einer gewissen Größe bedarf, um die Vorteile nutzen zu können und ein solches System ist im Rahmen dieser Arbeit nicht zu leisten.

6.3.1 Eine Workflow-Anwendung: Phoneshop

Ein Geschäft in dem Telefone bestellt werden können, ist die Beispielanwendung für das referenznetzbasierte Workflowmanagementsystem, das von Thomas Jacob im Rahmen einer Diplomarbeit vorgestellt wurde [Jacob 2002, S. 132ff]. In der Workflow-Anwendung Phoneshop werden Tasks zu drei anwendungsbezogenen Bereichen modelliert: Bestellung, Prüfung der Bezahl- und Adressdaten und Auslieferung. Diese Bereiche werden in insgesamt drei Workflow-Referenznetzen modelliert: ein übergreifender Workflow und zwei untergeordnete Workflows.

In der Ausführung können sich Angestellte des Phoneshops über eine RMI-Clientanwendung[4] mit ihrem PDA einloggen und werden dann über zutreffende Workitems und ihre laufenden Activities laufend informiert.

Ein Administrator kann über die Netzinstanz eines Workflows auf Prozessebene den Fortschritt der einzelnen Fälle beobachten und (eingeschränkt) auch steuern.

Über *Formular-Tasks* werden die nötigen Daten in strukturierter Weise erfasst, z.B. Adressdaten und Einzelheiten zur Bezahlweise und zur Auslieferung. *Manuelle Tasks* (z.B. das Verpacken eines Telefons zum Versand) werden bei der Fertigstellung vom

[4] RMI = Remote Method Invocation. Ein Kommunikationsprotokoll und eine Programmierschnittstelle der Programmiersprache Java [Network 2009].

Benutzer bestätigt. Für *automatisierte Tasks* wird eine externe Anwendung konfiguriert, so dass diese automatisch gestartet wird. Ein Beispiel hierfür ist das Versenden einer Bestätigungs-Email für eine Telefonbestellung. Wenn ein Benutzer ein Workitem für diesen Task anfordert, wird lokal auf seinem PDA oder Rechner ein Editor automatisch mit einem vorkonfigurierten Text gestartet. Die *automatischen Tasks* werden vollständig vom WFMS ausgeführt, ein Beispiel ist die Buchhaltung über alle Bestellungen und Kunden.

Für die anwendungsspezifischen Tasks, Benutzerrollen, Regeln und Formulare gibt es eine Datenbankschnittstelle, die entweder über property-Dateien oder über eine tatsächliche Datenbank bedient werden kann. Die Referenznetze, in denen die Präzedenzrelationen für die Workflows spezifiert sind, benutzen die Task-Transition, so dass die Benachrichtigung der Benutzer, Abbruch und reguläres Beenden von Activities möglich sind.

Bezogen auf diese Arbeit befindet sich Phoneshop auf Stufe I der im Kapitel 4 eingeführten Integrationsstufen, weil es sich um eine reine Workflowanwendung handelt. Die Prozesssicht steht beim Entwurf und bei der Ausführung im Mittelpunkt. Die Struktursicht wird lediglich in Form von konfigurierten Ressourcen in die Anwendung einbezogen und steht nicht im Vordergrund der Betrachtung. Für sehr große Anwendungen ist eine vordergründige Gestaltung der Struktursicht notwendig. Elaborierte Werkzeuge könnten die Modellierung einer Struktursicht unterstützen, was graduell zu Stufe II führen würde.

6.3.2 Eine Agenten-Anwendung: Siedler

Die Agentenanwendung Siedler basiert auf dem bekannten Brettspiel *Die Siedler von Catan* ([Teuber 2003]). Siedler ist ein rundenbasiertes Brettspiel mit reichen Interaktionsmöglichkeiten um vielfältige Ressourcen und dient als Beispiel für den Entwurf und die Implementierung einer komplexen Anwendung mit verteilten Prozessen mit Hilfe der Agententechnologie. Spieler wetteifern um Währungen (Holz, Wolle, Korn u.s.w.), um Bauplatz auf einer kleinen Insel und um Statusvorteile (z.B. die längste Handelsstraße zu besitzen).

Seit einigen Jahren wird Siedler wegen dem ausgewogenen Maß an Komplexität genutzt, um den Paose-Ansatz zu illustrieren ([Bosch et al. 2002], [Reese et al. 2003], [Cabac und Tell 2004], [Offermann et al. 2005], [Cabac et al. 2005a]). In Lehreprojekten wird Siedler entlang der Agentenrollen und entlang der Interaktionen neu entwickelt oder erweitert. Jedes Interaktionsprotokoll ist am Ende durch mehrere Protokollnetze implementiert (mindestens eines pro beteiligtem Agent), so dass der übergreifende Prozess und Entscheidungsfindung vollständig verteilt sind. Die Konzepte und der Paose-Ansatz sind sehr mächtig. Da die Interaktionsprotokolle jedoch nicht direkt ausgeführt werden, sind die agentenübergreifenden Prozesse zur Laufzeit im Sinne verteilter Workflows nur durch die Gesamtheit aller Agentenverhalten implizit modelliert.

Die Siedler-Anwendung besteht aus den administrativen Agenten, die das Spiel verwalten, und den Spieler-Agenten. Zu den administrativen Agenten gehören z.B. ein Insel-Agent und ein Bank-Agent. Für Spieler-Agenten gibt es verschiedene Im-

plementationen: verschiedene intelligente Agenten ([Meyer 2004], [Seegert 2004], [Brin 2008]) und GUI-Agenten für menschliche Mitspieler.

Durch eine Reihe von Interaktionsprotokollen ist z.B. der Rundenablauf oder das Zusammenspiel der administrativen Agenten beim Bauen oder Handeln der Spieler-Agenten beschrieben.

Eine reine Agentenanwendung befindet sich auf Stufe I der im Kapitel 4 eingeführten Integrationsstufen. Die Struktursicht steht durch die Agenten und Agentenrollen im Mittelpunkt der Anwendung. Für Siedler wurde jedoch mit Paose ein ausgeprägt prozessorientierter Entwicklungsansatz verwendet, deshalb kann es schon zu Stufe II gezählt werden. Zur Laufzeit geht der explizite Zusammenhalt mehrerer Nachrichten zu einer Interaktion verloren.

6.3.3 Eine Anwendung mit Workflow-Agent: Reiseagentur

Eine Agentur, die für Reisebuchungen mit Fluggesellschaften und Hotels kommuniziert, bildet den inhaltlichen Rahmen einer Beispielanwendung, welche das agentenbasierte Workflowmanagement nutzt. Ein Kunde fordert mit einer Reisebuchung einen komplexen Dienst an, woraufhin ein Workflow ausgeführt wird. Dieser Workflow enthält Teile, die nicht vor Ort in der Reiseagentur ausgeführt werden können. So wird ein Raum in einem Hotel gebucht und ein Flug bei einer Fluggesellschaft. Die Workflowinstanz ist also nicht vollständig spezifiziert und der Workflow-Agent sucht über den lokalen Verzeichnisdienst nach Workflowmanagementsystemen (WFMS), die Teile des Workflows ausführen können und migriert dort hin. Das jeweilige WFMS bietet die Fachkräfte und die Umgebungsdienste, die zur Ausführung eines Workflow-Teils nötig sind.

Normalerweise wird so ein Szenario durch mehrere voneinander abhängige Workflows realisiert: eine Reiseagentur kann nicht eigentlich einen Raum buchen, sondern kann ein Hotel beauftragen, einen Raum zu buchen oder zu reservieren. Der Workflow-Agent bietet nun die Möglichkeit, die Schnittstelle zwischen den verschiedenen Workflow-Umgebungen explizit durch einen einzigen Workflow darzustellen. Die Einzelheiten der Bearbeitung bleiben dabei im jeweiligen lokalen WFMS verborgen.

Die Reiseagentur als Beispielanwendung des in Kapitel 5 vorgestellten Workflowmanagementsystems zeigt prototypisch, dass die Konzepte für Multi-Agenten-Anwendungen mit starker Prozessorientierung und Konzepte von Workflowanwendungen kombiniert werden können. Allgemein lässt sich hier zusammenfassen, dass Multi-Agenten-Anwendungen wie Siedler größere Flexibilität ermöglichen, während Workflowanwendungen wie der Phoneshop eine durchgängige Modellierung der Prozesse ermöglichen. Die Reiseagentur bietet nun sowohl Klarheit durch explizite Darstellung übergreifender Prozesse als auch die Flexibilität einer Multi-Agenten-Anwendung.

Die Reiseagentur befindet sich auf Stufe III der in Kapitel 4 eingeführten Integrationsstufen, da in diesem Falle eine agentenorientierte Anwendung ein agentenbasiertes WFMS für einen ausgewählten Prozess einsetzt. Der Workflow-Agent und die noch nicht umgesetzten Konzepte für verteilte WFM-Systeme machen sich die Vorteile der agentenorientierten Entwicklung zunutze, um die Workflow-Ausführung mächtiger und flexibler zu gestalten. Alle diese Konzepte bleiben auf Stufe III. Erst, wenn

die Entwicklung von Werkzeugen zur systematischen Unterstützung beim Entwurf einer solchen gemischten Anwendung soweit fortgeschritten ist, dass die Entwickler innerhalb eines einzigen Paradigmas (Agenten oder Workflows) bleiben können, ist die Stufe IV erreicht.

6.3.4 Eine Anwendung für PIA: Software-Entwicklung

Die Vision einer integrierten Entwicklungsumgebung für kollaboratives, verteiltes Arbeiten an komplexen Softwareprojekten dient als Beispiel für eine Anwendung, welche die ebenfalls als Vision existierende Prozess-Infrastruktur benutzt. Im Kontext der verteilten Software-Entwicklung ist das Change Management von zentraler Bedeutung. In der zugehörigen Veröffentlichung [Markwardt et al. 2006] werden die wesentlichen Bestandteile beschrieben, mit denen mit Mitteln der prototypisch vorhandenen Stufe III, auf der Basis des WFMS, eine Anwendung für Change Management entwickelt werden kann. Die Prozess-Infrastruktur wie sie hier vorgestellt wird bietet die Möglichkeiten, die Anwendungsprozesse durch Softwareprozesse abzubilden und eine flexible, erweiterbare, adaptierbare, individuell konfiguierbare Anwendung in einer verteilten Umgebung aufzubauen.

Hier gilt es, die Prozesse auf allen Ebenen zu unterstützen, bis hin zu den Prozessen der Software-Entwicklung und auch die Prozesse der entwickelten Anwendung.

Eine solche verteilte Entwicklungsumgebung könnte Gegenstand der Entwicklung und Ausführung von Pia sein und befände sich dann auf Stufe IV der im Kapitel 4 vorgestellten Integrationsstufen.

Kapitel 7

Zusammenfassung und Ausblick

Die Arbeit hatte zum Ziel, die Prozesssicht in der agentenorientierten Software-Entwicklung besser zu unterstützen. Es wird dazu der Begriff einer Prozess-Infrastruktur geprägt und es wird die Architektur einer solchen Prozess-Infrastruktur für Agentenanwendungen (Apia) entwickelt.

Ausgangspunkt der Arbeit ist die Beobachtung, dass die Unterstützung der Struktursicht besonders gut durch die Agententechnologie realisiert ist, während die Unterstützung der Prozesssicht besonders gut durch die Workflowtechnologie realisiert ist. Beide Technologien werden mit Petrinetzen sowohl formal als auch in der Praxis bearbeitet und umgesetzt. In Paose werden die Prozesse einer agentenorientierten Anwendung als zusammenhängende Interaktionen entwickelt, zerfallen aber bei der Realisierung und Ausführung in Einzelteile, deren Zusammengehörigkeit nicht systematisch erkennbar ist.

Das Hauptergebnis dieser Arbeit ist die Architektur einer Prozess-Infrastruktur für Agentenanwendungen (Apia). Sowohl auf der konzeptionellen Ebene als auch auf der technischen Ebene wurden weitere Ergebnisse erarbeitet. Die Ausarbeitung von fünf Stufen zur Verbindung von Workflow- und Agententechnologie sind ein besonders zukunftsweisendes Ergebnis dieser Arbeit auf der konzeptionellen Ebene. Neben zahlreichen technischen Ergebnissen, die bereits verschiedentlich eingesetzt werden, stellt der Prototyp für ein agentenbasiertes Workflowmanagementsystem das zentrale technische Ergebnis dieser Arbeit dar.

Die Ergebnisse dieser Arbeit fügen sich in ein hochaktives Feld der Forschung und Entwicklung von Programmierkonzepten für verteilte, heterogene und interorganisationale Anwendungen. Die Weiterentwicklung solcher Programmierkonzepte ist bei der rasant wachsenden Vernetzung und einem wachsenden Anspruch an Interoperabilität von großem Allgemeininteresse. Der in dieser Arbeit ausgearbeitete Begriff einer Prozess-Infrastruktur für Agentenanwendungen kann die Lücke zwischen verteilter, strukturorientierter Entwicklung (im Sinne der Agentenorientierung) und zentraler Prozessverwaltung mit stabilen Konzepten überbrücken helfen. Es entsteht hier mit Pia ein Konzept der verteilten, autonomen und prozessorientierten Komponenten und der entsprechenden Sichten auf die Bestandteile einer Anwendung. Dieses Konzept findet in der Entstehung der sog. organsiationsorientierten Softwareentwicklung bereits Anwendung ([Wester-Ebbinghaus et al. 2007]) sowie in Arbeiten zur Unterstützung der verteilten Software-Entwicklung ([Cabac et al. 2008, Markwardt et al. 2009]).

Konzepte und vor allem konkrete Vorschläge für interorganisationale Workflows werden vielerorts benötigt. Um unternehmensübergreifende Workflows auf der Grundlage dieser Arbeit tatsächlich umzusetzen, reichen schon die Konzepte für die Stufe III der Integration, namentlich das agentenbasierte WFMS. Die dazu wesentlichen, hier ausgearbeiteten Konzepte betreffen die Verknüpfung von mehreren WFMS-Agenten:

einerseits zu hierarchischen Strukturen, andererseits zu föderierten, gleichberechtigten Systemen. Das Deployment von verteilten Anwendungen kann mithilfe der Ergebnisse zum Workflow-Agenten und zur Workflow-Fragmentierung umgesetzt werden. Die hier ausgearbeiteten Konzepte können mit der entsprechenden Finanzierung unternehmerisch umgesetzt werden und in ein marktreifes Produkt umgesetzt werden, um die Prozesssicht und Struktursicht in Unternehmens-Anwendungen wirksam zu unterstützen. Dies gilt gleichermaßen für das Agenten-Umfeld wie für verwandte Technologien wie WebServices und Grid.

Pia ermöglicht die Beobachtung und Steuerung von verteilten Prozessen. Pia ist im Hinblick auf die prozessorientierte *Entwicklung* von Anwendungen konzipiert, d.h. die erwünschten Prozesse werden möglichst vollständig spezifiziert und umgesetzt. In der Agententechnologie gibt es einen zweiten wichtigen Ansatz: das emergierende Verhalten von (ggf. intelligenten) Agenten in einer vorgegebenen Umgebung. Im ersteren Fall werden die *erwünschten* Prozesse spezifiziert und umgesetzt, im zweiten Fall müssen die *unerwünschten* Prozesse und Effekte wirksam ausgeschlossen werden. Pia dient zur Beobachtung und Steuerung von Prozessen und Strukturkomponenten. Diese Funktionen können zur Kontrolle und zur Durchsetzung von Richtlinien, Regeln und Gesetzen verwendet werden. In einer weiterführenden Arbeit kann der Einsatz von Pia als Überwachungsort für dynamische Komposition ausgearbeitet werden. Für wen allerdings solche Regeln von Vorteil sind, das kann bei der Entwicklung der Grundlagen nicht festgelegt werden. Insofern ist jede Entwicklung, insbesondere aber in der Informatik, ein zweischneidiges Schwert und die Verwendung kann nicht vorher festgelegt werden. Das gilt auch für diese Arbeit: Die Einsatzmöglichkeiten von Pia und Apia sind sehr vielfältig.

Wie eingangs in dieser Arbeit erwähnt, sind für die Entwicklung und den Betrieb von großen, verteilten, heterogenen Anwendungen mächtige Prinzipien zur Strukturierung notwendig. Insbesondere stellt sich die Frage nach der Beobachtbarkeit und Steuerbarkeit von Abläufen in Systemen mit autonomen Komponenten. In dieser Arbeit werden zwei allgemeine Prinzipien zur Strukturierung von komplexen Anwendungen betrachtet: Die Strukturgliederung am Beispiel der Agenten und die Vorgangsgliederung in Form von Prozessen. Die Prinzipien werden konzeptionell kombiniert, um Antworten zu erarbeiten. Die ausgearbeiteten Konzepte können unter Ausdifferenzierung im jeweiligen technologischen Kontext, insbesondere in den hier aufgegriffenen Bereichen WebServices und Grid, auf beliebige verteilte Systeme angewendet werden.

Literaturverzeichnis

[Aalst 1996] Aalst, Wil van der (1996). *Three Good Reasons for Using a Petri-net-based Workflow Management System*. In: Navathe, S. und T. Wakayama, Hrsg.: *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, S. 179–201.

[Aalst 2003] Aalst, Wil van der (2003). *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management.*. In: Desel, Jörg, W. Reisig und G. Rozenberg, Hrsg.: *Lectures on Concurrency and Petri Nets*, Bd. 3098 d. Reihe *Lecture Notes in Computer Science*, S. 1–65, Berlin. Springer-Verlag.

[Aalst 2008] Aalst, Wil van der (2008). *Woflan – The Woflan Website*. `http://is.tm.tue.nl/research/woflan/`. Aufgerufen am 19.10.2008.

[Aalst und Hofstede 2005] Aalst, Wil van der und A. t. Hofstede (2005). *YAWL: Yet Another Workflow Language*. Information Systems, 30(4):245–275.

[Aalst et al. 2003] Aalst, Wil van der, A. t. Hofstede, B. Kiepuszewski und A. P. Barros (2003). *Workflow Patterns*. Distributed and Parallel Databases, 14(3):5–51.

[Aalst et al. 1999] Aalst, Wil van der, D. Moldt, R. Valk und F. Wienberg (1999). *Enacting Interorganizational Workflows Using Nets in Nets*. In: Becker, Jörg, M. Mühlen und M. Rosemann, Hrsg.: *Proceedings of the 1999 Workflow Management Conference Workflow-based Applications, Münster, Nov. 9th 1999*, Working Paper Series of the Department of Information Systems, S. 117–136, University of Münster, Department of Information Systems, Steinfurter Str. 109, 48149 Münster. Working Paper No. 70.

[ac-net 2003] ac-net (2003). *Agentcities.NET, EU-Projekt*. Repräsentiert im Internet unter `http://www.agentcities.org/EUNET/`.

[ac-org 2003] ac-org (2003). *Agentcities (Organisation)*. Repräsentiert im Internet unter `http://www.agentcities.org/`.

[ac-rtd 2003] ac-rtd (2003). *Agentcities.RTD, EU-Projekt*. Repräsentiert im Internet unter `http://www.agentcities.org/EURTD/`.

[Alty et al. 1994] Alty, J. A., D. Griffiths, N. R. Jennings, E. H. Mamdani, A. Struthers und M. E. Wiegand (1994). *ADEPT - Advanced Decision Environment for Process Tasks: Overview & Architecture*. In: *Proc. BCS Expert Systems Conference (Applications Track, ISIP Theme), Cambridge, UK*, S. 359–371.

[Ambler 2004] Ambler, Scott W. (2004). *The object primer: Agile model-driven development with UML 2.0*. Cambridge Univ. Press, 3. Aufl.

[AristaFlow 2004–2007] AristaFlow (2004–2007). *AristaFlow Project – Next Generation Enterprise Process Management: Component-oriented Development of Adaptive Process-oriented Enterprise Software*. http://www.aristaflow.de/index.php. Zuletzt abgerufen am 22.10.2008.

[ASG 2004–2007] ASG (2004–2007). *ASG – Adaptive Services Grid*. http://asg-platform.org.

[Bellifemine et al. 2009] Bellifemine, Fabio, G. Rimassa, A. Poggi, T. Trucco, G. Caire, E. Cortese, F. Quarta und G. Vitaglione (2009). *Java Agent DEvelopment Framework. Homepage*. `http://jade.tilab.com/`. Zuletzt besucht im Juli 2009.

[Bergenti et al. 2003] Bergenti, Federico, O. Shehory und A. Sturm (2003). *Agent-Oriented Software Engineering*. In: Luck, Michael, W. van der Hoek und C. Sierra, Hrsg.: *AgentLink easss 2003*, S. 125–158. AgentLink.

[Bezancon 2005] Bezancon, Arnaud (2005). *Workflow and Service-Oriented Architecture (SOA)*. In: [WfMC 2005], S. 179–184.

[Blake 2002] Blake, M.B. (2002). *An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services*. In: *Proceedings of the 11th IEEE WETICE 2002, Pittsburgh, PA, June 2002*, S. 176–182. IEEE Computer Society Press.

[Böhme und Saar 2005] Böhme, Harald und A. Saar (2005). *Integration of heterogenous services in the Adaptive Services Grid*. In: *GSEM 2005: Erfurt, Germany, Proceedings*, Lecture Notes in Informatics, S. 220–232.

[Bosch et al. 2002] Bosch, Tobias, O. Gries, H. Kausch, M. Klenski, K. Lehmann, M. Morales, V. Seegert und A. Vilner (2002). *Agentenorientierte Implementierung des Spiels "Die Siedler von Catan"*. Projektbericht, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Braubach 2007] Braubach, Lars (2007). *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*. Doktorarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Brin 2008] Brin, Eugene (2008). *Das Belief Desire Intention-Modell und dessen prototypische Verwendung im Rahmen des Siedler V-Projektes*. Baccalaureatsarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Buhler und Vidal 2005] Buhler, Paul und J. M. Vidal (2005). *Towards Adaptive Workflow Enactment Using Multiagent Systems*. Information Technology and Management Journal, 6(1):61–87.

[Buhler 2004] Buhler, Paul Allen (2004). *A Software Architecture for Distributed Workflow Enactment with Agents and Web Services*. Doktorarbeit, Department of Computer Science and Engineering, College of Engineering and Information Technology, University of South Carolina.

[Cabac 2007] Cabac, Lawrence (2007). *Multi-Agent System: A Guiding Metaphor for the Organization of Software Development Projects*. In: Petta, Paolo, Hrsg.: *Proceedings of the Fifth German Conference on Multiagent System Technologies*, Bd. 4687 d. Reihe *Lecture Notes in Computer Science*, S. 1–12, Leipzig, Germany. Springer-Verlag.

[Cabac et al. 2008] Cabac, Lawrence, T. Dörges, M. Duvigneau, D. Moldt, C. Reese und M. Wester-Ebbinghaus (2008). *Agent Models for Concurrent Software Systems*. In: Bergmann, Ralph und G. Lindemann, Hrsg.: *Proceedings of the Sixth German Conference on Multiagent System Technologies, MATES'08*, Bd. 5244 d. Reihe *Lecture Notes in Artificial Intelligence*, S. 37–48, Berlin. Springer-Verlag.

[Cabac et al. 2007a] Cabac, Lawrence, T. Dörges, M. Duvigneau, C. Reese und M. Wester-Ebbinghaus (2007a). *Application Development with Mulan*. In: [Moldt et al. 2007], S. 145–159.

[Cabac et al. 2005a] Cabac, Lawrence, M. Duvigneau, M. Köhler, K. Lehmann, D. Moldt, S. Offermann, J. Ortmann, C. Reese, H. Rölke und V. Tell (2005a). *PAOSE Settler Demo*. In: *First Workshop on High-Level Petri Nets and Distributed Systems (PNDS) 2005*, Vogt-Kölln Str. 30, 22527 Hamburg. Universität Hamburg, Fachbereich Informatik.

[Cabac et al. 2005b] Cabac, Lawrence, M. Duvigneau, D. Moldt und H. Rölke (2005b). *Agent Technologies for Plug-in System Architecture Design*. In: *Proceedings of the Workshop on Agent-oriented Software Engineering (AOSE)*, Utrecht, The Netherlands.

[Cabac et al. 2007b] Cabac, Lawrence, M. Duvigneau, C. Reese, T. Dörges und M. Wester-Ebbinghaus (2007b). *Models and Tools for Mulan Applications*. In: Burkhard, H.-D., G. Lindemann, R. Verbrugge und L. Varga, Hrsg.: *Multi-Agent Systems and Applications V. Fifth International Central and East European Conference, CEEMAS'07, Leipzig. Proceedings*, Bd. 4696 d. Reihe *Lecture Notes in Computer Science*, S. 328–330, Berlin. Springer-Verlag.

[Cabac et al. 2003] Cabac, Lawrence, D. Moldt und H. Rölke (2003). *A Proposal for Structuring Petri Net-Based Agent Interaction Protocols*. In: Aalst, Wil van der und E. Best, Hrsg.: *24nd ICATPN 2003, Eindhoven, NL*, Bd. 2679, S. 102–120, Berlin. Springer-Verlag.

[Cabac und Tell 2004] Cabac, Lawrence und V. Tell (2004). *Demo of an Agent-Application: Settlers of Catan*. In: Moldt, Daniel, Hrsg.: *International Workshop on Modelling with Objects, Components, and Agents (MOCA 2004)*. Computer Science Department, Aarhus University.

[Carl 2004] Carl, Timo (2004). *Entwicklung eines agentenbasierten verteilten Workflow-Management-Systems mit Referenznetzen*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Constantinescu et al. 2003] Constantinescu, Ion, S. Willmott und J. Dale (2003). *D2.3: Agentcities Network Architecture*. Agentcities.RTD.

[Cosa 2008] Cosa (2008). *Die COSA BPM Suite*. `http://www.cosa.de`. Abgerufen am 19.10.2008.

[CrossFlow 1998–2000] CrossFlow (1998–2000). *CrossFlow: Cross-Organizational Workflow Support in Virtual Enterprises ESPRIT Project 28635*. `http://www.crossflow.org/flyer.html`. Visited: 2.1.2006.

[Dadam et al. 2007] Dadam, P., M. Reichert, S. Rinderle, M. Jurisch, H. Acker, K. Göser, U. Kreher und M. Lauer (2007). *ADEPT2 - Next Generation Process Management Technology*. Technischer Bericht, Heidelberger Innovationsforum, Heidelberg.

[Dadam und Reichert 2004] Dadam, Peter und M. Reichert (2004). *ADEPT – Prozess-Management-Technologie der nächsten Generation*. In: Spath, D. und K. Haasis, Hrsg.: *Aktuelle Trends in der Softwareforschung, Tagungsband zum doIT Software-Forschungstag 2003*. IRB Verlag Stuttgart.

[Dadam et al. 2009] Dadam, Peter, M. Reichert, S. Rinderle-Ma, K. Göser, U. Kreher und M. Jurisch (2009). *Von ADEPT zur AristaFlow BPM Suite – Eine Vision wird Realität*. EMISA Forum, 29(1):9–28.

[Dale et al. 2003] Dale, Jonathan, A. Hajnal und Others (2003). *Integrating Web Services into Agentcities Recommendation*. Agentcities Task Force.

[Dumas et al. 2005] Dumas, Marlon, W. M. van der Aalst und A. H. ter Hofstede (2005). *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley.

[Duvigneau 2002] Duvigneau, Michael (2002). *Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Duvigneau et al. 2002] Duvigneau, Michael, O. Kummer und F. Wienberg (2002). *Renew - User Guide*. Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, 1.6 Aufl.

[Duvigneau et al. 2003] Duvigneau, Michael, D. Moldt und H. Rölke (2003). *Concurrent Architecture for a Multi-agent Platform*. In: Giunchiglia, Fausto, J. Odell und G. Weiss, Hrsg.: *Agent-Oriented Software Engineering III. Third International Workshop, AOSE 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, Bd. 2585 d. Reihe *Lecture Notes in Computer Science*, S. 59–72, Berlin. Springer-Verlag.

[FIPA 2005] FIPA (2005). *FIPA: Foundation for Intelligent Physical Agents. Homepage*. `http://www.fipa.org`.

[FIPA 08 2003] FIPA 08 (2003). *FIPA SL Content Language Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00008/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 23 2005] FIPA 23 (2005). *FIPA Agent Management Specification*. Foundation for Intelligent Physical Agents. `http://www.fipa.org/specs/fipa00023/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 26 2003] FIPA 26 (2003). *FIPA Request Interaction Protocol Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00026/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 37 2003] FIPA 37 (2003). *FIPA Communicative Act Library Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00037/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 61 2003] FIPA 61 (2003). *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00061/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 67 2003] FIPA 67 (2003). *FIPA Agent Message Transport Service Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00067/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 70 2003] FIPA 70 (2003). *FIPA ACL Representation in String Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00070/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 71 2003] FIPA 71 (2003). *FIPA ACL Message Representation in XML Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00071/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 75 2003] FIPA 75 (2003). *FIPA Agent Message Transport Protocol for IIOP Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00075/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 84 2003] FIPA 84 (2003). *FIPA Agent Message Transport Protocol for HTTP Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00084/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 85 2003] FIPA 85 (2003). *FIPA Agent Message Transport Envelope Representation in XML Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00085/`. Zuletzt besucht am 6. Juli 2009.

[FIPA 93 2003] FIPA 93 (2003). *FIPA Messaging Interoperability Service Specification*. Foundation for Intelligent Physical Agents. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00093/`. Zuletzt besucht am 6. Juli 2009.

[Fleurke et al. 2003] Fleurke, Martin, L. Ehrler und M. Purvis (2003). *JBees – an adaptive and distributed framework for workflow systems*. In: Ghorbani, Ali und S. Marsh, Hrsg.: *Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA)*, S. 69–76. National Research Council Canada, Institute for Information Technology.

[Foster et al. 2001] Foster, Ian, C. Kesselman und S. Tuecke (2001). *The Anatomy of the Grid – Enabling Scalable Virtual Organizations*. Intl J. Supercomputer Applications.

[Frambach 1994] Frambach, Ludwig (1994). *Identität und Befreiung in Gestalttherapie, Zen und christliche Spiritualität*. Via Nova, Petersberg.

[Girault und Valk 2003] Girault, Claude und R. Valk (2003). *Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications*. Springer-Verlag, Berlin.

[Grefen 2002] Grefen, P. (2002). *A Taxonomy for Transactional Workflows*. Bd. 02 d. Reihe *CTIT technical reports series*, S. 1–20. University of Twente.

[Grefen et al. 2002] Grefen, P., K. Aberer und Y. Hoffner (2002). *CrossFlow: Cross organizational workflow management in dynamic virtual enterprises*. International Journal of Computer Systems Science & Engineering, 15:277–290.

[Hinz et al. 2005] Hinz, Sebastian, K. Schmidt und C. Stahl (2005). *Transforming BPEL to Petri Nets*. In: Aalst, Wil M. P. van der, B. Benatallah, F. Casati und F. Curbera, Hrsg.: *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, Bd. 3649 d. Reihe *Lecture Notes in Computer Science*, S. 220–235, Berlin. Springer-Verlag.

[Husemann et al. 2007] Husemann, Martin, M. von Riegen und N. Ritter (2007). *Transaktionale Kontrolle dynamischer Prozesse in serviceorientierten Umgebungen*. Datenbank-Spektrum, (20):6–14.

[Jacob 2002] Jacob, Thomas (2002). *Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Jacob et al. 2002] Jacob, Thomas, O. Kummer, D. Moldt und U. Ultes-Nitsche (2002). *Implementation of Workflow Systems using Reference Nets – Security and Operability Aspects*. In: Jensen, Kurt, Hrsg.: *Proc. of CPN*. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.

[Jadex 2009] Jadex (2009). *Jadex – BDI Agent System*. `http://jadex.informatik.uni-hamburg.de/bin/view/About/Overview`. Zuletzt besucht im Juli 2009.

[Jennings 2003] Jennings, Nick (2003). *Applying Agent Technology*. In: Luck, Michael, W. van der Hoek und C. Sierra, Hrsg.: *AgentLink easss 2003*, S. 263–280. AgentLink.

[Jennings 2000] Jennings, Nick R. (2000). *On Agent-Based Software Engineering*. Artificial Intelligence, 117(2):277–296.

[Jennings und Wooldridge 1998] Jennings, Nick R. und M. J. Wooldridge (1998). *Applications of Intelligent Agents*. In: Jennings, Nick R. und M. J. Wooldridge, Hrsg.: *Agent Technology: Foundations, Applications and Markets*, S. 3–28, Berlin. Springer-Verlag.

[Jensen 1992] Jensen, Kurt (1992). *Coloured Petri Nets: Volume 1; Basic Concepts, Analysis Methods and Practical Use*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin.

[Jessen und Valk 1987] Jessen, Eike und R. Valk (1987). *Rechensysteme. Grundlagen der Modellbildung*. Springer-Verlag, Berlin.

[Keller et al. 1992] Keller, G., M. Nüttgens und A.-W. Scheer (1992). *Semantische Prozeßmodellierung auf der Grundlage 'Ereignisgesteuerter Prozeßketten (EPK)'*. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken.

[Klusch 2003] Klusch, Matthias (2003). *Intelligent Information Agents*. In: Luck, Michael, W. van der Hoek und C. Sierra, Hrsg.: *AgentLink* easss *2003*, S. 51–88. AgentLink.

[Köhler und Wester-Ebbinghaus 2007] Köhler, Michael und M. Wester-Ebbinghaus (2007). *Petri Net-Based Specification and Deployment of Organizational Models*. In: [Moldt et al. 2007], S. 67–81.

[Kummer 2002] Kummer, Olaf (2002). *Referenznetze*. Logos, Berlin.

[Kummer et al. 2008] Kummer, Olaf, F. Wienberg und M. Duvigneau (2008). *Renew – The Reference Net Workshop*. Available at: http://www.renew.de/. Release 2.1.1. Renew is a Java-based high-level Petri net simulator and editor that provides a flexible modelling approach based on reference nets.

[Laak 1999] Laak, Dirk van (1999). *Der Begriff „Infrastruktur“ und was er vor seiner Erfindung besagte*. Archiv für Begriffsgeschichte, 41:280–299.

[Laka 2007] Laka, Wojciech (2007). *Ausbau einer Infrastruktur für offene agentenorientierte Anwendungen im Kontext von CAPA und OpenNet*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Langer 2004] Langer, Roman (2004). *Anerkennung und Vermögen. Eine sozialtheoretische Analyse von Selbstorganisationsprozessen in öffentlich-rechtlichen Bildungsinstitutionen*. Dissertation, Universität Hamburg.

[Laue und Liedtke 2000] Laue, Annette und M. Liedtke (2000). *Zustands- und prozeßorientierte Modellierung im Rahmen der Systemspezifikation*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Lehmann 2003] Lehmann, Kolja (2003). *Analyse und Bewertung von Agentenprotokollen auf Basis von Petrinetzen*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Leymann und Roller 2000] Leymann, Frank und D. Roller (2000). *Production workflow: concepts and techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Lienhard 2003] Lienhard, Heinz (2003). *Web Services and Workflow – a Unified Approach*. In: Fischer, Layna, Hrsg.: *Workflow Handbook 2003*, S. 49–60.

[Lilienthal 2007] Lilienthal, Carola (2007). *Komplexität von Softwarearchitekturen – Stile und Strategien*. Doktorarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Lohmann 2008] Lohmann, Niels (2008). *A Feature-Complete Petri Net Semantics for WS-BPEL 2.0*. In: Dumas, Marlon und R. Heckel, Hrsg.: *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007, Proceedings*, Bd. 4937 d. Reihe *Lecture Notes in Computer Science*, S. 77–91. Springer-Verlag.

[Lohmann et al. 2009] Lohmann, Niels, E. Verbeek, C. Ouyang und C. Stahl (2009). *Comparing and Evaluating Petri Net Semantics for BPEL*. Int. J. of Business Process Integration and Management, 4(1):60–73.

[Luck et al. 2003a] Luck, Michael, W. van der Hoek und C. Sierra, Hrsg. (2003a). *5th European Agent Systems Spring School (easss 2003)*, Barcelona, Spanien. AgentLink.

[Luck et al. 2003b] Luck, Michael, P. McBurney und C. Preist (2003b). *Agent Technology: Enabling Next Generation Computing. A Roadmap for Agent Based Computing*. AgentLink. Verfügbar im Internet unter `http://www.agentlink.org/roadmap`.

[Markwardt et al. 2009] Markwardt, Kolja, L. Cabac und C. Reese (2009). *A Process-Oriented Tool-Platform for Distributed Development*. In: *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*.

[Markwardt et al. 2006] Markwardt, Kolja, D. Moldt, S. Offermann und C. Reese (2006). *Using Multi-Agent Systems for Change Management Processes in the Context of Distributed Software Development Processes*. In: Sadiq, Shazia, M. Reichert und K. Schulz, Hrsg.: *The 1st International Workshop on Technologies for Collaborative Business Process Management (TCoB 2006)*, S. 56–66.

[MBala et al. 2005] MBala, Aloys, L. Padgham und M. Winikoff (2005). *Design Options for Subscription Managers*. In: Henderson-Sellers, Brian und M. Winikoff, Hrsg.: *Workshop-Proceedings AOIS at AAMAS 2005*, S. 26–33.

[Meyer 2004] Meyer, Simon (2004). *Planung im Siedlerprojekt*. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Milner et al. 1992] Milner, Robin, J. Parrow und D. Walker (1992). *A calculus of mobile processes, parts 1-2*. Information and computation, 100(1):1–77.

[Moldt 1996] Moldt, Daniel (1996). *Höhere Petrinetze als Grundlage für Systemspezifikationen*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Moldt 2005] Moldt, Daniel (2005). *Petrinetze als Denkzeug*. In: Farwer, Berndt und D. Moldt, Hrsg.: *Object Petri Nets, Processes, and Object Calculi; Report FBI-HH-B-265/05*, S. 51–66. Universität Hamburg, Fachbereich Informatik.

[Moldt et al. 2007] Moldt, Daniel, F. Kordon, K. van Hee, J.-M. Colom und R. Bastide, Hrsg. (2007). *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07)*, Siedlce, Poland. Akademia Podlaska.

[Moldt et al. 2005] Moldt, Daniel, S. Offermann und J. Ortmann (2005). *A Petri Net-Based Architekture for Web Services*. In: Cavedon, Lawrence, R. Kowalczyk, Z. Maamar, D. Martin und I. Müller, Hrsg.: *Workshop on Service-Oriented Computing and Agent-Based Engineering, SOCABE 2005, Utrecht, The Netherlands, July 26, 2005. Proceedings*, S. 33–40.

[Network 2009] Network, Sun Developer (2009). *Remote Method Invocation Home*. `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`. Zuletzt besucht am 5. März 2009.

[Nguyen et al. 2002] Nguyen, Giang T., T. Dang, L. Hluchy, M. Laclavik, Z. Balogh und I. Budinska (2002). *Agent Platform Evaluation and Comparison*. Technical report, II-SAS, Pellucid EU 5FP IST-2001-34519 RTD.

[OASIS 2009] OASIS (2009). *OASIS Web Services Business Process Execution Language (WSBPEL) TC*. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`. Abgerufen am 6.5.2009.

[Object Management Group (OMG) 2008] Object Management Group (OMG) (2008). *Object Management Group / Business Process Management Initiative*. `http://www.bpmn.org`. Abgerufen am 19.10.2008.

[Odell et al. 2000] Odell, James, H. van D. Parunak und B. Bauer (2000). *Extending UML for Agents*. In: Wagner, Gerd, Y. Lesperance und E. Yu, Hrsg.: *Agent-Oriented Information Systems. Workshop at the 17th National Conference on Artificial Intelligence (AAAI), AOIS 2000*, S. 3–17.

[Oesterreich 2005] Oesterreich, Bernd (2005). *Analyse und Design mit UML 2. Objektorientierte Softwareentwicklung*. Oldenbourg Verlag, 7. Aufl.

[Offermann 2003] Offermann, Sven (2003). *Ein Referenz-Netz-basiertes Modell zur dynamischen Komposition von Web Services*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Offermann et al. 2005] Offermann, Sven, J. Ortmann und C. Reese (2005). *Agent Based Settler Game*. In: [Willmott et al. 2005], S. 129–130. Part of NETDEMO, demonstration at international conference on Autonomous Agents and Multi Agent Systems, AAMAS-2005.

[OMG 2003] OMG (2003). *Common Object Request Broker Architecture (CORBA)*. Object Management Group (OMG). Verfügbar im Internet unter `http://www.omg.org/technology/documents/corba_spec_catalog.htm` Zuletzt besucht: Apr. 2009.

[Pokahr 2007] Pokahr, Alexander (2007). *Programmiersprachen und Werkzeuge zur Entwicklung verteilter agentenorientierter Softwaresysteme*. Doktorarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Purvis et al. 2002a] Purvis, Martin, S. Cranefield, M. Nowostawski, R. Ward, D. Carter und M. A. de Oliveira (2002a). *Agentcities Interaction Using the Opal Platform*. In: *1st International Workshop on "Challenges in Open Agent Systems"which was held at AAMAS'02 in Bologna, Italy in July 2002*.

[Purvis et al. 2002b] Purvis, Martin, M. Nowostawski und S. Cranefield (2002b). *A multi-level approach and infrastructure for agent-oriented software development*. In: *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, S. 88–89, New York. ACM Press.

[Purvis et al. 2001] Purvis, Martin K., M. Purvis und S. Lemalu (2001). *A Framework for Distributed Workflow Systems*. In: Sprague, Ralph H., Hrsg.: *34th HICSS*, Bd. 9, S. 9039, Los Alamitos, Calif. IEEE Computer Society.

[Purvis et al. 2005] Purvis, Maryam, M. Purvis, A. Haidar und B. T. R. Savarimuthu (2005). *A Distributed Workflow System with Autonomous Components*. In: Barley, M.W. und N. Kasabov, Hrsg.: *PRIMA 2004*, Nr. 3371 in *Lecture Notes in Artificial Intelligence*, S. 193–205. Springer-Verlag.

[Purvis et al. 2000] Purvis, Maryam, M. Purvis und S. Lemalu (2000). *An Adaptive Distributed Workflow System Framework*. The Information Science Discussion Paper Series 2000/12, Department of Information Science University of Otago, Dunedin, NEW ZEALAND.

[Purvis et al. 2004] Purvis, Maryam, B. T. R. Savarimuthu und M. K. Purvis (2004). *Evaluation of a Multi-agent Based Workflow Management System Modeled Using Coloured Petri Nets.*. In: Barley, Mike und N. K. Kasabov, Hrsg.: *PRIMA*, Bd. 3371 d. Reihe *Lecture Notes in Computer Science*, S. 206–216, Berlin. Springer-Verlag.

[Reese 2003] Reese, Christine (2003). *Multiagentensysteme: Anbindung der petrinetzbasierten Plattform CAPA an das internationale Netzwerk Agentcities*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Reese et al. 2003] Reese, Christine, M. Duvigneau, M. Köhler, D. Moldt und H. Rölke (2003). *Agent–based Settler Game*. In: *Agentcities Agent Technology Competition, Barcelona, Spain*.

[Reese et al. 2005] Reese, Christine, J. Ortmann, S. Offermann, D. Moldt, K. Lehmann und T. Carl (2005). *Architecture for Distributed Agent-Based Workflows*. In: Henderson-Sellers, Brian und M. Winikoff, Hrsg.: *Workshop on Agent-Oriented Information Systems, AOIS 2005, Utrecht, The Netherlands, July 26, 2005. Proceedings*, S. 42–49, Utrecht, The Netherlands.

[Reese et al. 2006a] Reese, Christine, K. Markwardt, S. Offermann und D. Moldt (2006a). *Distributed Business Processes in Open Agent Environments*. In: Manolopoulos, Yannis, J. Filipe, P. Constantopoulos und J. Cordeiro, Hrsg.: *Eighth International Conference on Electronic Information Systems (ICEIS) 2006*, S. 81–86.

[Reese et al. 2006b] Reese, Christine, S. Offermann und D. Moldt (2006b). *Architektur für verteilte, agentenbasierte Workflows*. In: Schoop, Mareike, C. Huemer, M. Rebstock und M. Bichler, Hrsg.: *Service-oriented Electronic Commerce im Rahmen der Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006)*, Bd. P-80 d. Reihe *Lecture Notes in Informatics - Proceedings*, S. 73–87, Bonn. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

[Reese et al. 2006c] Reese, Christine, J. Ortmann, S. Offermann, D. Moldt, K. Markwardt und T. Carl (2006c). *Fragmented Workflows Supported by an Agent Based Architecture*. In: Kolp, Manuel, P. Bresciani, B. Henderson-Sellers und M. Winikoff, Hrsg.: *Agent-Oriented Information Systems III, 7th International Bi-Conference Workshop, AOIS 2005, Utrecht, The Netherlands, July 26, 2005, and Klagenfurt, Austria, October 27, 2005, Revised Selected Papers*, Bd. 3529 d. Reihe *Lecture Notes in Artificial Intelligence*, S. 200–215. Springer-Verlag, Berlin.

[Reese et al. 2007] Reese, Christine, M. Wester-Ebbinghaus, T. Dörges, L. Cabac und D. Moldt (2007). *A Process Infrastructure for Agent Systems*. In: Dastani, Mehdi, A. E. Fallah, J. Leite und P. Torroni, Hrsg.: *MALLOW'007 Proceedings. Workshop LADS'007 Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS)*, S. 97–111.

[Reese et al. 2008] Reese, Christine, M. Wester-Ebbinghaus, T. Dörges, L. Cabac und D. Moldt (2008). *Introducing a Process Infrastructure for Agent Systems*. In: Dastani, Mehdi, A. E. F. Seghrouchni, J. Leite und P. Torroni, Hrsg.: *Languages, Methodologies and Development Tools for Multi-Agent Systems – First International Workshop, LADS 2007 – Revised Selected and Invited Papers*, Nr. 5118 in *LNAI*, S. 225–244, Berlin. Springer.

[Reichert und Dadam 1998] Reichert, M. und P. Dadam (1998). *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, 10(2):93–129.

[Reichert et al. 2003] Reichert, M., P. Dadam und T. Bauer (2003). *Dealing With Forward and Backward Jumps in Workflow Management Systems*. International Journal of Software and Systems Modeling (SoSyM), 2(1):37–58.

[Reichert et al. 1999] Reichert, Manfred, T. Bauer und P. Dadam (1999). *Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows*. In: *Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*.

[Reichert und Dadam 1997] Reichert, Manfred und P. Dadam (1997). *A Framework for Dynamic Changes in Workflow Management Systems*. In: *DEXA Workshop*, S. 42–48.

[Renew 2008] Renew (2008). Renew – *The Reference Net Workshop homepage*. URL `http://www.renew.de/`.

[Ricordel und Demazeau 2000] Ricordel, Pierre-Michel und Y. Demazeau (2000). *From Analysis to Deployment: A Multi-Agent Platform Survey*. In: *Engineering Societies in the Agents World*, Bd. 1972 d. Reihe *Lecture Notes in Artificial Intelligence*, S. 93–105, Berlin. Springer-Verlag.

[Rodenhagen 1997] Rodenhagen, Jörg (1997). *Darstellung ereignisgesteuerter Prozessketten (EPK) mit Hilfe von Petrinetzen*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Rölke 2004] Rölke, Heiko (2004). *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, Bd. 2. Logos Verlag, Berlin.

[Savarimuthu et al. 2004] Savarimuthu, Bastin Tony Roy, M. Purvis und M. Fleurke (2004). *Monitoring and controlling of a multi - agent based workflow system*. In: J.Roddick, Hrsg.: *Australasian Workshop on Data Mining and Web Intelligence (DMWI 2004), Dunedin. Conferences in Research and Practice in Information Technology*, Bd. 32.

[Savarimuthu et al. 2006] Savarimuthu, Bastin Tony Roy, M. Purvis und M. Purvis (2006). *Agent Based Web Service Composition in the Context of a Supply-Chain Based Workflow*. The Information Science Discussion Paper Series 2006/04, Department of Information Science University of Otago, Dunedin, NEW ZEALAND.

[Savarimuthu et al. 2005] Savarimuthu, Bastin Tony Roy, M. Purvis, M. Purvis und S. Cranefield (2005). *Agent-Based Integration of Web Services with Workflow Management Systems*. The Information Science Discussion Paper Series, 2005/05.

[Scheer und Thomas 2005] Scheer, A.-W. und O. Thomas (2005). *Geschäftsprozessmodellierung mit der ereignisgesteuerten Prozesskette*. Das Wirtschaftsstudium, 34(8–9):1069–1078.

[Schleinzer 2007] Schleinzer, Benjamin (2007). *Flexible und hierarchische Multiagentensysteme — Modellierung und prototypische Erweiterung von Mulan und Capa*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Schmidt und Stahl 2004] Schmidt, Karsten und C. Stahl (2004). *A Petri net semantic for BPEL4WS - validation and application*. In: Kindler, Ekkart, Hrsg.: *Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets*, S. 1–6, Universität Paderborn.

[Seegert 2004] Seegert, Valentin (2004). *Untersuchung von Planerkonzepten für Mulan-Agenten*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Singh 2003] Singh, Munindar P. (2003). *Distributed enactment of multiagent workflows: temporal logic for web service composition*. In: *2nd AAMAS*, S. 907–914, New York, NY, USA. ACM Press.

[Teuber 2003] Teuber, Klaus (2003). *Die Siedler von Catan. Homepage*. Weitere Informationen: `http://www.diesiedlervoncatan.de` und `http://www.KlausTeuber.de`.

[Tibco 2008] Tibco (2008). *TIBCO Staffware Process Suite*. `http://bpmleader.tibco.com/de/docs/staffware_processsuite_datasheet.pdf`. Abgerufen am 19.10.2008.

[Valk 1998] Valk, Rüdiger (1998). *Petri Nets as Token Objects: An Introduction to Elementary Object Nets*. In: Desel, Jörg, Hrsg.: *19th ICATPN*, Nr. 1420 in *LNCS*, S. 1–25, Berlin. Springer-Verlag.

[ViCon GmbH 2009] ViCon GmbH (2009). *Die Swimlane-Darstellung*. `http://swimlane.info/index.htm`. Besucht am 8.4.2009.

[Vidal et al. 2004] Vidal, J. M., P. Buhler und C. Stahl (2004). *Multiagent Systems with Workflows*. Internet Computing, 8:76–82.

[Wagner 2009] Wagner, Thomas (2009). *Modeling of a Centralized Petri Net- and Agent-based Workflow Management System*. Baccalaureatsarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, 22527 Hamburg.

[Wester-Ebbinghaus et al. 2007] Wester-Ebbinghaus, Matthias, D. Moldt, C. Reese und K. Markwardt (2007). *Towards Organization–Oriented Software Engineering*. In: Züllighoven, Heinz, Hrsg.: *Software Engineering Konferenz 2007 in Hamburg: SE'07 Proceedings*, Bd. 105 d. Reihe *LNI*, S. 205–217. GI.

[Westphal und Weyer 2007] Westphal, Ralf und C. Weyer (2007). *.NET 3.0 kompakt*. Elsevier Spektrum Akademischer Verlag.

[WfMC 1995] WfMC (1995). *The Workflow Reference Model*. URL `http://www.wfmc.org/standards/model.htm`.

[WfMC 1997] WfMC (1997). *Workflow Handbook*. John Wiley & Sons, Chichester, UK.

[WfMC 2005] WfMC (2005). *Workflow Handbook 2005*. WfMC.

[WfMC 2008] WfMC (2008). *XML Process Definition Language*. `http://www.wfmc.org/standards/XPDL.htm`.

[Wheeler und Bortolotty 2005] Wheeler, Zachay B. und R. Bortolotty (2005). *Using SOA and Web Services to Improve Business Process Flow*. In: [WfMC 2005], S. 113–128.

[White 2004] White, Stephen A. (2004). *Introduction to BPMN*. Object Management Group/Business Process Management Initiative.

[Whitestein Technologies and the JADE Board 2005] Whitestein Technologies and the JADE Board (2005). *JADE Web Services Integration Gateway*. In: *openNet Networked Agents Demonstration for AAMAS 2005*, Utrecht, The Netherlands.

[Wiering 2003] Wiering, Marco (2003). *Course on Reinforcement Learning*. In: Luck, Michael, W. van der Hoek und C. Sierra, Hrsg.: *AgentLink easss 2003*, S. 333–352. AgentLink.

[Willmott 2005a] Willmott, Steven (2005a). *openNet (Agentennetz)*. `http://x-opennet.net/`. Last visited: Apr. 2009.

[Willmott 2005b] Willmott, Steven (2005b). *openNet Project*. `http://www.x-opennet.org/`. Last visited: Apr. 2009.

[Willmott et al. 2005] Willmott, Steven, M. Beer, R. Hill, D. Greenwood, M. Calisti, I. Mathieson, L. Padgham, C. Reese, K. Lehmann, T. Scholz und M. O. Shafiq (2005). *NETDEMO: openNet Networked Agents Demonstration*. In: Pechoucek, Michael, D. Steiner und S. Thompson, Hrsg.: *AAMAS 2005. Proceedings (Industry Track)*, S. 129–130. 2 individual demos: (1) CAPA: The CAPA Mobile Chat Agent & Web Services Gateway Agent and (2) Settler: AgentBased Settler Game.

[Wolf 2008] Wolf, Karsten (2008). *LoLa – A Low Level Petri net Analyzer*. `http://www2.informatik.hu-berlin.de/top/lola/lola.html`. Besucht am 19.10.2008.

[Wooldridge und Jennings 1995] Wooldridge, Michael und N. R. Jennings (1995). *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review, 10(2):115–152.

[Wooldridge und Luck 2003] Wooldridge, Mike und M. Luck (2003). *Introduction to Agents*. In: Luck, Michael, W. van der Hoek und C. Sierra, Hrsg.: *AgentLink easss 2003*, S. 3–34. AgentLink.

[Yan et al. 2001] Yan, Yuhong, Z. Maamar und W. Shen (2001). *Integration of Workflow and Agent Technology for Business Process Management*. In: *The Sixth International Conference on CSCW in Design*, S. 420–426, London, Ontario, Canada.

Index

Anmerkung zum Index: Manche Einträge verweisen bewusst nur auf die Stelle, an der der Begriff erläutert ist.