Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Multi-Agent Approach for Managing Workflows in an Inter-Cloud Environment

**Dissertation zur Erlangung des Doktorgrades**
**an der Fakultät für Mathematik, Informatik und Naturwissenschaften**
**Fachbereich Informatik**
**der Universität Hamburg**

vorgelegt von

**Sofiane Bendoukha**
**aus Oran, Algerien**

Hamburg, 2016

**Gutachter:**
Dr. Daniel Moldt
Prof. Dr. Norbert Ritter
Prof. Dr. Bernd Page

**Tag der Disputation:**
14. November 2016

**Dedication**

*To my Mother*

# Abstract

The Cloud technology offers several attractive features for a number of applications. Nevertheless, managing, controlling processes and resources are among the serious obstacles that Cloud application developers need to overcome. These issues increase when the execution of processes needs to exploit services from different Cloud providers to fulfill application requests and requirements. In this case, Cloud application developers need to deal with some critical problems like communication, coordination, collaboration and heterogeneity between all kinds of participants. Designing, building and managing Cloud systems and applications running in the Cloud are challenging issues. Powerful and expressive software systems have to be provided, which are required to handle the complexity of such Cloud systems.

This dissertation contributes at different stages to enhance the development of Cloud-based applications with an emphasis on processes. Several contributions provided here concerns the modeling phase. New modeling techniques and tools based on Petri net formalisms are provided. Furthermore, agents and workflows provide intuitive and powerful concepts to bring more intelligence and structure to the Cloud system. The features of agents and workflows are suitable for designing and implementing software systems that operate in distributed and open environments such as the Clouds. The main goal of this thesis is to propose means like modeling techniques, methods, tools and architectural design to facilitate the integration of Cloud and multi-agent systems concepts and technologies for managing workflows in distributed service-oriented environments, explicitly, in an Inter-Cloud environment.

# Zusammenfassung

Die Cloud Technologie bietet zahlreiche, attraktive Funktionen für eine Reihe von Anwendungen. Nichtsdestotrotz sind sowohl schwierige Verwaltung, als auch die komplizierte Handhabung von Prozessen und die jeweiligen Ressourcen noch Hindernisse, die die Entwickler von Cloud Anwendungen überwinden müssen. Diese Probleme treten vermehrt auf, wenn das Verfahren in seiner Ausführung auf Dienste unterschiedlicher Cloud Anbieter zugreifen muss, um die Anwendungsanfragen und -anforderungen zu gewährleisten. Zu diesem Zweck sollten Entwickler von Cloud Anwendungen sich eingehend mit solch kritischen Problemen beschäftigen, indem man dem Zusammenspiel von Kommunikation, Koordination, Zusammenarbeit und Heterogenität der TeilnehmerInnen nachgeht. Die Aufmachung des Designs, die Art und Weise der Implementierung und Verwaltung von Cloud Systemen, als auch die einzelnen Anwendungen, die in der Cloud laufen, sind herausfordernde, wichtig zu gestaltende Kernpunkte. Ziel müssen leistungsstarke Softwaresysteme sein, die in der Lage sind solch komplizierte Systeme zu unterstützen.

Diese Dissertation leistet einen Beitrag zu den unterschiedlichen Entwicklungsstufen von Cloud Anwendungen mit Betonung auf den Prozessen. Mehrere Abschnitte der vorliegenden Arbeit befassen sich mit der Phase der Modellierung. Hierzu werden neue Techniken und Werkzeuge der Modellierung auf der Basis von Petrinetzformalismen vor- und zur Verfügung gestellt. Darüber hinaus bieten Agenten und Workflows die Möglichkeit sowohl intuitive, als auch leistungsstarke Konzepte zu entwickeln, die erhöhte Intelligenz und verbesserte Struktur der Cloud Systeme bewirken. Die Ausstattungen der Agenten und Workflows sind auf den ersten Entwurf und die spätere Implementierung von Softwaresystemen ausgerichtet, die, wie Cloud, für verbreitete und offene Umgebungen geeignet sind. Das Ziel dieser Arbeit ist mittels Modellierungstechniken, Methodiken, Werkzeugen und architekturellen Entwürfen, für die Integration von Cloud- und Multiagentensysteme, die Konzepte und Technologien der Verwaltung von Workflows in verteilten, dienstorientierten Umgebungen, insbesondere Inter-Cloud Systemen, zu erleichtern.

# Acknowledgments

Although this thesis is my own contribution, the work described in this manuscript would not seen the light without the many helpful people around me. I would like to thank them herewith. First and foremost, I would like to express my special appreciation and thanks to my supervisors: Dr. Daniel Moldt (Theoretische Grundlagen der Informatik (TGI)) and Prof. Dr. Norbert Ritter (Datenbanken und Informationssysteme (ISYS)). You have been an exceptional guide for me. I would like to thank you for the continued support and encouragement throughout this thesis. I am deeply grateful to Dr. Moldt for his patience and for the long discussions that helped me sculpt my work. Further I would like to acknowledge the jury members of this thesis: Prof. Dr. Walid Maalej and Prof. Dr. Bernd Page. My sincere thanks also go to Prof. Dr. Rüdiger Valk for his implication to get funding and to join the TGI group.

I would also to thank all past and current members of the TGI group at the university of Hamburg, for their various forms of support during the last awesome years. In particular, Michael Haustermann for his guidance and assistance during the integration of my work into Renew. I also thank my friend Dr. Hashim Iqbal Chunpir for his helpfulness and support.

More importantly, I would like to thank all members of my big family for their love, continuous support and encouragement at all times. I am very grateful to my sister Dr. Lahouaria Diab. Your presence contributed to reduce my homesickness and was a good support during the difficult times. I am also greatly indebted to Houssem for proofreading. Special thanks go to my sister Hayat. I can not thank you enough. I have learned a lot from you. Your encouragement, advices and your trust in my abilities prevented me several times to give up.

Finally, I appreciate the financial support from the Deutscher Akademischer Austauschdienst (DAAD), the MIN Graduate School International, the department for research funding at the University of Hamburg and Prof. Dr. Norbert Ritter.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This chapter introduces the context of the research, motivates this study and details the contributions of the thesis. It is structured as follows: Section 1.1 presents the problem, which is addressed in this work. Section 1.2 introduces a use case, which will be used through the thesis to show the usability of the contributions. Section 1.3 defines the goals objectives of the thesis. Section 1.4, outlines the contributions of this thesis. Section 1.5 presents the structure of this manuscript.

## 1.1 Problem Statement and Scope

Service Oriented Computing (SOC) is the field of computer science that revolves around the concept of "service": Web and Cloud services. The latter are currently the most common forms of service for implementing service-oriented computing [Koehler and Alonso, 2007]. While Cloud services provide the foundation for the distributed execution of complex applications using a standardized and stateful service interface, Web services provide the basis for the development and execution of workflows that are distributed over the network and available via standard interfaces and protocols.

Based on the Internet, Cloud computing provides on-demand computing and storage capacities to individuals and businesses in the form of heterogeneous and autonomous services. Furthermore, recently there is an emergence of *Inter-Cloud Computing*, which could be seen as a Cloud of Clouds [Kelly, 2007]. The reason lies in the fact that a single Cloud infrastructure does not have unlimited resources to satisfy complex application requirements and the latter may receive requested services from different Cloud providers [Buyya et al., 2010]. This new computing paradigm needs to deal with the problem of heterogeneity, communication, coordination and collaboration among all participants.

Hence, the construction of such complex systems remains a problem as soon as there are several independent / autonomous partners involved in the design and execution of these systems. Currently mainly data is stored in the Cloud. (Web) services in the Cloud are designed to be realized in a static fashion. Missing is the support of processes in this environment. For complex systems with independent partners expressive and powerful software systems have to be provided.

Multi-Agent Systems (MAS) and workflow concepts are strong candidates to address this issue. On the one hand, common characteristics of agents are social ability, autonomy, pro-activity, adaptability and mobility [Jennings, 2000]. Also in this perspective, according to the literature, *mobile agents* are used to construct a Cloud computing federation mechanism to permit portability and interoperability among different Cloud computing platforms [I. Foster, 2004, M. Spata, 2011a, Zhang and Zhang, 2009]. On the other hand automation of processes and efficient coordination and collaboration among various entities are some advantages of workflow concepts.

However, Workflow Management Systems (WfMSs) usually do not address the special aspects of Cloud-based systems. Current inter-organizational WfMSs are designed to control the autonomous entities (agents or Web services) from another location. So it is not embedded within the systems. This causes problems with respect either to the autonomy of the participating partners or their efficient coordination. New concepts and constructs to overcome this problem are necessary.

Therefore, this thesis provides a conceptual and technical solution for the modeling and design of complex systems in Cloud-like environments with a special emphasis on processes. The aim is to provide concepts, techniques and tools to design and to implement agent-based WfMSs, which support definition, deployment and monitoring of distributed inter-organizational workflows for independent complex partners within Cloud environments.

Usually autonomy for Web services is not a desired property. A Centralized process management system to control the execution of cross-organizational processes is often neither technically nor organizationally desired [Zaplata et al., 2009, Zaplata, 2012]. For Inter-Cloud applications each Cloud should keep its (relative) autonomy, so that autonomy in this context is an inherent property. In order to execute an overall application within the Inter-Cloud environment, autonomy of Clouds must be possible or even supported.

Moreover, the proposed modeling concepts and techniques as well as their tools facilitate the integration between Cloud environment and MAS for an efficient management and execution of workflows in environments qualified to be distributed and scalable. Specifically, that means that, concepts and technologies from agents, workflows and Clouds field are coupled to offer a powerful environment to the users for the deployment of applications based on multiple Cloud platforms.

In this work techniques, models and tools, which are a part of the Petri net-based, Agent- and Organization-oriented Software Engineering (Paose) approach are exploited. On the basis of high-level Petri nets the above mentioned concepts like agents, workflows or services are integrated. The Multi-Agent Nets (Mulan) architecture [Cabac, 2010] and the Reference Nets Workshop (Renew) [Cabac et al., 2016] tool[1] developed at Theoretische Grundlagen der Informatik (TGI) group at the University of Hamburg[2] provide the technical background for this.

[1] http://www.renew.de/

[2] http://www.informatik.uni-hamburg.de/TGI/

## 1.2 Motivating Example

In order to give an overview of the objectives of this thesis and to present the contributions, a working example related to the remote sensing domain is provided. First and foremost, it is noteworthy to mention that *scientific workflows* are considered in this work. The reason behind considering scientific workflows as use case is related to the nature of tasks and their dependencies[3]. In contrast to business workflows (see Chapter 4), scientific workflows contain usually compute and storage intensive tasks. Thus it is recommended to distribute the execution of these tasks across many computing nodes. The following scenario will be utilized in several chapters of the thesis as a showcase. It shows the applicability of the proposed concepts, techniques and tools to real scenario.

Consider the scenario from Figure. 1.1, it is modeled by Petri nets, where the transitions represent the tasks to be performed. It represents a simple processing workflow of a satellite imagery. The workflow calculates the Normalized Difference Vegetation Index (NDVI) from two raster data files (NIR and Red images). The input to the workflow are the two images and the output is a single image containing the NDVI values. The complete implementation of the NDVI workflow is described in Chapter 8.

Due to the lack of resources at local site[4], workflow can not be executed. Thus users should move the execution of the whole workflow or some of its tasks to other available computing nodes, for example in an Inter-Cloud environment. Through a graphical user interface (GUI)[5] workflow definitions as well as the required data are uploaded/submitted to the execution target. Consequently, the entity (software) that calculates the NDVI values as well as the required data are hosted in the Cloud.

The life-cycle of this workflow can be summarized as follows:

- (T1) Workflow definitions are uploaded as well as all required images to the Cloud.
- (T2) On the selected Cloud, specific softwares or Web services calculate the NDVI values and generate an image as output.
- (T3) In parallel with (T2), the original image is stored in a Cloud-based database.
- (T4) The calculated NDVI values are displayed or transmitted to the entity initiating the workflow.

[3]Data and other resources required for the execution.

[4]The lack of resources may take several forms. For example, unavailability of required softwares or performance issues.

[5]Web or desktop are both possible.

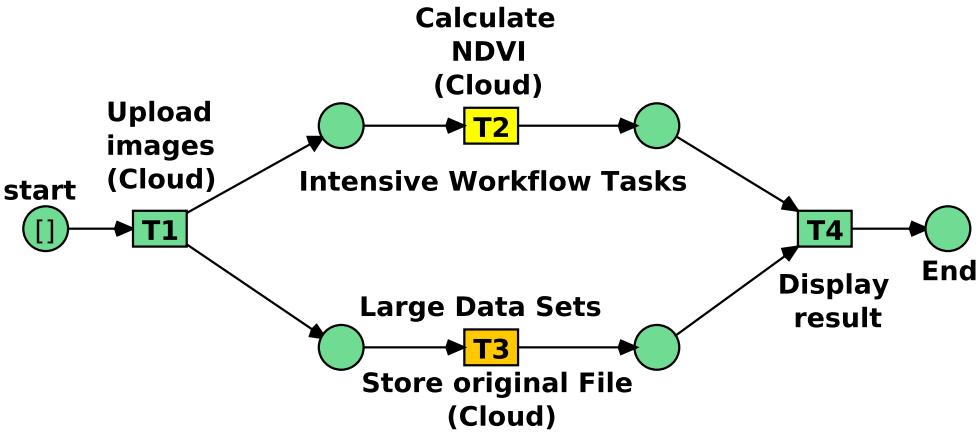



Figure 1.1: Cloud-based Image Processing Workflow

Even if the workflow described above seems to be straightforward in terms of complexity, moving the execution of workflows to the Cloud is not trivial. In fact, executing workflow tasks in a Cloud environment or invoking Cloud services requires special skills and must be based on a solid strategy. Adopting a Cloud solution has also many challenges. Which Cloud provider is appropriate for the execution of workflow? How to find and invoke services from different Cloud providers? are some of the questions that should be investigated prior moving to the Cloud. These issues apply for both Cloud developers and Cloud consumers. Furthermore, functions of the WfMS should be adapted to the Cloud. According to [Yu and Buyya, 2005], there are build-time and run-time functions. The build-time functions are concerned with defining and modeling workflow tasks and their dependencies; while the run-time functions are concerned with managing the workflow execution and interactions with resources for processing workflow applications. In contrast to Grid computing, these functions need to be adapted to the Cloud environment. Especially at the modeling level there are different criteria that should be considered to cover the special nature of the Cloud environment. This can for example include costs or performance parameters (see Chapter 10). Moreover, all required software must be installed and configured at the execution target (Cloud instance). Mechanisms should be provided in order to ease the provision of Cloud instances.

Concerning the example scenario, transitions T1, T2 and T3 require Cloud services. In order to retrieve these services, an appropriate selection mechanism needs to be available in order to select the suitable Cloud provider that fits their requirements. Furthermore, Cloud consumers should be able to express the requirements of their applications properly (at design time). Another important issue is Cloud interoperability. With the growth of the Cloud computing technology there are currently plenty of Cloud providers. Nevertheless, each of these providers has its own data models, interfaces as well as authentication and authorization mechanisms. Thus programming against the Cloud is arduous. Adopting applications to be executed in a Inter-Cloud environment There should be a strategy to communicate with different Cloud providers. A run-time

API (vendor independent) is the appropriate solution for this issue. The contributions presented in this thesis strive to overcome the issues mentioned above and to provide a user-friendly environment for the development of Cloud-based applications.

## 1.3 Goals and Objectives

As mentioned above, the aim of this thesis is to propose means like modeling techniques, methods, tools and architectural design. The contributions presented in this thesis facilitate the integration between Cloud systems and MAS for an efficient management and execution of workflows in environments qualified to be distributed and scalable. The description and implementation of this work are carried out in several steps/phases, which are iteratively applied to have several prototypes. Firstly, an accurate state of the art is elaborated to evaluate the existing conceptual basis of the presented work and technological solutions. The basic research areas are: Cloud computing, workflow management systems, modeling techniques and tools, agent systems and Petri nets. In the next step new requirements are defined for modeling workflow execution in complex environments. The focus is on the actual problems in Inter-Cloud environments such as heterogeneity, communication, coordination and collaboration between the participants.

Taking into account the newly defined requirements appropriate modeling techniques and concepts are proposed. They should constitute a conceptual basis for the management of Inter-Cloud applications. The Unified Modeling Language (UML) and Petri nets are the major modeling techniques that are investigated to elaborate the new proposed techniques. This step also includes provisioning semantics based constructs that allow for an efficient design of process management of Inter-Cloud applications.

In order to evaluate and validate the results, direct modeling tools support are introduced, which will be implemented on the basis of Renew for the elaborated techniques. Most of prototypes are based just on Renew in terms of a drawing and simulation tool for Petri nets and Uml models.

Moreover, the integration of the proposed tool support within the Paose approach is considered. This allows for simulation and execution of agents system using the Mulan and the Concurrent Agent Platform Architecture (Capa) framework. Due to the Foundation for Intelligent Physical Agents (FIPA) compliance also distributed execution is possible. An extension for workflows is provided by [Jacob et al., 2002] for Petri nets and by [Reese, 2009] and [Wagner and Moldt, 2011] for workflow and WfMS.

As a prove of concept for the conceptual solution proposed in the dissertation, a prototype distributed over above mentioned prototypes will allow for the investigation of heterogeneous implementation of the MAS approach for the management of workflow in an Inter-Cloud environment.

## 1.4 Contributions

This thesis provides several contributions towards improving the development life-cycle of (Inter-) Cloud-based applications and towards advancing the accessibility to the Cloud environment. The major contributions are as follows:

1. This thesis provides an accurate state of the art, which evaluates the existing conceptual basis of this work and technological solutions. Part I presents different concepts, fundamental techniques and tools that are essential to this work. It gives a deep insight of Service Oriented Computing (SOC) and Grid/Cloud computing. It defines the key workflow concepts: process, business process, scientific workflows, WfMS. Moreover, an overview of the Paose approach and its related frameworks is given.

2. This thesis investigates the migration of workflow applications to the Cloud. Based on [Strauch et al., 2013, Han et al., 2010], different scenarios are analyzed and extensions are proposed. The study in Chapter 7 represents the conceptual basis of the work presented in Part II and Part III. A framework named ICWorkflow is implemented for the development of (Inter-) Cloud-based workflows. It provides the required layer for the ICNets. It covers several steps when modeling and executing workflows in the (Inter-) Cloud.

3. This thesis defines the requirements for modeling and executing workflows in complex environments. Moreover, it proposes appropriate modeling techniques and concepts that should constitute a conceptual basis for the management of Inter-Cloud applications. There is a strong emphasis on modeling in this thesis. For instance, RenewGrass (see Chapter 8) is a tool that allows the specification of image processing workflows by Petri nets. The Cloud Task Transition (Ctt) and the Inter-Cloud Nets (ICNets) (see Chapter 10) are significant contributions. The objective is also to enhance the modeling of Cloud-based workflows by Petri nets. The Ctt is a new modeling concept, which consists of introducing a special kind of Petri net transitions. These transitions have the ability to communicate with Cloud providers and invoke services. Also in this direction, ICNets ((see Section 10.3)) are predefined Petri net models that can be used to perform Cloud operations such as database transactions.

4. This thesis investigates the relation between Cloud computing and workflows concepts. In Part II, different prototypical implementations are provided to show how Cloud technology can benefit from the integration of workflows and vice versa. The relation between workflows and Clouds is addressed in two different ways. On the one hand, this thesis proposes and provides new mechanisms to enable workflow execution in the Cloud (see Chapter 9). On the other hand, Clouds need structured and mature workflow concepts and high level languages to handle issues like managing complex task and data dependencies.

5. This thesis elaborates a multi-agent approach for the management of workflows in an Inter-Cloud environment. The concept of a Drop-Engine (DE) is presented

in Part III, which shows the impact of integrating the agent metaphor to the development of Cloud-based applications.

## 1.5 Structure of the thesis

The thesis is organized in three main parts: the objective of Part I is to elaborate a state of the art of the concepts, techniques and tools which are used to achieve the contributions. Part II addresses the notion of Cloud-based workflow management, it constitute the basics of this work. Part III shows the applicability of the agent concepts to the results obtained in Part II.

After a short introduction to the notion of SOC, Part I provides a deep understanding of (Inter-) Cloud computing, workflows and Petri nets. The part emphasizes specifically on modeling techniques. It also introduces agents and multi-agent systems. The Paose approach and the related Mulan/Capa framework are also presented. This part concludes with a brief summary.

Part II presents the essential contributions of the thesis. Here the relation between Clouds and workflows is clearly demonstrated. In this work the relation is shown through different implementations covering the execution of workflows in the Cloud. The first chapter of Part II can be seen as state-of-the-art introduction on how to enable the migration of existing systems to the Cloud. A plug-in named RenewGrass for the implementation of image processing workflows by Petri nets is presented. Next, two important topics are addressed, namely: Cloud computing for workflow execution and workflow concepts for the Cloud. Finally, the ICWorkflow plug-in for Renew is presented, which enables the specification and execution of Petri nets-based workflows for the Cloud. A summary concludes this part. Part III shows how agent concepts and techniques participate in the management of Cloud-based workflows. The works presented in Part II are extended to fit to the Paose approach techniques. Part IV presents a summary of the presented work. It includes a description of the achieved contributions and future work.

## 1.6 Publications

Parts of this thesis have been released in the following publications:

[Bendoukha, 2013] Bendoukha, S. (2013). Multi-agent approach for managing workflows in an inter-cloud environment. In Lomuscio, A., Nepal, S., Patrizi, F., Benatallah, B., and Brandić, I., editors, *Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers*, volume 8377 of *Lecture Notes in Computer Science*, pages 535–542. Springer.

[Bendoukha et al., 2015a] Bendoukha, S., Bendoukha, H., and Moldt, D. (2015a). Building cloud-based scientific workflows made easy: A remote sensing application. In Marcus, A., editor, *Design, User Experience, and Usability: Users and Interactions - 4th International Conference, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*, volume 9187 of *Lecture Notes in Computer Science*, pages 277–288. Springer.

[Bendoukha et al., 2015b] Bendoukha, S., Bendoukha, H., and Moldt, D. (2015b). ICNETS: Towards designing inter-cloud workflow management systems by Petri nets. In Barjis, J., Pergl, R., and Babkin, E., editors, *Enterprise and Organizational Modeling and Simulation - 11th International Workshop, EOMAS 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Selected Papers*, volume 231 of *Lecture Notes in Business Information Processing*, pages 187–198. Springer.

[Bendoukha et al., 2015c] Bendoukha, S., Bendoukha, H., and Moldt, D. (2015c). RenewGrass - A tool for building scientific workflows: Application to the remote sensing domain. In Rubin, S. H. and Chen, S.-C., editors, *2015 IEEE International Conference on Information Reuse and Integration, IRI 2015, San Francisco, CA, USA, August 13-15, 2015*, pages 311–317. IEEE.

[Bendoukha and Cabac, 2013] Bendoukha, S. and Cabac, L. (2013). Cloud transition for QoS modeling of inter-organizational workflows. In Moldt, D., editor, *Modeling and Buisness Environments MODBE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 355–356. CEUR-WS.org.

[Bendoukha et al., 2013] Bendoukha, S., Moldt, D., and Wagner, T. (2013). Enabling cooperation in an inter-cloud environment: An agent-based approach. In Morvan, F., Tjoa, A. M., and Wagner, R., editors, *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on*, pages 217–221. IEEE Computer Society.

[Bendoukha and Wagner, 2012] Bendoukha, S. and Wagner, T. (2012). Cloud transition: Integrating cloud calls into workflow Petri nets. In Cabac, L., Duvigneau, M., and Moldt, D., editors, *Petri Nets and Software Engineering. International Workshop PNSE'12, Hamburg, Germany, June 2012. Proceedings*, volume 851 of *CEUR Workshop Proceedings*, pages 215–216. CEUR-WS.org.

[Bendoukha and Wagner, 2015] Bendoukha, S. and Wagner, T. (2015). Improving performance of complex workflows: Investigating moving net execution to the cloud. In Moldt, D., Rölke, H., and Störrle, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings*, volume 1372 of *CEUR Workshop Proceedings*, pages 171–189. CEUR-WS.org.

# Part I

# Foundations

# Introduction

The work presented in this dissertation is based on several concepts, fundamental techniques and tools. For instance: Multi-agent systems, workflow modeling, Petri nets. These concepts, techniques and tools proved their efficiency for modeling, verifying and executing processes, that can either run on-premises or in distributed environments such as Grids or Clouds. In the following chapters an overview of the basic concepts and the key techniques are given. The *Foundations* part represents the conceptual and the technical background of all proposed solutions and implemented tools in order to ensure the success of a Cloud-based workflow management strategy.

The structure of this part is organized as follows: Chapter 2 gives an insight to Service Oriented Computing (SOC) and its core paradigms. It presents the concept of Service Oriented Architecture (SOA) and the related technologies such as Web services and service composition principles. With respect to process execution, (Inter)-Clouds are selected as the execution target. Clouds (and also Grids) are described in Chapter 3. This chapter answers questions like: what is Cloud computing? What are the different types of Clouds? Existing Cloud standards and formal aspects of Cloud computing are also outlined.

Chapter 4 concerns all concepts and techniques related to the notion of process management and workflow. It defines the key workflow concepts: Process, business process, scientific workflows. Afterward, a general overview of a workflow management system and its structure is given. Mainly, the workflow reference model from the WfMC is described.

Specification (modeling), deployment and execution of workflows in Cloud-like environments fills a huge part of this dissertation. New techniques and tools have been developed in this direction. Furthermore, since there is a strong emphasis on the modeling of workflows by Petri nets, Chapter 5 deals exactly with this topic. As already mentioned in the introduction, this dissertation exploits techniques, concepts and tools from the Paose approach. Chapter 6 gives an overview about Mulan and the Paose approach. Furthermore, the chapter provides also an overview of agents and multi-agent systems. The description of the different concepts is strongly based on the FIPA specifications.

# Chapter 2

# Service Oriented Computing (SOC)

This chapter describes the fundamental basic concepts and technologies that enable SOC. It includes definitions of SOC and explains the SOA paradigm as well as Web service technology.

## 2.1 Introduction

Nowadays, the key technologies in distributed systems are SOA, Web services and Cloud computing. These three technologies can be encapsulated under the global term, SOC. SOC is a paradigm that is changing the way modern software is designed and developed. It utilizes software services as fundamental elements for developing and deploying distributed software applications. Services are autonomous entities that can be dynamically discovered and composed to provide more complex systems. SOC and Cloud computing have many similarities and challenges such as maintaining high service availability, providing end-to-end secure solutions, managing longer-standing service workflows and service discovery through federated Clouds [Wei and Blake, 2010]. Cloud computing provides the processing of services, and SOC offers services for computing. All along the thesis, the notion of (Cloud) services is frequently mentioned. This chapter provides a description of the technologies enabling SOC, whereas Cloud computing will be described in Chapter 3. The reason behind introducing SOC is that several architectures and implemented tools in this thesis follow the SOA paradigm. For instance, Web services have been used to explain the concept of *synchronous channels* (see Chapter 5) and the deployment of RenewGrass in a Cloud environment (see Chapter 8).

## 2.2 Definitions

SOC is a cover term to design a distributed computing platform based on services. The latter are utilized in order to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments

[Papazoglou et al., 2007]. According to [Francisco et al., 2006], SOC is defined as follows:

> *"Service-Oriented Computing (SOC) is a new computing paradigm that utilizes services as the basic construct to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments. The visionary promise Service-Oriented Computing is a world of cooperating services where application components are assembled with a little effort into a network of services that can be loosely coupled to create flexible dynamic business processes and agile applications that may span organisations and computing platforms."* [Francisco et al., 2006]

At the heart of service-oriented computing are services that are autonomous, loosely coupled software components with publicly available interfaces that can be invoked by a client or composed by a third party to achieve a more complex goal. According to [MacKenzie et al., 2006] a service is:

> *"a mechanism to enable access to a set of one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by one entity – the service provider – for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider."*

In the following sections, some background is given about service-oriented architectures, Web services and service composition.

## 2.3 Service-Oriented Architecture

Most system architectures follow a service-oriented approach and Cloud computing is no exception. SOC is mainly driven SOA. SOA is an architectural style for building software systems based on services. Services are loosely coupled components that can be discovered and composed. SOA is not an architecture in itself, but it instead leads to a concrete architecture. Hence, SOA is rather a paradigm than a standard. Thus there are several definitions of SOA [Oasis, 2012, Erl, 2005, Mimoso, 2004]. Generally, one can say that SOA is a style of designing large distributed information systems based on autonomous, QoS-capable, vendor diverse, inter-operable, discoverable and potentially reusable service. Early examples of technologies that at least partly service-oriented are CORBA, DCOM, J2EE or .NET.

In considering the term service-oriented architecture, it is useful to review the key terms:

- An **architecture** is a formal description of a system, defining its purpose, functions, externally visible properties, and interfaces. It also includes the description of the system's internal components and their relationships, along with the principles governing its design, operation, and evolution.

- A **service** is a software component that can be accessed via a network to provide functionality to a service requester.
- The term **service-oriented architecture** refers to a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.

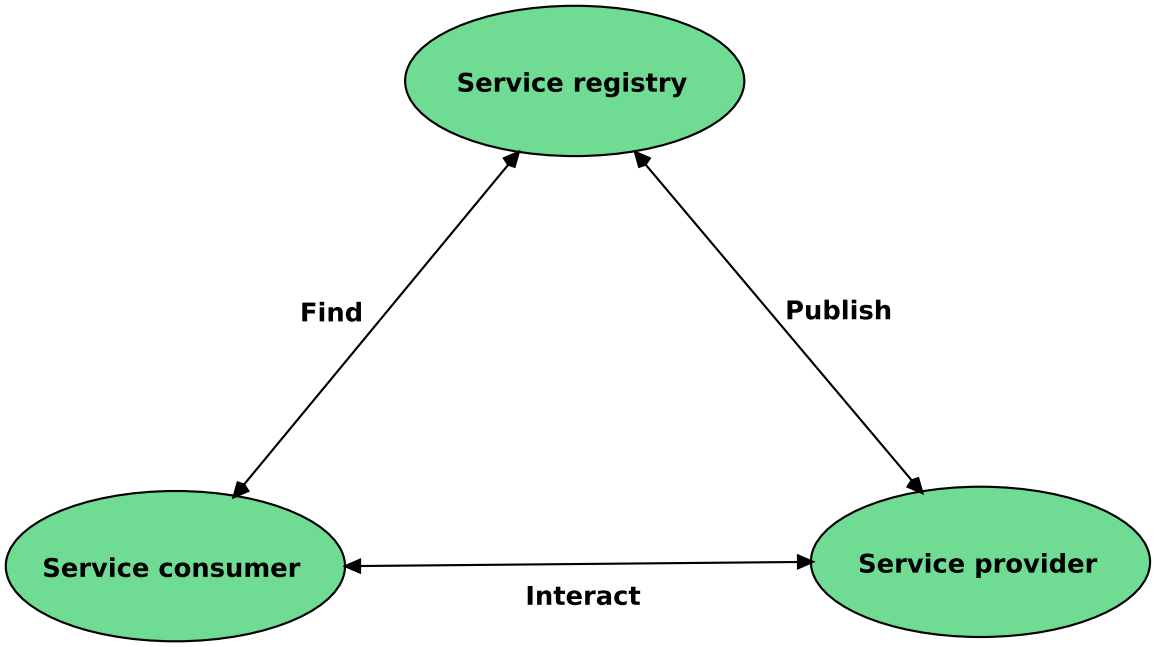



Figure 2.1: Service Oriented Architecture

- **Logical view**: The service is an abstract, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
- **Message orientation**: The service is formally defined in terms of the messages exchanged between provider and requester agents, and not the properties of the agents themselves.
- **Description orientation**: A service is described by machine-processable meta data.
- **Granularity**: Services tend to use a small number of operations with relatively large and complex messages.
- **Network orientation**: Services tend to be oriented toward a use over a network.
- **Platform neutral**: Messages are sent in a platform-neutral, standardized format delivered through the interfaces, Extensible Markup Language (XML) is the most obvious format that meets this constraint.

## 2.4 Web Services

The term Web services is a set of technologies based on XML, for creating distributed software components, describe their interfaces and use them regardless of the chosen implementation language and the hosting platform. One of the most important objective of Web services, is to achieve interoperability.

*"A Web Service is a software system designed to support inter-operable machine to machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."* [Booth et al., 2004].

In the following section technologies that are used to implement Web services will be presented. Restful implementation style also described as well as the Web services composition.

### 2.4.1 SOAP Web Services

The key specifications used by Simple Object Access Protocol (SOAP) Web services are:

- XML (eXtensible Markup Language) A markup language for formatting and exchanging structured data. XML is one of the most widely used formats for sharing structured information among programs. It includes the following concepts:
  - XML document (data object): a composition of declarations, elements, comments, character, references and processing instructions
  - XML namespaces: are used to uniquely identify named elements and attributes in an XML document. XML, the default namespace is defined by using the *xmlns* attribute in the start tag of an element
  - Characters: a character is an atomic unit of text
  - Markup: is a series of characters in XML document. It can be distinguished from text because it always begins either with the character < (in which case it ends with the character >) or the character & (in case it ends with the character ;)
  - Character data: a text other than markup
  - Entities: they are identified by a *name* and have contents. Each XML document has a unique entity called DOCUMENT ENTITY which serves as the root of the tree and a starting point for the XML processor
  - XML processor: it is a software module that is used to read XML documents and provide access to their content and structured
  - Parsed data: this term refers to the entities that are processed or read by the XML processor. A parsed entity contains a sequence of characters, called text.

- SOAP: This specification describes the basic format of an exchanged message between two Web services. A message is an XML document composed by two different parts: the *header* and the *body*. The former part is an optional part which can contain special control information which characterize the communication such as, e.g., security references for retrieving cryptographic keys or reliability tags for guaranteeing message delivery, whereas the latter part contains the information to be communicated.

- Universal Description, Discovery and Integration (UDDI): This specification deals with the programming interface exhibited by a discovery registry which is a particular kind of service that allows for the retrieving of Web service depending on their functionalities. The Uddi specification introduces the concept of dynamic discovery of a Web service.

- Web Services Description Language (WSDL): This specification deals with the description language which allows for the standard definition of a Web service interface. It is a fundamental specification which fixes the basic communication primitives, the operations, exploited by a Web Services for exchanging messages.

### 2.4.2 RESTful Web Services

Representational State Transfer (Rest) is a software architecture style first described in a doctoral dissertation by a researcher named Roy Fielding [Fielding, 2000] that turns around the transmission of data over HTTP. Rest has received a lot of attention recently as architectural style for distributed systems made up of loosely coupled resources.

RESTful architectures adhere to the following basic principles:

- Application state and functionality are divided into resources

- Resources are addressable using standard Unified Resource Identifiers (URIs) that can be used as hypermedia links

- All resources use only the four Hypertext Transfer Protocol (HTTP) methods

  - DELETE to delete resources
  - GET it queries the representation of a resource
  - POST to change the state of a resource
  - PUT to create new resources or to replace the content of existing ones

- All resources provide information using the Multipurpose Internet Mail Extensions (MIME) types supported by HTTP

- The protocol is stateless

- The protocol is cachebale

- The protocol is layered

By now, many developers are using a REST approach because it is easy to use and background knowledge about WSDL, Common Object Request Broker Architecture (CORBA), or Remote Method Invocation (RMI) is not required. They can easily be called from Java, Hypertext Preprocessor (PHP), Ruby, Python, C#, or even a shell script. Petri nets are an adequate modeling technique for Web services behavior and can be used as a composition model and language for RESTful Web services [Alarcón et al., 2010] [Decker et al., 2008]. In the Mulan/Capa framework (see Chapter 6), RESTful architecture is followed to create a gateway architecture that makes it possible to interconnect multi-agent systems and Web services [Betz et al., 2011]. Furthermore, several of the prototypes in this work are based on RESTful Web services. Concerned are mainly front-end applications provided for the Cloud consumers to interact with the Cloud provider.

## 2.5 Service Composition

Different services are often required to work together to complete a task. A *service composition* is an aggregate of services collectively composed to automate a common task. Figure. 2.2 illustrates a service composition that highlights which service capabilities are being invoked in a particular sequence.

Service A is acting as the service consumer of Service B, C and D. The arrows indicate a sequence of modeled message exchanges. When taking part in a composition, services can perform different roles depending on how they are positioned within the overall composition configuration. As a *composition controller*, the service is located at the head of a composition hierarchy. This happens when the service capability that is being executed contains and carries out logic that invokes capabilities in other services (see Service A in Figure. 2.2)[1].

On the other side, a *composition member*, represents a service being composed by another. As shown in Figure. 2.2, it is the fact that the service's capability is being invoked by another service that places the service into this role. A composition member may compose other compositions members, which can, in turn, compose others as well (see Figure. 2.3)[2].

[1] Capability A of Service A composes two other services. This assigns the Controller's role to Service A. Capability A of Service B does not compose other services but is composed. This places it as a Member role

[2] Service B is placed into the role of controller because Capability B composes two other services. Service C fulfills both the controller and the member role because its Capability B is composed by Service B and also composed by Service D

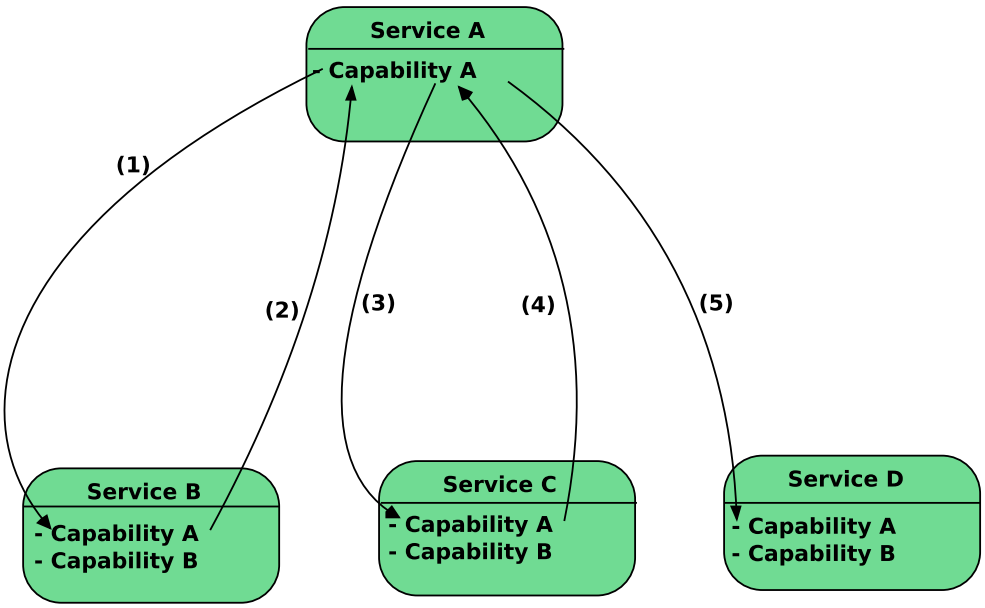



Figure 2.2: A Service Composition comprised of four Services (adapted from [Erl et al., 2012])

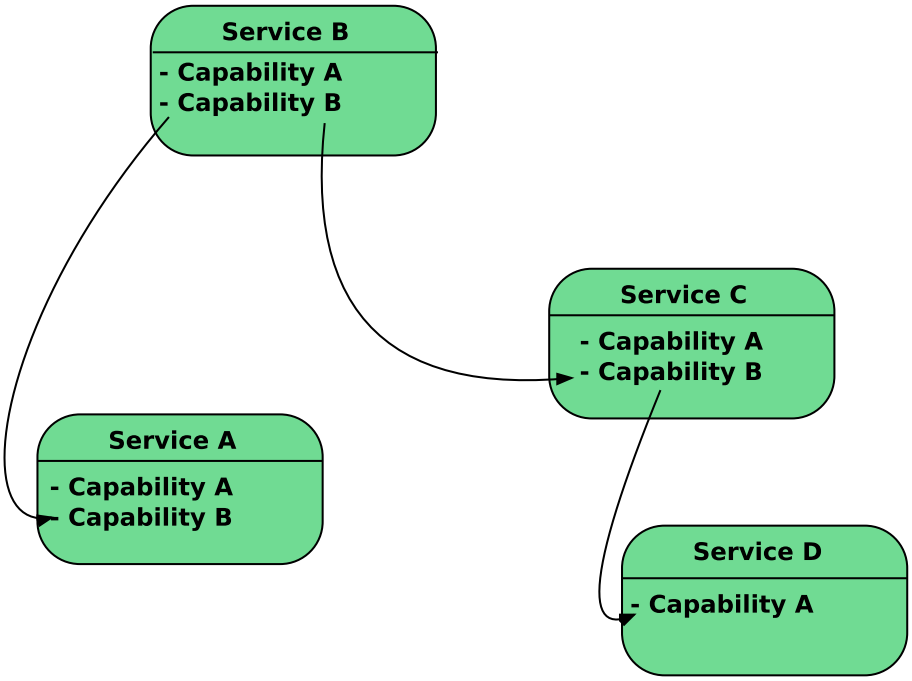



Figure 2.3: Composition Members and Controllers (adapted from [Erl et al., 2012])

There are two terms (i) *orchestration* and (ii) *choreography* which explain two features of workflow creation from composite Web services. Furthermore, in the current work

there is more focus on *orchestration* then *choreography*. For instance, the prototypes presented in Part II makes use of services from different Cloud providers. Invoking these services is performed by Petri nets transitions.

Concerning the implication of Petri nets or specifically High Level Petri Nets (HLPN) like reference nets (see Chapter 5) several research works addressed this topic [Moldt et al., 2004, Offermann, 2003, Hamadi and Benatallah, 2003, Cardinale et al., 2013]. In [Hamadi and Benatallah, 2003], the authors proposed a Petri net-based algebra for modeling Web service control flows and composing Web services (WS). The formal semantics of the composition operators is expressed in terms of Petri nets by providing a direct mapping from each operator to a Petri net construction. In [Cardinale et al., 2013] an overview is presented. It shows how Petri Nets have been used in all phases of WS composition (specification phase, verification, validation, and evaluation phase, automatic selection phase and execution phase). The authors in [Moldt et al., 2004] propose a modeling technique and a framework based on high level Petri nets (reference nets). In the work of [Offermann, 2003] a model is introduced for the dynamic composition of Web services based on reference nets.

### 2.5.1 Web Service Orchestration

The objective of modeling service composition is usually the definition of collaborative processes (business processes or workflows)based on service oriented architecture. The process logic of the orchestration contains the order and the execution conditions to invoke services and manages in a controlled way the messages exchanged between involved parties. Figure. 2.4 illustrates how can different services be orchestrated to a process, which in turn becomes a complex service and can be accessed via its own service interface.

The term *orchestration* is used for workflow (business process) that is executable and interacted with both internal and external Web services. The interaction is done at the message level containing business logic and task execution order. In orchestration, there is control from one party. while choreography is concerned with all parties' involvement and their roles played in the interaction process. In order to meet changing business needs in organizations, Web services orchestration is required to be dynamic, flexible, and adaptable. An orchestration engine is responsible for overall process flows, calling the appropriate Web services and determining what steps to complete as well as separation of process logic and Web services which support flexibility. The Business Process Execution Language (BPEL) is the de-facto standard for the implementation of business processes using Web services. Implemented business processes are classified in several different scenarios: for automating scientific applications and for orchestrating Grid services [Bendoukha et al., 2012b, Turner and Tan, 2007, Emmerich et al., 2005]. [Bendoukha et al., 2012b] provide JASMIN, a framework for the modeling and execution of workflows in Grid-like environments. The modeling is based on the UML activity diagrams and the composition of services is based on BPEL. They also provide a framework to translate the diagrams into BPEL executable processes. In the work of [Emmerich et al., 2005] the authors investigate the use of BPEL for the

specification of scientific workflows. These workflows rely on Grid services, which need to be orchestrated in order to achieve other tasks. A brief introduction to BPEL is given in Chapter 5.

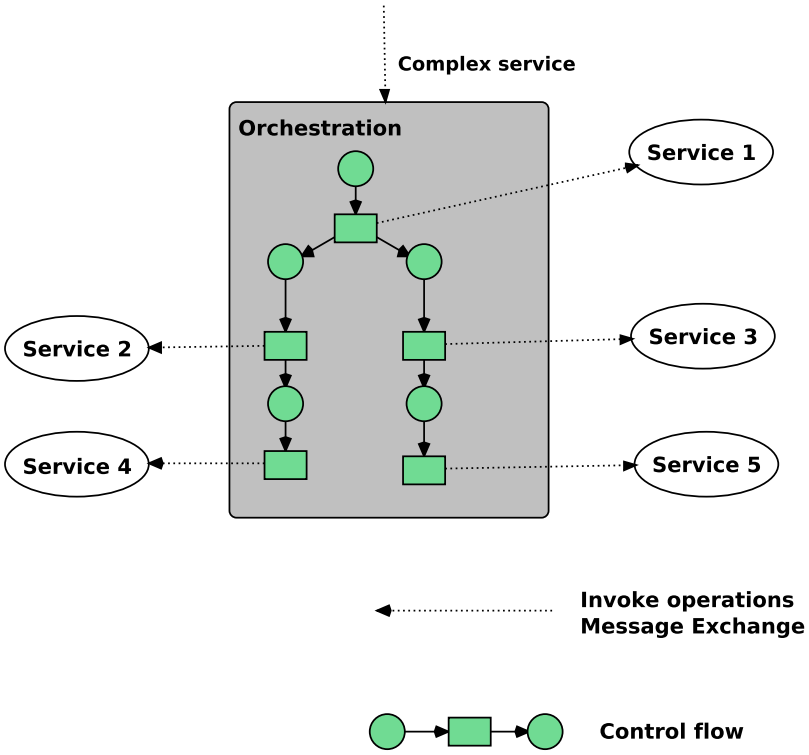



Figure 2.4: Orchestration (adapted from [Peltz, 2003])

### 2.5.2 Web Services Choreography

A choreography (Figure. 2.5) describes the tasks and the interaction of several processes under the cooperation aspect. It can be seen as a description of an abstract communication protocol between the different parties. Each participant has its own process which is referred to as an orchestration process. Choreography is responsible for the message sequence tracking among various parties and sources. Choreography is concerned with all partie's involvement and their roles played in interaction process. Each part acts in an individual way. Here too are some interesting works dealing with the choreography of Web services by Petri nets. Authors in [Caliz et al., 2011] propose a model based on Colored Petri Nets (CPN) to analyze the Web Services Choreography Description Language (WS-CDL). The work consists of generating a CPN model from the WS-CDL document. The CPN model generated is then analyzed using the formal verification environment and simulation capability provided by CPN-Tools. [Vidal et al., 2012] proposes a Petri net-based approach for modeling the choreography of semantic Web services, which are described following the OWL-S specification. The aim of the work is to translate each OWL-S service to a Petri net and to use it for checking the correctness

of its choreography model and/or for facilitating agents (or clients) to coordinate the execution of the service processes through a chore of messages.

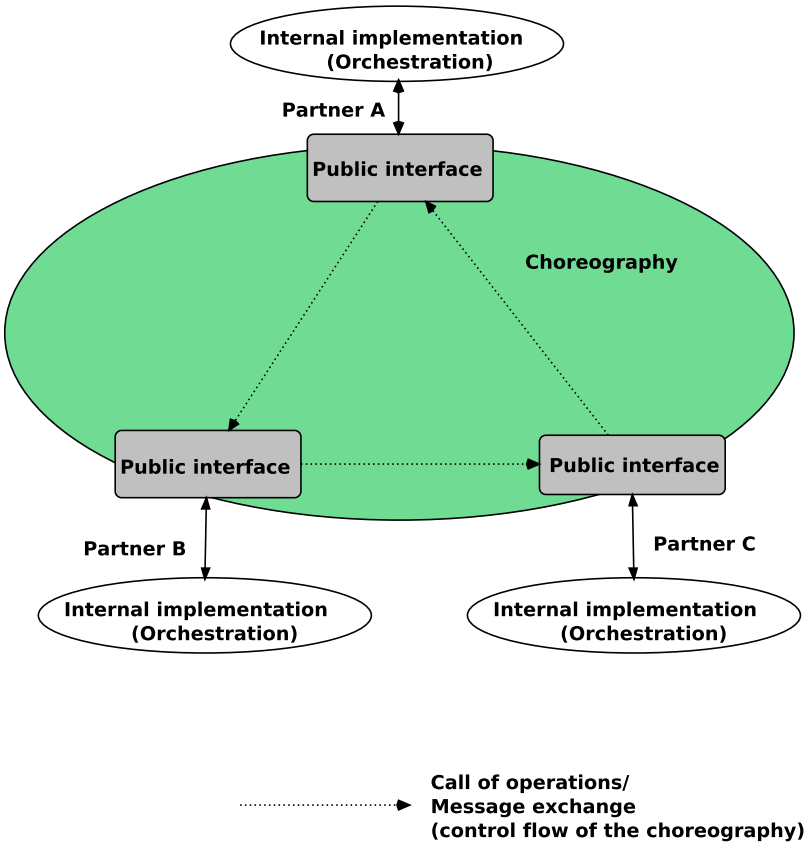



Figure 2.5: Choreography (adapted from [Peltz, 2003])

## 2.6 Web Services and Paose

As stated above, the majority of software systems follows a service oriented architecture. The Paose approach (see Chapter 6) is no exception. In fact, a gateway is implemented in order to interconnect FIPA-compliant multi-agent systems and RESTful Web services. Concretely, it creates a bridge between Petri net-based (Mulan) agents with arbitrary Web service providers or clients [Betz et al., 2011, Betz et al., 2013]. The gateway architecture, its message routing core and its multi-agent interface are modeled and implemented in Java reference nets.

A possible integration is even planed to provide the ICWorkflow plug-in (see Chapter 11) with WebGateway agents. A WebGateway agent has two different kind of interfaces. An *internal* interface which is responsible for the communication between agents. This is performed via the FIPA compliant communication infrastructure of the Mulan framework (see Chapter 6). The *external* interface controls the communication with Web service platforms, generally in form of a Web server. In the case of Mulan

framework, a Jetty[3] servlet container has been integrated. For each MULAN host there is a Web server and for MULAN platform a correspondent WebGateway agent. A MULAN host may include multiple agent platforms. The above description is depicted in Figure. 2.6.

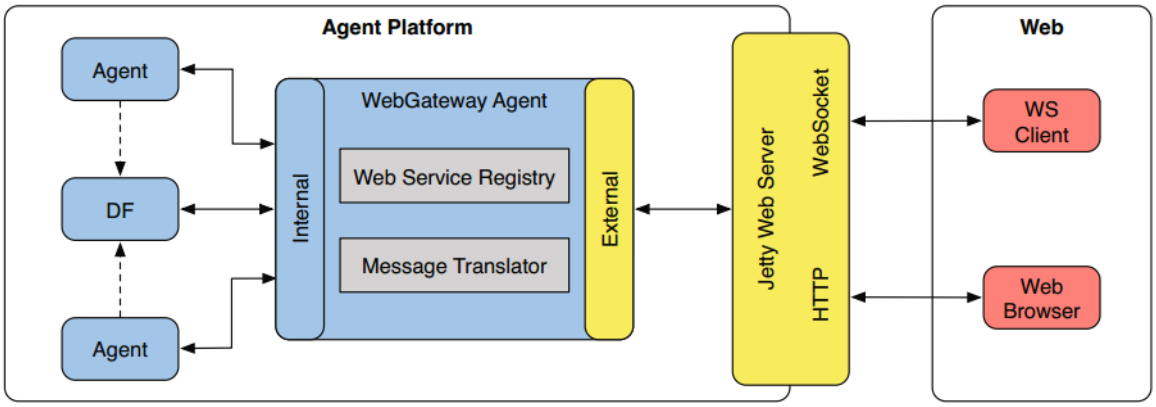



Figure 2.6: The MULAN Gateway Architecture (from [Betz et al., 2011])

## 2.7 Conclusion

In this chapter, the concept of SOC is presented. The different concepts and technologies enabling SOC are outlined. Moreover, in Chapter 8, the deployment of image processing services into the Cloud is based on Web services. Web services, (SOA) and Cloud computing are strongly related to each oder. Web services cover Cloud computing because it uses those services for enabling different connections between several entities. Nowadays, most system architectures follow a service-oriented approach and Cloud computing is no exception. A service provider can be in the Cloud or not. The main system architecture can involve both Cloud and non Cloud service providers. Web services and their related technologies play an important role in the approaches proposed in this thesis. Many proposed solutions have been implemented based on Web services. For instance, the example to illustrate the features of reference nets (see Chapter 5) uses both REST Web services and Cloud-based storage service. In the following chapter, Grids and Clouds are described in details.

[3] http://www.eclipse.org/jetty

# Chapter 3

# Grid and Cloud Computing

In this thesis it is considered that the processes are deployed and executed on external resources rather than on-premises. The reason is seek for more performance and efficiency that can be missing in local sites. As an execution platform, Clouds have been chosen to run long-running processes[1]. This chapter presents an overview of the Cloud/Grid technology with a major emphasis on (Inter-) Cloud computing. It includes definitions and architectures that are relevant for the contributions presented in this thesis. Moreover, a possible formalization of Cloud computing is also discussed.

## 3.1 Introduction

The last few years have seen the emergence of a new generation of distributed systems that scale over the Internet, operate under decentralized settings and are dynamic in their behavior, where participants can leave or join the system at any time. Cloud computing is a recent computing paradigm, in which distributed resources are dynamically provisioned based on a pay-per-use model. There is no doubt that Cloud computing has evolved from Grid computing. Both Grids and Clouds offers huge storage and computing capabilities to different stakeholders. Concerning the work presented in this dissertation, Cloud services are invoked from the process model, exactly from Petri net transitions. Several contributions in form of modeling techniques, deployment and executions mechanisms are provided to (i) specify Cloud-based workflows (by Petri nets) (ii) deploy and (iii) execute processes on Cloud resources. Most of these contributions are presented in Part II. Also in Part II, migration issues are also discussed and solutions to solve them are provided. It is important to mention that the use of Grids is optional and out of scope of the current work. Nevertheless, in the context of a Magister thesis, new concepts have been introduced and tools implemented to specify and execute processes in Cloud/Grid-like systems. The main difference with the current work is the modeling techniques[2]. This chapter aims to give a deep introduction to Cloud

[1] Later on the term workflow is used to designate a process execution life-cycle.

[2] As modeling techniques UML activity diagrams are used. For more information see the work of [Bendoukha et al., 2012b, Bendoukha et al., 2012a]

computing. The information mentioned in this chapter are essential to understand the contributions presented in Part II.

## 3.2 Grid Computing

Although Grid computing is the ancestor of Cloud computing, the main reason to include it in this work is that a Grid node or a service can be also considered as a Cloud resource and provided on-demand. Not all concepts and tools related to Grid computing are presented, but only the notion of resource and topology since they are related to the Inter-Cloud concept addressed in Chapter 10. Typically, applications that are good candidates for a Grid implementation take many hours (possibly days or weeks) to execute (large problem sizes). In general, an application should posses one or more of the following characteristics to be considered a good candidate for a Grid implementation:

- Problem to be solved results in a long execution time.
- Problem to be solved involves many replicated runs of the same fundamental tasks. These types of problems are massively parallel.
- Problem to be solved requires processing vast amounts of data.
- The problem to be solved allows the decomposition into multiple execution units and/or subsets.

### 3.2.1 Grid Concepts

Grid computing was developed in 1998 by Ian Foster and Carl Kesselman [Foster, 2002]. The term Grid has its origin in comparison to the electricity Grid (Electrical Power Grid). Grid computing is a form of distributed computing in which the use of disparate resources such as compute nodes, storage, applications and data, often spread across different physical locations and administrative domains, is optimized through virtualization and collective management.

#### 3.2.1.1 Definitions

The term "Grid" was first formalized in 1999 in the book *The Grid: Blueprint for a New Computing Infrastructure*:

*"A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities"* [Kesselman and Foster, 1998]. This early definition of Ian Foster and Carl Kesselman is very general and focuses on a hardware and software infrastructure. At this time, the Grid was not yet realized. A second refined definition was given by the same authors in 2001:

*"Grid computing is **coordinated resource sharing** and problem solving in **dynamic**, multi-institutional **virtual organizations**"* [Foster et al., 2001].

The final official definition of the Grid computing is given by [Foster and Kesselman, 2004], in which Ian Foster refined the earlier definitions of Grids to include resource sharing among a set of participating parties.

In [Foster, 2002], a Grid is described as a system that: (i) Coordinates resources that are not subject to centralized control, (ii) uses standard open, general-purpose protocols and interfaces and (iii) delivers nontrivial qualities of service Before going too much further, let's take a quick look at a computer's resources.

#### 3.2.1.2 Grid Resources

A Grid computing system enables the *sharing* of a variety of resources and the load across multiple computers to complete tasks more efficiently and quickly.

- **Central Processing Unit (CPU)**: A CPU is a microprocessor that performs mathematical operations and directs data to different memory locations. Computers can have more than one CPU.
- **Memory**: In general, a computer's memory is a kind of temporary electronic storage. Memory keeps relevant data close at hand for the microprocessor. Without memory, the microprocessor would have to search and retrieve data from a more permanent storage device such as a hard disk drive.
- **Storage**: In Grid computing terms, storage refers to permanent data storage devices like hard disk drives or databases.

Grid computing systems link computer resources together forming a integrated system to provide end users with excellent computing power and storage capacities for the special tasks. To the individual user, it's as if the user's computer has transformed into a supercomputer [Foster et al., 2001].

### 3.2.2 Grid Topologies

Topologies are essential to design and configure any Grid system. There are three types of Grid topologies: Intra-Grid, Extra-Grid and Inter-Grid. These topologies are depicted in Figure. 3.1. They are better defined as evolution steps. The complexity when designing a Grid system is related to the number of organizations that the Grid is intended to support. It is also related to geographical and administrative constraints. As more organizations require access to Grid resources, the requirements for increased security, higher availability become more and more complicated. The resource sharing is not only a simple file exchange but rather direct access to softwares, data and other resources, as is required by many collaborative problem-solving in industry, science and engineering.

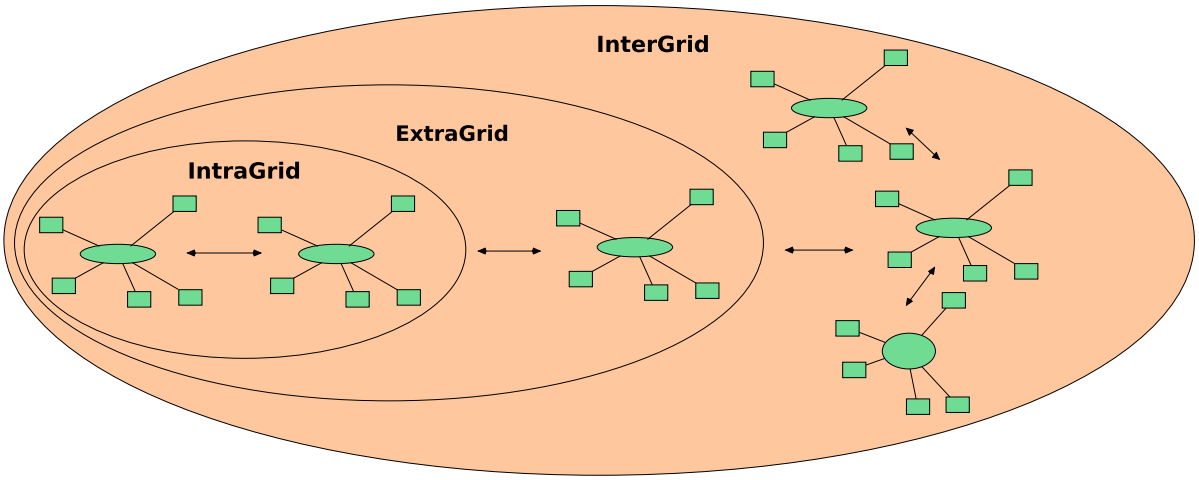



Figure 3.1: Grid Topologies

#### 3.2.2.1 Intra-Grid

The Intra-Grid refers to Grid environments, which are internal to some existing organizations, which define policies to share, access and use the resources. It is the simplest one of the three topologies. It consist of set of basic services within single organizations only. Resources are shared easily between organizations in static fashion. There is one single security provider and the bandwidth is high and always available.

#### 3.2.2.2 Extra-Grid

The Extra-Grid is a combination of two or more Intra-Grid (multiple organizations) and resulting from the pooling of resources across organizational entities that did not follow common architectural guidelines to deploy their IT infrastructures. This assumes that connectivity between the two enterprises is through some trusted service, such as a private network or virtual private network. The main features of such a Grid is the presence of a heterogeneous interconnection network and broadband (LAN / WAN), several different security domains, and a more or less dynamic resources.

#### 3.2.2.3 Inter-Grid

The notion of Inter-Grid consists of the ability that one provider allows access into one other provider. It allows a dynamic integration of applications, resources, and services with partners, customers and organizations that will obtain access to the Grid via the Internet. It Provides the ability to share resources across the network. The main component in this topology is the *Gateway* that mediates the access to the resources of participating Grids. Furthermore, it is also useful for the deployment of applications in different Grids.

### 3.2.3 Grid vs Cloud

From a technological view, Grid computing is the most related technology to Cloud computing. Though, they differ in many aspects, such as the idea of resource sharing. In this section, a short comparison between Grid and Cloud computing from various angles is given.

- Grid and Cloud computing both are designed upon service-oriented architectures
- Grid computing provides access to mostly scientifically used compute resources and data
- Cloud computing is a commercially-driven, recently emerging technology that offers services to access compute power, platforms, and software solutions

In the literature, there is no common Cloud architecture. Nevertheless, Ian Foster et al. [Foster et al., 2008] proposed a four layer architecture for Cloud computing (see Figure. 3.3) based on the five layer Grid architecture (see Figure. 3.2). This architecture has been well-received by many researchers and practitioners.

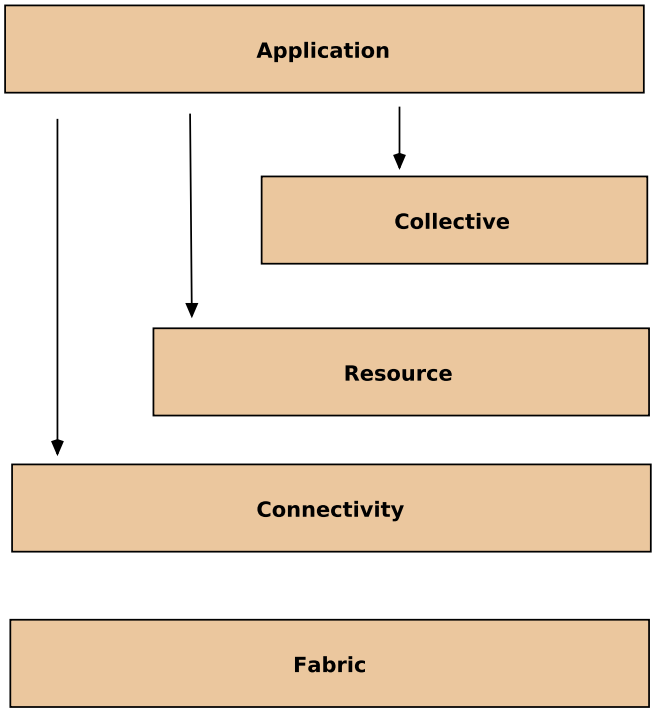



Figure 3.2: Grid Architecture (adapted from [Foster et al., 2001])

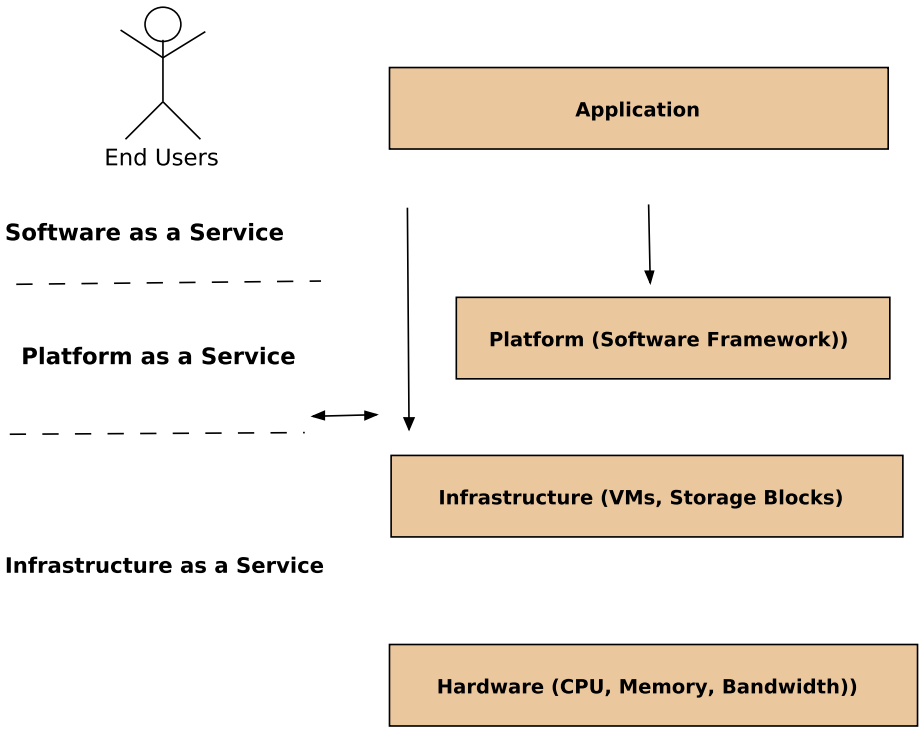



Figure 3.3: Cloud Architecture (adapted from [Foster et al., 2008])

The *hardware layer* consists of raw hardware resources, such as computing units, memory and network bandwidth. It is similar to Grid computing, most of the resources are heterogeneous. The *infrastructure (unified resource) layer* represents the resources which are usually abstracted/encapsulated by using virtualization tools so that they can be exposed to upper layer and end users as integrated resources, for example, a virtual computer/cluster, a logical file system, a database system, etc. On top of the unified resources, there is the *platform layer*, it contains a set of resource management tools, middleware and services to provide a development and/or deployment platform. For instance, a Web hosting environment, a scheduling service, etc. Finally the *application layer* consists of user applications that would be executed in the Cloud, such as Cloud-based workflows, social networking tools and e-commerce.

## 3.3 Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). According to Figure. 3.4, an overview of Cloud computing is presented. In the following section, several definitions and architectures are given.

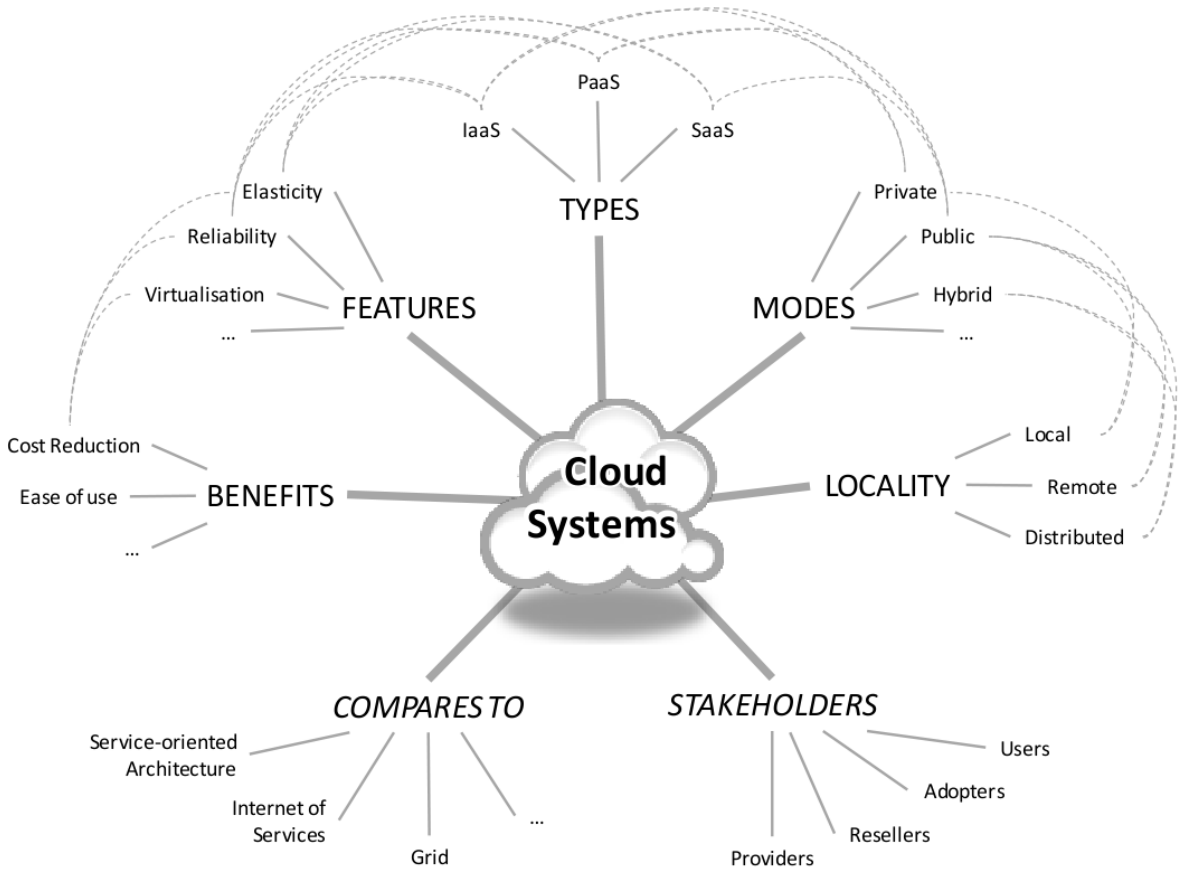



Figure 3.4: Overview of Cloud computing (from [Keith and Burkhard, 2010])

### 3.3.1 Definitions and Architectures

Giving a definition to "Cloud" or "Cloud computing" is not trivial, [Vaquero et al., 2008] reviewed more then 22 different definitions. This difficulty comes from the fact that (i) there are many interpretations of these terms and (ii) that Cloud computing is not a new technology, but it is related to other concepts or technologies such as Grid computing, utility computing, virtualization, etc.

After taking a look at severals definitions and interpretations of the term “Cloud” [Buyya et al., 2009] [Buyya et al., 2010] [Hoffa et al., 2008] [MEDIA, 2008], the definition given by the National Institute of Standards and Technology (NIST) is adopted. It includes all aspects of the Cloud computing features.

NIST’s definition of Cloud computing:

> *“Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”* [Peter Mell, 2011].

While not all Cloud services are freely available, users must in this case to subscribe to the service and establish a Service Level Agreement (SLA) with the service provider defining the Quality of Service (QoS) parameters under which the service is delivered. This issue is clearly expressed by [Buyya et al., 2008]:

> *"A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service level agreements established through negotiation between the service provider and consumers".*

Delivering the computer power as a utility is not a new idea but already mentioned by John MacCarthy in the 1960's [Zhang et al., 2010] or by Leonard Kleinrock in 1969 when he stated [Kleinrock, 2003]:

> *"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of, "computer utilities", which, like present electric and telephone utilities, will service individual homes and offices across the country".*

Ian Foster et al. in [Foster et al., 2008] defines the Cloud as:

> *"A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualiz ed, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet".*

In other words, Cloud computing is:

- Informatics as a service
- Often based on the virtualization
- Composed of three layers (see Section. 3.3.2):

  **Infrastructure** (IaaS: Infrastructure as a Service)

  **Platform** (PaaS: Platform as a Service)

  **Application** (SaaS: Software as a Service)
- "Self-service" or pay as you go
- Abstraction, virtualization and dynamic physical resources allocation.

Cloud computing permits elasticity[3] and flexibility. These features are beneficial not only for providers but also for customers and allow for scaling of applications to

[3]Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time [Peter Mell, 2011].

handle peak loads. Thus, the computing power and storage capacity could be easily improved by provisioning new OS instances or allocating hardware resources to an end user it can appear infinite. Within a Cloud computing environment there are two roles of service provider: The *infrastructure providers* who manage Cloud platforms and lease resources according to a usage-based pricing model, and *service providers*, who rent resources from one or many infrastructure providers to serve the end users [Zhang et al., 2010].

#### 3.3.1.1 NIST Cloud Computing Reference Architecture

Since 2008, the NIST is a recognized organism, that leads an international activity on defining conceptual and standard base in Cloud computing. Definitions and standards provided by the NIST are widely accepted. The following documents create a solid base for Cloud services development and offering:

- NIST SP 800-145, a NIST Definition of Cloud Computing [Peter Mell, 2011].
- NIST SP 500-292, Cloud Computing Reference Architecture [Liu et al., 2011].
- NIST SP 800-146, Cloud Computing Synopsis and Recommendations [Lee Badger and Voas, 2011].

A high level architecture from the NIST [Liu et al., 2011] is depicted in Figure. 3.5. It defines the different actors[4] (Cloud Consumer, Cloud Service Provider, Cloud Auditor, Cloud Broker and Cloud Carrier) and their activities and functions in Cloud computing. In the following, a brief overview of these entities:

[4] An actor is an entity, that can be either a person or an organization.

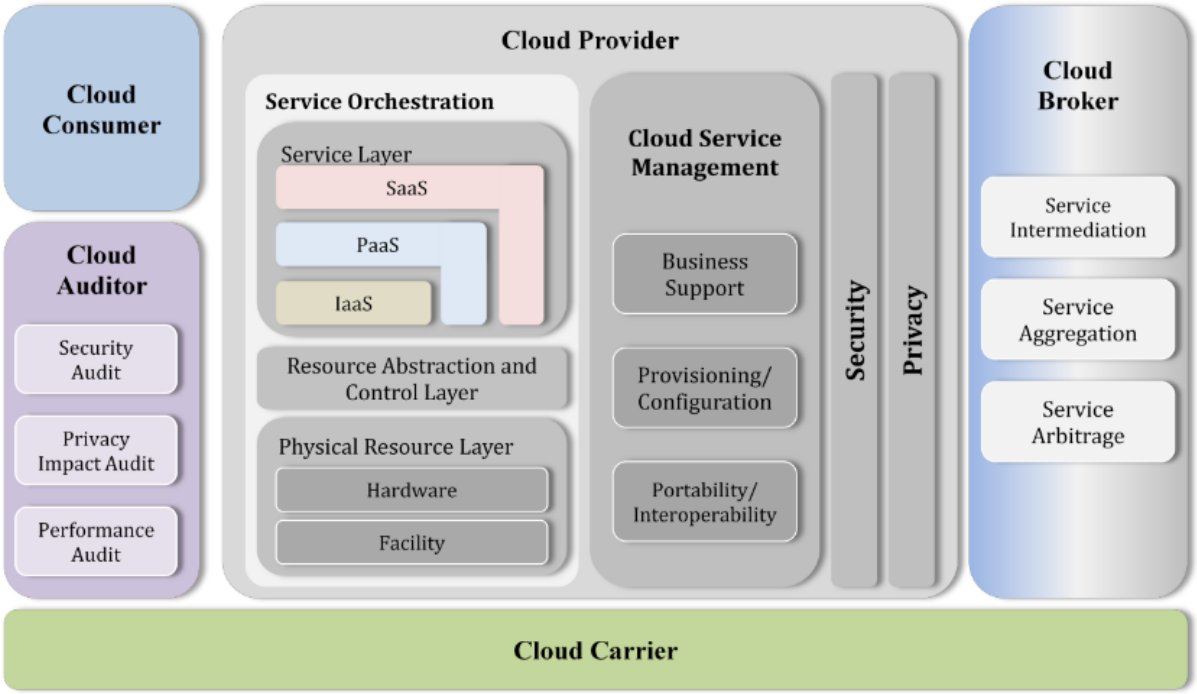



Figure 3.5: NIST Cloud Computing Reference Architecture(from [Liu et al., 2011])

- **Cloud consumer** The Cloud consumer can be a person or organization and may request Cloud services from a Cloud provider directly or via a Cloud broker.
- **Cloud Service Provider** A Cloud provider builds the requested software/platform/ infrastructure services, manages the technical infrastructure required for providing the services, provisions the services at agreed upon service levels, and protects the security and privacy of the services.
- **Cloud Auditor** A Cloud auditor conducts independent audits and may contact the other actors to collect necessary information.
- **Cloud Broker** A Cloud broker is an entity that manages the use, performance, and delivery of Cloud services and negotiates relationships between Cloud providers and Cloud consumers. This entity can provide services in three categories: Service Intermediation, Service Aggregation and Service Arbitrage (see [Liu et al., 2011] for further explanation).
- **Cloud Carrier** It provides connectivity and transport of Cloud services to Cloud Service Providers and their customers.

#### 3.3.1.2 IBM's Cloud Computing Reference Architecture

The IBM's Cloud Computing Reference Architecture [Behrendt, 2014] defines three main roles typically encountered in any Cloud system (see Figure. 3.6):

1. Cloud Service Creator: It uses service development tools to develop new Cloud services. This includes both the development of runtime artifacts and management-related aspects (e.g., monitoring, metering, provisioning, etc.).
2. Cloud Service Provider: runs the created Cloud services and provides them to service consumers, which can also be IT systems. They deliver the Cloud Services to the clients. They consist of basic models (see Section. 3.3.2):
   (a) Infrastructure-as-a-Service
   (b) Platform-as-a-Service
   (c) Software-as-a-Service
3. Cloud Service Consumer or Clients: It can be billed for all (or subset of) their interactions with Cloud services and the provisioned service instances.

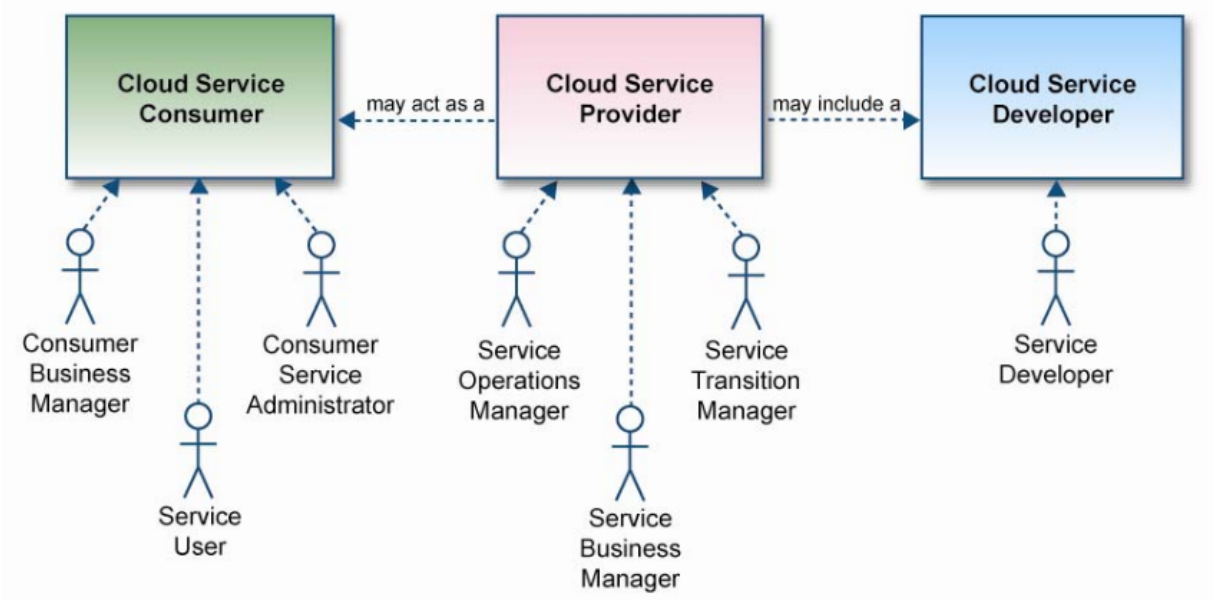



Figure 3.6: IBM's Cloud Computing Reference Architecture (from [Behrendt, 2014])

#### 3.3.1.3 DMTF Architecture and Life Cycle Model

The objective of the Distributed Management Task Force (DMTF) Cloud Management Initiative[5] is to develop inter-operable Cloud infrastructure management standards and to promote adoption of those standards in the industry. The DMTF Cloud Management Initiative has published three interesting white papers about Cloud computing, The first one focuses on Cloud Interoperability [DMTF, 2009], the second on Cloud Architecture and Management [DMTF, 2010a] and the third one on Use Cases for Cloud Management [DMTF, 2010b]. A summary of the important Cloud aspects as well as the relevant terms are outlined [DMTF, 2010a]:

[5] www.dmtf.org

- **Cloud service**: a publicly available service or a private service that is used within an enterprise.
- **Provider interface**: an interface through which Cloud service consumers access and monitor their contracted services.

  The interface also covers service level agreement (SLA) negotiation, service access, service monitoring and billing. This interface is also the interface through which a Cloud service developer interacts with a Cloud service provider to create a service template that is added to the service catalogue.

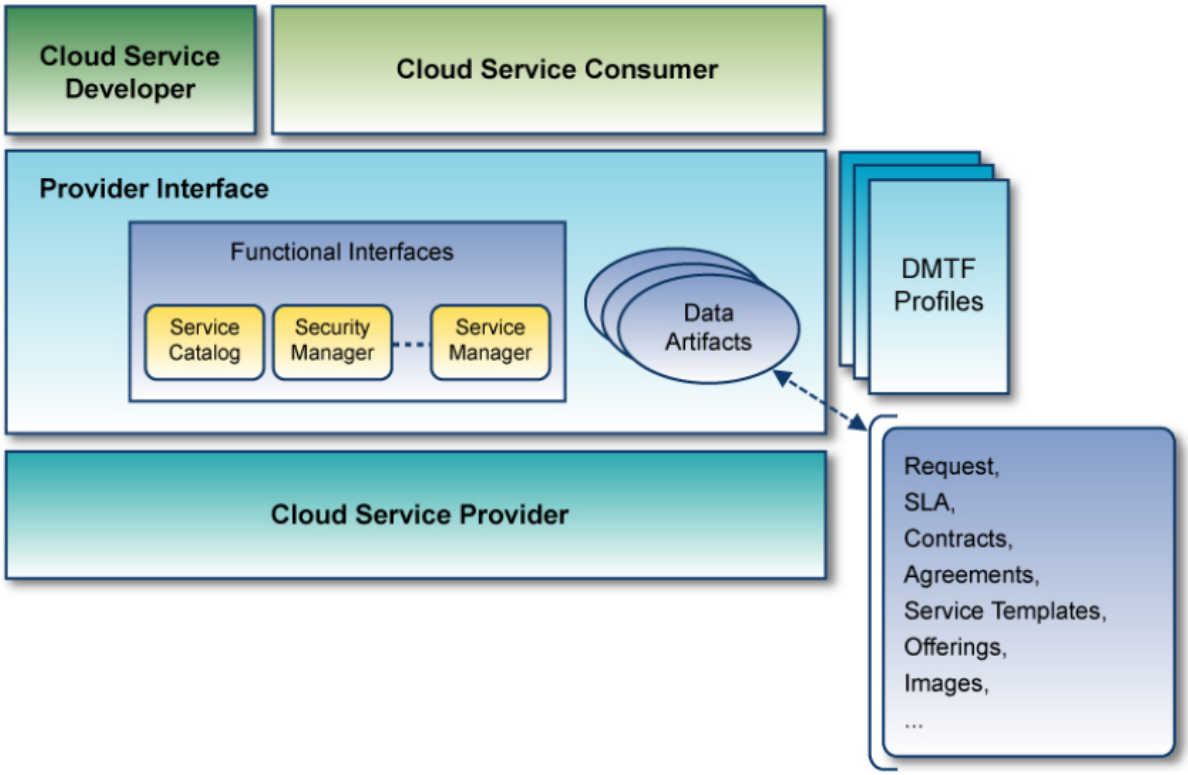



Figure 3.7: DMTF Cloud Service Reference Architecture [DMTF, 2010a]

The Figure. 3.7 shows the key components such as actors, interfaces, data artifact and profiles with the relations among them. The DMTF introduces several stages and artifacts (see Figure. 3.8) around Cloud services and derive use cases and management functionality from these artefacts. The six life cycle stages are:

- **Template**: A developer defines the service in a template that describes the content of and interfaces to a service.
- **Offering**: A provider applies constraints, costs and policies to a template to create an offering available for request by a consumer.

- **Contract**: A consumer and provider enter into a contract for services, including agreements on costs, SLAs, etc.
- **Provision Service**: A provider deploys (or modifies) a service instance per the contract with the consumer.
- **Runtime Maintenance**: A provider manages a deployed service and all its resources, including monitoring resources and notifying the consumer of key situations.
- **End of Service**: A provider halts a service instance, including reclaiming resources for redeployments to support other service.

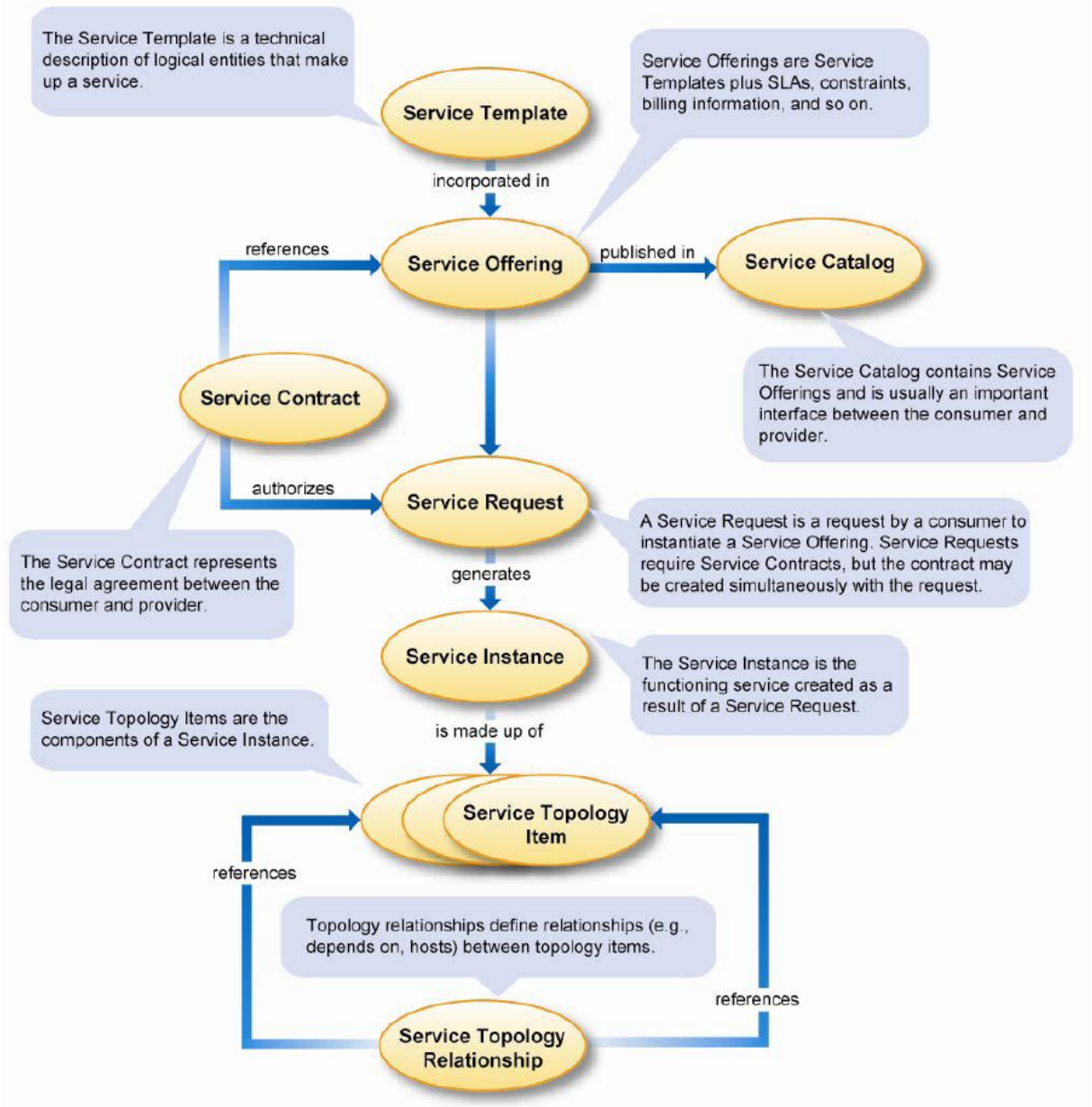



Figure 3.8: DMTF Service Artefacts [DMTF, 2010b]

The use cases introduced in [DMTF, 2010b] are related to the different stages of the latter life cycle. The use cases are specified through textual descriptions, sequence charts, detailed class diagrams of the Unified Modeling Language (UML).

#### 3.3.1.4 CCUCDG Cloud Computing Taxonomy

The role of the Cloud Computing Use Case Discussion Group (CCUCDG) is to *highlight the capabilities and requirements that need to be standardized in a Cloud environment to ensure interoperability, ease of integration and portability* [Group, 2010]. The use cases presented in the white paper should:

- Provide a practical, customer-experience-based context for discussions on interoperability and standards.
- Make it clear where existing standards should be used.
- Focus the industry's attention on the importance of Open Cloud Computing.
- Make it clear where there is standards work to be done. If a particular use case can't be built today, or if it can only be built with proprietary APIs and products, the industry needs to define standards to make that use case possible.

The CCUCDG introduces a Cloud taxonomy (see Figure. 3.9) in which service consumers use the services provided through the Cloud, service providers manage the Cloud infrastructure and service developers create the service themselves.

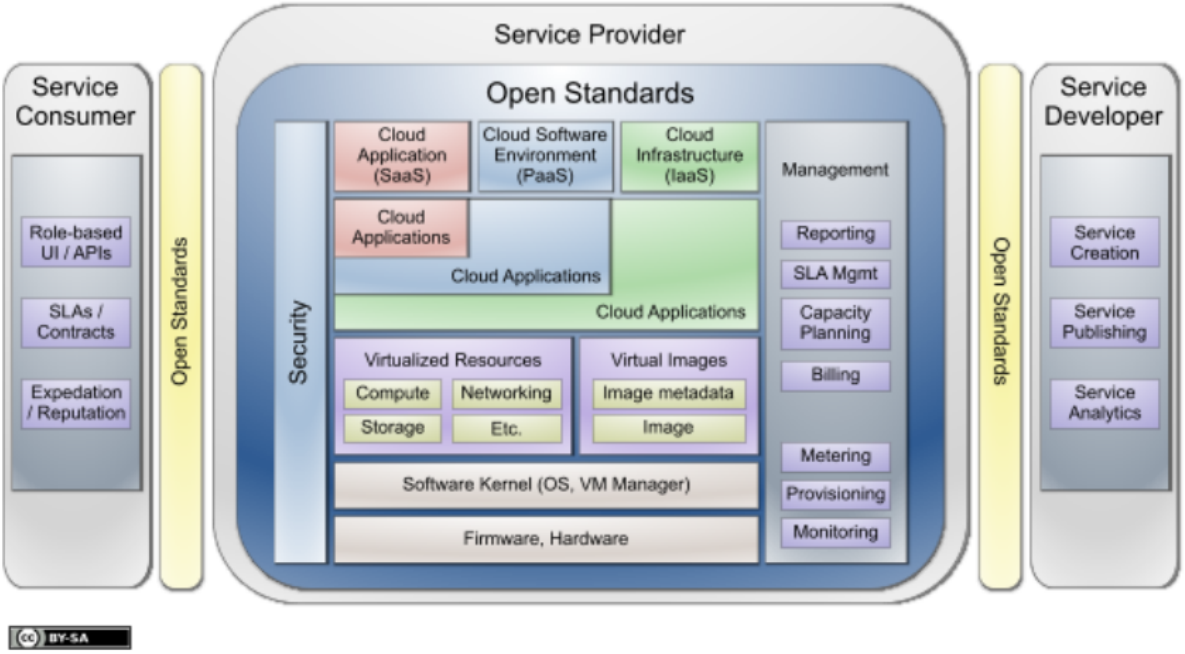



Figure 3.9: Cloud Computing Taxonomy (from [Group, 2010])

Figure. 3.10 shows the three actors categories (Cloud service consumer, provider and developer) defined by the CCUCDG.

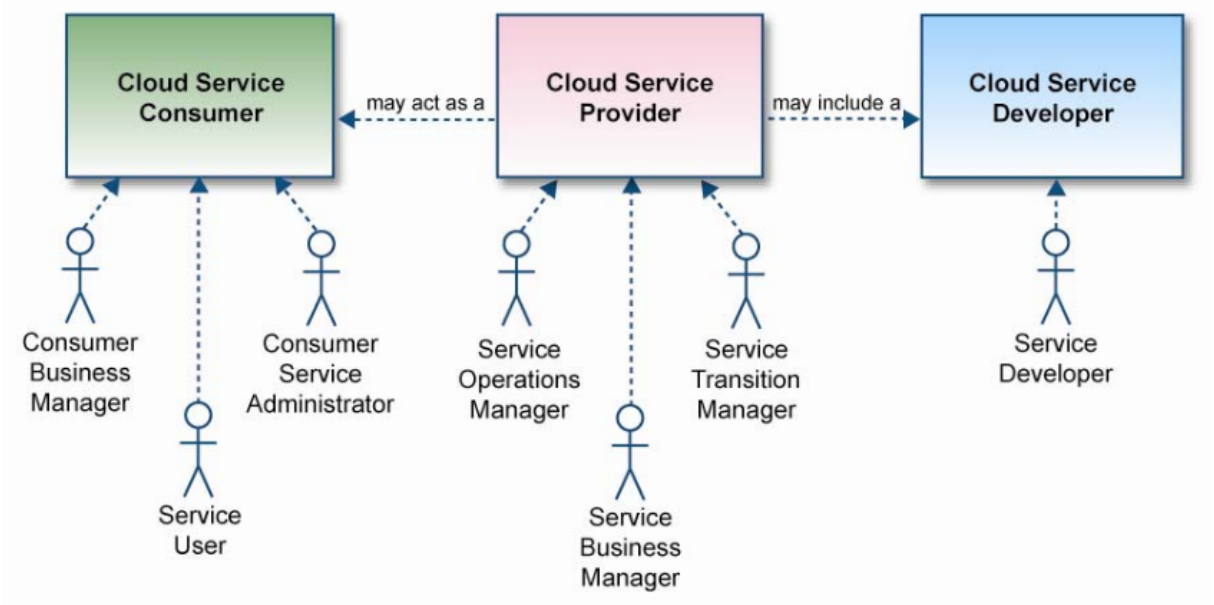



Figure 3.10: Cloud Actors (from [DMTF, 2010b]

1. Service Consumer: This is the end user or company that actually uses the Cloud service, whether it is software, platform or infrastructure as a service. Depending on the type of service and their role, the consumer works with different user and programming interfaces. Some of these user interfaces can be ordinary applications (he does not need to have knowledge about Cloud technology when using the application) or other administrative functions such as starting and stopping virtual machines, etc.

2. Service Provider: It delivers the service to the consumer. It also depends on the type of the service. For SaaS (see Section. 3.3.2): the provider installs, manages and maintains the software. For PaaS (see Section. 3.3.2): the provider manages the infrastructure for the platform, typically a framework for a particular application. For IaaS (see Section. 3.3.2): the provider maintains the storage (databases) and compute (CPUs) capabilities.

3. Service Developer: The service developer creates, publishes and monitors the Cloud service. These are typically "line-of-business" applications that are delivered directly to end users via the SaaS model.

### 3.3.2 Layers of Cloud computing (A Business Model)

Cloud computing can be viewed as a collection of services, which can be presented as a set of loosely-coupled layers. The NIST identifies three service models for Cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [Peter Mell, 2011]. These service models are closely related and can be seen as three layers, as it shown in Figure. 3.11.

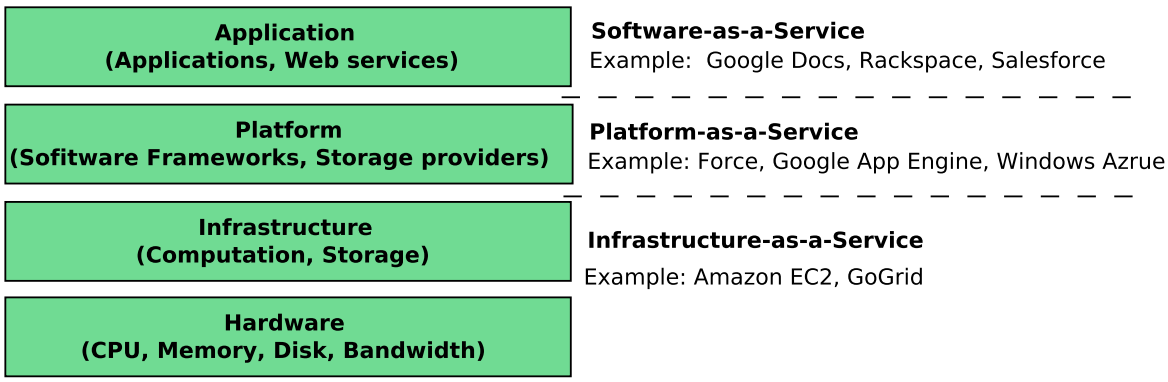



Figure 3.11: Layers of Cloud computing (Adapted from [Zhang et al., 2010])

- **Infrastructure** (IaaS) Also known as the virtualization layer, this is an abstracted layer which gives the users a view on the hardware. the infrastructure layer creates a pool of storage and computing resources by partitioning the physical resources using virtualization technologies such as Xen[6], KVM[7] and VMware[8]. The infrastructure layer is an essential component of Cloud computing, since many key features, such as dynamic resource assignment, are only made available through virtualization technologies. Examples: Amazon Elastic Compute Cloud (Amazon EC2)[9] (which is one of the most widely used infrastructure), GoGrid[10].
- **Platform** (PaaS) This layer allows the users to deploy their applications on the provider's infrastructure, these applications have to be based on programming languages and tools supported by the provider. The user has no control over the underlying infrastructure (servers, storage, operating systems, networks,...) but has control over the deployed applications.
  Examples: Force.com[11], Google App Engine[12], Windows Azure[13].
- **Application** (SaaS) Applications are available at distance and run on the provider's infrastructure and are accessed via a web browser. Users have no control over the infrastructure of the Cloud whatsoever on the technical (data center, network, servers, storage,...) or functional aspects (application version, available features, etc.). Only a limited layer of customization is available to users.
  Examples: Rackspace[14] and Salesforce[15].

Thesis focuses on a special kind of Cloud named Inter-Cloud.

[6]XenSource Inc, Xen (`http://www.xensource.com`.)
[7]Kernal Based Virtual Machine (`http://www.linux-kvm.org/page/MainPage`).
[8]VMWare ESX Server (`http://www.vmware.com/products/esx`).
[9]Amazon Elastic Computing Cloud (`aws.amazon.com/ec2`).
[10]GoGrid (`http://www.gogrid.com`).
[11]Force.com (`http://www.salesforce.com/platform`).
[12]Google App Engine (`http://code.google.com/appengine`).
[13]Windows Azure (`http://www.microsoft.com/azure`).
[14]Rackspace Cloud (`http://www.rackspacecloud.com`).
[15]Salesforce CRM (`http://www.salesforce.com/platform`).

### 3.3.3 Types of Clouds (Deployment Models)

Cloud computing offers several deployment options:

- **Public Cloud** (or external Cloud) (see Figure. 3.12) is a Cloud in which service's providers offer their resources as services to the general public. These resources are dynamically provisioned over the network. A public Cloud is an IaaS, PaaS or a SaaS proposed and hosted by a third party. Amazon, Google and Microsoft offers a public Cloud in which any individual or any company can host their applications, services or data.

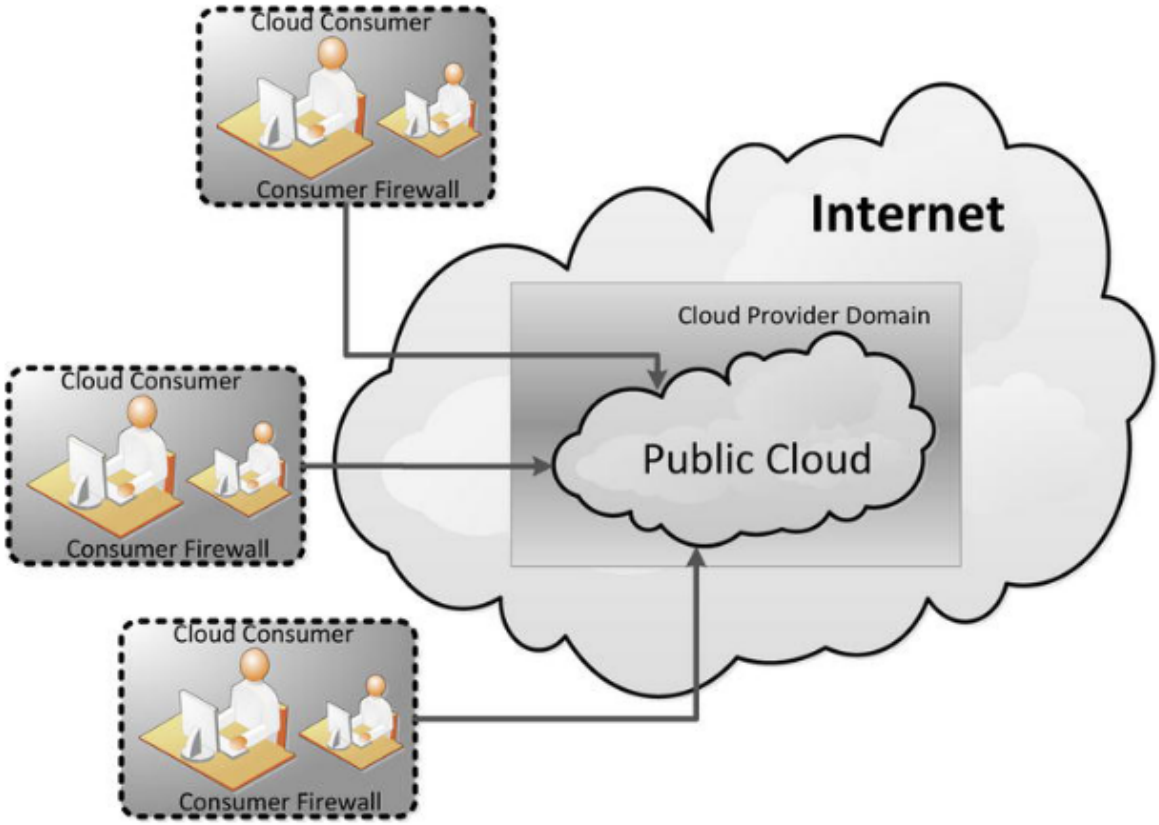



Figure 3.12: Public Cloud (from [Waschke, 2012])

- **Community Cloud** (see Figure. 3.13) the infrastructure is shared by several organizations and supports a specific community that generally has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party [Peter Mell, 2011]. The *Cloud infrastructure* could be either a solely-owned data center or a network (federation or community) of (smaller) data centres [Keith and Burkhard, 2010].

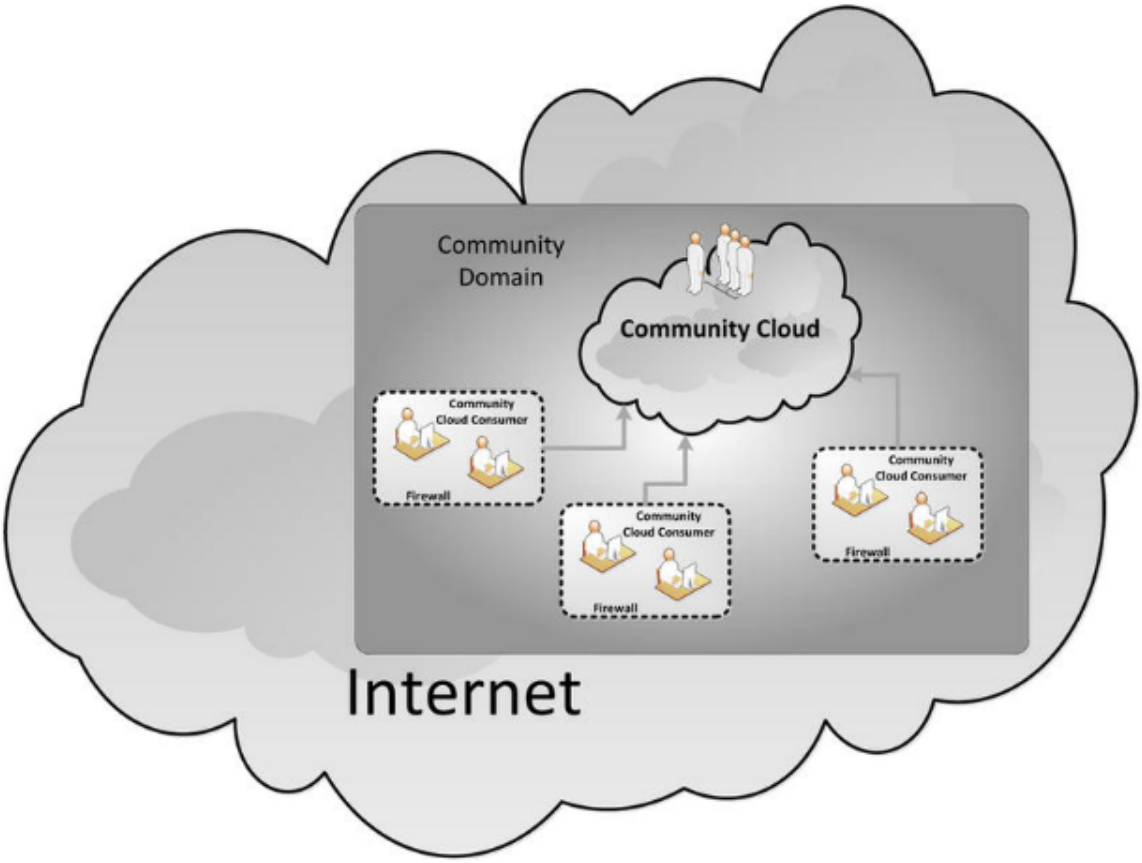



Figure 3.13: Community Cloud (from [Waschke, 2012])

- **Private Cloud** (or internal Cloud) (see Figure. 3.14) in this type of Cloud services and infrastructure are maintained on a private network. The Cloud infrastructure is operated only by a single organization. It may be managed by the organization or a third party, and may exist on the premises or off the premises.

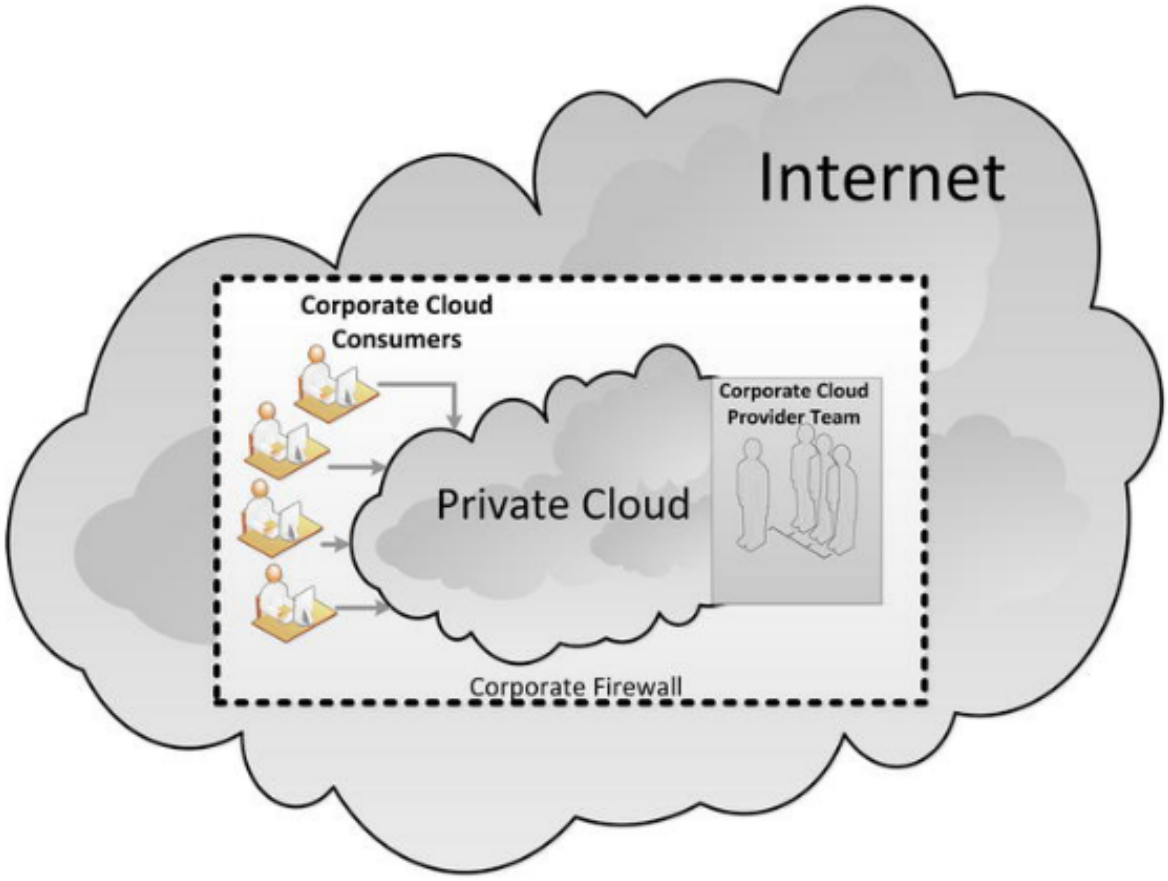



Figure 3.14: Private Cloud (from [Waschke, 2012])

- **Hybrid Clouds** (see Figure. 3.15) a Hybrid Cloud is the use of multiple Clouds, public or private. Applications can be pushed to a public Cloud that will consume the data stored and exhibited in a private Cloud, or to communicate two applications hosted in two separate private Clouds, or consume multiple services hosted in various public Cloud.

  A possible configuration for a hybrid Cloud is represented by a private Cloud that scales out into a public Cloud.

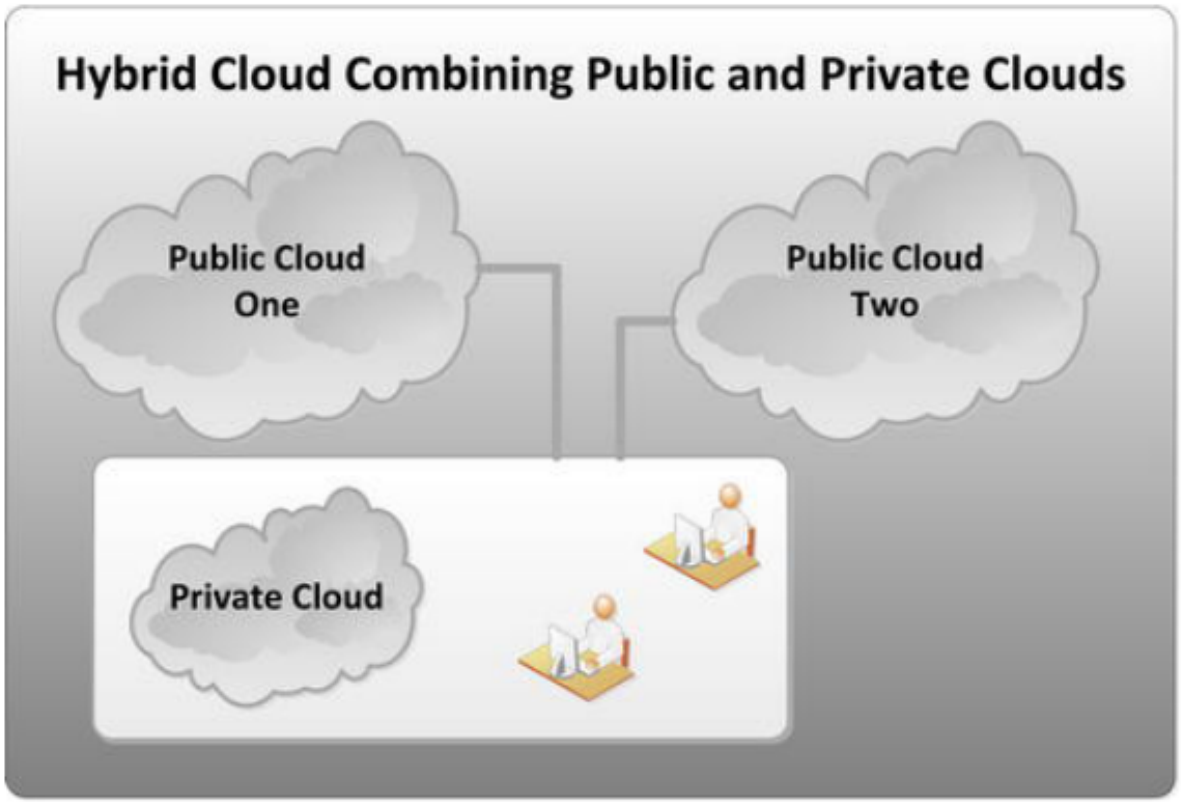



Figure 3.15: Hybrid Cloud (from [Waschke, 2012])

### 3.3.4 Cloud Standardization

Many researchers are working on Cloud computing standards. Loutas et al. [Loutas et al., 2010] identified the following working groups as the most active: CloudAudit[16], the Cloud Security Alliance (CSA)[17], the Distributed Management Task Force (DMTF)[18], the European Telecommunications Standards Institute Technical Committee (TSI TC Cloud)[19], the Open Grid Forum (OGF)[20], the Object Management Group (OMG)[21], the Open Cloud Consortium (OCC)[22], the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA TC)[23], the Storage Networking Industry Association (SNIA)[24], the Open Group Cloud Work Group[25], the Cloud Computing Interoperability

[16] http://www.CloudAudit.org/
[17] http://www.cloudsecurityalliance.org/
[18] http://www.dmtf.org/home
[19] http://www.etsi.org/WebSite/Technologies/GRID_CLOUD.aspx
[20] http://www.ogf.org/
[21] http://www.omg.org/
[22] http://openCloudconsortium.org/
[23] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[24] http://www.snia.org/home
[25] http://www.opengroup.org/Cloudcomputing/

Forum (CCIF)[26], the Cloud Standards Coordination Working Group[27]. The OCCI Working Group of OGF develops mainly a practical specification related to IaaS. This API introduces three fundamental concepts: compute, storage and network. The Cloud Manifesto[28] is an initiative supported by several companies arguing that Cloud computing should capitalize on standards. The DMTF's Open Cloud Standards Incubator (OCSI) and focuses on standardizing interactions between Cloud environments by developing resource management protocols.

The Open Virtualization Format (OVF)[29] is a DMTF standard that describes virtual appliances for deployment across heterogeneous virtualization platforms (i.e different hypervisor), allowing the users to deploy their virtual appliances at every Cloud provider. The CDMI is a SNIA standard for data management specifying a functional manner on how applications create, retrieve, update, delete data from the Cloud. These two standards are the most widely adopted by the Cloud community [Petcu, 2011].

### 3.3.5 Related technologies

Cloud computing is not a new technology, but is related to other concepts and technologies (see Figure. 3.16) such as Grid computing, utility computing, virtualization, etc. These technologies are briefly introduced.

- Grid computing (see Section 3.2) consists of the exploitation of underutilized resources (PCs and servers). It is a kind of distributed computing, permits dynamic resources sharing between participants, organizations and enterprises in order to allow intensive computing applications to be executed or important data volumes to be treated.

- Utility computing defines a "pay-per-use" model for using computing services. Cloud computing implements the idea of utility computing, which was first suggested by John McCarthy in 1961, where computing is viewed as a public utility. It adopts a utility-based pricing model in which users only pay for what they use.

- Virtualization forms the foundation of Cloud technology and provides virtualized resources for high-level applications. Virtualization is the technology that creates a layer between Hardware and operating system. It includes virtualization technologies such as: Xen, KVM and VMware.

- Autonomic computing is an initiative started by IBM in 2001. The objective was to develop computer systems capable of self-management and to overcome the management complexity of IT systems. In other words, the primary goal of autonomic computing is that *"systems manage themselves according to an Administrator's goals ..."* [IBM, 2001].

[26] http://www.Cloudforum.org/
[27] http://Cloud-standards.org/wiki/index.php
[28] http://www.openCloudmanifesto.org/index.htm
[29] http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf

Furthermore, there are other technologies and concepts that help the development of Cloud-based applications and systems such as Web services and Service-Oriented Architecture, workflows, Web 2.0 and Mashups[30] [Keith and Burkhard, 2010].

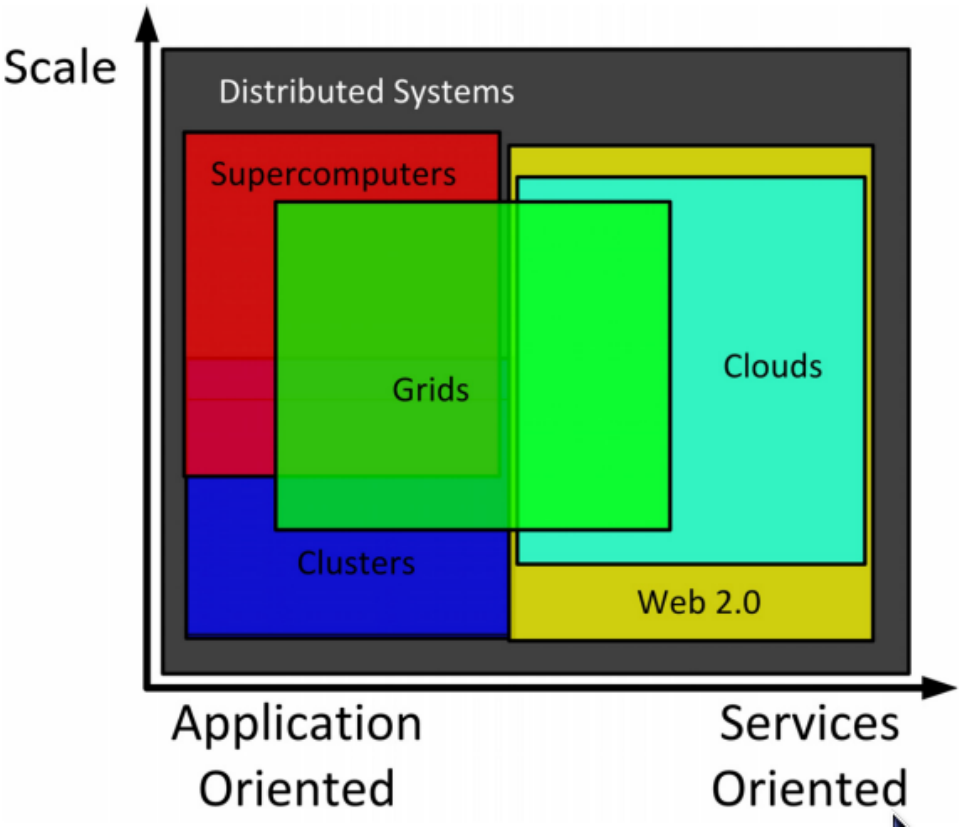



Figure 3.16: Overview of Cloud Computing (Adapted from [Keith and Burkhard, 2010])

### 3.3.6 Formal Models of the Cloud

There is currently a very limited community working on formal aspects of Cloud computing. This due principally to the technical implication of Clouds into the process management . They are tackled only as external resources, that should be mapped to workflow tasks. Although Petri nets have a strong mathematical basis, it is out of scope of this dissertation to provide a formal description of Cloud computing. In Chapter 10, a semantic for a refined Petri net transition called *Cloud Transition* is proposed.In the following section, a state-of-the-art introduction towards defining a Cloud ecosystem by Petri nets (or other modeling techniques) is presented. The proposed model can be in the future extended and improved to establish a robust modeling strategy that takes into account all participating entities. This ranges from clients, workflows to Cloud providers and services.

[30]Mashup is a Web page or application that combines data from two or more external on-line sources. Their data may be obtained in various ways, including, but not limited to APIs and XML feeds.

#### 3.3.6.1 The Axiomatic Cloud Theory

Several formal models of sequential and distributed processes are established. For example: finite state automata, Turing machines and Petri nets. In the following, a mathematical Cloud system, process definitions and Cloud axioms are proposed.

The author in [Weinman, 2011] is the first to introduce the term *Axiomatic Cloud Theory*. He tackles the Cloud environment from a different angle that is not used often. The idea is to abstract Cloud computing participants (clients, providers) using a formal model. Even if this model is suitable for Cloud computing, it can be applicable to other dynamic domains where distributed resources are shared and dynamically allocated and usually priced. However, there exists no transformation of it so far. The model is based on well-known disciplines such as: metric spaces, measure theory, linear algebras, graph theory etc. [Weinman, 2011] defines a Cloud as a 6-tuple structure (S, T, G, Q, $\sigma$, $q_0$) and satisfying five formal axioms (1) commons, (2) location-independent, (3) online, (4) utility, (5) on-demand:

- S is space,
- T is time,
- G = (V, E) is a directed graph,
- V is a set of vertices,
- E is a set of edges, E $\subseteq V \times V$, and (u,v) $\in$ E implies (v,v) $\notin$ E and (v,u) $\notin$ E,
- Q is a set of states where each state combines assignments of resource capacity and demand, resource allocations, node location, and pricing;
- $q_0$ is an initial state; and
- $\sigma$ is a transition function that determines state trajectories over time: mapping resources, allocations, locations and pricing to a next state of resources, allocations, locations and pricing.

Figure. 3.17 shows a possible scenario that can use the previous model [Weinman, 2011]. The figure shows 4 vertices (Atlanta, Boston, Carolyn, Dave) and connected by three edges with correspondent distance. The resources are denoted by <processor,storage> which can have positive or negative values. Atlanta < 5,3 > means that this node has 5 cores and 3 GB and Dave <-2,-2> means that this node needs 2 cores and two GB. The notion of distance plays an critical role regarding the resource allocation, the node Dave changed its location. Hence, conditions change and pricing also changes. Therefore, having intermediary nodes or brokering agents is of high necessity. In this work, the matching process is handled using an SLA-based technique with respect to QoS constraints.

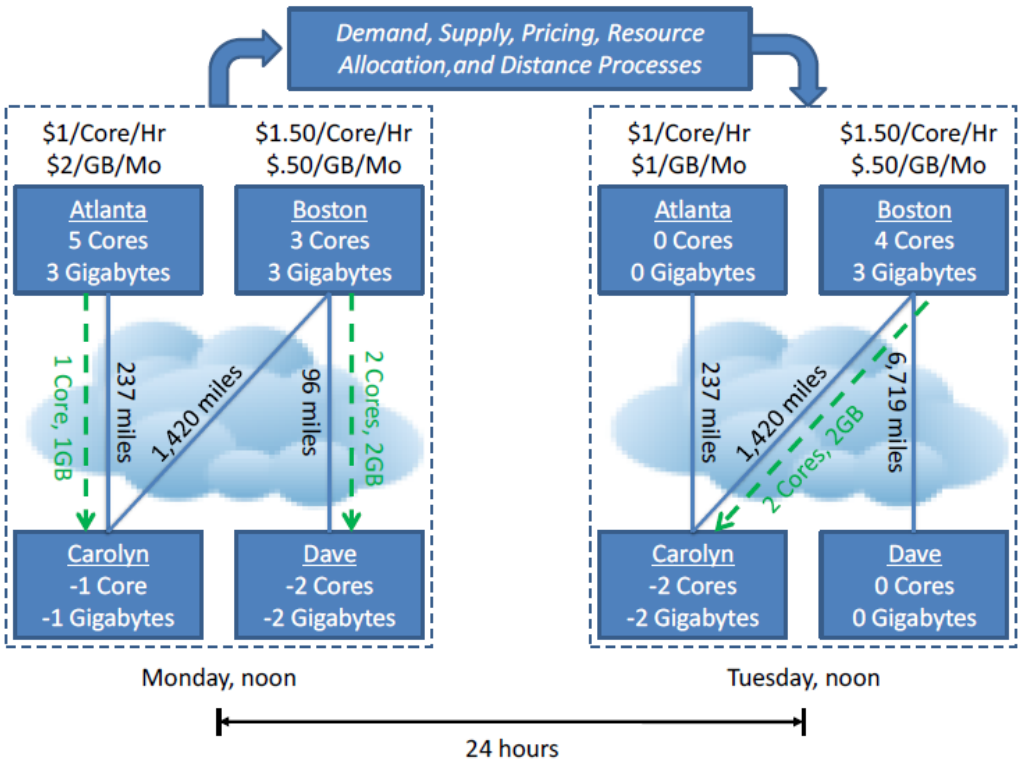



Figure 3.17: Possible Scenario (from [Weinman, 2011])

The main problem that the model in [Weinman, 2011] strives to solve is: matching the demand of the Cloud customers with resources capacity available at the Cloud providers. In this dissertation the problem is also tackled but from different angle including cooperation and collaboration between different partners. Two situations are addressed: Not all Cloud providers are available to all Cloud customers and there is a need to collaborate multiple Cloud services from different Cloud providers to satisfy the customers. In the current work, this idea is explicitly named as Inter-Cloud (see Part II). Some elements of the formal model can be used to enhance the current approach especially the parameters for the Cloud transition, for example the notion of distance between the customer and the provider.

The Cloud scenario is modeled as a directed graph (see Figure. 3.18) where each customer has some level of demand $d_i$, and each Cloud provider $s_i$ has some level of resources $r_i$.

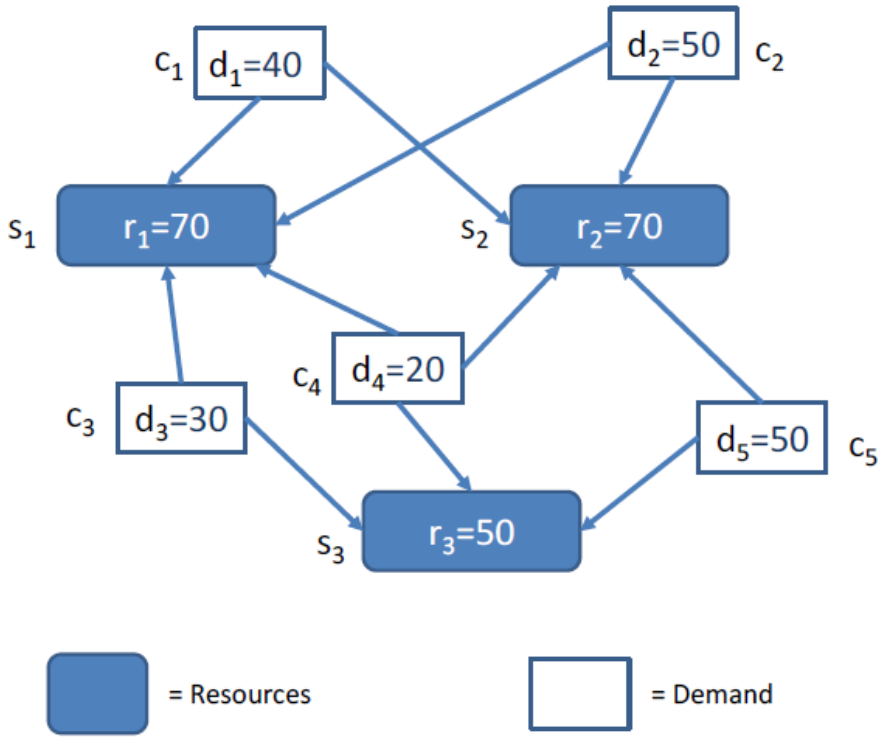



Figure 3.18: The Cloud Computing Graph (from [Weinman, 2011])

The Inter-Cloud scenario can be also modeled using the above model (see Figure. 3.19).

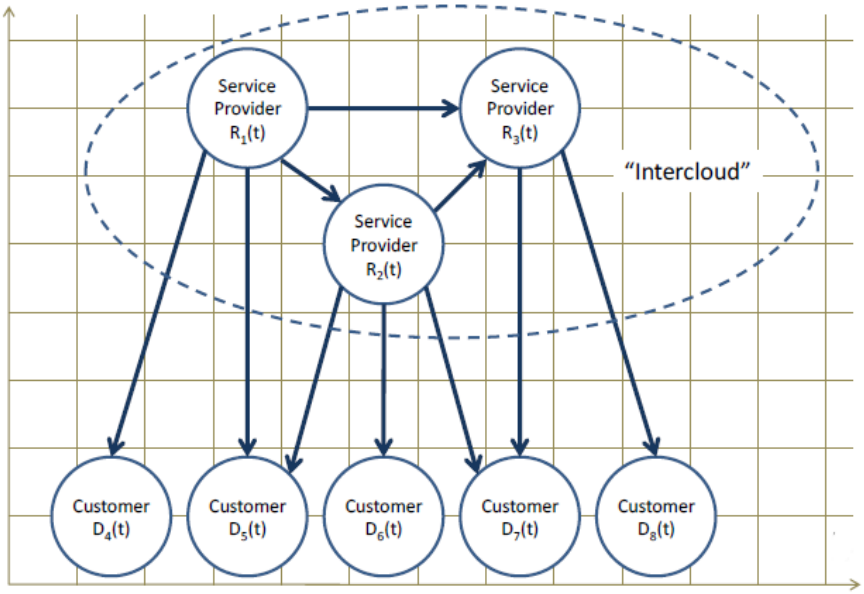



Figure 3.19: The Inter-Cloud Scenario (from [Weinman, 2011])

#### 3.3.6.2 The Inter-Cloud Nets (ICNets)

In opposite to the formal aspects discussed above, a new kind of Petri net constructs are proposed and will be later presented. In this dissertation, solution named Inter-Cloud Nets (ICNets) is introduced. ICNets are predefined Petri net models that allows not only the modeling of Cloud-based workflows but also the management of those workflows in an Inter-Cloud environment. They allows more performance during the execution of processes, avoidances Cloud vendor Lock-in and the interoperability issues. ICNets are explained in detail in Chapter 10.

## 3.4 Inter-Cloud Computing

This section introduces the notion of Inter-Cloud computing which adds a supplementary management layer to the classical Cloud computing architecture.

### 3.4.1 Preliminaries

> *The long term vision of Cloud computing is that IT services are traded as utilities in an open market, without technological and legal barriers. In this Cloud marketplace, Cloud service providers and consumers, trading Cloud services as utilities, play a central role* [Buyya et al., 2013, p.3].

> *..., the discovery of such services is mostly done by human intervention: a person (or a team of people) looks over the Internet to identify offerings that meet his or her needs. We imagine that in the near future it will be possible to find the solution that matches our needs by simply entering our request in a global digital market that trades Cloud computing services [Buyya et al., 2013, p.3].*

> *The existence of such a market will enable the automation of the discovery process and its integration into existing software systems, thus allowing users to transparently leverage Cloud resources in their applications and systems [Buyya et al., 2013, p.3].*

For some years, the Inter-Cloud notion has emerged, which could be seen as Cloud of Clouds [Kelly, 2007]. The notion of Inter-Cloud is discussed much more and cited in many research papers [Petcu, 2011, Papazoglou, 2012a, Bernstein et al., 2009, Nuñez et al., 2011, Buyya et al., 2010]. The ability of one Cloud to participate in managing another Cloud becomes critical to scaling a Cloud. It provide a means for a private Cloud to temporarily use the resources of a public Cloud as part of an elastic resource capacity strategy. It also will make possible to more immediately share functionality, information, and computing resources.

As the short history of Cloud computing shows, Cloud services may be unavailable for a short, or even for an extended period of time; such an interruption of service is likely to impact negatively the organization and possibly diminish, or cancel completely,

the benefits of utility computing for that organization. The potential for permanent data loss in case of a catastrophic system failure poses an equally greater danger.

The other reason lies in the fact that one Cloud infrastructure does not have unlimited resources to satisfy client's requirements and the latter may receive requested services from different Cloud servers [Buyya et al., 2010]. Thus, cooperation between multiple Clouds (Inter-Cloud) is desired to make the Cloud able to satisfy consumer's requests and permits to ensure a high quality of service availability and performance (see Figure. 3.20).

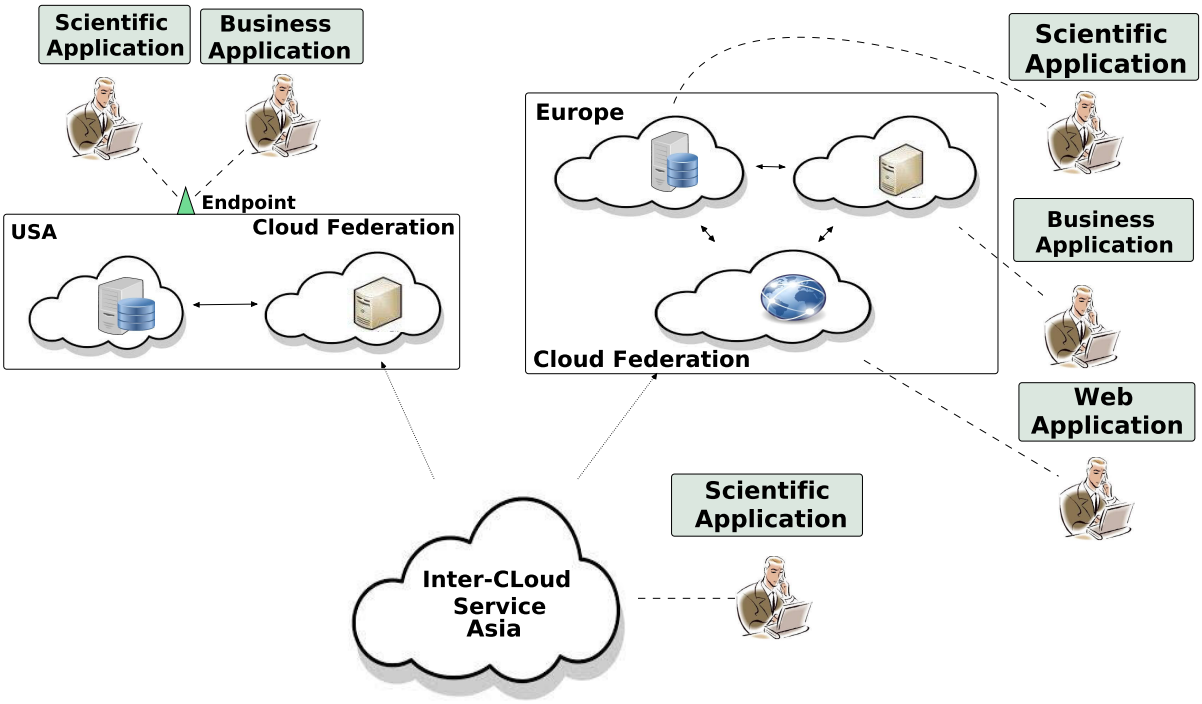



Figure 3.20: Inter-Cloud Vision (Adapted from [Grozev and Buyya, 2012a])

The alternative in this case is to switch to another provider; unfortunately, this solution could be very costly due to the large volume of data that needs to be transferred from the old to the new provider. Transferring tera or possibly peta bytes of data over the network takes a fairly long time, it incurs substantial charges for the bandwidth used and requires substantial manpower.

### 3.4.2 Definitions

As stated in Chapter 3, there are several definitions for Cloud computing and this terminological ambiguity still remain. The Inter-Cloud is no exception to the rule due to it's infancy. The question that also remains is who is attempting to initiate the Inter-Cloud- the Cloud provider or the Cloud client.

The authors in [GICTF, 2010] define the Inter-Cloud as : *A Cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each*

*service, allows on-demand reassignment of resources and transfer of workload through a interworking of Cloud systems of different Cloud providers based on coordination of each consumer's requirements for service quality with each provider's SLA and use of standard interfaces*.

Definitions by [Bernstein et al., 2009] and [Buyya et al., 2010] are implicitly similar. Papazoglou et al. [Papazoglou, 2012b], introduce the notion of federated[31] Clouds without using explicitly the term "Inter-Cloud". He shows how individual services which are provided by diverse Cloud's providers can be combined to offer a Business Process as a Service (BPaaS).

There are other terms to design the Inter-Cloud: *Federation* and *Multi-Cloud*. The term *Federation* is used when a group of Cloud providers decide voluntarily to share their resources among each other [Rochwerger et al., 2009, Arjuna Agility, 2014]. The *Multi-Cloud* notion is used to design a situation when different Clouds are used by a client or a service. This means that Cloud clients are responsible for managing provisioning and scheduling of the Cloud resources. The *Hybrid Cloud* is also used as a type of Inter-Cloud, which means that two or more different Cloud (a private and a public Cloud) are composed together. This situation occurs when for example the local resources are insufficient. In [Petcu et al., 2011, Keahey et al., 2009], the authors invoke the term *Sky Computing*, which almost represent Cloud environments that allow interconnection and inter operation of Cloud resources from multiple providers.

### 3.4.3 Participating Actors

There are different actors participating in the interactions:

**Inter-Cloud users,** who are the actors that request resources and services, such as human users, external applications (e.g., an IT application from a company), internal applications or Cloud providers.

**Inter-Cloud service providers,** who are Cloud providers that are able to offer services or resources to Inter-Cloud users.

**Inter-Cloud identity providers,** who are Cloud providers that are able to authenticate Inter-Cloud users and to share the result of this authentication to Inter-Cloud service providers. They are also responsible for issuing, certifying and managing the identity information of their associated Inter-Cloud users.

### 3.4.4 Standards for Inter-Cloud

In July 2009, the Inter-Cloud Technology Forum (GICTF) was established, for the investigation of possible Inter-Cloud's schemes and to promote appropriate technology standards. It has also the objective of studying different functional requirements and use cases for Inter-Cloud computing. In addition, a number of other initiatives exist

[31] A Cloud federation uses both external (under the control of a vendor) and internal (under the control of the enterprise) Cloud capabilities

in relation to the technical problems in the context of the Inter-Cloud computing. One of the important projects is RESERVOIR (Resources and Services Virtualization without Barriers) which aims to develop systems and service technologies, utilizing virtualization and Grid-technologies across administrative domains [Celesti et al., 2010, Rochwerger et al., 2011]. Another related project is one led by the DTMF (Distributed Management Task Force) Industry Consortium which includes also Cisco, EMC and Microsoft. The objective of the Consortium is to focus on the interoperability and portability issues and standardization of the interactions between different Cloud environments [DMTF, 2009, DMTF, 2010a].

Besides manufacturers eg. Cisco, Vmware, the idea is to establish a standardization for the exchange of computing. In 2011, the IEEE sets off a technical standards called P2302- Standard for Inter-Cloud Interoperability and Federation (SIIF) (see Figure. 3.21). The objective as stated by the working group is to establish a standard as such:

*"The standard defines topology, functions and governance for Cloud-to-Cloud interoperability and federation. Topological elements include Clouds, roots, exchanges (which mediate governance between Clouds), and gateways (which mediate data exchange between Clouds). Functional elements include names spaces, presence, messaging, resource ontologies (including standardized units of measurements), and trust infrastructure. Governance elements include registration, Geo-independence, trust anchor, and potentially compliance and audit. The standard does not address Intra-Cloud (within Cloud) operation, as this is Cloud implementation-specific, nor does it address proprietary hybrid-Cloud implementations"* [SIIF, 2011].

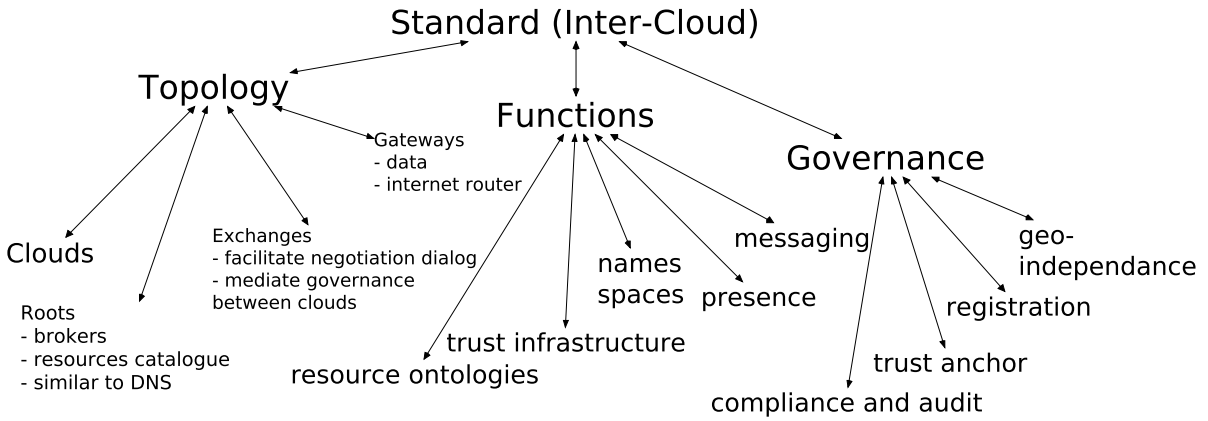



Figure 3.21: Elements of an Inter-Cloud Standard (adapted from [SIIF, 2011])

## 3.5 Conclusion

This chapter provides a deep understanding of what (Inter-) Cloud/Grid computing is. It focuses more on Cloud computing, since it is the chosen storage and computing technology required for the execution the workflows. Nevertheless, approaches from Grid domain can also be exploited to resolve problems in Clouds such as service

composition or workflow scheduling. The chapter starts by a general description of Grid computing, its concepts, topologies and applications. Then, it tackles Cloud computing. Here, architectures, layers and types of Clouds are presented. Furthermore, some formal models of Cloud computing are discussed. The next chapter is devoted to all what concern the notion of workflow: concepts, workflow management systems and the reference model from the WfMC.

# Chapter 4

# Process Management and Workflows

Workflows and their management in Cloud-like environments are the area of interest of this dissertation. Concepts, techniques and tools are provided in order to facilitate the specification of complex Cloud-based workflows and their management (deployment and execution). This chapter aims to provide first an introduction of workflows in general and furthermore Cloud-based workflows. The objective is to define the main workflow concepts and the class of workflow, which is used in this work.

## 4.1 Introduction

Workflows (Figure. 4.1)[1] have been initially introduced in companies to automate administrative procedures in which documents move between different services. Subsequently, they reached the field of software engineering for the management of business processes. For some years, the scope of workflows has been broadened to scientific applications, such as astronomy, bio-informatics, ecology, meteorology, etc, these applications are expressed as *scientific workflows* [Griphyn, 2010, Stevens et al., 2003, Kepler, 2012]. These last are seen as an adaptation of the first models for the conception of distributed applications, typically intensive computing on massive data and for dynamic allocation of resources. Scientific workflows are characterized as long-running and compute/storage intensive. With the significant advance of distributed systems, the Cloud technology is the most suitable technology for running these kinds of workflows. On the one hand, Clouds afford workflows with a considerable number of resources (software and hardware) with easy access to support their deployment and execution. On the other hand, workflows serve concepts and techniques for the Cloud to support processes in all stages. For instance, workflow systems provides development tools to build up applications by using modeling techniques (Petri nets or DAG based modeling) instead of script-based workflow specification.

In this dissertation, several prototypes cover the remote sensing domain, especially processing of satellite imagery, which are considered as *scientific workflows*. Therefor, a tool named RenewGrass has been implemented (see Chapter 8). Its role is to support

[1] This workflow is already explained in the introduction (see Chapter 1).

users to model and execute their image processing workflows by Petri nets and using external GIS software. In addition to the definitions and the description of different workflow types, an overview of the theme workflow is given. It includes supplementary the general structure of a Workflow Management System (WfMS). Furthermore, a state-of-the-art introduction is provided and addresses the notion of Cloud (Grid)-based workflows. In this chapter, the objective is not to lead a deep study of workflow technology. The objective is to put the emphasis on fundamental concepts of workflow.

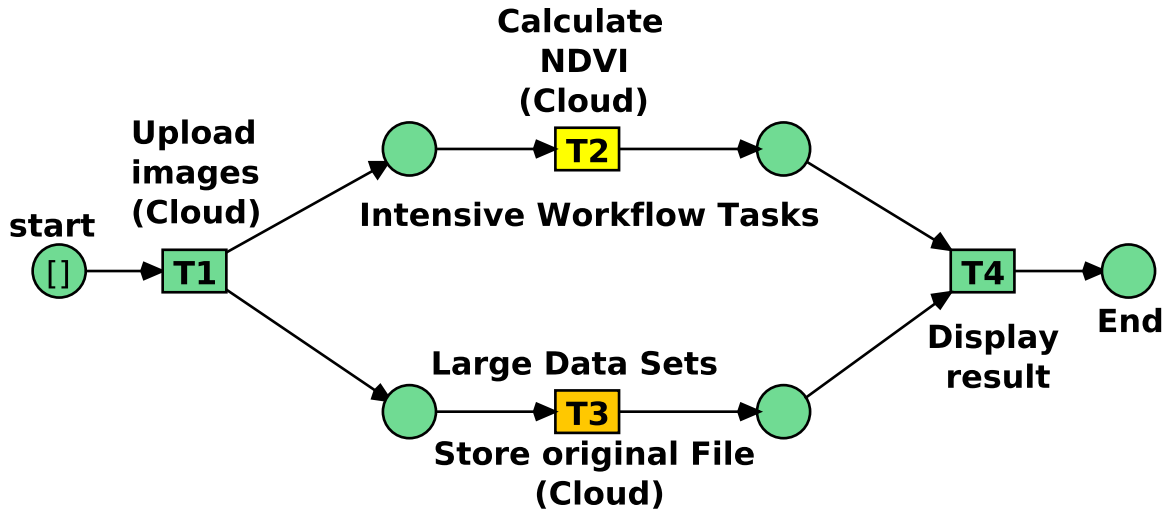



Figure 4.1: Example of a Workflow

## 4.2 Definitions

The workflow concepts are summarized by the "3Rs" of Marshak [Marshak, 1994]:

- **Routes** are the flows or decisions that connect the steps and define the path that an entity will take throught the process. Routing or movement of documents, information or tasks was the first major function of workflows.
- **Rules** are the second major function of the workflow. It is complementary to the first concept as far as the route of a process depends on rules, which define both the nature of information and method of transition from one activity to another. These rules can be simple or complex, but they are essential in the workflow execution. Generally speaking, they take care of delivering the right piece of work to the right resource at the right time.
- **Roles** (the capability of various participants) the third major function of the workflow is the assignment of roles to the workflow's users (who complete steps in the process). A role is associated with the realization of one or more tasks. It can be assigned to several users and one user may perform several roles.

The following sections present some of the basic terminologies, which are mentioned in this thesis. Most of the terminologies are taken from the Workflow Management Coalition (WfMC)[2]. The WfMC established a document containing technical definitions used to describe the concepts and a general structure of a workflow management system [WfMC, 1999].

### 4.2.1 Process

A *process* is a formalized view of a business process, represented as a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common goal. It consists of a number of tasks (which are connected in a form of a directed graph) that need to be carried out and a set of conditions that determine the order of the task.

### 4.2.2 Task

A *Task* is a logical single unit of work that is carried out as a single whole by one resource[3]. Four distinct types of task are denoted: *atomic*, *block*, *multiple-instance* and *multiple-instance block*. A definition has been given to these tasks by [Nicholas Charles, 2007]. An *atomic task* has a simple, self-contained definition (i.e. one that is not described in terms of other tasks) and only one instance of the task executes when it is initiated A *block task* is a complex action which has its implementation described in terms of a *subprocess*. When it starts, it passes control to the corresponding *subprocess*, which when finished gives the control back to the *block task*. The *multiple-instance block* is the one that may have multiple distinct execution instances running concurrently within the same process instance. Finally, a multiple-instance task is a task that may have multiple distinct execution instances running concurrently within the same process instance. Each of these instances executes independently.

### 4.2.3 Work Item

A *Work Item* is the term to design the invocation of a task. Generally there is a Work Item initiated for each task in a given *case*. However, in case of a multi-instance task, there may be several associated work items created when the task is instantiated.

### 4.2.4 Case

A *Case* is the executing instance of a process model. Furthermore, there may be multiple cases of a particular process model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other.

[2]The WfMC has been founded primary to provide a reference model to develop a WfMS with the objective that vendor-independent modules, which are linked together and operate with each other.

[3]A resource is a generic name for a person, machine or software that can perform specific tasks.

Another important term is the *Activity*, which designs a *Work Item* being executed by a specific resource. The authors in [van der Aalst and van Hee, 2004] present the relation of all these concepts. This is shown in Figure. 4.2. Work items link cases and tasks. Activities link cases, tasks, and resources.

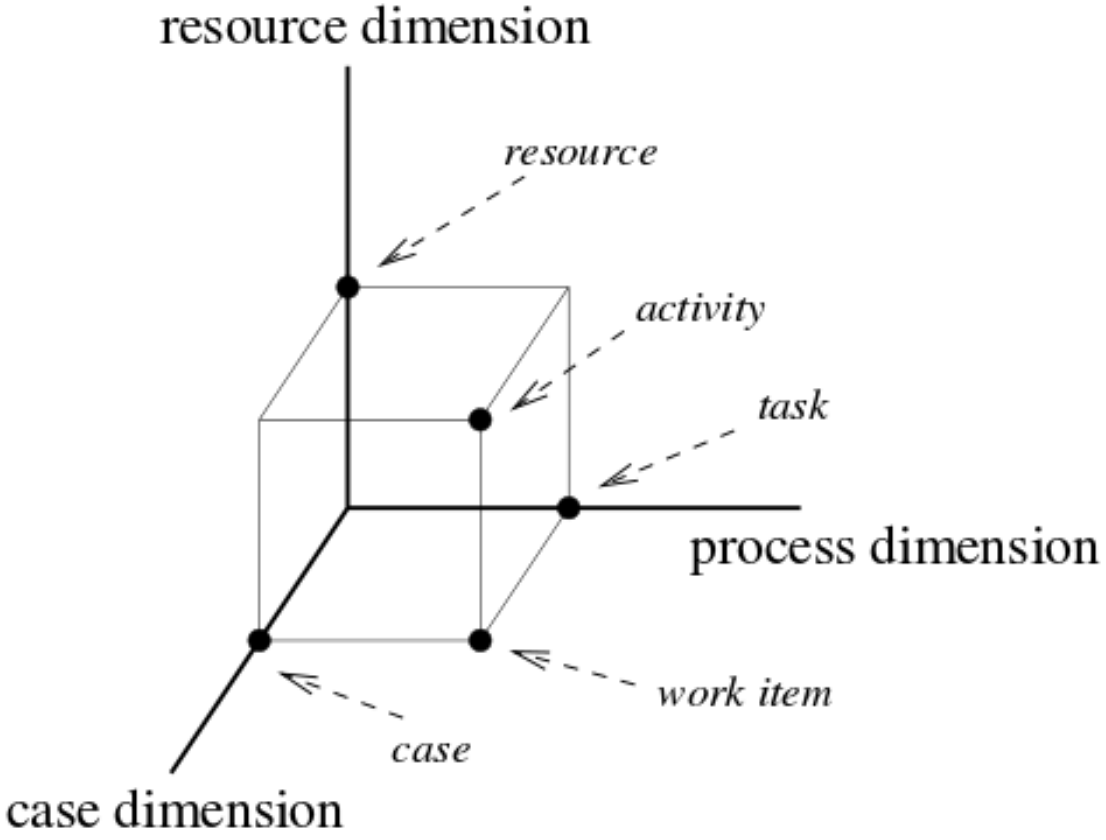



Figure 4.2: A Three Dimensional View of a Workflow
(from [van der Aalst and van Hee, 2004])

### 4.2.5 Business Process

A business process is a set that includes one or more linked procedures or activities. which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships [Hollingsworth, 1995]. Examples of business processes are: Bank Loan, Application for Building Permit submitted to a public authority or the development of a software. It is a business transaction of business entity with a defined *start* and *end*, which may take a different path between the start and the end, respectively, and thus different functions are executed. A business process is initiated by an event and is a sequence of activities that are logically connected. The flow of the business process is influenced by the input of required data entries as well as internal and external events. These entries are carried out by the activities.

### 4.2.6 Workflow

A business process is technically supported by a workflow. A Workflow is defined by the WfMC as [Hollingsworth, 1995]: *"The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules".* That is, not all modeled activities must be supported by a WfMS, but can also be organizationally performed by deployed software tools. A workflow runs always according to the same or at least a similar schema. After a successful execution of the activities, the workflow must return to a defined state. The activities are performed by different users (roles), which are provided by necessary software tools and informations.

The WfMC distinguishes four basic workflow types:

- *Ad-hoc Workflow*: supports single or strongly varying processes that are little structured and not predictable.
- *Collaborative Workflow*: supports the joint development of a result; this term is also used as a synonym for *groupware*. It has high business value to the company and involves a single large project and possibly many individuals.
- *Administrative Workflow*: supports structured routines that are not strategic, rarely critical in time and cost. It refers to enterprise activities such as internal bookkeeping, database management, and maintenance scheduling,
- *Production Workflow*: supports fixed structured and pre-definable operations, most of which are time critical and strategic.

### 4.2.7 Scientific Workflows

In the current work a special kind of workflows called *Scientific Workflows* is addressed. A scientific workflow is a flow of tasks, mostly computational tasks, that are part of a scientific experiment. Scientific workflows are usually executed on distributed systems because of their great deal of demand of computation power [Qin and Fahringer, 2012]. Scientists and scientific applications impose new requirements on the employed workflow technology. One of the obstacles that the application of the traditional WfMS to scientific workflow faces, is that a scientific workflow is mostly considered data/compute centric and time consuming. The major goal of scientific workflow is enabling scientists to focus on domain-specific (science) aspects of their work, rather than dealing with complex data management and software issues. They deal with huge amounts of data and/or complex calculations and hence utilize large storage capacities and computing resources. They do not have a rich control flow structure. The control flow found in business workflows may not be expressive enough for highly concurrent scientific workflows and data. Thus scientific workflows are often executed in a cluster or Grid/Cloud environment. There are several techniques and tools developed for business workflows that can be used for scientific workflows. Nevertheless, there are differences between the two concepts. There is obviously a gap between the features

business workflows provide and the requirements scientists and scientific applications have [Sonntag et al., 2010b]. There are a several research works to exploit the business workflow technology in the scientific field [Sonntag et al., 2010a, Wassermann et al., 2007]. The objective of this section is to give an overview of scientific workflows. Ludäscher et al. define scientific workflows as:

> *"networks of analytical steps that may involve, e.g., database access and querying steps, data analysis and mining steps, and many other steps including computationally intensive jobs on high performance cluster computers."* [Ludäscher et al., 2006].

In [Qin and Fahringer, 2012] the authors give a detailed definition of scientific workflows. They describe it as: *"a distributed application that consists of a set of activities that are processed in a well-defined order to accomplish a specific goal."* A scientific workflow $w$ is formalized as a pair:

w = (A,$\overrightarrow{D}$), where A is a set of activities, and $\overrightarrow{D}$ is a set of dependencies. Each dependency $\overrightarrow{d} \in \overrightarrow{D}$ is either a control flow dependence or a data flow dependence associated with an ordered pair of activities ($a_m$,$a_n$), where $a_m$,$a_n \in$ A.

An *activity* $a \in$ A in a scientific workflow is a computational task. It may refer to a computational entity, which will be invoked or executed when the scientific workflow is executed.

A *computational entity* is a logical resource such as a Web service, an executable, a shell script, a software component, a Java class, etc. that can be assembled in a scientific workflow as an *activity*.

The *composition* of a scientific workflow refers to the process of the identification of the activities of a workflow, and the establishment of necessary control flow and data flow dependences among identified activities. Such that a specific goal, e.g., to get resulted NDVI image of a satellite imagery (see Chapter 8), can be fulfilled in case the scientific workflow is correctly executed.

## 4.3 Workflow Management System

A WfMS is according to the WfMC subdivided in three functional areas (see Figure. 4.3). These areas are oriented towards the creation, operation and control of a process. A distinction is made amongst these functionality: *Build-Time*, *Run-Time Process Control* and *Run-Time Activity Interaction*. With Build-Time functionalities, processes are translated from the real world into a computer-readable definition, this is performed after the process is analyzed, optimized and modeled. For the instantiation and the management the process is the Run-Time Process Control responsible, and for the interaction between the user and the computer-supported process, the Run-Time Activity Interaction. The Workflow Reference Model defines these functions (see Section 4.3.1) and gives a good overview of what the WfMC aims.

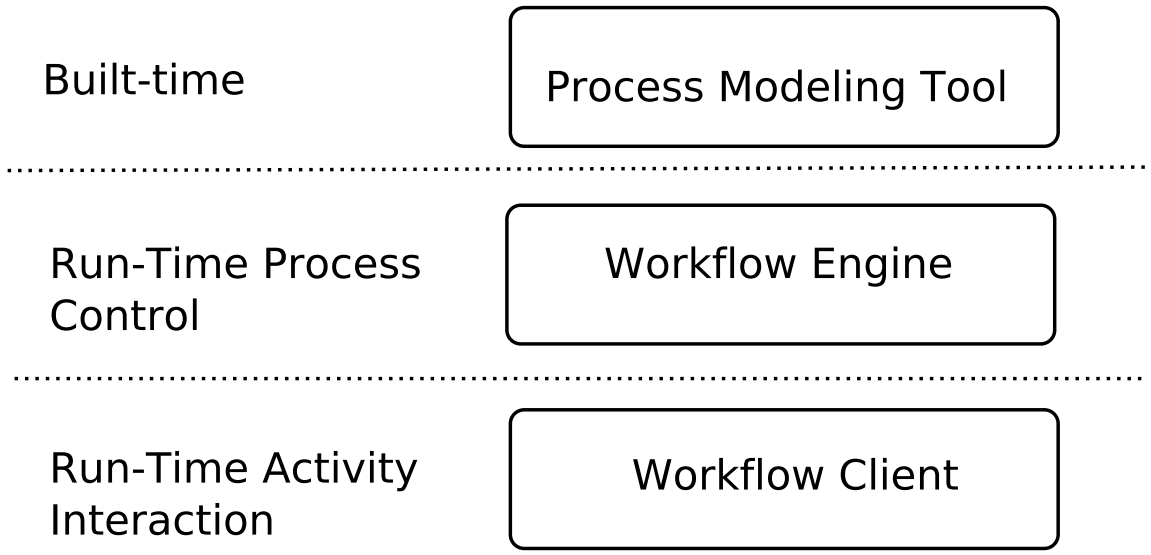



Figure 4.3: WfMS Character

WfMS are increasingly being used to manage business processes associated with distributed global enterprises. A WfMS[4] supports the functionality of workflows through three important modules: modeling, execution and monitoring. The first module provides a set of modeling and definition tools needed to define the following components: activities, entities as well as the control flow between activities and the pre-conditions or post-conditions. The nesting of workflows is also part of this module.

When designing a WfMS, the challenge is to design an environment in which various technologies, from databases to distributed processing, must be integrated in a simple and flexible way. In fact, every technology involved in the operation of a workflow has its structural and functional characteristics. Therefore, issues of interoperability[5] arise again.

A definition of a WfMS is given by the Workflow Management Coalition (WfMC):

> *"A Workflow Management System is the computerized execution of a process definition."* [Hollingsworth, 1995].

There are two main concepts: process definition and computerized execution. Moreover, the WfMS is a software entity that can be used to support the definition, management and execution of workflow processes.

A WfMS is also defined as:

> *"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."* [Lawrence, 1997].

[4]In this work, there is an emphasis on *processes perspective* rather then data and resources

[5]As general term, interoperability is a property referring to the ability of diverse systems and organizations to work together (inter-operate).

There are many advantages by using a WfMS, these can be summarized as follows:

- Ability to visualize the overall process and interdependencies amongst various tasks,
- Automation of the processes, and
- Automated coordination and collaboration among various business entities.

The workflow system is adaptable in three ways: i) from the process perspective (the most dominant one) in which the process model can be changed dynamically and ii) from the resource perspective, the choosing of a particular resource could be done at run time based on the history data and iii) from individual task/activity point of view where an appropriate service can be invoked on the fly.

### 4.3.1 Workflow Reference Model

The aim of a WfMS is to define business processes in order to support their computerized execution. To implement this functionality, the WfMC defines a WfMS architecture that is widely adopted. Figure. 4.4 shows the basic components of the reference model of a workflow and the interfaces among these components. The reference model is considered to be an effort for standardize terminology in the area and define a series of interfaces for various aspects of workflow systems thus vendors could adopt to promote the opportunity for interoperability between distinct offerings. A brief overview about the WfMS architecture is given below.

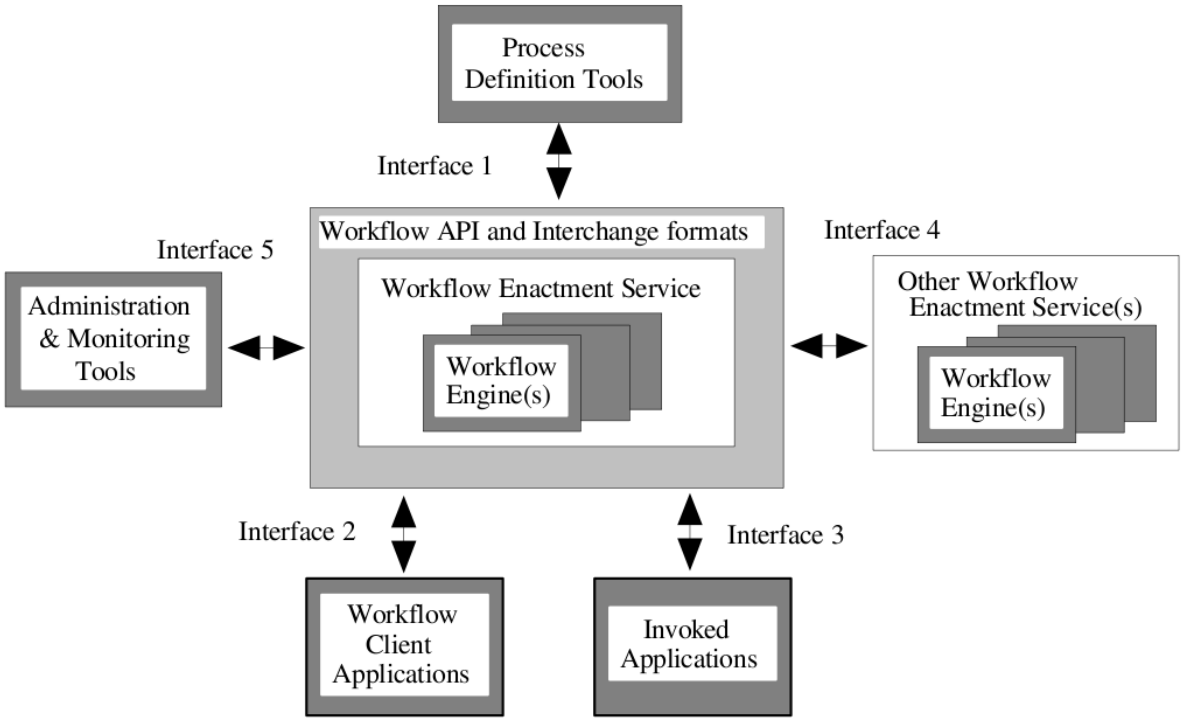



Figure 4.4: Workflow Reference Model (from [Hollingsworth, 1995, page 20])

The reference model of a workflow defines five components:

- **Process Definition Tools**: these tools are used to specify in an abstract notation the execution logic of the process. Process definition tools can use any visual modeling language such as Petri nets [Salimifard and Wright, 2001] or Directed Acyclic Graph (DAG) [van der Aalst and van Hee, 2002] to model a workflow process. These tools are integrated for linking resources to specific activities in the process definition.
- **The Workflow Enactment Service**: is the component in a workflow system, responsible of processes and workflows management. It consists of one or several workflow engines and is in this way able to produce, manage and execute workflow instances in the engines.
- **The Workflow Engine**: is the heart of a the workflow management system and responsible for the design and runtime control of workflow instances. It maintains all the data about the available and running processes. It is located within the Workflow Enactment Service. It can be a single workflow engine, or a collection of several workflow engines.
- **Workflow Client Application**: provides an interface for end-users to the WfMS. This allows the user to take and edit tasks. Moreover, it can thus instantiate workflows. The workflow client application is divided into a worklist handler and an user interface. The worklist handler interacts with the user's worklist, which is maintained in the Workflow Enactment Service. The user interface displays tasks to the user and gives him tools for editing.
- **External Applications**: definition of workflow interoperability models and the corresponding standards to support interworking. Business processes are made of many different functions. These functions are performed by applications and must communicate its state to the workflow engine. A specific set of functions is defined for applications that are invoked.
- **Administration and Monitoring Tools**: A monitoring tool allows one to query the detailed results from the workflow management database. All the information about the business process is stored in the administration tool for analysis. An ideal monitoring tool permits to retrieve selective views of performance and to handle predictions.

The Process definition tool is used to create the process descriptions. The Workflow enactment service:

- interprets the process description
- controls the instantiation of processes, sequencing of activities, adding work items to the user work lists and invoking application tools as necessary.

The above is done through one or more co-operating workflow management engines, which manage(s) the execution of individual instances of the various processes.
The reference model of a workflow has five interfaces between its components:

- **Interface 1** (Server-Designer) enables to use a process definition as an input to a workflow engine. It defines a point of separation between the build-time and run-time workflow environment.
- **Interface 2** (Client-Server) supports interactions between an application and the workflow engine.
- **Interface 3** (Invoked Application) defines an interface for calling an application to handle an activity. The invoked application may be local to the workflow engine or located on a separate network accessible platform.
- **Interface 4** (Server-Server) enables one workflow system to pass a work item seamlessly to another workflow system. This allows multiple workflow engines to work together on achieving the goal of a business process.
- **Interface 5** this interface includes the following operations: user management, resource control, process supervisory functions and process status functions.

### 4.3.2 Workflow Perspectives

Next to the control flow, workflows can model other aspects. These perspectives of workflow modeling will be shown here. The modeling of workflows is not limited to the modeling of possible sequences, but also requires the consideration of; on the one hand data plus information and on the other hand the resources. For this reason, most of the workflow modeling tools have established different views of a workflow. [Weske and Puhlmann, 2005, Jablonski Stefan (Hrsg.), 1997, Aalst, 1999, Scheer, 2002] designate five different views. [Jablonski Stefan (Hrsg.), 1997] emphasizes, however, that in a concrete application case, these views must be extended by other views.

The Figure. 4.5 is adapted from [Scheer, 2002] and represents the ARIS-House, which is considered as a model for the modeling with the ARIS tool and which shows the different views. However, the views (in [Scheer, 2002]) are slightly different than those shown here.

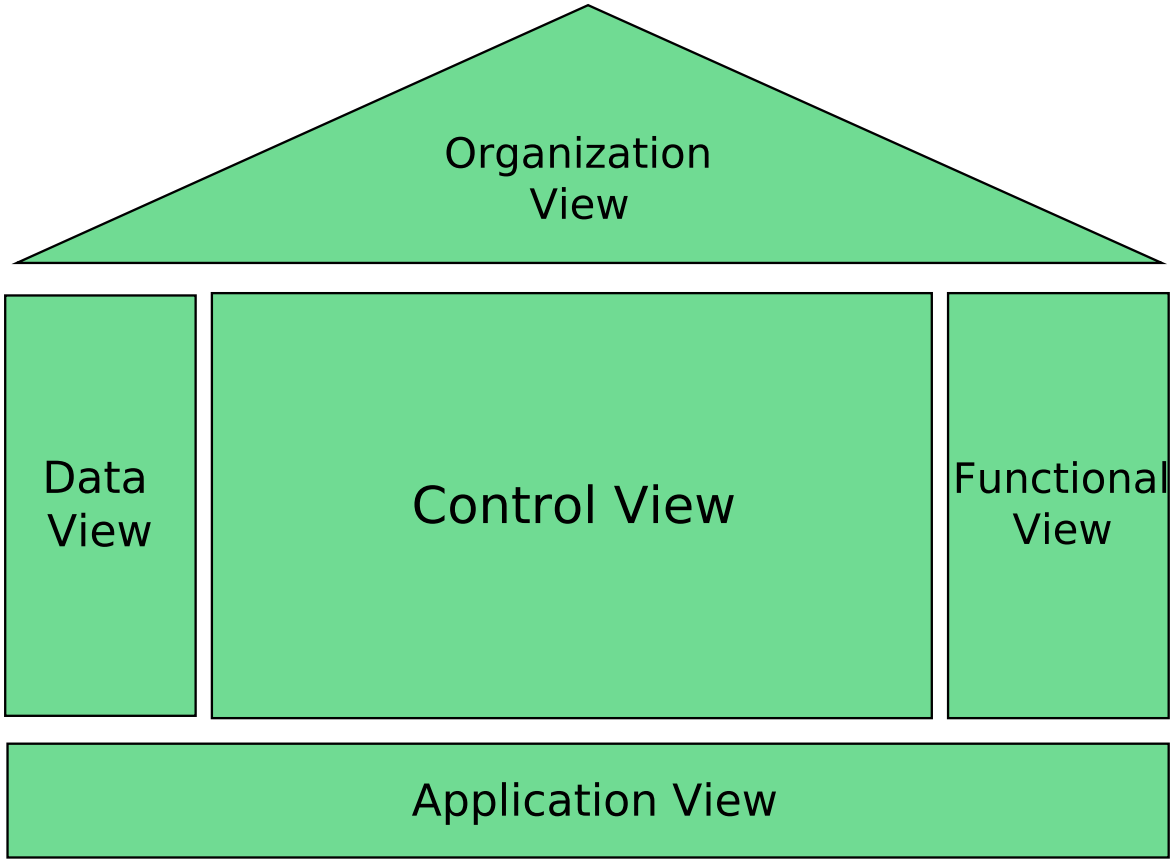



Figure 4.5: Workflow Perspectives (from [Scheer, 2002, p. 610])

*The Activities-, task- or function view* characterizes the activities and sub-workflows that must be processed during the execution of the workflow. This view also describes how workflows are partitioned into complex (sub-workflows) and atomic workflows [Weske and Puhlmann, 2005]. This is often represented by a hierarchical structure based on a tree. In the following, the perspectives from [Weske and Puhlmann, 2005] are described:

- *Functional Perspective*: concerns the functional decomposition of the workflow, often represented by a hierarchical structure. It determines which activities have to be executed. It specifies what has to be done during the workflow execution but does not specify how, when or under which conditions the tasks are performed. It specifies who performs a given task and which data and applications are used.
- *Behavioral (or process) Perspective*: This perspective designs the constraints on the functions performed in a workflow. One of the important components of this perspective is the control flow constraints. It specifies under which conditions the sub-workflows are executed during workflow execution. Therefore, the WfMS has to take into account the interrelationships of the workflow's tasks.

- *Informational Perspective*: concerns the data and means the modeling of the relevant application data of the workflow. It controls the transfer of the generated (control), e.g., variables as well as the processed (production) data by the activities. Such a form of data dependency is called data flow and it is important to guide and control data transfer between related workflow tasks.
- *Organizational Perspective*: It gives an overview of the organizational structure and the structure of the resources available to a workflow. The main objective of this perspective is to enhance the efficiency of application processes by assigning work to resources. For this, the WfMS need information about the organizational and the technical environment in which the workflow is executed. Often, the word role resolution is used and designs the selection of one or more persons (software in case of automated workflows), which are allowed to perform the task.
- *Operational Perspective*: The operational perspective covers mainly technical issues. It describes the elementary operations performed by resources and applications. For example, entering customer and credit request data is typically done by forms-based softwares.

## 4.4 Workflow Patterns

Workflow patterns address comprehensive workflow functionality. In [Moldt and Rölke, 2003], authors showed that Reference nets are suitable to model the workflow patterns in an elegant and easy way. They developed a plug-in for the Renew Petri net editor to support the construction of workflows by simply putting workflow patterns together in a drag-and-drop manner. In the following are, six categories of workflow patterns addressed from a reference net view:

- Basic control patterns
  - sequence,
  - parallel split (and),
  - synchronization (parallel, and),
  - choice and
  - simple merge (exclusive, xor)
- Branching and Synchronization patterns
  - multi-choice (or),
  - synchronizing merge (or)
  - multi-merge
  - discriminator
- Structural patterns

  - arbitrary cycles (loop)
  - implicit termination
- Multiple instance patterns
  - multiple instances without synchronization (fork)
  - multiple instances with a Priori design time knowledge
  - Multiple Instance with a Priori Runtime Knowledge
  - Multiple Instance without a Priori Runtime Knowledge
- State-based patterns
  - Deferred Choice (implicit choice)
  - Interleaved Parallel Routing
  - Milestone
- Cancellation patterns
  - Cancel Activity
  - Cancel Case (Instance)

The authors in [Moldt and Rölke, 2003] investigate how these patterns are supported in the modeling tool Renew.

## 4.5 Workflow and Cloud Computing

In contrast to traditional execution scenarios, the aim of this dissertation is to investigate the integration of both workflow concepts and the Cloud technology and later on with the agent paradigm. This is addressed exclusively in Part II. To combine workflow and Cloud computing, two research directions are distinguished and will be presented in Part II. On the one hand, the applicability of the Cloud technology to workflow execution is investigated. Here, the objective is to increase the performance of the whole execution system, by using Cloud resources as execution targets. For this purpose, mechanisms are provided to deploy and enact workflows in Cloud environments (see Chapter 9). On the other hand, workflow concepts for Cloud computing are introduced. They help researchers and practitioners in the design, development and execution of their application in the Cloud. It also includes among others improved modeling techniques based on Petri nets for the management of Cloud-based workflows. The contributions covering this research are exclusively presented in Chapter 10.

The following sections outline some important works, which are somehow related to the proposed approaches in this dissertation. Furthermore, since Grids and Clouds are strongly related in term of resource sharing and allocation as well as data management, it is also included a short overview about Grid-based WfMSs.

## 4.6 Grid- and Cloud-based Workflow Management Systems

Over the past years, several works have been dedicated to the use of workflow for scientific applications. Yu et al. give a detailed taxonomy of workflow management systems [Yu and Buyya, 2005] based on workflow design, workflow scheduling, fault management and data movement. The study includes also a characterization and classification of different approaches for building and executing workflows on Grids. The authors also outline existing Grid workflow systems with key features and differences.

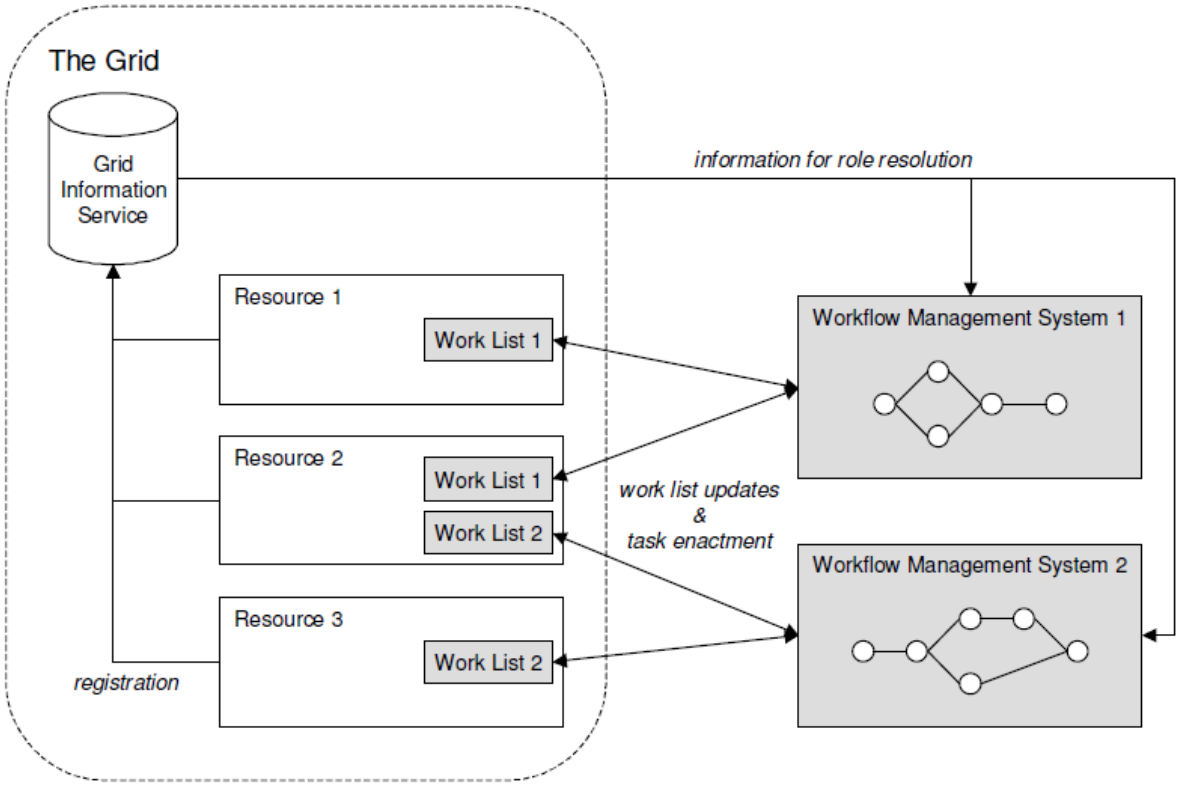



Figure 4.6: Workflow Management in Grid (from [Kuropka et al., 2006])

Korupa et al. [Kuropka et al., 2006] give a model of grid workflow management. Figure. 4.6 shows how the workflow management systems are related to grid nodes. The latter register their services in the grid information service to perform role resolution. The WfMSs belong to different organizations and they are responsible of scheduling and monitoring of their own tasks. It is the same for the grid nodes which perform local scheduling on the basis of the work lists they receive from the WfMSs. In this model, one resource can holds several work lists. Hence, further scheduling parameters need to be included such as priorities and deadlines.

The development of distributed scientific workflow systems with grid paradigm has significantly grown during the last decade. Workflow enactment service can be built on

top of the low level grid middleware (e.g. Globus Toolkit[6], UNICORE[7] and Alchemi[8]), through which the workflow management system invokes services provided by grid resources [Yu and Buyya, 2005]. At both the build-time and the run-time phases, the state of the resources and applications can be retrieved from grid information services. There are actually many grid workflow management systems in use. Following are a few examples of such systems: ASKALON[9], GridAnt[10], Pegasus[11], Taverna[12], Kepler[13], GridBus[14], GridFlow[15], GrADS[16] and Triana[17].

In [Yu and Buyya, 2005], is a comparison of several Grid workflow management systems given. It takes into account scheduling architecture, decision making, planning scheme, scheduling strategy and performance estimation.

### 4.6.1 JASMIN: Applications Management in Service-oriented Grid Systems

JASMIN [Bendoukha et al., 2012a], is another interesting grid-based WfMS. JASMIN[18] is a visual framework to make grids more efficient and more transparent to both experts and not- experienced users, by making easy interaction between them and the grid environment. For workflow modeling, the JASMIN framework is based on UML activity diagrams, the latter is converted to a BPEL process, which will be executed on a BPEL engine (like ActiveBPEL). To realize this work, many UML refinements have been made especially for mapping the UML activity diagrams into BPEL [Bendoukha et al., 2012b].

#### 4.6.1.1 The JASMIN Architecture

JASMIN framework covers workflow definition, service composition and enactment. Figure. 4.7 shows the architecture of JASMIN and its components. There are two main layers [Bendoukha et al., 2012a, Bendoukha et al., 2012b]:

**Workflow Definition** JASMIN framework provides a tool for the specification of workflow tasks and their dependencies. As modeling techniques, it uses UML activity diagrams. All the interactions between users and JASMIN are supported by a graphical user interface. The result of this layer is a model (in form of activity diagram).

[6] http://www.globus.org/toolkit/
[7] http://www.unicore.eu/
[8] http://www.cloudbus.org/~alchemi/
[9] http://www.askalon.org/
[10] http:// www.globus.org/cog/projects/gridant/
[11] http://pegasus.isi.edu/
[12] http://www.taverna.org.uk/
[13] https://kepler-project.
[14] http://www.gridbus.org/
[15] http://gridflow.ca/
[16] http://www.iges.org/grads/
[17] http://www.trianacode.org/
[18] I participated actively to this project since it was related to my Magister thesis, which was in cooperation with the GRID Computing Laboratory of the University of Calabria (Italy) (`http://gridlab.dimes.unical.it/`)

**Service Composition** , the role of this layer is to take the BPEL model generated above and to transform it into an executable script. There are specific tools which are responsible of the transformation. In order to convert the model, the activity diagram elements are mapped into BPEL elements. Therefore, a library called *UML2BPEL* has been implemented. This layer is also considered as the gateway to the grid resources.

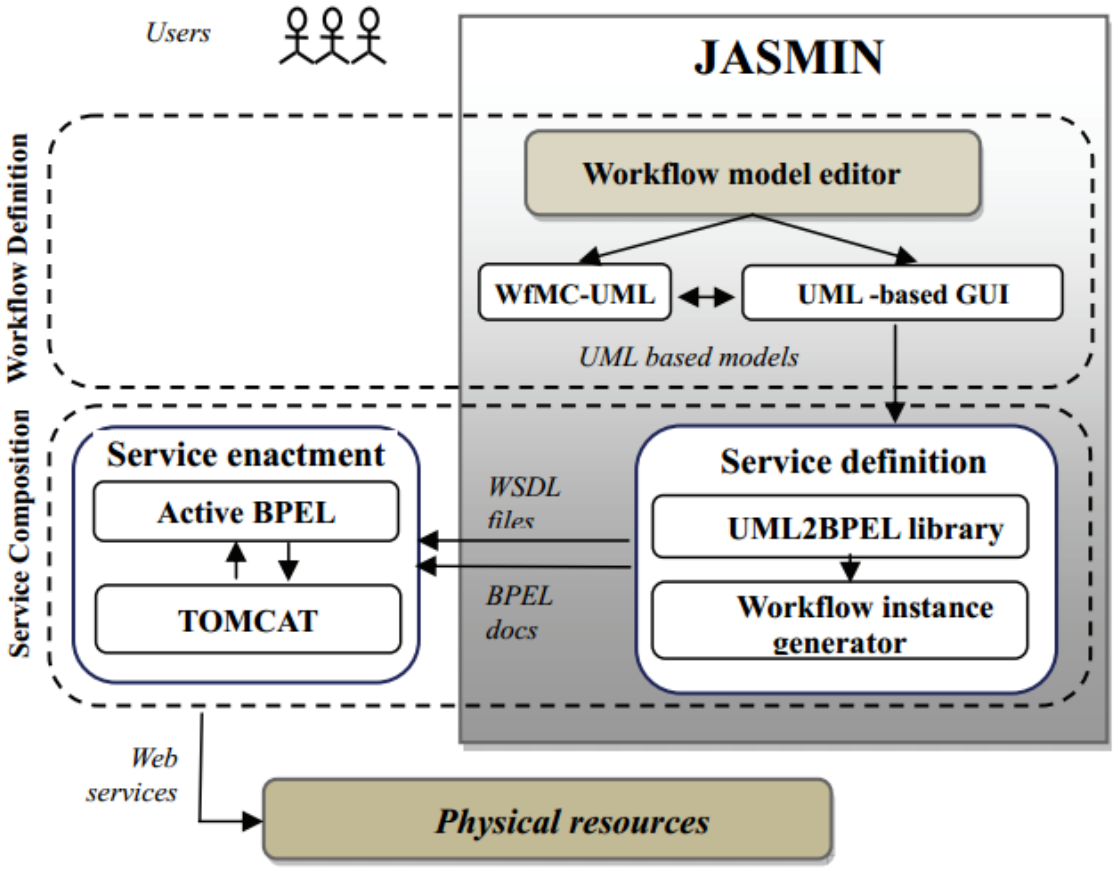



Figure 4.7: The architecture of the JASMIN framework
(from [Bendoukha et al., 2012a])

#### 4.6.1.2 The JASMIN Scenario

In this section, the life-cycle of a workflow using the JASMIN framework is described. In order to specify the workflow tasks and their dependencies, the framework provides a *Workflow Model Editor* based on UML activity diagrams. Since the workflow tasks involve Web services that need to be *orchestrated* using BPEL, modeling a workflow takes all required information into account. For example, Partner Link Types, Endpoints references and variables following the BPEL specifications [Juric, 2006, OASIS, 2007]. Furthermore, the editor provides predefined workflow patterns such as *Sequential*, *Fork*, *Join* and *Switch*. These are dedicated especially to non-expert users during the modeling step.

The generated model can not be directly executed. Therefore, it is converted to an executable BPEL document. This is achieved by the UML2BPEL, which converts each UML construct to a BPEL element. JASMIN uses Globus Toolkit[19] as grid middleware. Such middleware allows *hosting* grid services in a container.

### 4.6.2 Cloud-based Workflow System Architecture

Based on the discussion in the section 3.2.3, Xiao Liu et al. [Liu et al., 2010] proposed a general architecture for Cloud-based workflow (see Figure. 4.8). It is considered as a mapping of the Cloud system architecture (see Figure. 3.3).

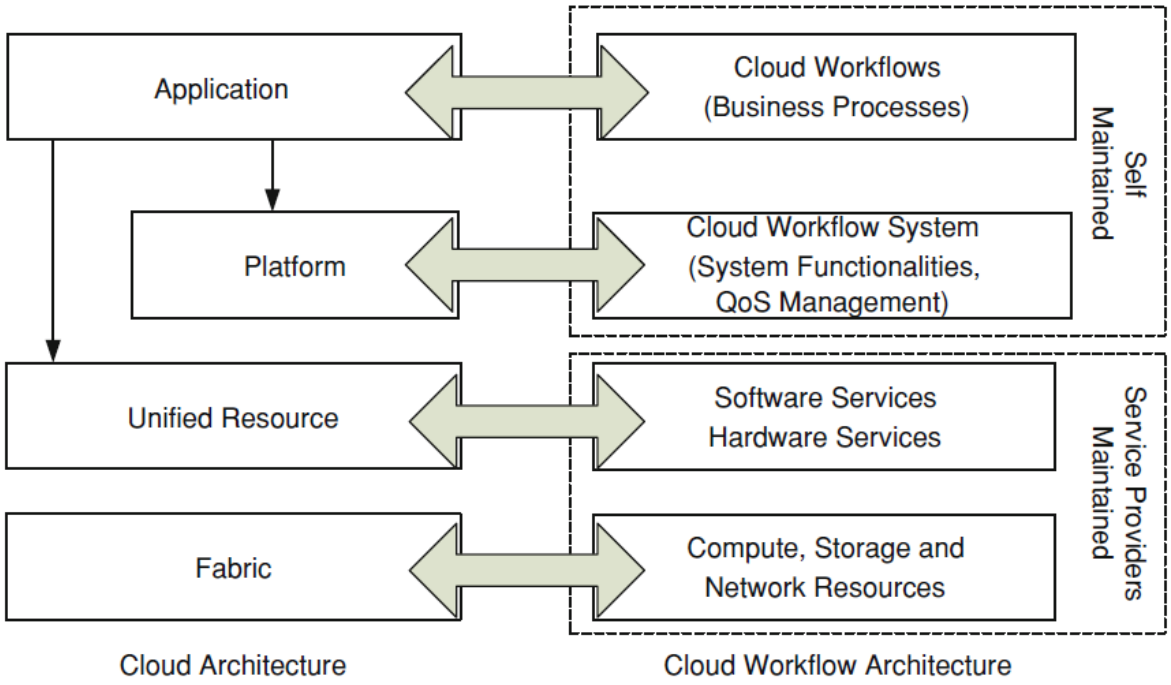



Figure 4.8: Cloud Workflow System Architecture (from [Liu et al., 2010])

The *application layer* contains the Cloud-based workflows (business processes). The *platform layer* includes the Cloud workflow functionalities such as workflow management, Cloud resource management and QoS management. The *unified resource layer* consists of software and and hardware services which are required for the execution of the Cloud workflow. These services are encapsulated and delivered as VMs (virtual machines). The *fabric layer* contains low level hardware resources such as computing, storage and network resources.

As shown in Figure. 4.8, the unified layer and fabric layer are usually maintained by external Cloud service providers. The application layer and the platform layer are usually self-maintained by the business organization.

[19] http://toolkit.globus.org/toolkit/s

### 4.6.3 Amazon Simple Workflow Service

Launched in 2012, Amazon Simple Workflow Service (SWF)[20] (see Figure. 4.9) is an orchestration service for building scalable applications. Amazon recognized the usefulness of workflow concepts to develop Cloud-based applications. It maintains the execution state of the workflow in terms of consistency and reliability. It permits to structure various processing steps in an application that runs across one or more systems as a set of tasks[21]. These systems can be Cloud-based, on-premises, or both. Amazon SWF manages dependencies between the tasks, schedules the tasks for execution, and runs any logic that needs to be executed in parallel. The service also stores the tasks, reliably dispatches them to application components, tracks their progress, and keeps their latest state. Nevertheless, the important drawback of SWF is its incompatibility with other workflow systems. Amazon provides SDKs for different programming languages like Java, Ruby, etc. The following terminology is taken from the original documentation of SWF [OASIS, 2014]. A workflow in SWF *"is a set of activities that carry out some objective, together with logic that coordinates the activities."*. Each workflow runs in an AWS resource called a *domain*. An Amazon SWF *Actors* can be workflow starters, deciders, or activity workers. A *decider* is an implementation of a workflow's coordination logic. Deciders control the flow of activity tasks in a workflow execution. A workflow starter is any application (event) that can initiate workflow executions. An activity worker is a process or thread that performs the activity tasks that are part of the workflow. Activity tasks need to be registered using the Amazon SWF console.

[20] `aws.amazon.com/swf`

[21] There are two types of tasks in Amazon SWF: Activity task that tells an activity worker to perform its function, such as to check inventory or charge a credit card. The activity task contains all the information that the activity worker needs to perform its function. **Decision task** that tells a decider that the state of the workflow execution has changed so that the decider can determine the next activity that needs to be performed. The decision task contains the current workflow history.

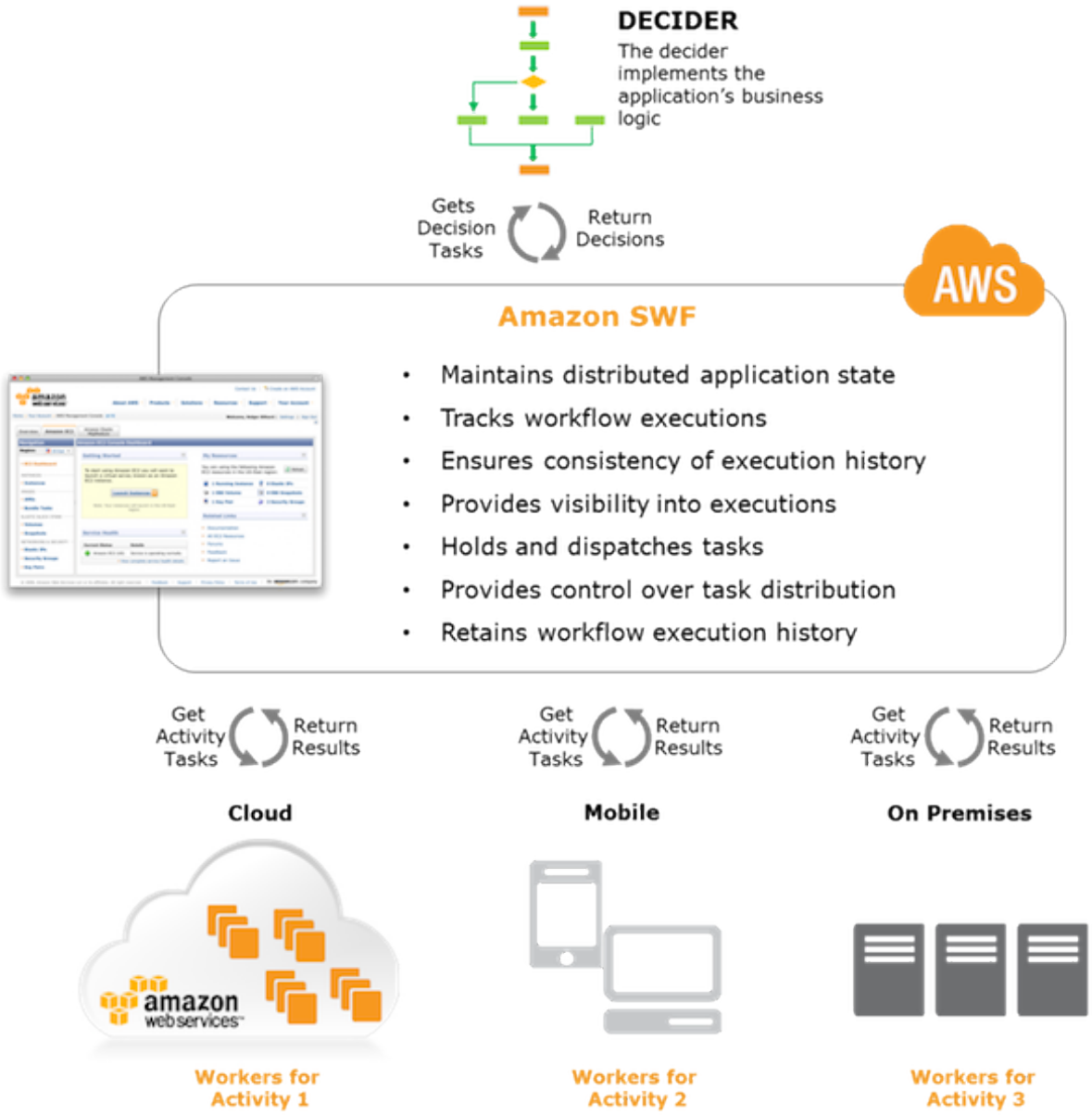



Figure 4.9: The Amazon Simple Workflow Service [Amazon, 2012]

### 4.6.4 The SwinDeW-C Architecture

The Swinburne Decentralized Workflow for Cloud (SwinDeW-C) system [Liu et al., 2010] is for running large scale workflow applications. It inherits many features of its ancestor SwinDeW-G [Yang et al., 2007] but with significant modifications to accommodate the novel Cloud computing paradigm. The architecture of SwinDeW-C is depicted in Figure. 4.10, it is composed of four basic layers from top to bottom: application layer (user applications), platform layer unified resource layer (abstracted/encapsulated resources by virtualisation) and fabric layer (physical hardware resources). They cover three aspects of large-scale workflows: QoS management, Data management

and Security management. At build-time, workflow applications are modeled by a cloud modeling tool (Web portal) as Cloud workflow specifications (consist of such as task definitions, process structures and QoS constraints). These specifications are next submitted to the coordinator peers on the platform layer. Afterwards, workflow tasks will be assigned to suitable peers through peer to peer based communication between SwinDeW-C peers.

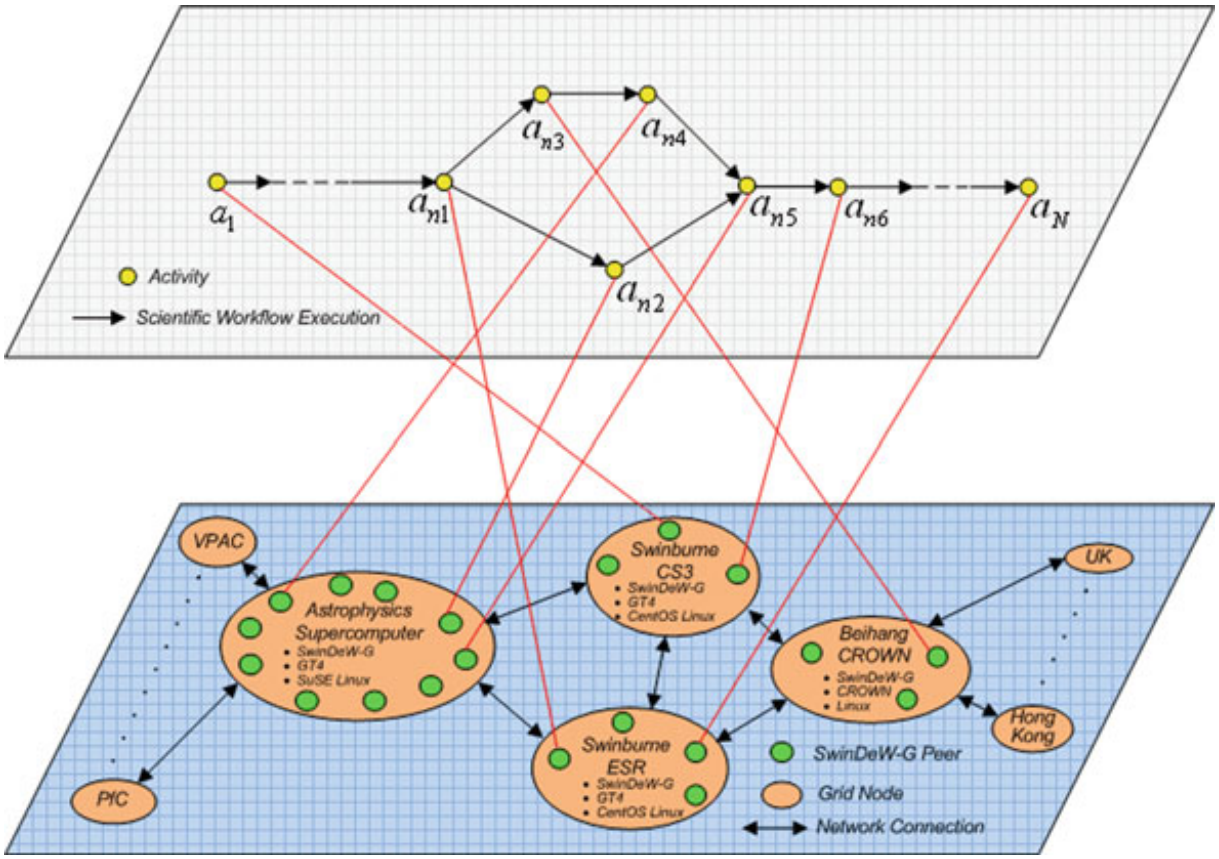



Figure 4.10: The SwinDeW-C Architecture (from [Liu et al., 2010])

## 4.7 Conclusion

In this chapter, workflow concepts have been presented. The first sections concern the terminologies used in this dissertation. Moreover, a special emphasis is made on *scientific workflows*. The rest of the chapter covers the relation between workflows and Cloud computing. It is stated that this relation can be addressed in different ways. On the one hand, there is Cloud for workflow which consists of using Cloud resources to execute complex workflow tasks especially scientific workflows [Hoffa et al., 2008] [Juve et al., 2009]. Such kind of workflows are more resource-centric and focus on the computational tasks. On the other hand, designing Cloud ecosystems needs structured and mature workflow concepts and high-level languages to handle issues like managing complex task and data dependencies. In this dissertation, both scenarios are tackled by

providing new concepts and techniques for modeling, deployment and execution of Cloud-based workflows. The next chapter tackles workflow specification techniques and tools. It will concentrate on Petri nets-based modeling since it is the modeling techniques used in the current work.

# Chapter 5

# Workflow Modeling: Techniques and Tools

Workflow modeling plays an important role in this dissertation. Most of the effort has been dedicated to provide appropriate modeling techniques for Cloud-based workflows. Furthermore, existing techniques and tools have been improved in order to adapt them for new execution environments (Clouds). In this research work Petri nets are exclusively used as a modeling technique. The objective of this chapter is first to address workflow specification in general. Next, a detailed introduction to Petri nets as well as related technologies are given.

## 5.1 Introduction

Defining workflows or processes is an essential step towards the design and the development of Cloud-based applications and their management. It allows understanding the functionality of each entity taking part to the management system. The main role of a process definition is to model routing patterns (sequence, iteration, parallelism and selection). It describes the order of the tasks that should be performed and the dependencies among them. Moreover, it also allows the verification of the workflow model to detect (semantic) errors like deadlocks. Process definition permits also the reusability of the workflow model several times and by other workflow engines, that can be hosted on different Cloud providers. Workflow modeling helps analyzing the whole system and its components. This chapter is composed of two main sections. Firstly, an introduction is given to differentiate between script-like and graph-based workflow description languages. After that, some existing workflow specification languages and tools are presented such as BPEL, BPMN and Petri nets. The latter are explained in details in the second part of the this chapter. The most important are:

- Petri nets (see Section 5.4): In this section, Petri nets are defined and classified into three categories (Elementary nets, P/T nets and high level petri nets). The basic formalisms and operations of P/T nets are presented.

- Workflow nets (see Section 5.4.3): They are a special sub class of Petri nets used for modeling and verification of business processes.
- Reference nets (see section 5.4.4): Reference nets are based on the nets-in-nets paradigm and play an important role in this thesis, especially in Chapter 10. This formalism are explained through a working example (DropBox Storage nets).
- Renew (see Section 5.5): Renew is the chosen tool for modeling and simulating different kinds of Petri nets. In this section, the Renew Editor, Renew Simulator and the Plug-in system are presented.

## 5.2 Workflow Specification

Workflow specification or definition is an important step and strives to provide a process description that can be interpreted by the software to support the corresponding process. It defines a collection of tasks and the order of task invocation.

In general, users can define applications using definition languages and tools. Existing workflow description languages can be regrouped into two categories [Hoheisel and Alt, 2007, page. 13]. Script-like workflow descriptions using a textual "programming language" that has complex semantics and extensive syntax, and graph-based workflow description languages. In terms of script-based definition language, markup language, such as Extensible Markup Language (XML) [W3C, 2012] is widely used specially for workflow specification as it facilitates information description in a nested structure. Therefore, many XML-based application definition languages have been adopted in Grids. Some of these languages, such as WSDL [Chinnici et al., 2012] and BPEL [OASIS, 2007] have been standardized by the industry and research community (i.e. W3C [W3C), 2012]), whereas some of them, such as xWFL [Rajkumar Buyya, 2009] and AGWL [Fahringer et al., 2005b] are customized according to the requirements of the system.

Although language-based definition of applications is convenient for expert users, it requires users to learn a lot of language-specific syntax. However, the general users prefer to use graph-based definition languages and support tools for application definition, where the application composition is better visualized. This graphical representation is later converted into other forms for further manipulation.

## 5.3 Workflow Specification Languages

Although a standard workflow language like the Business Process Execution Language (BPEL) is defined [OASIS, 2007, Juric, 2006], some workflow systems developed their own workflow model for making it possible users to specify workflows. Besides BPEL, other formalisms like UML Activity Diagrams [OMG, 2015], Petri nets [Petri, 1962, Alt et al., 2006], Business Process Modeling Notations (BPMN) [OMG, 2013] and XML-based languages [Fahringer et al., 2005b] are also used to define workflows. This makes it difficult sharing workflow models between different partners and diminishes

the interoperability of the workflow applications when trying to use different workflow management systems. The following sections provide a short overview of these specification languages.

### 5.3.1 WS-BPEL

The Business Process Execution Language for Web service known as BPEL4WS, more recently as WS-BPEL (or BPEL for short) is a widely used standard for implementing business processes that are based on SOAP services and a standard workflow language. It is a domain specific, imperative programming language for the definition of executable business processes. BPEL is also classified in the group of service orchestration languages. Although one can understand BPEL as a programming language, this is not *"programmed"* but *"modeled"*. For this reason, no text editor (although it is possible) is utilized, but "modeling tool" that supports the configuration of the Web service details. This also assumes the creation and updating of individual files (BPEL, WSDL, XSD) so that the developer can focus on the business logic. Each element in the BPEL process is called an *activity*. An activity can be either *primitive* or *structured*. Primitive activities allow invoking operations on a Web service, waiting for a message from an external source, copying data from source to target, etc. The structured activities permit for example to define the execution order, grouping and routing activities, etc. The BPEL specification distinguishes explicitly between executable and abstract processes and defines which details in abstract models can be omitted. Unlike conventional programming, business processes are typically long running, concurrent programs, where the control- and data flow are specified and interpreted (by a Workflow engine) differently.

The reason mentioning BPEL here is related to a previous research work (Magister thesis). The objective was to build a Grid-based Workflow Management System called Jasmin (see the previous Chapter 4). It supports workflow modeling, deployment and execution in Grid-like environments with an emphasis on the specification level. Therefor, a modeling tool based on UML activity diagrams has been implemented. The next step of the project was to translate the activity diagrams into executable BPEL-based processes, which are deployed after in a BPEL engine. The BPEL engine runs on a container as a Grid service. More details about this work can be found in [Bendoukha et al., 2012a].

### 5.3.2 UML Activity Diagrams

UML Activity Diagrams are used to specify the control flow of a workflow. An activity diagram consists of a set of *nodes* [Qin and Fahringer, 2012], which can be classified as: *Action nodes* (see Figure. 5.1a), *Control nodes* (see Figure. 5.1b, c, e-h) and *Object nodes* (see Figure. 5.1d, i). *Actions nodes* or *Action states* represent the performance of a step within a workflow. *Control nodes* indicate the control and data flow. *Object nodes* contain the data that circulate through the graph. A *control flow edge* (see Figure. 5.1i) indicates the dependency relationship between two interconnected nodes. An *Object node* specifies the flow of objects along interconnected *Action nodes*. As stated before,

UML Activity Diagrams have been chosen to specify Grid-based workflows. A tool has been implemented to allow the specification of workflows and then their deployment and enactment in a Grid environment [Bendoukha et al., 2012b].

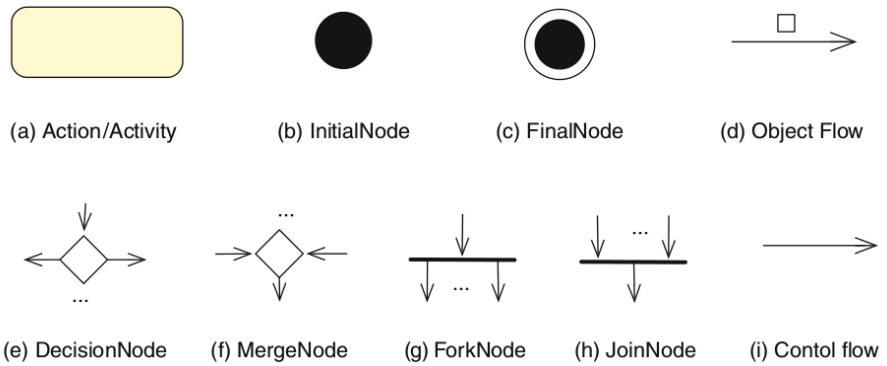



Figure 5.1: UML-based Scientific Workflow Modeling
(from [Qin and Fahringer, 2012])

### 5.3.3 Business Process Model and Notation (BPMN)

The process description language Business Process Modeling Notation (BPMN) provides a graphical representation of process models (see Figure. 5.2). It has become one of the most widely used languages to model business processes and includes since version 2.0 an executable XML representation. BPMN is supported by many tool vendors and has been standardized by the *Object Management Group* (OMG). The aim of the current version of the process description language *Business Process Model and Notation* [OMG, 2013] is to avoid as far as possible the gap between the strategic-operational modeling at the business level and the executable descritpion at the technical level.

This goal is also laid out in the standards documents, which states that *"The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation."* [OMG, 2013, page 1]. On that point, a graphical modeling language by the BPMN version 2.0.2, to support various human experts, and which can be transfered directly into a corresponding machine readable XML-format. Following BPMN, a process consists of a number of flow objects (tasks, gateways and events), connecting objects (sequence flows and message flows), distribution objects (pools and swimlanes) and other artifacts (data objects and groups, etc.) [Barkhordarian et al., 2012].

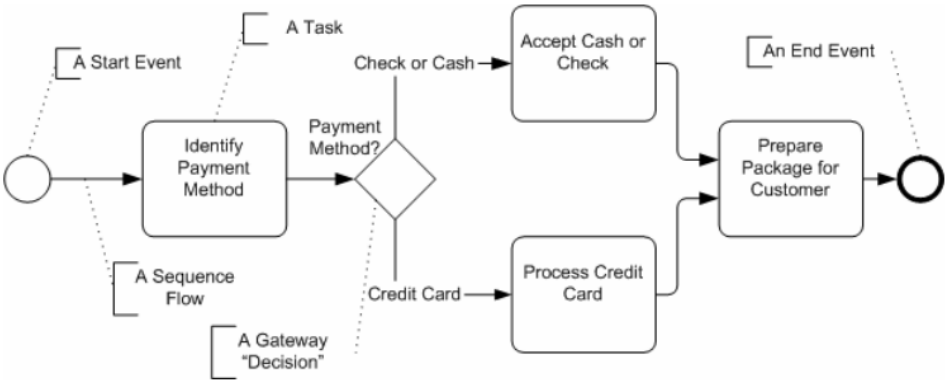



Figure 5.2: Example of a Simple Business Process (from [White, 2008])

### 5.3.4 Petri Nets

Petri nets are used for modeling and analyzing workflows. They can serve as a solid foundation for Business Process Management and workflow management. Workflow Petri nets (see Section. 5.4.3) are a class of Petri nets used for the representation, validation and verification of processes following the principles laid out in [van der Aalst, 1997]. More specifically the reference net formalism [Kummer, 2002] is extended with a specialized workflow task transition [Jacob et al., 2002] (see Section. 5.4.4). Petri nets can model different activities in a distributed system; a transition may model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on. The input places of the Cloud Task Transition model the pre-conditions of an event, the input data for the computational task. The output places of the transition model the postconditions associated with an event, the results of the computational task. Petri nets are presented with more details within next sections (see Section 5.4).

## 5.4 Petri Nets

Petri net formalisms are used for modeling complex systems thanks to their advantages in supporting concurrency and their capacity to offer both operational semantics (validation) and formal semantics (verification). They were designed by Carl Adam Petri in 1962 in his PhD thesis: *"Kommunikation mit Automaten"*. Petri nets (see Figure. 5.3) are widely seen as the scientific fully functional workflow modeling method [van der Aalst, 1997]. They are a special class of directed graphs that can model sequential, parallel, loops and conditional execution of tasks. A vast number of algorithms and tools for Petri Nets analysis have been developed along the years. There are several kinds of Petri nets. A classification was elaborated by [Bernardinello and De Cindio, 1992, page. 306], who distinguishes between three levels of Petri nets:

- **Level 1**: Petri nets that are characterized by places that can represent Boolean values; i.e., a place is marked by one unstructured token. Examples of level 1

nets are Condition/Event (C/E) systems, Elementary nets (EN) systems and State Machines (SMs).

- **Level 2**: Petri nets characterized by places that can represent Integer values; i.e., a place is marked by a number of unstructured tokens. Examples of level 2 Petri nets are Place/Transition (P/T) nets and Free Choice nets.
- **Level 3**: Petri nets characterized by places that can represent high-level values; i.e., a place is marked by a multi-set of structured tokens. Examples of high-level Petri nets are Colored Petri nets (CPN).

First, the basic formalisms and operations of Petri nets (P/T nets) are presented. A special subclass of Petri nets called Workflow nets are used to model and to verify business processes and other structured processes. For the modeling and the implementation the reference nets formalism is used [Kummer, 2002]. In reference nets, nets can be used as marking in other nets, whereby a large expressive power for modeling complex systems is given.

### 5.4.1 P/T Nets

Petri nets consist of three types of net elements: places, transitions and arcs. Places and transitions are connected by edges represented by Arrows. These arcs always go from one place to a transition or from a transition to a place, never between two net elements of similar type. Places may contain tokens that may move to other places by executing ("firing") actions.

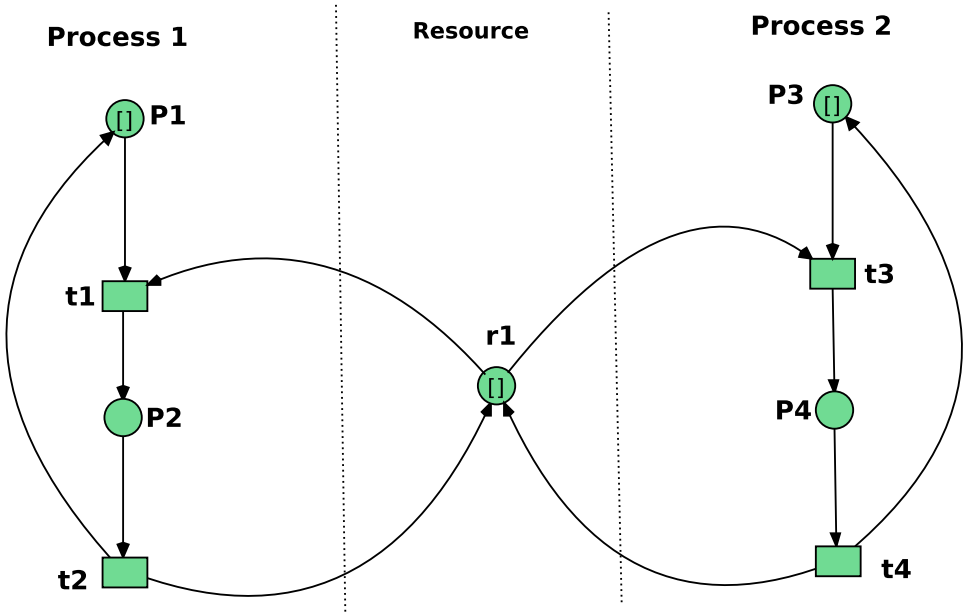



Figure 5.3: Petri Net

**Definition 1 (Net)** A net is a triple N = (P, T, F) where:

- P is the set of places of $N$
- T is the set of transitions
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation

A marking of $N$ is a multi-set of places $m \in \mathbb{N}^P$. The firing of a transition is represented by $m \xrightarrow{t} m'$. Firing sequences are represented by $m \xrightarrow{t_1 \ldots t_n} m'$. The set of the *reachable markings* is defined as $RS(m) = \{m' \mid \exists t_1 \ldots t_n : m \xrightarrow{t_1 \ldots t_n} m'\}$.
The preset of a node (a place or a transition) y is ${}^\bullet y := \{x \mid xFy\}$. The post-set of a node y is $y^\bullet := \{x \mid yFx\}$.

P/T nets extend the above definition of Petri nets by adding capacities to places, an initial marking function and weights to arcs.

**Definition 2 (P/T Net)** A P/T Petri net ($PN$) is a 5-tuple, $PN = (P, T, F, W, M_0)$, where:

- $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places
- $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation)
- $W : F \to \{1, 2, 3, ...\}$ is a weight function
- $M_0 : P \to \{0, 1, 2, 3, ...\}$ is the initial marking

In this work, Petri nets are used to enable the combination of agents and workflows, which are solid and proven models ( [Rölke, 2004] for agent and Multi-agent systems and [van der Aalst, 1997] for workflow and workflow systems).

### 5.4.2 Nets-in-Nets Formalism

The *nets-in-nets* formalism was introduced by Rüdiger Valk (see [Valk, 1998, Valk, 2004]) and will be briefly described in this section. The given formal definition of the Elementary Object System (EOS) by Valk describes a two-level system architecture: At the first level there is the so called *System Net* and at the secong level all contained *Object Nets* therein. Each of these nets is a P/T net and use usual black tokens unlike the system net's places, which are marked with either black tokens or object nets. The underlying conceptual idea is applicable to any nested *Object Nets*. Michael Köhler [Köhler, 2004] elaborated a further formalization of the object nets. The concept of nets-in-nets can in principle (under certain constraints) be interpreted as a variant of the hierarchy of Petri

nets. This paradigm allows tokens in a Petri net place to be interpreted as a net again. It can occur any number of object net tokens with different markings in a system net. In their simple form such as the Elementary Object System (EOS), the object net tokens hold the same net's structure (transitions, places and arcs). In the complex form the object nets may appear in different structures.

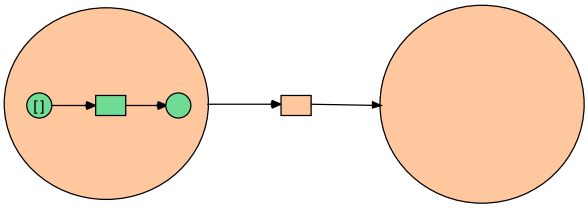


Figure 5.4: Net as a Token

**Definition 3 (EOS)**: An *Elementary Object System* (EOS) is a tuple OS = ($\hat{N}$,$N$, d, l) such that:

- $\hat{N}$ is a P/T net, called the system net
- $N$ is a finite set of disjoint P/T nets, called object nets
- d: $\hat{P} \rightarrow N$ is the typing of the system net places
- l = ($\hat{l}$,($l_N$) N∈$N$) is the labeling

Another detailed example is given by [Heitmann and Köhler-Bußmeier, 2011] and depicted in Figure. 5.5. It shows an EOS with the system net $\hat{N}$ and the object nets $N = N_1, N_2$. The system has four net-tokens: two on place $p_1$ and one on $p_2$ and $p_3$ each. The net-tokens on $p_1$ and $p_2$ share the same net structure, but have independent markings.

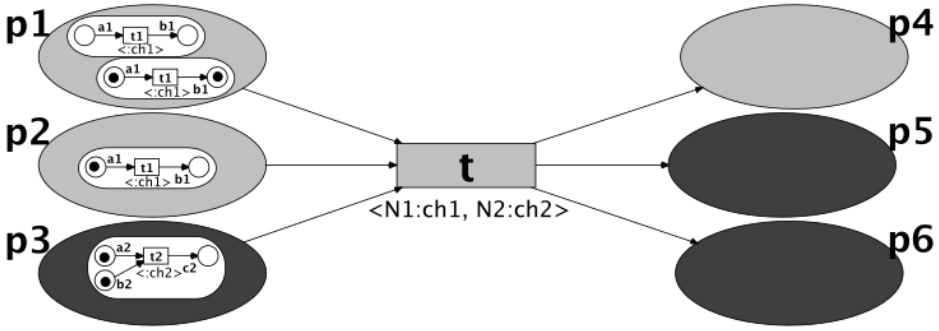



Figure 5.5: An Elementary Object Net System
(from [Heitmann and Köhler-Bußmeier, 2011])

### 5.4.3 Workflow Petri Nets

A Petri net which models the process aspect of a workflow, is called *Workflow Net* (WF-net) [van der Aalst, 2011b]. A WF-net is a reference net with one source place and one sink place (see Figure. 5.6), It is a workflow description technique with powerful modeling and formal analysis. Formally, a net N = (P, T, F,W,$m_o$) is a *workflow net* if and only if there exist places $i,o \in$ P such that $^\bullet$i = $\emptyset$ = o$^\bullet$, $m_0$(P)=1 for p=i and $m_0$(P)=0 otherwise, and the net $\tilde{N}$= (P,T$\cup${r}, F$\cup${(o,r),(r.i)},W$\cup${(o,r)$\mapsto$1,(r,i)$\mapsto$1},$m_o$) where r $\notin$ T, is strongly connected.

A firing sequence $\omega$ of a workflow net N is a *run* if $m_0 \xrightarrow{\omega r} m_0$ in $\tilde{N}$. The runs of N are the formalization of the use cases of the business process modelled by the workflow net. [] denotes a simple black token.

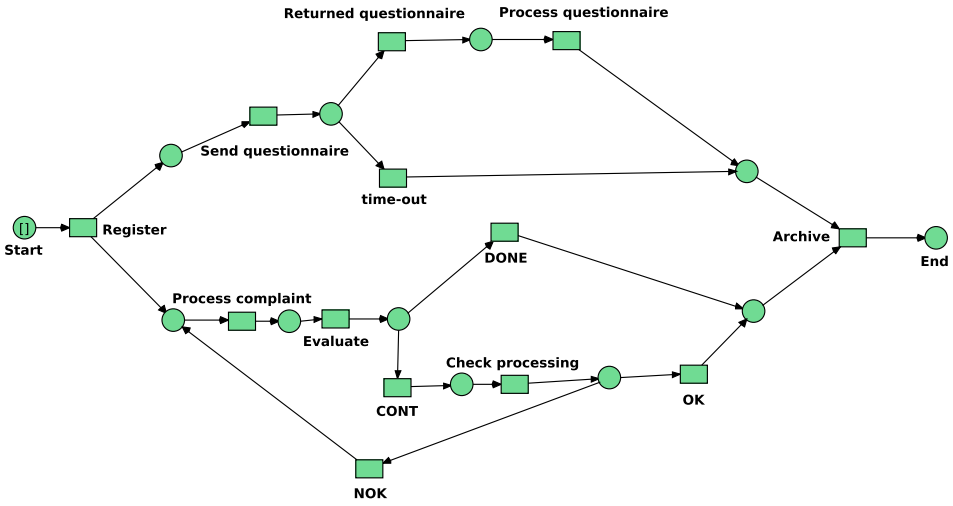



Figure 5.6: Workflow Petri net

### 5.4.4 Reference Nets

Reference nets were introduced in 2002 by Olaf Kummer (see [Kummer, 2002]). Reference nets are modeled and simulated using Renew the editor and simulation tool, both are presented in the Renew manual[1]. In reference nets, tokens can be anonymous, basic data types, Java objects or reference nets.

Firing a transition can also create a new instance of a subnet in such a way that a reference to the new net will be put as a token into a place. This allows for a specific, hierarchical nesting of nets, which is helpful for building complex systems in these formalisms. The creation of instances is similar to object instances in object-oriented programming languages and the usage of references allows to construct reference net systems, whose structures are not fixed at the build time. New net instances can be created by transitions that carry creation inscriptions, which consist of a variable name,

[1]The latest version of Renew, documentation and articles are published on the Internet (http://www.renew.de)

a colon (:), the word new and the name of the net. Net instances can be created destroyed dynamically at the run time.

One main idea is that two transitions can be connected via a named channel. If this is the case, both transitions can fire simultaneously and synchronously, but only if they were both activated before. During firing there is also a bidirectional information/data exchange possible, so that input data from one transition can be used as output data of the other transition. Connecting two transitions with a synchronous channel can be seen as fusing them together for the firing process.

Synchronous channel inscriptions consist of two types of inscriptions, *up-links* and *down-links*. Synchronous channels between reference net instances are specified in [Christensen and Hansen, 1994]. They consist of at least two transitions where one of the transitions is seen as the initiator of the communication having a down-link inscription. Transitions can have only one down-link and multiple up-links. Channels are designated by a string :channelname(parameters), in which the parameters are the object/token/variable that is transmitted through the channel. Synchronous channels play an important part in reference nets and the multi-agent architectures Mulan /Capa.

**Working Example** To show how Reference nets are created, Figure. 5.7 shows two separate nets that can communicate. The net (a) represents the Web authentication step to the DropBox service [DropBox, 2012], which consist of (i) getting request token (ii) having the user authorize the application and finally getting an *access*[2] token from a request token. The class *DropTransition* abstracts the three authentication steps mentioned above. Net (b) is using information from Net (a) to perform an upload of files to the repository (DropBox). The second transition in Net (a) fires and creates new instance of Net (b).

In Java the reserved word *this* denotes the object whose method is currently executed. In Reference nets this denotes the net instance in which a transition fires. It is used when in the case where two net instances within the same net synchronize.

Channels can also take a list of parameters. Although there is a direction of invocation, this direction need not coincide with the direction of information transfer. Indeed it is possible that a single synchronization transfers information in both directions. Figure. 5.7 shows that the Net (b) needs information from Net (a). These information can be transmitted as a parameter.

[2]For clarity there is no relation with Petri net token.

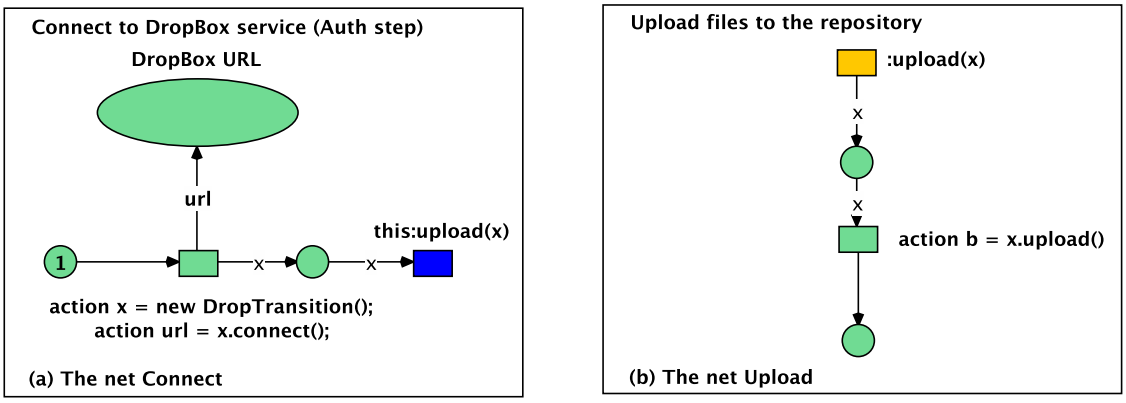



Figure 5.7: (a) The System Net, (b) The Sub-net Upload

The Reference net formalism is used to build workflow systems [Jacob et al., 2002, Moldt and Rölke, 2003], especially in scientific workflows [Tolosana-Calasanz et al., 2012]. In the latter, the authors used reference nets to model the montage of workflow using cloud infrastructures (see Figure. 5.8).

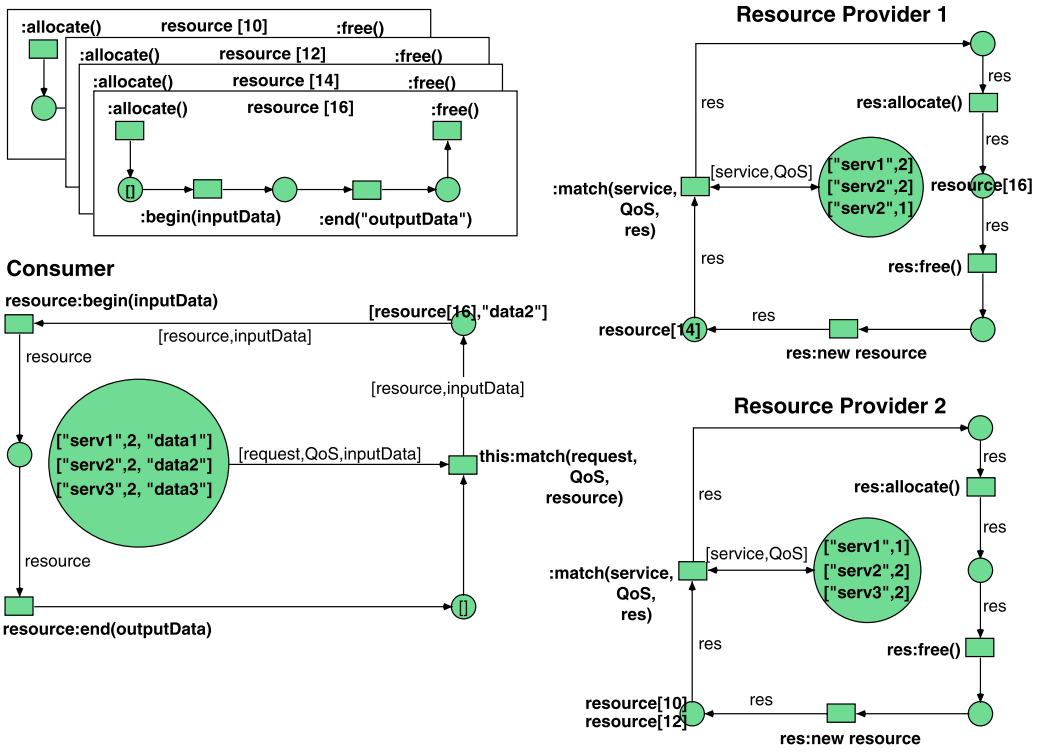



Figure 5.8: A Reference net model for Cloud-based Workflow (from [Tolosana-Calasanz et al., 2012])

## 5.5 Renew

Renew is a graphical tool for creating, editing and simulating reference nets. It is maintained by the TGI group (see http://www.renew.de/). With Renew it is possible to draw and simulate Petri nets and reference nets. Nets can be drawn comfortably using the graphical user interface and loaded nets can be executed directly in Renew.

### 5.5.1 Renew Editor

The Renew editor provides features to facilitate the creation of Petri net models. Figure. 5.9 shows the editor's graphical user interface, it contains several icons and a status bar. The first level of the icons provides features that can be added to the net model for more clarity. These features are ignored and not taken into account during the simulation. The second level concerns really the Petri net creation (transitions, places, arcs, tokens and inscriptions).

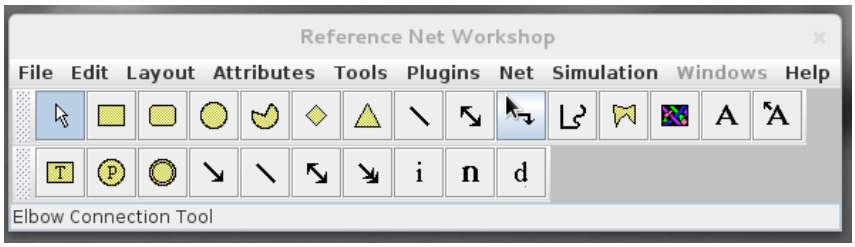



Figure 5.9: Renew Editor

### 5.5.2 Renew Simulator

In Renew, reference nets can be created as "shadow net system" files (.sns) or as graphical files (.rnw). Shadow nets are used for simulation and are basically an abstraction of the graphical nets. Shadow nets strip all unnecessary information such as color, position of net elements and leave only the needed information for use in the simulation. Those nets are required when moving simulation to the Cloud (see Chapter 9). A graphical file can be compiled in Java. After compilation of the graphical file a simulation can be started. During the simulation, a net instance is created and can be viewed in a separate window as its active transitions fire. Simulation is used in Renew to view firing sequences of active transitions in reference nets. Simulation can run in a one step modus where users can progress in steps where only one transition fires. Renew also offers the possibility to set breakpoints to hold the simulation process. Breakpoints can be set to places as well as to transitions. By changing the compiler, Renew can also simulate P/T nets, timed petri nets, WF-nets and boolean nets, etc.

### 5.5.3 Plug-in System

Renew is built on a plug-in architecture since version 1.7. The Renew plug-in architecture, which was developed and introduced in [Schumacher, 2003]. It allows the

extension of Renew with additional functionality through the use of interfaces from Renew components without changing the core of Renew. Additional functionality can be added to Renew through providing the classes of the new plug-in. This means to literally add the Java archive of the classes to the "plug-ins" folder inside the Renew application folder structure. Plug-ins can be included after Renew has started with load command. The Java archive has to be present in the "plug-ins" folder at the start of Renew.

## 5.6 Conclusion

There are several techniques to specify workflows such as the UML Activity Diagrams, BPEL and Petri nets. Petri nets and Renew represent the conceptual and technical background of the current work. They are selected among others thanks to their capability in term of modeling complex systems [van der Aalst, 1998]. This chapter concerned workflow specification in general with an emphasis on Petri nets. It started by defining the importance of workflow definition (see Section 5.2). Then in Section 5.3 important workflow specification techniques are presented. Section 5.4 gives a deep understanding overview of Petri nets. The last concepts that need to be presented concerns agent and multi-agents systems (see Chapter 6). It focuses on the Petri net-based, Agent- and Organization-oriented Software Engineering (Paose) approach and particularly on the Mulan (**Mul**ti-**A**gent **N**ets)/ Concurrent Agent Platform Capa framework.

# Chapter 6

# Multi-agent systems; Mulan and the Paose Approach

Agent-based systems have emerged significantly during the last two decades. The development of such systems includes, among others, artificial intelligence, distributed systems and software engineering. In this dissertation, the contributions concern both distributed systems and software engineering. This chapter presents an introduction to the concept of agents and multi-agent systems. Agent concepts are defined first, followed by an introduction to multi-agent systems. Furthermore, the **Mul**ti-**A**gent **N**ets (Mulan) [Rölke, 2004], the Concurrent Agent Platform Architecture (Capa) [Duvigneau et al., 2003] and the Paose approach [Moldt, 2006] are described.

## 6.1 Introduction

One of the objectives of this dissertation is to investigate the integration of Cloud, agents and workflows on a conceptual and technical level. The combination of individual agent and workflow concepts is being addressed in a parallel research work (see [Wagner, 2012, Wagner and Moldt, 2011]). Relative to earlier work, more current work emphasizes a greater deal on the integration between Clouds and workflow concepts. The advantages of Cloud computing for different domains are already well defined (see Chapter 3). Concerning the relation between Clouds and agents, it is clear that both can benefit from each other. Cloud computing offers a reliable and scalable infrastructure for the execution of complex processes especially in terms of modeling and simulating. In other side, agents have attractive features like: autonomy, pro-activity, cooperation and mobility. These features help to handle Service Level Agreement (SLA) negotiation, Cloud service discovery and composition, etc. Thus, agents can be used as basic components to bring intelligence in Cloud systems, in order to make them more adaptive, flexible in both resource management, service discovery/provisioning and running complex applications. The most relevant references in this area are [Talia, 2012, Sim, 2012, Aversa et al., 2010, M. Spata, 2011a, I. Foster, 2004, Jander and Lamersdorf, 2013, Braubach et al., 2011]. For example, in [Aversa et al., 2010], the notion of *Cloud agency* is presented. Authors used a mobile agent platform that permitted the

user to dynamically add and configure services on the virtual clusters provide by the Cloud. Multi-agent systems were also used to construct a Cloud computing federation mechanism to permit portability and interoperability among different Grid/Cloud computing platforms (see [M. Spata, 2011a] and [Foster and Kesselman, 2004]).

An overview of the investigation can be found in [Bendoukha et al., 2013,Bendoukha, 2014]. Furthermore, in Part III, the notion of a Drop-Engine is introduced. Its purpose is to provide support to Cloud administrators and developers to move the execution of processes to the Cloud or to invoke Cloud services from workflow models. Moreover, a generic agent-based Cloud architecture is proposed in Chapter 12. It focuses on the system architecture, leaving most of Cloud management services (pricing, accounting and virtualization) and other functional modules aside.

The results obtained in Part II, which concerns mainly the relation between Clouds and workflows will certainly enhance the development of Cloud-based application following the agent paradigm. The integration of all these concepts and technologies is investigated particularly in Part III. In the latter, the concept of a Drop-Engine is presented as an example for a general process execution engine that is specifically developed for Cloud environments.

The main objective of this chapter is to understand the Mulan/Capa framework (see Section 6.4). It describes the four layers namely the Infrastructure, the Agent platform, Mulan Agents and Mulan Protocols. Following the four layers, the concurrent agent platform (Capa), which is an extension of the Mulan architecture is presented. After, the Paose approach and it's six steps are explained: Requirements analysis, coarse design, ontology implementation, role implementation, interaction implementation and integration. The notion of team organization is also presented, which is a central concept of the Paose approach.

## 6.2 Agents and Multi-agent Systems

A multi-agent system is an association of synchronized, autonomous agents, which interact with each other to achieve common goals (objectives). In order to address the research topics mentioned above, the area of multi agent systems is introduced here.

Other properties that an agent may have is the ability to learn, to perceive and influence an environment as well as to interact with other agents or human users of a computer system. Although the notion of agents is central for an agent-oriented approach of software development, there are different views about what constitutes an agent. In [Wooldridge, 2009], the author states that autonomy is a central feature of an agent. To achieve its goals, the actions of the agent can be performed in response to a perceived state of the environment or be proactively initiated by the agent.

### 6.2.1 Definitions

There are several definitions for the notion of agent, on which multi-agent systems are based [Wooldridge, 1997, Ferber, 1999].

> *"An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives"* [Wooldridge, 1997].

According to Ferber [Ferber, 1999, p. 9] An agent is a physical or virtual entity which:

- is capable of acting in an environment,
- can communicate directly with other agents,
- is driven by a set of tendencies,
- possesses resources of its own,
- is capable of perceiving its environment
- has only a partial representation of this environment
- possesses skills and can offer services,
- may be able to reproduce itself,

and whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perceptions, its representations and the communications it receives.

An agent is defined as an autonomous software entity, embedded in the environment and maybe proactive or cooperative. It is important to keep in mind that an agent, unlike other programs, must be at least simultaneously:

1. Located: it perceives the world / environment in which it is situated;
2. Autonomous: it makes decisions and acts on its environment to achieve its goal;
3. Interactive: it has the ability to interact with other agents.

Below are some of the characteristics of agents [Talia, 2012]:

- Autonomy: is the ability to behave autonomously on behalf of users or other programs.
- Pro-activity: the ability to pursue their own individual set goals, including by making decisions as result of internal decisions.
- Re-activity: the ability to react to external events and adapt their behavior and make decisions to carry out their tasks.
- Communication and Cooperation: the ability to communicate and interact with other agents to achieve a goal.
- Learning: the capability to improve performance and decision making when interacting with external environment.

### 6.2.2 Types of Software Agents

In order to systematically consider the term *agent*, several taxonomies have been proposed that try to classify the aspect of agents into categories. The most common taxonomy comes from Franklin et al. [Franklin and Graesser, 1997], it is inspired from the biological model. At the first level, there are living creatures *biological agents*, Artifacts (Robotic agents) and abstract concepts (Computational). At the next level, Artificial Life Agents are distinguished from software agents. The difference between these two types is primarily in the environment, in which they find themselves. Software agents exist in a computer system, which fulfill its own purpose. Artificial Life agents "live" in an artificial world that was created specifically for these agents, eg for research purposes (simulation). Agents are further classified into Task-specific agents, entertainment agents and computer viruses.

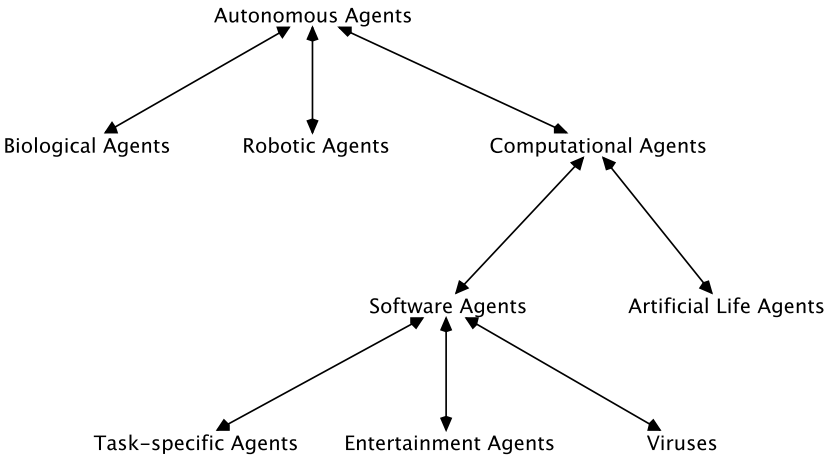



Figure 6.1: Agents Taxonomy [Franklin and Graesser, 1997]

### 6.2.3 Multi-Agent Systems

Multi-agent systems are one of the latest generation of intelligent systems. They arose from the research in distributed artificial intelligence in the 80's. Multi-agent systems are distributed computing systems and like all distributed systems, they are composed of a number of interacting computational entities. However, unlike classical distributed systems they, and their constituent entities, are intelligent.

Again, Ferber [Ferber, 1999] delivers an useful definition: *The term multi-agent system (MAS) is applied to a system comprising the following elements:*

1. *An environment, E, that is, a space which generally has a volume.*
2. *A set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E. These objects are passive, that is, they can be perceived, created, destroyed and modified by the agents.*

3. *An assembly of agents, A, which are specific objects ($A \subseteq O$), representing the active entities of the system.*

4. *An assembly of relations, R, which link objects (and thus agents) to each other.*

5. *An assembly of operations, Op, making it possible for the agents of A to perceive, produce, consume, transform and manipulate objects from O.*

6. *Operators with the task of representing the application of these operations and the reaction of the world to this attempt at modification, which we shall call the laws of the universe.*

*[Ferber, 1999, p. 31]*

As stated in [Sycara, 1998], characteristics of MASs are that:

- each agent has incomplete information or capabilities for solving a problem and, thus, has a limited viewpoint of the global task to be done;
- there is no system global control;
- data are decentralized; and
- computation is asynchronous.

## 6.3 Foundation for Intelligent Physical Agents

The Foundation for Intelligent Physical Agents (FIPA) is a standardization committee, which was established in 1996 in Switzerland and belongs to the Institute of Electrical and Electronics Engineers (IEEE). The goal of FIPA is to define common standards for agent systems to enable interoperability between different systems. Specifications are focused on the basic structure and main components of an agent system, and the communication between agents. A description of the individual components and their relationships have been summarized in an abstract architecture ( [FIPA, 2002a]). It represents a collection of concepts which are developed for the integration of agents and agent systems of various concrete architectures and software systems. A more detailed description of this abstract architecture can be gleaned in [Duvigneau, 2002]. The next two sections deal with the administration of agents and of the agent communication.

### 6.3.1 Agents Management

The specification [FIPA, 2004] specifies the abstract architecture and defines a reference model, and summarized all the ingredients to an agent platform. This agent platform is used to manage agents and provides an infrastructure in which agents can act. Below, the components of the *Agent Management Reference Model* are described (see Figure. 6.2).

**Agent** Agents are the basic unit of an agent platform according to FIPA. They communicate via the *Message Transport System* (MTS) in the *Agent Communication Act*. FIPA agents have a unique global identifier, an *Agent Identifier* (AID), and has to be

assigned at least to one owner. This can include, for example, a human user or an agent. As indicated in Figure. 6.2 agents can also access other softwares when needed.

**Agent Management System** The *Agent Management System* (AMS) must be exactly one time present for each agent platform. It manages and controls the life-cycle of the agent and the access to the platform. Agents must register at the AMS. Registering is necessary different actions, for example, if they want to unsubscribe or if they want to migrate to/from another platform or its execution is completed. When registering, the AMS also awards the valid agent identifier and stores it along with the associated addresses. Via the AMS, it is possible to search an agent platform and furthermore information about registered agents can be requested.

**Directory Facilitator** The *Directory Facilitator* (DF) is an optional element of the agent platform. Agents can register their offered services at the DF, which will be via the yellow pages-like form accessible for other agents. The DF is constantly trying to have accurate and current information about the registered agents.

**Message Transport System** The *Message Transport System* (MTS) is used for agent's communication within and outside the agent platform, and consists of several components The *Message Transport Service* is the communication service for agents, they can with each other communicate internally and externally to the agent platform. This service is offered by the *Agent Communication Channel* (ACC) and through it the messages are exchanged. Each agent must be registered with at least one such ACC. Messages are sent via the *Message Transport Protocol* using the *Agent Communication Language* (ACL). The structure of the ACL message is described in the next section.

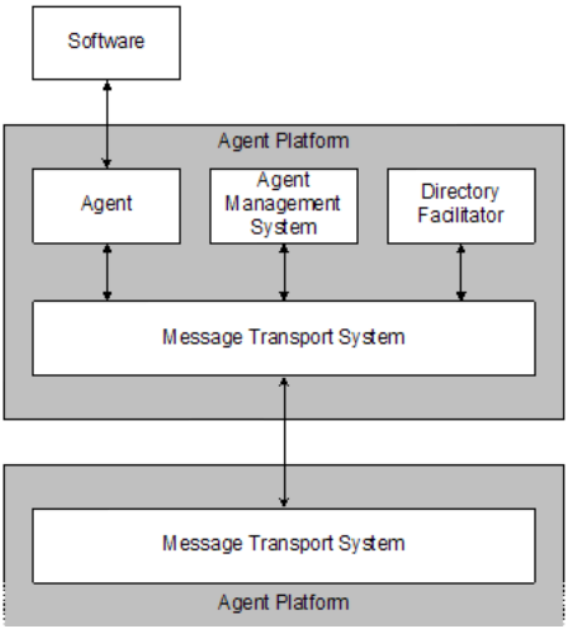



Figure 6.2: FIPA Agent Management reference Model(from( [FIPA, 2004]))

| Parameter | Description |
|---|---|
| per-formative | communicative act that will be executed with this message |
| sender | sender of the message |
| receiver | one or more receivers of the message |
| reply-to | receiver of a reply to this message |
| content | message content |
| language | content language |
| ontology | that is used for the message |

Table 6.1: Structure of an ACL-message

### 6.3.2 Agent Communication

Agents in multi-agent system (MAS) must be able to interact and communicate with each other. This usually requires a common language, an Agent Communication Language, or ACL. In order to communicate, agents must be able to:

- Deliver and receive messages - at this physical level, agents must communicate over agreed physical and network layers to be able to deliver and receive strings or objects that represent messages
- Parse the messages - at the syntactic level, agents must be able to parse messages to correctly decode the message to its parts, such as message content, language, sender
- Understand the messages - at the semantic level, the parsed symbols must be understood in the same way

An ontology describing the symbols must be shared or explicitly expressed and accessible to be able to decode the information contained in the message. A message content ontology helps agents to describe facts, beliefs, hypotheses and predication about a domain. Ontologies range in abstraction from very general terms to terms that are restricted to specific domain of knowledge [Singh, 1998].

Message-based communication between agents is an important component for an agent system. Messages are represented in form of a FIPA-compliant speech acts. Verbs like ('inform', 'ask', etc.) are called performatives. The structure of a message is defined by the [FIPA, 2002c] specification and uses the *Agent Communication Language* (ACL). Accordingly, an ACL consists of a set of parameters, which are represented as key / value pairs. A selection of the most common parameters are listed in Table 6.1.

The only non-optional parameter is the per-formative and must appear in each message. It specifies the purpose of the message and helps the agent to detect what kind of communicative act it is. On the basis of these acts, the agent decides what actions to perform. The content of a message is specified in a certain language that is determined by the parameter 'language'. A possible content language is specified by the

FIPA is Semantic Language (SL) [FIPA, 2002d]. In the specification [FIPA, 2002b] the given communicative acts are described using formal modes (Sl) with the appropriate performative.

## 6.4 Mulan/Capa and Paose Approach

Mulan represents the conceptual reference architecture. Capa represents the technical embedding of Mulan and is also used for the technical framework. Paose is the whole approach of the development of agent-based systems in the context of Mulan and Capa. In the following, a brief overview of Mulan/Capa as well as the Paose approach is given.

### 6.4.1 Mulan/Capa Framework

Mulan (**Mul**ti-**A**gent **N**ets) [Cabac, 2010] is a reference architecture of a multi-agent system which is based on Renew and modeled in reference nets. Mulan is a specification for multi-agent systems that was presented by Heiko Rölke [Rölke, 2004] in his dissertation (for the first version of Mulan see his diploma thesis: [Rölke, 1999]). The resulting architecture is strongly based on the abstract architecture of FIPA [FIPA, 2002a] and extends it to the concept of agent platform [FIPA, 2004]. Figure. 6.3 shows the structure of Mulan and includes four layers *Infrastructure*, *Agent Platform*, *Agent* and *Protocol*.

#### 6.4.1.1 Infrastructure

The top level (labeled with *Infrastructure*) represents the Agent system and connects the agent platforms with each other. Each place of the *Infrastructure* net contains an agent platform (more specifically a reference to the platform), the transitions and arcs represent the communication channels to provide cross-platform communication. In this way agents can communicate with agents from different platforms and can also migrate to another platform.

#### 6.4.1.2 Agent Platform

The structure of an *agent platform* is displayed at the top right (labeled as "Agent Platform"). At the place in the middle of the net, all agents of the platform are stored (only their references). Transitions of the platform manage the agent's life-cycle and can also create and terminate agents. In addition, the platform offers communication channels for internal and cross-platform messages.

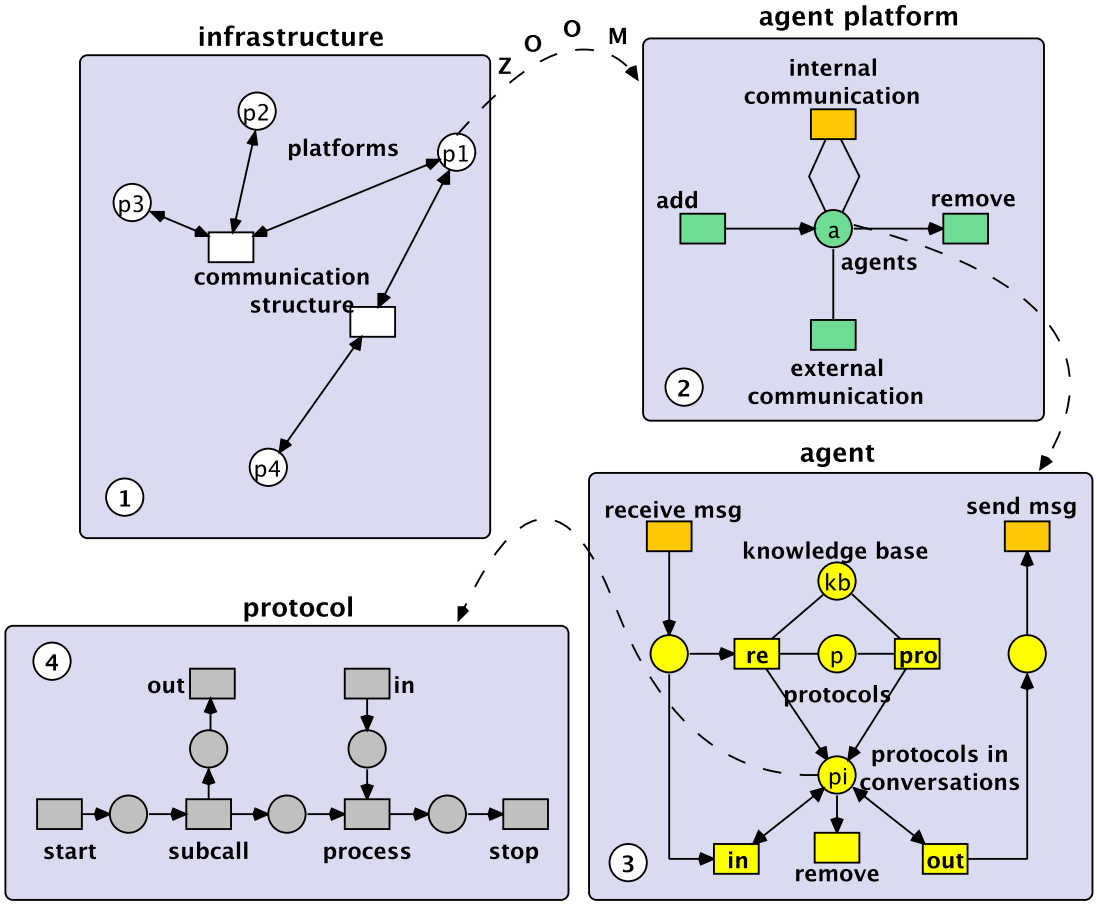



Figure 6.3: Mulan Architecture (from( [Cabac, 2010]))

#### 6.4.1.3 Mulan Agents

The third layer is shown in Figure. 6.4 is a simplified version of the agent layer. Mulan agents reside on platforms. Each place of the platform holds all the agents of that platform. Mulan agents can receive or send messages to other agents over the platforms in which they are situated. The incoming and outgoing synchronous channels of the agent provide this functionality.

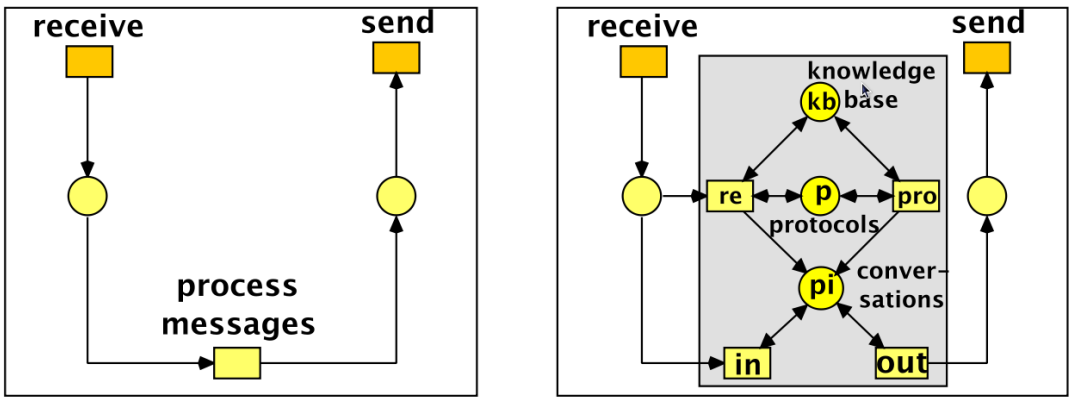



Figure 6.4: Mulan Agent

The three central places represent the state of the agent, the top place contain the agent's knowledge base, the middle one contains the protocols and the bottom one contains instantiated protocols that was started as conversations. The two central transitions with the labels **re** and **pro** are responsible for the instantiation of the protocols. They represent the actions of the agent and can fire either *re-actively* (re) to a received message or *pro-actively* (pro). To act pro-actively, the necessary preconditions have to be agreed in the knowledge base. Then, a suitable protocol is instantiated and stored on the conversation' s place. Reactive action requires a received message, which belongs to no existing conversation. With the help of the knowledge base, the agent determines which of the available protocols is adapted to react to the message. The selected protocol is instantiated as a proactive action and stored at the **conversation's** place. Messages of the agent platform reaches the agent via the transition **receive**. According to the incoming message on an ongoing conversation can the transition **in** fire and forwards the messages to the appropriate protocol net. Via the transition **out**, a protocol instance can send messages. The knowledge base serves as a central repository of all data and should be available to the agent persistence. It is a special protocol that is only once instantiated and during the creation of the agent and has to be active during the lifetime. The content of the knowledge base is individual for each agent and equipped with a basic knowledge at the initialization. From the protocols, entries from the knowledge base can be read, added and removed.

#### 6.4.1.4 Mulan Protocols

Mulan protocols are Reference nets which define the behavior of the agents during the communication. They control the communication between agents. They define the activities of each participating agent at a certain time. They include sequence, concurrency or decisions. An agent can use numerous protocols and instantiate multiple instances of various protocols at the same time. Figure. 6.5 shows an abstract Petri net model to illustrate a simple scenario. With each communication, Mulan agents have to instantiate a protocol net and messages are transmitted via a medium provided by

agents. In case the agents are hosted in the same platform, TCP/IP protocols are not used.

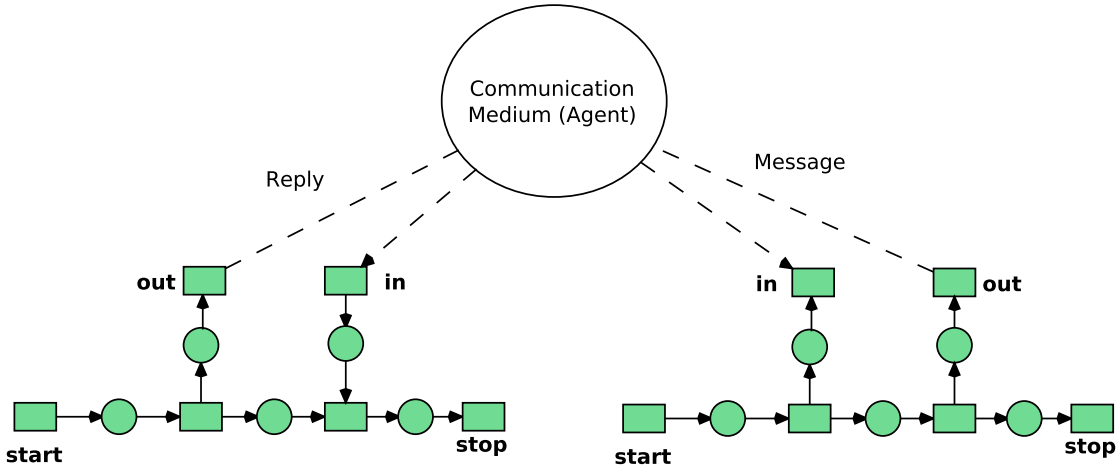



Figure 6.5: Mulan Protocols (from( [Cabac, 2010]))

#### 6.4.1.5 Capa

Capa is an extension of the Mulan architecture to reach the FIPA-compliance. It especially focuses on communication between different agent platforms. The objective of Michael Duvigneau in his master thesis was the adaptation of Mulan to the FIPA's standards. Thereby Capa makes mainly adaptations to the top two levels of the Mulan architecture and leaves the other levels almost unchanged. Agents are hosted in several platforms that are connected through a technical communication infrastructure and together build the multi-agent system as whole. The communication between agents is performed in terms of agent communication language FIPA-ACL and domain specific ontology. Agents can be either *service providers* or *consumers*. A (distributed) *directory service* is used to identify the other partners. In order to comply to FIPA, Capa has to provide for its agents the management and directory services *Agent Management System* (AMS) and *Directory Facilitator* (DF), a local Message Transport System (MTS) and an interface for communication with external platforms, the Agent Communication Channel (ACC).

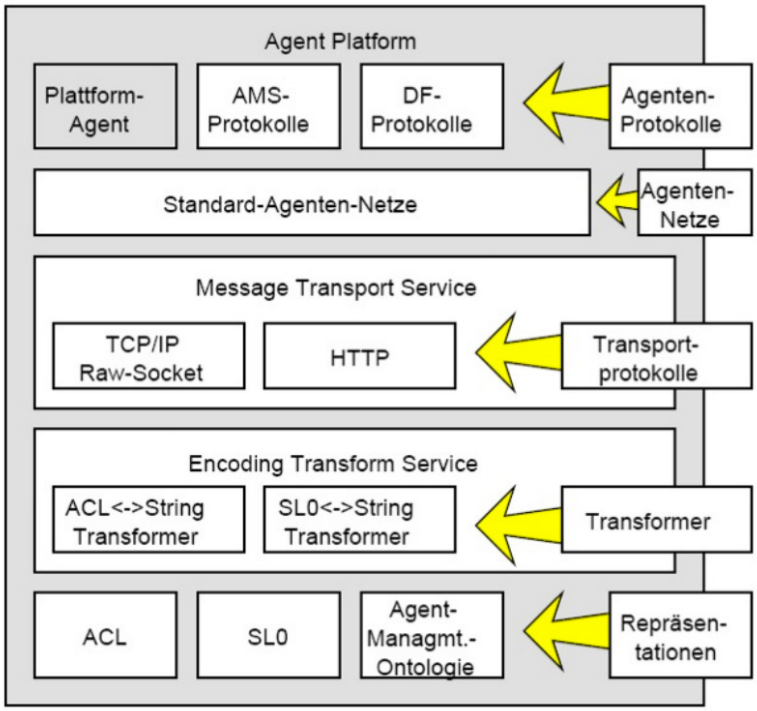



Figure 6.6: Capa-Platform (from( [Duvigneau, 2002]))

### 6.4.2 Paose

For developing agent-based applications, the Paose approach is exclusively followed. This approach will be presented in the following sections. For more information about the Paose approach, it is recommended to refer to [Cabac, 2010].

#### 6.4.2.1 Overview

First approaches of agent-oriented software development led to the development of the agent platform Mulan [Rölke, 1999]. Based on this, and on the extension Capa [Duvigneau, 2002], various teaching projects were accomplished. The Paose development approach was formulated from the experience that have been made over a number of teaching projects. This approach consists of a framework(Renew/Mulan/Capa) and there are continuously developing tools thanks to the plug-in system already introduced.

The Paose[1] (Petri net-based Agent-Oriented Software Engineering) approach joins the benefits of Petri nets and software engineering paradigms. It follows the multi-agent paradigm to structure complex distributed systems in a comprehensible manner. It allows modeling the processes explicitly and on the same level of importance as the system structure. Paose not only covers technical issues of implementing multi-agent

[1] In alternative interpretations of the P stands for *Processes* or *Persons* and the O for *Organizations*.

systems with Petri nets, but it also provides methods, techniques and tools that build upon the multi-agent paradigm and Petri net theory to guide the development process.

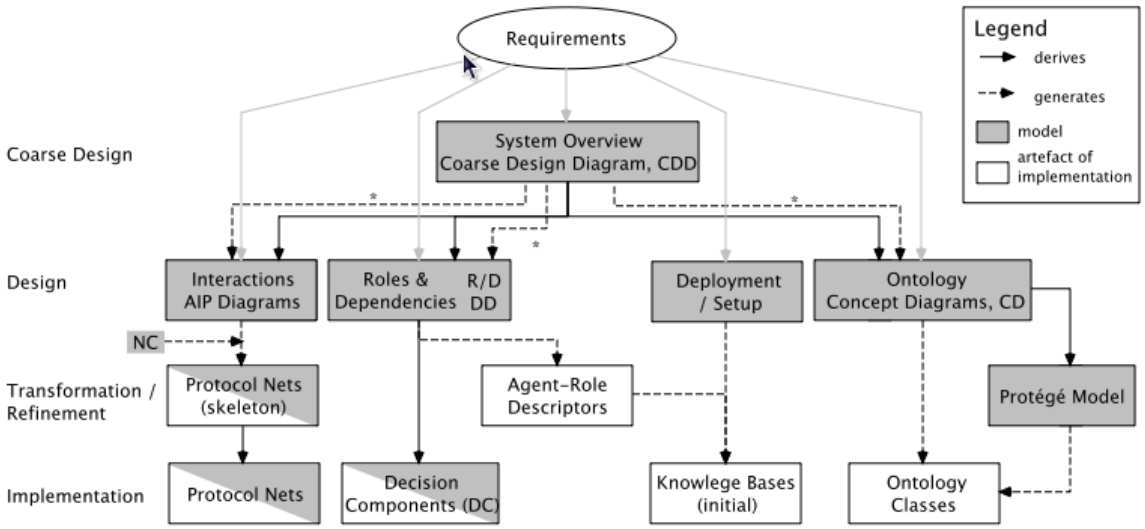



Figure 6.7: Overview of the Paose Approach (from [Cabac, 2010])

In the Paose approach the development process is sketched by six tasks types:

1. Requirements analysis
2. Coarse design
3. Ontology implementation
4. Role implementation
5. Interaction implementation
6. Integration

During the development of multi-agent systems with Capa, three types of software techniques artifacts have to be generated:

- Agents, characterized by their knowledge base and their protocols;
- Interactions between agents, specified in form of agent protocols and AUML;
- Ontology's classes, to describe the concepts that occur in the multi-agent system, and over which the agents communicate with each other.

#### 6.4.2.2 Team Organization

A central concept in the Paose approach is the organization of the development team according to the concepts identified in the development of multi-agent systems. Accordingly, there is an allocation of tasks to different agents/roles, as well as to the identified interactions. This leads to a matrix organization, where in one dimension appear the different agent's roles and in others the interactions. Figure. 6.8 shows an example of such a matrix.

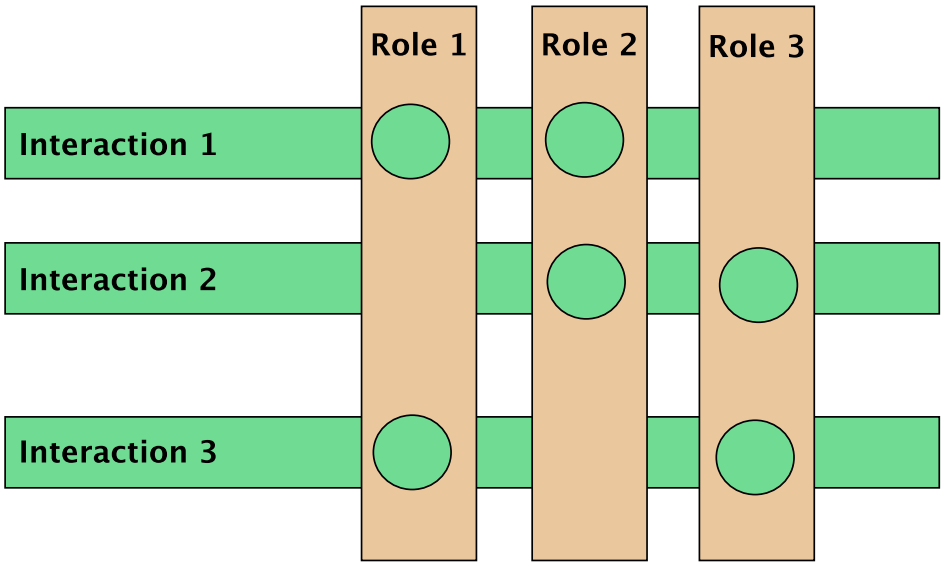



Figure 6.8: Matrix Organization (from [Cabac, 2010])

A circle in the diagram indicates that the corresponding role is involved in the interaction. The developer of the role must coordinate with the developers of the interactions in which their agents are involved. The development and maintenance of the ontology represents another dimension, which is also orthogonal to the above mentioned dimensions. Coordination at the intersections and the respective responsibilities must be clearly understood. Within the development of an interaction, the creator has the responsibility of the agent protocols and messages that will be exchanged. The processing and the decision-finding within the agent is then the responsibility of the agent creator who cares about the knowledge base and the decision components. Also the registration of the protocol in the knowledge base falls within its scope. However, it is difficult to precisely determine the boundary.

## 6.5 Conclusion

This chapter contains the last concepts, that are used in this dissertation. It includes an introduction to the the notion of agents and multi-agent systems. Based on this introduction, the Mulan/Capa framework is presented. This framework is exclusively

used to design and build agent-based applications following the Paose approach, which is also described in this chapter. Part III serves the purpose of demonstrating how an agent can help develop Cloud-based applications. The next chapter concludes Part I and summarizes all the conceptual and technical background already presented.

# Summary

This chapter gives a summary of the first part. The contributions brought by concepts, techniques and technologies (Web services, Cloud computing, Petri nets and multi-agent systems) are outlined. Part I provides an introduction to SOC, Clouds, Petri nets and multi-agent systems as conceptual and technical background of this work. In Chapter 2 SOC and its related concepts: Web services and SOA are presented. Web services, service-oriented architectures (SOA) and Cloud computing are strongly related to each other. Web services cover Cloud computing because the latter uses those services to enable different connections between several entities. Nowadays, most system architectures follow a service-oriented approach and Cloud computing is no exception. A service provider can be in the Cloud or not. The main system architecture can involve both Cloud and non Cloud service providers. Web services and their related technologies play an important role in the proposed approaches and are used in different part of thesis. Many proposed solutions have been implemented based on Web services. For example, Web services have been applied in Chapter 5 and Chapter 8 to explain the notion of *synchronous channels* and implement some prototypes. Here, the objective is to demonstrate how external Web services can be invoked from Petri net models.

In Chapter 3, both Grid and Cloud computing are introduced. Also the similarities between these two computing paradigms have been presented. Approaches from Grid domain can also be exploited to resolve problems in Clouds such as service composition or workflow scheduling. Middlewares[2] are crucial entities when designing and developing Grid applications. It is considered to be out of scope to mention them since they are not related to the current work.

Chapter 4 and Chapter 5 concern process management and workflow modeling. Petri nets and related concepts/tools such as *reference nets* and Renew editor are presented in detail. In Chapter 6, agents, multi-agent systems and Mulan are presented.

In the following part, the relationship between Clouds and workflow concepts is deeply investigated. There are three main topics that will be addressed. First, a state-of-the-art study is elaborated to detect the issues when *deploying* processes to the Cloud. Second, the question: How Cloud computing can support developers when developing complex and large-scale workflows is investigated. Finally, workflow concepts will be introduced to improve building workflows in terms of modeling and execution.

[2] A Grid Middleware Distribution is a software stack or a set of cooperating components, services and protocols which enable users access to the distributed resources of a grid.

# Part II

# Workflow Management in the Cloud

# Table of Contents

# Introduction

According to [Bell et al., 2009]: *"In the future, the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, workflow management, visualization, and Cloud computing technologies"*. The latter citation characterizes approximately the work described in this part. In fact, introduced approaches and methodologies cover the following issues: data management, workflow management and visualization of results.

This part presents the relation between Cloud computing and workflows and the contributions of the current work that enable this relation. It shows how Cloud technology can benefit from the integration of workflow concepts and vice versa. The relation between workflow and Cloud can be addressed in different ways. On the one hand, there is *Cloud for workflow* which consists of using Cloud resources to execute workflows and especially scientific workflows [Hoffa et al., 2008] [Juve et al., 2009]. Such works are more resource-centric and focus on the computational tasks. On the other hand, Clouds needs structured and mature workflow concepts and high level languages to handle issues like managing complex task and data dependencies. Existing workflow architectures need to be adapted into the Cloud and workflow management systems should be integrated with Cloud infrastructure and resources [Pandey et al., 2011a].

This part introduces techniques and tools as means for enabling an efficient integration between workdflows and Clouds. The focus in this part is on the techniques and tools that support Cloud application developers to specify, deploy and execute workflows in Cloud and later in Inter-Cloud environments. Chapter 7 gives an introduction to the challenges that one is confronted to, when moving applications to the Cloud. Patterns are also presented, which shows different scenarios of moving data, activities and process engine to the Cloud. The chapter discusses existing approaches and scenarios to deploy applications in one of the three deployment modes of Clouds, (infrastructure, platform and software) Moreover, the chapter shows how the implemented prototypes fit to the the existing approaches.

Chapter 8 presents the first technical contribution of the thesis. It consists of the RenewGrass plug-in for Renew. The reason behind the implementation of the plug-in is related to the intention to have a use case to showcase the features delivered by other contributions. In fact, several examples are based on RenewGrass. RenewGrass allows modeling and execution of scientific workflows by reference nets with Renew. The domain of application of RenewGrass is remote sensing.

Chapters 9 and 10 present several essential contributions, which are the Cloud Task Transition (CTT), Inter-Cloud Nets (ICNETS) and the ICWorkflow plug-in.

Chapter 9 presents how one can execute workflows specified by Renew in the Cloud. It introduces techniques to allow Cloud instances support the simulation of Petri net models. The notion of CTT is first introduced (see Chapter 10). It follows the concept of the workflow task transition [Jacob et al., 2002] and extends it to adopt the Cloud computing technology. After that, ICNets are presented. ICNets are predefined Petri net-based models that allows the specification of Cloud-based workflows and their execution. A tentative towards the formalization of the ICNets is also presented.

# Chapter 7

# Moving Processes to the Cloud

This chapter focuses on the notion of *moving* the execution of processes to the Cloud. In the current work, processes are Petri net-based. The investigation is driven by the following works [Anstett et al., 2009,Han et al., 2010,Strauch et al., 2013]. It is necessary to note that the emphasis of the current work is on enabling execution of workflows in the Cloud. Since workflows are specified by Petri nets with Renew, the objective is concretely to move Renew from on-premise to the Cloud. Migrating applications between Clouds or communicating different workflows with each other is out of scope. However, a conceptual as well as technical solution is provided to overcome this issue. This chapter is essential to understand the contributions presented within the next chapters.

## 7.1 Introduction

Cloud computing is currently one of the most important technology trends. The word *Cloud* covers the IT services as well as the compute- and storage capacities, which are configured and delivered dynamically and adapted for application requirements. Nowadays, Business Process Management (BPM)/Workflow management in the Cloud is a hot topic. There is a lot of research on how to deploy a part or all the components of the WfMS into the Cloud. In this chapter migration issues and methodologies are discussed. Several approaches have been proposed to manage workflows in the Cloud, each of them focuses on different perspectives of the workflow.

For now, most worflow systems are based on a single Cloud. Workflow tasks are executed on a Cloud provider, which is in priori selected. This has a limitation in term of performance and flexibility working with different Cloud providers.

In this dissertation, an emphasis is made on the *process* view. Coupling Cloud technology and workflow concepts has different aspects, which are addressed in this work. It concerns the following topics:

- Invoking Cloud services to execute workflow tasks
- Moving existing workflow applications to the Cloud, to get more performance

- Provision of adapted modeling techniques for Cloud-based workflows

Based on the work of [Anstett et al., 2009, Han et al., 2010, Strauch et al., 2013] an investigation is made on the challenges when developing workflow management solutions for the Cloud. The main issue is which entity and functionality should be moved to the Cloud? The security aspect of the migration process is also crucial, but it is out of scope of this dissertation.

## 7.2 BPM (Workflow Management) in/for the Cloud

Despite several attractive features that the Cloud technology offers, moving an existing application to the Cloud should be based on a solid strategy. It is necessary that these applications should be adapted to the new computing paradigm. There are many issues when putting the business management system (or WfMS) or a part in the Cloud. For example, Cloud users could lose control of their own data in case of a solution based fully on the Cloud. Some activities, which are not compute-intensive can be executed on-premise rather than moving them to the Cloud. This transfer can be time- and cost-consuming because of the pay-per-use model.

In [Anstett et al., 2009], different ways of moving an application to one of the Cloud service model (Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS)) are investigated. In the following sections, the scenarios presented in Figure. 7.1, Figure. 7.2 and Figure. 7.3 are discussed.

### 7.2.1 Business Process Infrastructure as a Service (BPIaaS)

In that case, the Cloud users are responsible for the applications, the operating system and the middleware. This situation is similar to working on-premise, i.e the Cloud user is free to install the required softwares for his BPM system. Additionally, security matters should be taken into consideration to avoid attacks, by locking ports and enforcement of access control policies.

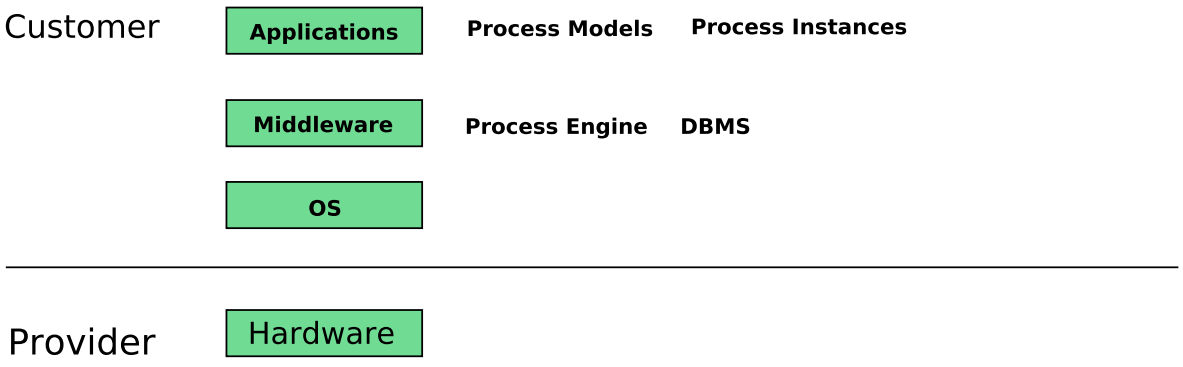



Figure 7.1: BPIaaS (Adapted from [Anstett et al., 2009])

### 7.2.2 Business Process Platform as a Service (BPPaaS)

In the platform layer, Cloud users have no control over the underlying infrastructure (servers, storage, operating systems) but have control over the deployed applications. Unlike in the IaaS solution, the PaaS providers own the operating system, middleware (process engine) and the hardware. From confidentiality perspective, both configuration data such as the process models and runtime data (which are processed by the the business process) need to be encrypted and signed order to avoid an unauthorized transactions in the system. This ensures that the process models are only readable for the authorized entities and not from the intruders. The access to the data base can also be a problem because the process engine read and store process models and also instances. Such data need an encryption too.

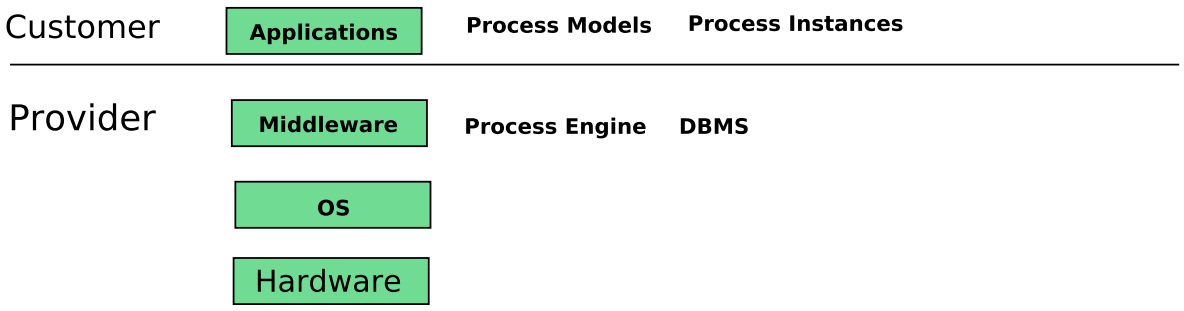



Figure 7.2: BPPaaS (Adapted from [Anstett et al., 2009])

### 7.2.3 Business Process Software as a Service (BPSaaS)

The provider does not have to worry about the middleware, operating system and even the application itself. Nevertheless, offering an application to multiple leads to two kind of architectures (single- and multi-tenant). The first one implies the installation of one process engine for each process model. In the multi-tenant architecture, the process engine can serve multiple process models (or Cloud users). There are here also many issues related to the security and solutions are discussed in [Anstett et al., 2009].

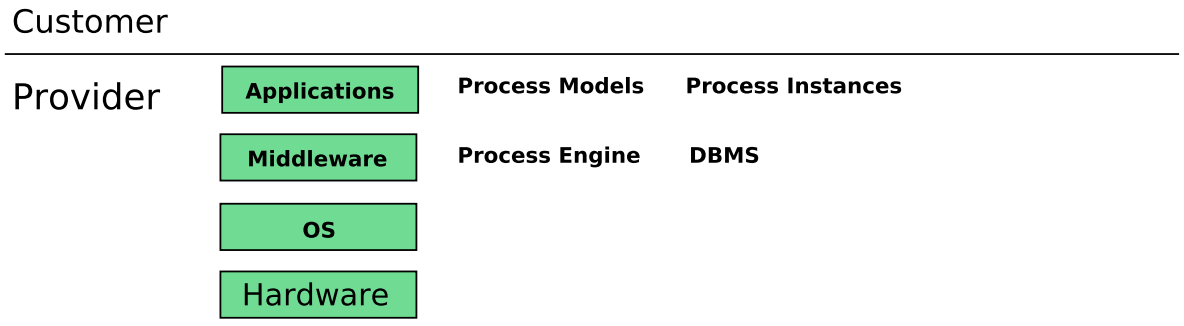



Figure 7.3: BPSaaS (Adapted from [Anstett et al., 2009])

## 7.3 Combination

In a normal situation, BPM solutions and their correspondent entities such as process engine, activities and required data are on the same platform. Later when Cloud solution is adopted, these entities are either on the user side or hosted in the Cloud. In [Han et al., 2010], the authors propose a model for a distributed BPM in the Cloud. Based on [Han et al., 2010] and [Anstett et al., 2009] an analogy is made with the contributions provided by this thesis. It will be the basis background for different prototypes developed within the next chapters. In the works cited above, there are three main questions, which are investigated:

1. where to enact processes?
2. where to execute the activities?
3. where to store the data?

They introduce a PAD model (see Figure. 7.4), the P designates Process enactment engine (responsible for the execution and the monitoring of the activities), A designates the Activities that need to be executed by the business process and D designates the Data that are required by the business process.

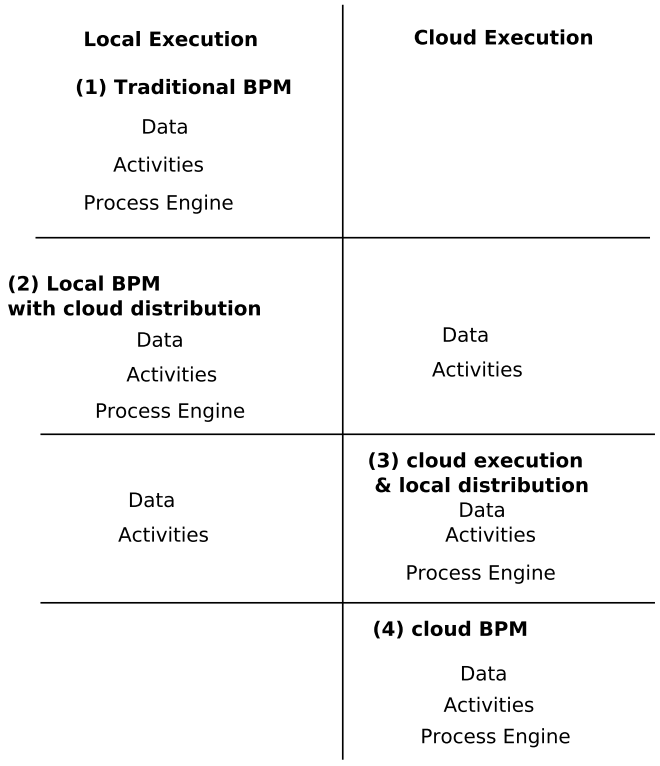



Figure 7.4: Patterns for Cloud-based BPM (Adapted from [Han et al., 2010])

The first pattern is to design a BPM system where all the components are in the same side precisely in the user side. The second pattern represents a case when the business

process contains computation-intensive activities, so they are moved to Cloud to offer more performance. The third pattern design a situation, where the end-users do not have a BPM system, so they use a Cloud-based BPM system that is utilized on-demand. In the latter case, the non computation-intensive activities can be executed locally. The fourth situation (pattern) is used when the whole BPM strategy is based on a Cloud solution, i.e that all the activities, the data and BPM system is hosted on a Cloud.

A first observation on the previous model is the absence of some scenarios. For instance, a situation where the process engines are both on-premise and on the Cloud. There might be also a cooperation between these process engines during the execution of workflows. This scenario is very significant for the current work. This is due to the aim of creating an Inter-Cloud environment, where services from different Cloud providers are invoked. In that scenario, different activities (tasks) can be distributed over the Cloud resources.

The authors in [Han et al., 2010] proposes an architecture of a Cloud-based BPM system with respect to user-end distribution (see Figure. 7.5). There is an emphasis on the data-flow aspect of the application. They started from the fact that data might contain sensitive data. Therefore, the management system deployed in the Cloud should be able to protect this data. They propose to use an encrypted tunnel to transfer data between local site and the Cloud. Sensitive data is stored at the user-end and non-sensitive data is stored in the Cloud.

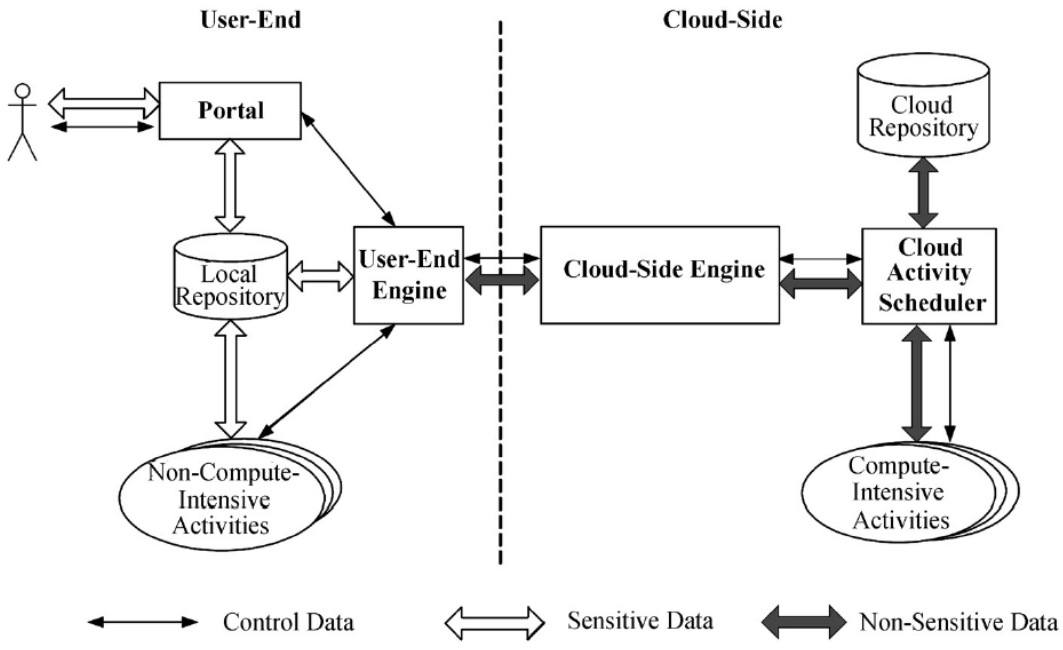



Figure 7.5: Architecture of Cloud-based BPM combined with user-end distribution [Han et al., 2010]

## 7.4 Execution Scenarios

The migration to the Cloud as defined in this thesis takes different forms and progress in terms of complexity. The prototypes implemented within the next chapters cover multiple execution scenarios. Based on concrete examples from the developed prototypes, different execution scenarios and their relation with above mentioned patterns are presented. The RenewGrass plug-in for image processing of satellite imagery will be frequently used as use case to give more details about the scenarios.

### 7.4.1 Local Execution

The first scenario (see Figure. 7.6) concerns a classical case, where all entities are hosted on-premise. There is no need to perform the execution of the workflow on external nodes. Concerning image processing, Renew, RenewGrass and data are all hosted on-premise. The contribution here consists of the implementation of the RenewGrass plug-in (see Chapter 8).

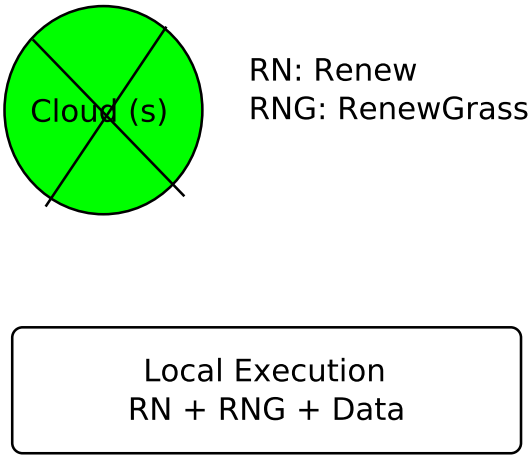



Figure 7.6: Local Execution

### 7.4.2 Local Execution with Local Distribution

This scenario concerns a situation where Renew simulations are performed on VM's but always on the same host. This is the first step towards moving to the Cloud. The first challenge is to enable Renew simulations on the remote host. Therefore, a whole mechanism is developed to create VM's and provision them with required softwares. Even this solution does not show clearly the advantage of using distributed systems for workflow execution. However, it can serve for testing purposes before moving to the Cloud. Another reason is purely related to Renew. In fact, since VM's can run different operating systems, so it is possible to evaluate Renew simulations on different OS. More details about enabling Renew simulations on external machines is described in Chapter 9.

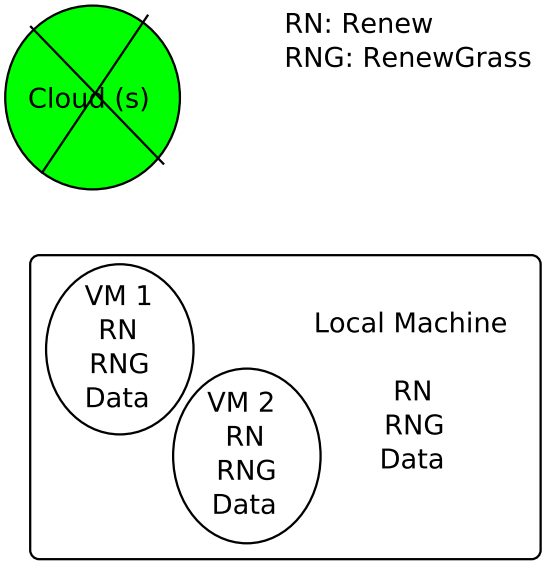



Figure 7.7: Local Execution within External Resources

### 7.4.3 Execution in the Cloud

This scenario concerns a situation where all the entities (data. activities and process engine) are hosted in the Cloud. It corresponds to the last case presented in Section 7.2.3 and to the third pattern from Figure. 7.9 (Cloud execution and local distribution). There are here different scenarios, which are presented below.

#### 7.4.3.1 Renew Simulations in the Cloud

The objective is mainly to gain more performance in case there is not enough power in local site. The idea consists of the execution of Renew processes (Petri nets) in the Cloud. The problem with this scenario is that the target Cloud execution platforms are generally not able to execute such kind of processes. First mechanisms should be implemented to allow Cloud instances to support Renew models. Figure. 7.8 shows the the big picture of the presented idea. The prototype is based on OpenStack Cloud. Cloud instances can be configured to be equipped by Renew and other required softwares. Furthermore, in that work the notion of *interfaces* is introduced and discussed. Different kinds of interfaces are proposed, from simple to complex. This is explained in depth in Chapter 9.

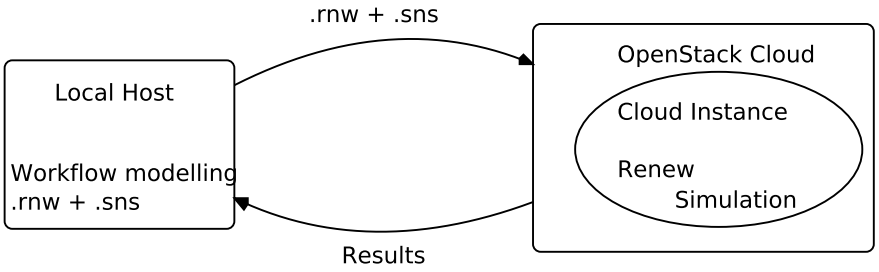



Figure 7.8: Renew Simulation in the OpenStack Cloud

#### 7.4.3.2 RenewGrass in the Cloud

RenewGrass is a tool for Renew, which allows the specification and execution of image processing workflows related to the remote sensing domain. To show the relation between the approaches from Section 7.2 and Section 7.3, Renew is the process engine, the activities are the geoprocessing tasks (performed by the Grass services), which are related to the satellite images (data). The latter (data and Grass GIS) can be either on-premise or hosted in the Cloud. Based on these elements and the illustration presented above, Figure. 7.9 presents an overview of the diverse approaches (patterns) to design the workflow system based on the Cloud technology.

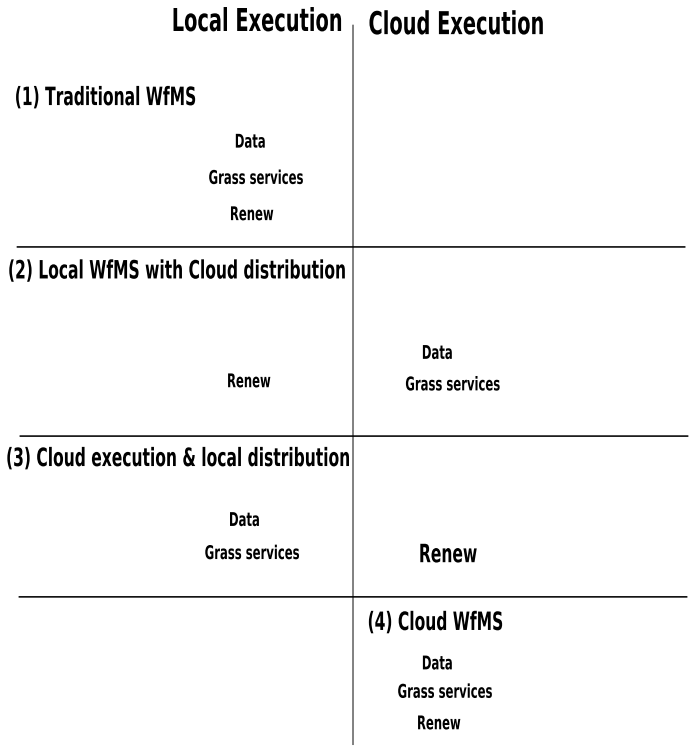



Figure 7.9: Patterns for Cloud-based Workflow Systems (Adapted from [Han et al., 2010])

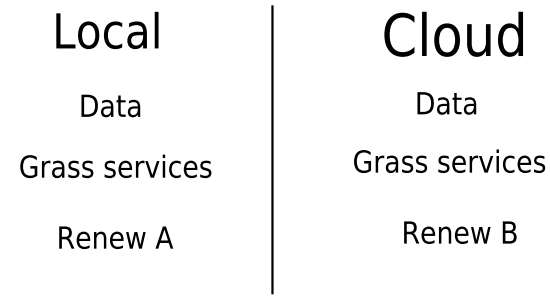



Figure 7.10: A Pattern Designing Multiple Process Engines Integration

Figure. 7.10 shows approximately how this scenario looks like. The proposed solution consists of the idea to transfer the data and to execute the process by another process engine. In order to provide an explicit understanding, this has been applied to the RenewGrass plug-in (see Chapter 8). In [Bendoukha et al., 2015b], issues and migration patterns are discussed. They concern moving applications and application data to the Cloud. Based on these patterns, an agent-based architecture proposed to integrate the current implementation in Cloud-like environments, i.e., the components making up the workflow management system are mapped into services provided by specific agents. In general, agent concepts are employed for improving Cloud service discovery, composition and Service Level Agreement (SLA) management. The elaborated architecture is based on the Mulan/Capa framework and following the Paose approach. The objective is that agents perform the Cloud management functionalities such as brokering, workflow submission and instance control. Technically, the OpenStack framework was adopted for the creation and the management of the Cloud instances. The architecture is presented in Part III.

#### 7.4.3.3 Cloud services to Workflow Tasks

The scenario addressed in this section corresponds to the first case (see Section 7.2.1 and to the second pattern (see Section 7.3). It includes the work presented in Chapter 10 and Chapter 11, respectively ICNets and ICWorkflow. In both contributions, the objective is to enable invoking services from different Cloud providers in order to execute workflow tasks. This is the first difference with the solutions presented above. In fact, not the whole workflow is concerned by Cloud integration, but only some of its tasks. For each workflow task it is possible to specify the Cloud service to use as well as the required data that need to be processed. ICNets provide conceptual as well as technical solutions. They are based on reference nets and support Cloud application developers when building Cloud-based workflows and also their management. In this scenario the process engine (Renew) and the data are at local site. The activities (workflow tasks) are also executed on-premise. On the other side, the ICWorkflow plug-in in Renew provides ICNets with means to communicate with the Cloud.

## 7.5 Conclusion

In this chapter moving processes to the Cloud is investigated. It is proved that moving local executions to the Cloud is not trivial and should be based on a solid strategy. One of the obstacles is security of data that need to be transfered to the Cloud. However, despite the security of data, there are several ways to technically enable the migration to the Cloud. After presenting existing approaches based on the work of [Anstett et al., 2009, Han et al., 2010, Strauch et al., 2013], contributions from this dissertation are briefly introduced. These contributions prove that several scenarios concerning the use of Cloud computing and workflows are supported. With respect to the above study, one can perceive that both [Anstett et al., 2009] and [Han et al., 2010] do not address all possible situations. For instance, the following situation has been not addressed: the process engine is available on the user side but due to circumstances (internal failure, not sufficient compute or storage resources), remote process engines need to be integrated and remotely invoked. The next chapter (see Chapter 8) presents the first technical contribution of the dissertation. It consists of the RenewGrass plug-in for Renew.

# Chapter 8

# Image Processing with Renew: The RenewGrass Plug-in

In this chapter, a new plug-in for Renew named RenewGrass is presented. It allows the development of image processing workflows based on the Geographic Resource Analysis Support System (Grass) Geographic Information System (GIS).

## 8.1 Introduction

In order to show case the functionality and the usability of the introduced approaches, examples from the remote sensing domain have been modeled, deployed and executed using new implemented tools. One of these tools is RenewGrass. The idea behind RenewGrass is that Renew is originally not adapted to handle the special kind of *scientific workflows* especially image processing automatically. Renew's users are obligated to implement all the processing steps using Java and use them as net inscriptions, which is arduous and not accessible to unexperienced users. Therefore, RenewGrass is a new plug-in for Renew to exactly overcome the above issue and makes it possible modeling and implementing image processing workflows by references nets.

Nevertheless, Renew is principally dedicated to teaching purposes (AOSE project). It is not adapted to deal automatically with image processing workflows and by far with concrete distributed systems such as Grids or Clouds. The domain of application covers mostly business workflows. One of the significant works in this direction are the development of a WfMS by [Jacob et al., 2002] and [Carl, 2004] based on reference nets. In order to support scientific workflows, Renew users are obligated to implement all interfaces and processing steps (Java) and use them for example as net inscriptions over the transitions, which is arduous and not accessible to unexperienced users. RenewGrass extends Renew in term of supporting new kind of workflows such as scientific workflows. The main aspect of the plug-in is its usage of *reference nets* for modeling/execution of image processing workflows. Reference nets and Renew have been already presented in Chapter 5. As case study, remote sensing especially the processing of satellite imagery is selected. Modeling and implementing those workfows need specific techniques, tools and external libraries, which are unfortunately not

available in Renew. In order to overcome these issues, RenewGrass is strongly based on the Grass GIS [Neteler et al., 2012]. Grass is a popular open source GIS that provides powerful raster and vector GIS capabilities. The modules of Grass (see Table. 8.1) are now accessible from the Petri net transitions as net inscriptions. This allows users to invoke Grass GIS modules directly in their Petri net models, which are executed later. Furthermore, as soon as the workflow requirements become locally unsatisfied, the workflow tasks need to be mapped to distributed resources. This issue is also taken into account, since the long-term perspective is to provide service-oriented environment built on top of Cloud resources and to allow flexible deployment of scientific workflows.

To summarize, the main objectives of this new plug-in are:

- Adapt Renew to handle scientific workflows
- Provide unskilled users by means for ease modeling of their workflows
- Exploit the power of the Grass GIS, in order to implement various image processing workflows
- Reduce the gap between the specification of the workflow and its implementation

In Section 8.2, a prototype, which consists of calculating the Normalized Differences Vegetation Index (NDVI) and the Enhanced Vegetation Index (EVI) values is explained. Next in Section 8.3, an overview of the Grass GIS is given as well as the integration issues in Renew. In Section 8.4 the architecture of RenewGrass is illustrated. Grass *Net Components* are introduced in Section 8.5. Section 8.8 concludes the chapter and evaluates the obtained results.

## 8.2 First Prototype: The Image Processing Workflow

In order to show how RenewGrass is used in concrete development process, an implementation of the NDVI (Figure. 8.1) as well as the EVI (Figure. 8.2) workflows is given. It was applied to a satellite imagery taken from the LANDSAT-TM7. The first workflow is adapted from (`https://trac.osgeo.org/grass/wiki/HowToTestGrass6`) with the OSGEO educational data set North Carolina. With the same manner, more complex vegetation indexes or image processing workflows in general can be implemented. The NDVI is a numerical indicator that uses the Red and near-infrared bands. It subtracts the red reflectance values from the near-infrared and divides it by the sum of red and near-infrared band.

$$NDVI = (NIR\text{-}Red)/(NIR+Red)$$

The whole process as described in
`https://trac.osgeo.org/grass/wiki/HowToTestGrass6` consists of:

1. set current region/resolution to map:

```
$ g.region rast=lsat7_2002_40 -p
```

2. display metadata:

```
$ r.info -h lsat7_2002_40
```

3. generate NDVI:

```
$ r.mapcalc "ndvi = 1.0 * (lsat7_2002_40 - lsat7_2002_30)
 / (lsat7_2002_40+ lsat7_2002_30)"
```

4. display metadata of the ndvi:

```
$ r.info -r ndvi
```

5. set color to ndvi:

```
$ r.colors ndvi color=ndvi
```

6. set current region/resolution to map:

```
$ g.region rast=lsat7_2002_40 -p
```

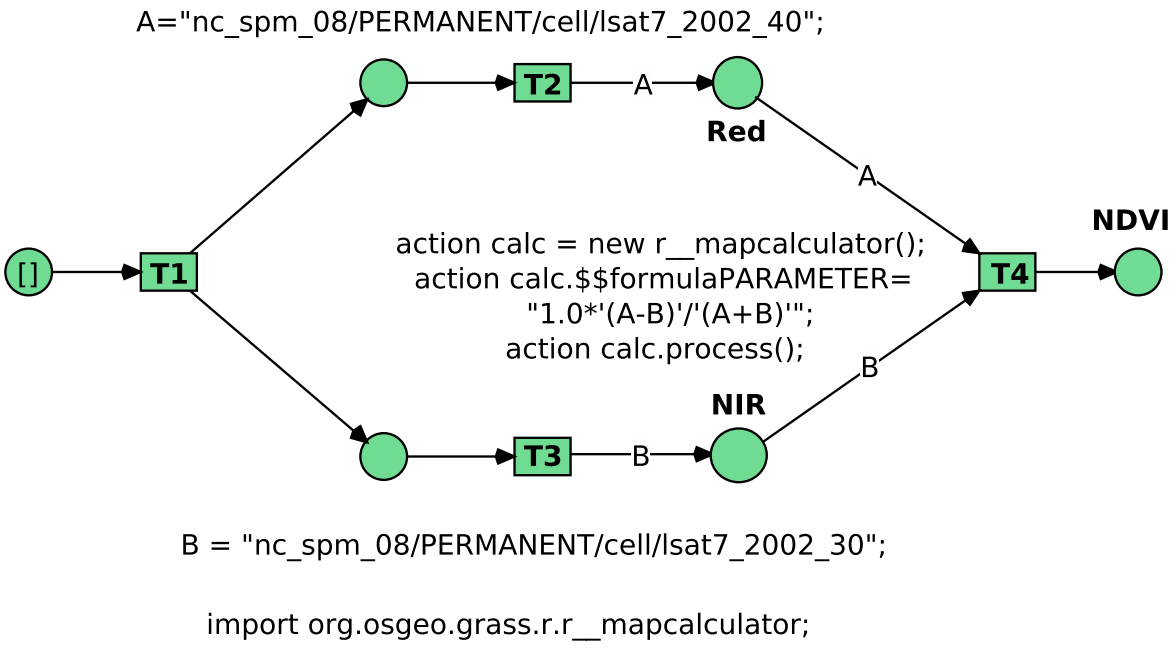



Figure 8.1: NDVI Calculation using RenewGrass

lsat7_2002_40 and lsat7_2002_30 are the third and fourth spectral bands. Figure. 8.1, shows the simplified net model corresponding to the NDVI workflow described above. As it is shown, not all transitions require Grass commands, but only (T4). When a Grass command is required (for example r.mapcalculator), the package (module)

containing the command should be declared exactly the same in Java with *import*. After the completion of the workflow, the results are displayed through the monitor. Display command are also supported and can be called directly from the net (see Figure. 8.1). With respect to the modules presented in Table. 8.1 raster (r.*) and display (d.*) are required for the computation and the visualization of the NDVI value.

On the other side, the EVI is a also a numerical indicator that uses Red, Near-infrared and Blue bands. It is an optimized index designed to enhance the vegetation signal with improved sensitivity in high biomass regions. Figure. 8.1, shows the Peri net model corresponding to the EVI workflow. After analyzing these two workflows, one can easily observe that there are some tasks that are repeatable. For example, one needs always to upload images and specify the mathematical operation that calculates the values. The Grass modules (see Table 8.1) required are also the same: *display (d.*)*, *imagery* (i.*) and *raster* (r.*). To avoid that, Grass *Net Component* are introduced, which allow using predefined Petri nets blocks dedicated to perform specific processing. The EVI formula is as follows:

*2.5 * (nirchan - redchan) / (nirchan + 6.0 * redchan - 7.5 * bluechan + 1.0)*

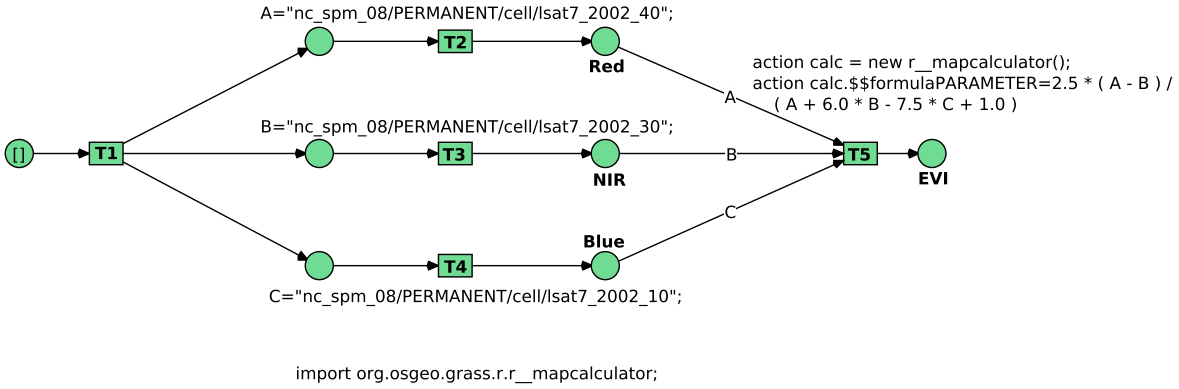



Figure 8.2: The EVI Workflow Modeled and Calculated using RenewGrass

The results can be easily displayed by calling the *d.mon* module. Figure. 8.3 shows both the original satellite image (left side) and the calculated NDVI on the right.

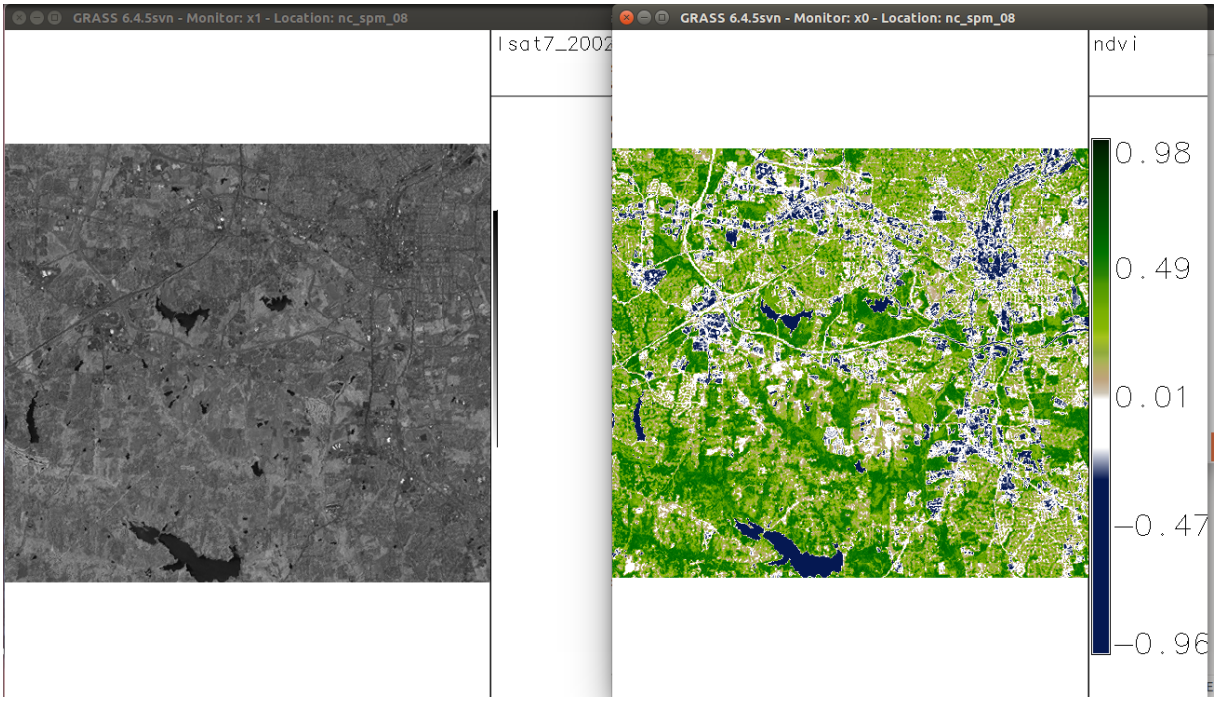



Figure 8.3: The NDVI of a Landsat-TM Image

## 8.3 The Integration of Grass GIS in Renew for Remote Sensing Applications

This section explains the process of integration of Grass GIS in Renew. First, a short overview of the Grass GIS is given. It includes the Grass modules as well as the required structure of a GIS project. Next, the architecture is introduced to show all the components taking part to the integration.

### 8.3.1 The Grass GIS

Here an overview of the Grass GIS is given. Grass is a multi-purpose open source GIS, which can be used for geoprocessing applications such as: geospatial data production, analysis and mapping. It can handle raster as well vector data. The most important specificity of the Grass GIS is its modularity, which diminishes overhead. This allows to run only the required modules (same as Renew). These modules are organized in categories (general GIS modules, raster modules, vector modules, etc.). Table. 8.1 shows the modules provided by the Grass GIS [Neteler et al., 2012].

Table 8.1: Grass GIS Commands (from [Neteler et al., 2012])

| Prefix | Function class | Type of command |
|---|---|---|
| d.* | display | graphical output |
| db.* | database | database management |
| g.* | general | general file operations |
| i.* | imagery | image processing |
| m.* | misc | miscelaneous commands |
| ps.* | postscript | map creation in Postscript format |
| r.* | raster | 2D raster data processing |
| r3.* | 3D raster | 3D raster data processing |
| v.* | vector | 2D and 3D vector data processing |

In order to allow Renew invoking these commands directly from the Petri net transitions, RenewGrass offers a wrapping layer, which makes the Grass modules available at runtime. Before using the modules for processing the data, the latter should be first imported into a Grass *DATABASE*. Within the DATABASE, the projects are organized as subdirectories called "LOCATIONS". Each LOCATION can have one or more MAPSETS. Each *MAPSET* may represent a sub region within a given LOCATION. These are mostly the important variables that need to be set.

### 8.3.2 Integration Issues

The first obstacle when trying to integrate Grass GIS with Renew is that these tools are written in different programming languages (The Grass GIS is written in C and Renew in Java), which makes a direct communication between them arduous. Thus the Grass GIS needs to be adapted to the running environment of Renew. Moreover, a proper environment variables need to be pre-specified. In order to achieve this integration, there are two possibilities:

- *Desktop integration*: this means that use the Grass GIS locally and creates interfaces to provide geoprocessing functions for Renew
- *Web-based integration*: in this case, the objective is to publish and execute geo-processes over the web, following the Web Processing Service (WPS) interface specification[1] from the Open Geospatial Consortium (OGC)[2]

In this chapter, the first solution is chosen. The integration using the WPS is here omitted but this is discussed in Chapter 13 and concerns moving the current work to be

[1] http://www.opengeospatial.org/standards/wps
[2] http://www.opengeospatial.org/

executed in the Cloud. Figure. 8.4 gives a simple overview of the integration of Grass GIS as desktop application. With Renew, scientific workflows are specified as Petri net models. When some of workflow tasks require Grass commands, this can be easily performed directly at the transitions (see section 8.2). In order to communicate directly with the Grass GIS, an interface or a wrapper is necessary.

Over the last few years, some effort has been dedicated to leverage the strength of Java and Grass GIS. There are only few works dealing with this issue. Two candidate projects are interesting for the current work, which are the *JGrasstools*[3] and the *vtkGRASSBridge*[4]. The *vtkGRASSBridge* provides a VTK/C++ interface to most of the GIS GRASS raster, voxel[5] and vector C library functions This library can be used to build comprehensive 3D visualization of GIS GRASS data with Java, Python and C++. Although the project seems promising, it was quickly rejected, due to building issues. For this work, tools from the JGrasstools project have been taken. JGrasstools is a library that is extracted from the Java Geographic Resources Analysis Support System (JGrass) project[6].

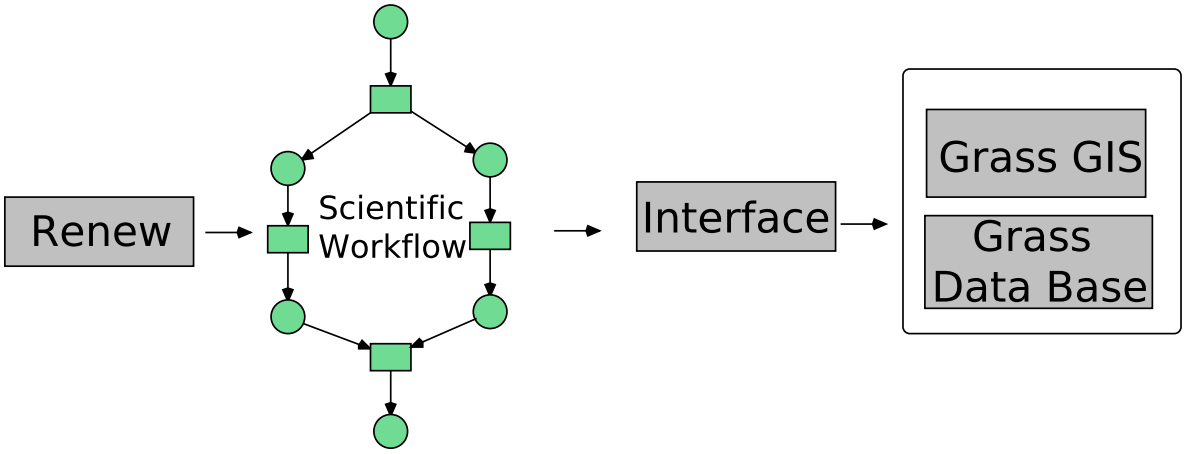



Figure 8.4: Grass GIS Integration in Renew

## 8.4 Architecture

As mentioned above, Renew's architecture has been decomposed into several components. Each component is characterized as a plug-in. This provides more flexibility and extensibility. Thus new features can be easily integrated. The so called plug-in system [Duvigneau, 2010] is responsible for adding and removing of plug-ins at runtime.

[3] http://moovida.github.io/jgrasstools/

[4] https://code.google.com/p/vtk-grass-bridge/

[5] A voxel represents a value on a regular grid in three-dimensional space. Voxel is a portmanteau for "volume" and "pixel" where pixel is a combination of "picture" and "element" (Wikipedia).

[6] JGrass is a free, multi platform, open source GIS based on the GIS framework (see www.jgrass.org).

New features include some new plug-ins as for instance the *Workflow* and the *WFNet*. which which provide workflow management functionality. An overview of these plug-ins is given in [Jacob et al., 2002]. RenewGrass is also built following the plug-in architecture of Renew. Figure. 8.5, shows simplified view of the position of RenewGrass in Renew.

The *Workflow* and the *WFNet* plug-ins are not required when using RenewGrass. They are especially used when workflow management functionality are required such as log-in, tasks management, etc. As you can see in the figure, the RenewGrass plug-in is built on top of the JGrassTools, which is adapted for Renew. The main requirements are the Grass GIS installation and the Grass data. The first one provides all the modules presented in Table 8.1. The Grass data is a directory, that holds all the required files (raster or vector images). Both Grass GIS installation and the Grass Data path should be specified to RenewGrass prior any utilization. The current implementation of the tool allows local execution only, since both Renew and Grass GIS are installed on-premise.

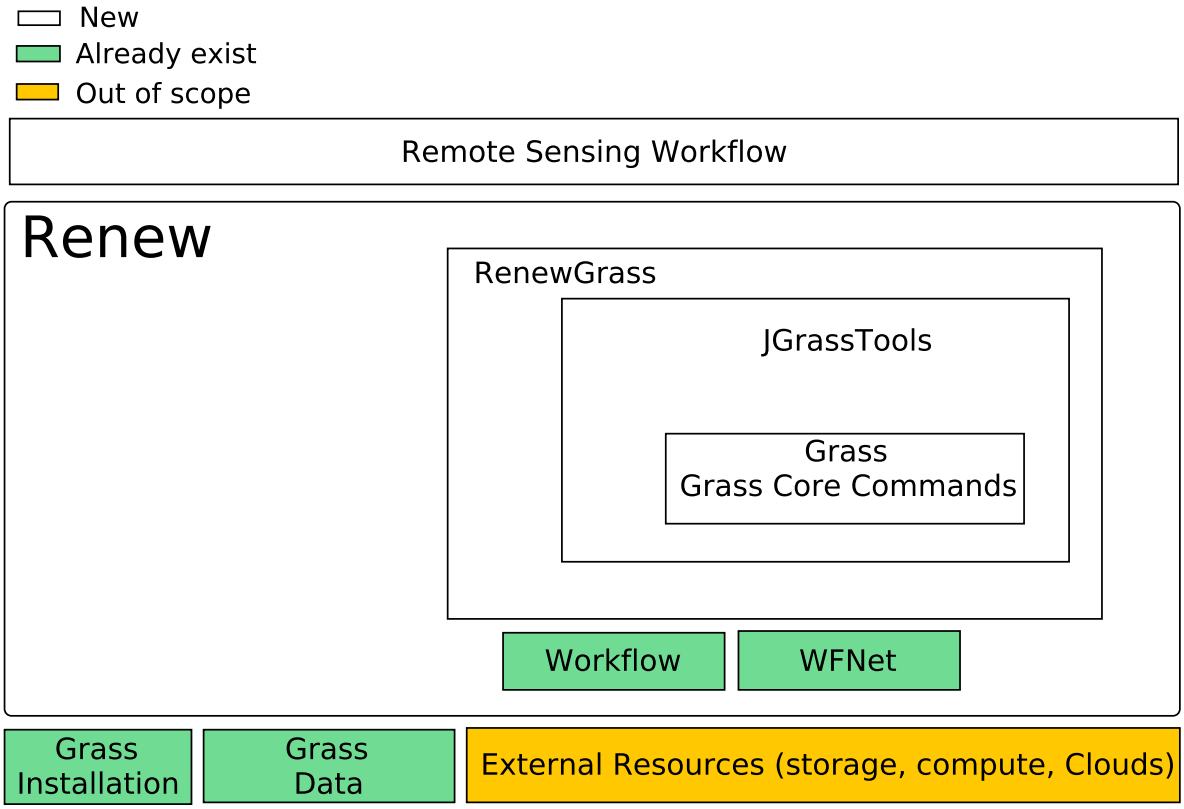



Figure 8.5: Architecture of the RenewGrass Tool

Although RenewGrass does not require an additional graphical user interface for the modeling and the simulation of workflows, front-end functionalities have been implemented. For instance, Figure. 8.6 and Figure. 8.7 shows the Grass Nets Components and other features, which can be selected from the Renew palette.

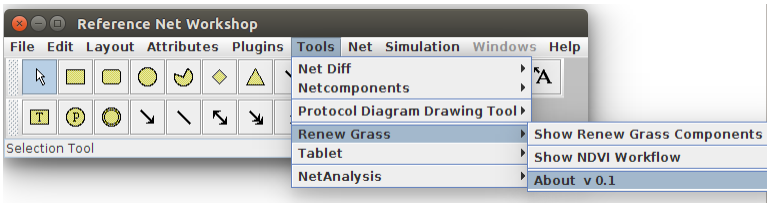



Figure 8.6: RenewGrass Menu in Renew

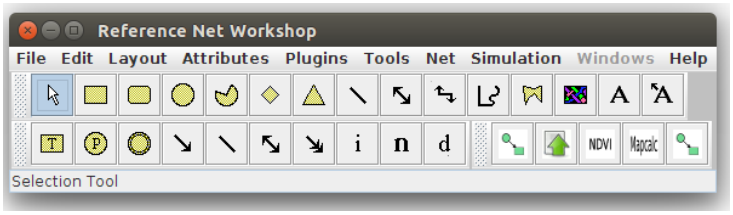



Figure 8.7: The Grass Net Components

## 8.5 Grass Net Components

Image processing workflows contain often repetitive tasks, especially inputs from users. As a use case the NDVI (see Section 8.2) is presented, transition calculating the NDVI value always require two images (Bands) as inputs The mathematical expression still the same. To avoid this and to reduce modeling's time, Grass Nets Components are provided and are based on the work of [Cabac, 2010]. Net components (NC) are Petri net structures (subnets), which can be composed or combined to from a large net. The net component can provide additional help, such as a default inscription or comments. Functionality already implemented can be used without going through the the process of 'low level' implementation again [Cabac, 2010]. With *Net Components*, it is possible not only to model and analyze the system, but also for the implementation and to accelerate the modeling process. Thus the model can be transformed to an implementation without the change of the formalism. Additionally, there are also Mulan Net Components, which are designed for Mulan protocols (see Chapter 6). Net components can be classified into two categories such as generic net components (control flow), Mulan Protocol Specific Net Components (Protocol Management, Messaging) [Cabac et al., 2006].

For example, Figure. 8.8 shows some of the predefined nets structure, which are configured and added to the Renew palette. As a result, Figure. 8.9 shows the NDVI workflow modeled using the Grass Net Components. This reduces considerably the modeling time, especially when the workflow is more complex than the one reported here.

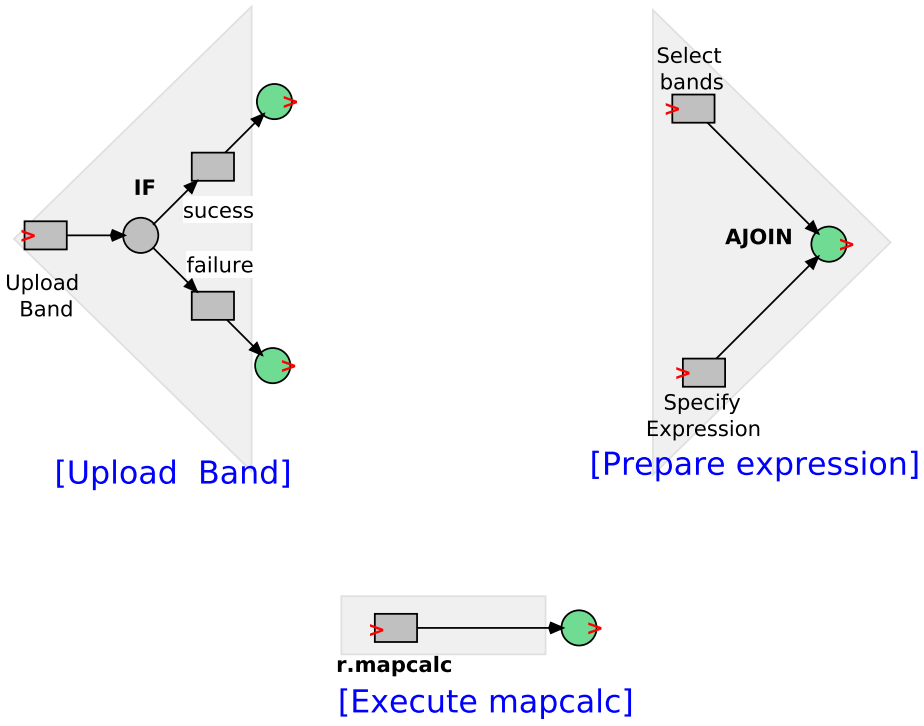



Figure 8.8: Grass Net Components

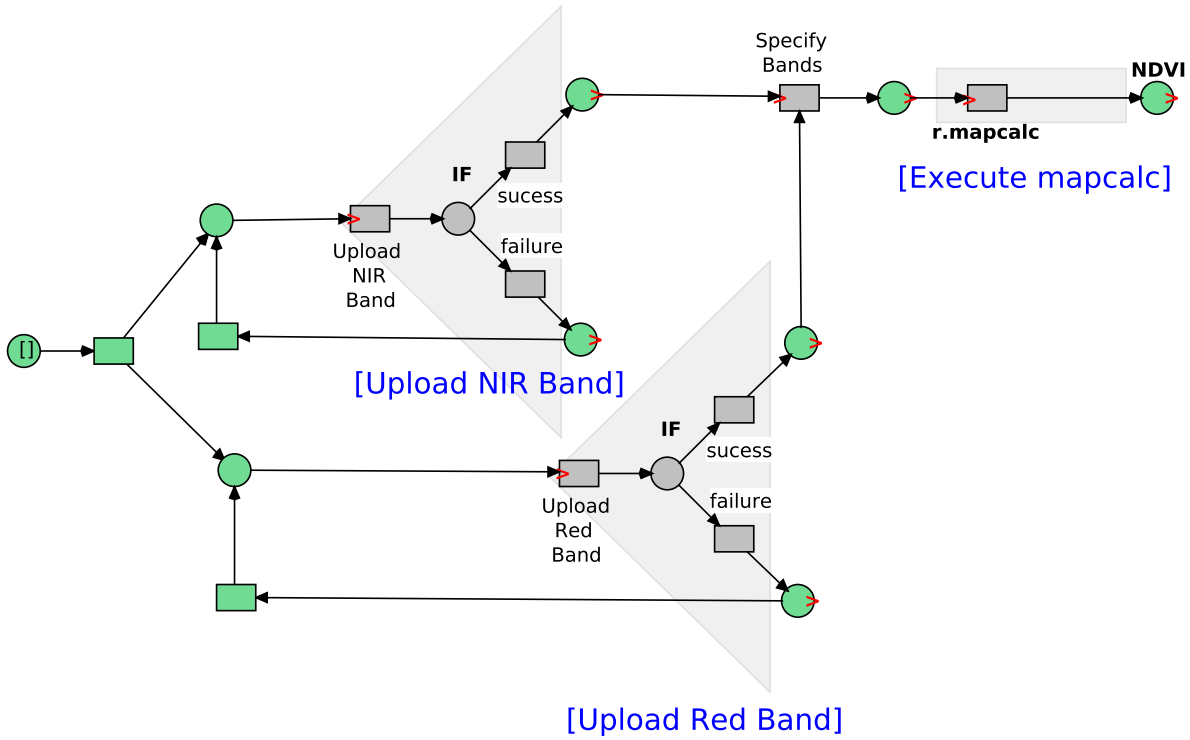



Figure 8.9: An Abstract NDVI Workflow Modeled by the Grass Net Components

## 8.6 Related Work

A model is a simplified representation of a phenomenon or a system. Models in the remote sensing domain are important to analyze and to understand the behavior of the whole processing steps. There are different kind of models, the complete classification. The models are classified by purpose, methodology and logic. In this work there is an focus on *Process Models*, which integrate existing knowledge into a set of relationships and equations for quantifying the physical process. These models are typically raster-based. Most of GIS (commercial or open source) provide tools and assist scientists creating their models created by dragging and dropping tools. A ModelBuilder is a graphical user interface that helps creating models. In ModelBuilder, a process is composed of input data, the tool to apply to the data and the resulting output data. Famous existing ModelBuilders are the ArcGIS ModelBuilder ( [Armstrong, 2009]) and the Grass Graphical Modeler[7]. In ArcGIS, the model is represented by a flux diagram in a graphic user interface that facilitates to create, visualize, edit, and execute geoprocessing workflows, to use and reuse them, to share and apply them to different geographic areas. In the other side, Grass GIS provides also commands for preparing input variables, running and viewing the model. Figure. 8.6 shows the GUI of both ArcGIS ModelBuilder and the Grass Graphical Modeler.

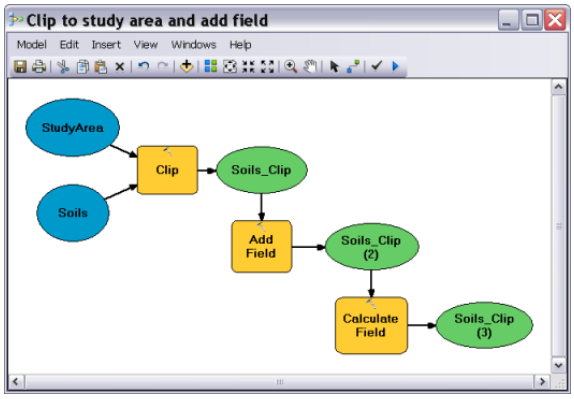



Figure 8.10: The ArcGIS ModelBuilder

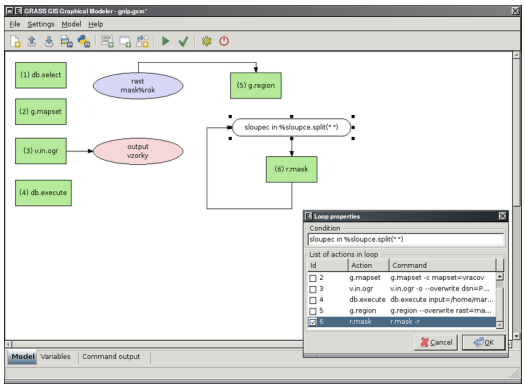


Figure 8.11: Grass Graphical Modeler

## 8.7 Further Documentation

More documentation of RenewGrass is maintained at these web sites: `www.paose.net` and `http://sofianeb.github.io/` respectively. Users can find technical details on how to compile, install and use RenewGrass. A How-To is also available at the end of the thesis (see Appendix A).

[7] `http://grasswiki.osgeo.org/wiki/WxGUI_Graphical_Modeler`

## 8.8 Conclusion

RenewGrass has been implemented for a specific purpose, which is the design and the development of remote sensing applications. In opposite to many other tools dedicated to this purpose, RenewGrass combine the power of the Java programming language and the use of a well-known graphical and mathematically based modeling technique, which makes use of Petri nets. By the integration of RenewGrass in Renew, the latter is now able to handle new kind of workflows, which are scientific workflows. The main objective behind this integration is the test of the approaches and techniques proposed in this thesis. Thus, in each part there is a prototype based on RenewGrass. In the current chapter, simple image processing workflows have been provided. These workflows are not data and computing intensive. In Part II, RenewGrass is also used to implement workflows that are resource-centric, which means that they require external computing and storage resources. Concretely, in this part it is about the integration of the Cloud technology and later the notion of Inter-Cloud computing and here two other prototypes are presented. In Part III another version of the image processing workflow built by RenewGrass is elaborated. The management of the workflow is performed by specific agents. The multi-agent system is designed following the Paose approach and the Mulan/Capa framework.

# Chapter 9

# Cloud Computing for Workflow Execution

This chapter presents techniques and tools for the deployment and the execution of Petri net-based processes in the Cloud. This concerns many aspects of the life-cycle of a workflow like modeling of processes, configuration of Cloud instances and moving the execution of processes to the Cloud. The main objective of the current work is investigate the use of Cloud resources to support Petri net simulations. In addition, how can a Cloud instance support the simulation/execution of Renew modeled processes is investigated.

In this chapter, the idea of *Cloud for workflow* is introduced and questions like "why is the Cloud technology important for application development? and how it can be integrated in the application life-cycle?". For the latter, the benefits that the agent-based applications can take from the Cloud technology are outlined in Chapter 12. The chapter is structured as follows. Section 9.2, presents the conceptual and technical background as well as related work. Section 9.3 introduces the approach and methodology for moving net simulations to the Cloud. Section 9.4 proposes different kinds of interfaces. Section 9.5 discusses the approach and Section 9.6 concludes the chapter and presents future work.

## 9.1 Introduction

Several long-running and high-throughput applications can be designed as complex workflows, which describe the order and relationships between the different activities and related data (input, output). In such scenarios, these tasks often need to be mapped to distributed resources, possibly due to a lack of on-premise resources or failures.

Cloud computing is a recent computing paradigm. It provides an environment that allows the user to dynamically allocate resources for the execution of workflow tasks following an on-demand and pay-as-you-go model. In this dissertation, these resources are used to improve the performance of the applications. These applications are, specified as Petri nets using Renew. In order to make this possible, mechanisms and strategies need to be provided. These mechanisms are based on the integration of

workflow concepts and Cloud technology (and furthermore the agent paradigm). There are different ways to address this. On the one hand, Cloud for workflow uses Cloud resources to execute complex workflows and especially scientific workflows [Hoffa et al., 2008] [Juve et al., 2009]. Such works are more resource-centric and focus on the computational tasks. On the other hand, Clouds need structured and mature workflow concepts and high-level languages to handle issues like managing complex tasks and data dependencies. It should be noted that only moving the execution of entire nets or systems of nets into the Cloud is concerned.

The research described in this chapter focuses more on performance issues, which can be considerably improved by using Cloud resources. Here, an approach is presented to provide techniques and tools to move the execution of complex workflows modeled in Petri nets to the Cloud. The migration to the Cloud is based mainly on user requirements. Thus Quality of Service (QoS) parameters are specified in advance. Modeling and execution of Petri net models is performed exclusively through the Renew editor. Furthermore, different realization possibilities are discussed. Three different types of interfaces are examined, which define how input and output to/from the Cloud are defined. Simple interfaces provide only basic functionality to initiate Cloud workflows and receive results. Simulation interfaces are used to run extensive simulations of workflows in a Cloud environment. Lastly, advanced interfaces feature advanced mechanisms to process input and output data for the Cloud. The main avenue of thought for the advanced interfaces is to utilize autonomous software agents and their characteristics.

## 9.2 Background and Related Work

The approach for managing workflows in (Inter) Cloud-like environments is based on several concepts, techniques and tools. In order to comply to the Paose approach, many of the techniques are based on Petri nets. Petri/Reference nets (see Chapter 5) and agent concepts (see Chapter 6) for example have been already presented.

Originally, WfMS were not conceived to be used in Cloud-like environments. With the growth of Cloud computing, several traditional WfMS improved their kernel and are now able to provide interfaces to communicate with external Cloud services. The prevalent (scientific) WfMS are: Taverna [Oinn et al., 2004], Pegasus [Deelman et al., 2005], Triana [Taylor et al., 2003], Askalon [Fahringer et al., 2005a], Kepler [Altintas et al., 2004] and the General Workflow Execution Service (GWES) [Alt et al., 2006]. The originality of these systems is that they run on parallel and distributed computing systems in order to reach a high level of performance and get access to wide range of external resources. The Pegasus system allows scientists to execute workflows using different resources including clusters, Grids and Clouds. This has been adapted later to execute scientific workflows in the Cloud (within an Amazon EC2 Instance) [Nagavaram et al., 2011]. Compared to current work, the *migration* to the Cloud is almost similar, the difference lies at the modeling level, where reference nets are used as modeling technique. GEWES is an interesting project that makes use of high-level Petri Nets (HLPN) for the description of workflows. The GWES coordinates the composition and

execution process of workflows in arbitrary distributed systems, such as SOA, Cluster, Grid, or Cloud environments. In the workflow specifications, transitions represent tasks while tokens represent data flowing through the workflow.

There have also been much more efforts to infuse Cloud and distribution aspects into general workflow management. The ADEPT project [Dadam and Reichert, 2009, Reichert et al., 2009] focuses on flexible and adaptive workflow management but also deals with distribution and migration aspects to avoid performance bottlenecks in the network. Another interesting combination of Clouds and workflows is the OpenTosca project [Binz et al., 2013]. It utilizes management plans implemented as workflows to configure Cloud applications for organizations. [van der Aalst, 2011a] also deals with configuration issues but focuses explicitly on the configuration of inter-organizational business processes in the Cloud.

## 9.3 Investigating Moving Net Execution to the Cloud

In this section, mechanisms and implementation issues are proposed and discussed. They concern moving the execution of computation- and time- consuming workflows into the Cloud. These complex workflows are specified by Petri nets, more precisely, reference nets using the Renew tool. Cloud technology is a suitable solution to (i) overcome the lack of resources on-premises and to (ii) improve the performance of the whole system based on quality of service (QoS) constraints. As an execution target for simulations, tests have been performed on an OpenStack Cloud (TryStack). Furthermore, the integration and interfaces between workflows, Cloud computing and agent concepts are also addressed.

### 9.3.1 Bringing Renew to the Cloud

Renew *in the Cloud* is a new computing idea, which designs the process of simulating Petri net processes in the Cloud rather than on-premises. There are different reasons to move (Renew) simulations to external (Cloud) platforms, but the main reason is to seek gains in performance. Particularly, (Petri net) models that contain complex and time consuming tasks are of interest here. In this work the design/modeling step is performed at the user's side since it does not require computing or storage capabilities. After this, the models are pushed to the Cloud provider. The Cloud provider should be able to provide instances, that support Petri net simulations. Therefore, Cloud instances need to be *provisioned* by external Petri net editors and simulators. Since Renew is used it will be installed and configured before starting the simulation. The whole process consists of the following steps:

1. Modeling the workflow
2. Configuring the Cloud instance
3. Starting/connecting to the Cloud instance
4. Uploading the required nets

5. Executing the simulation and getting the results

Technically, the implementation is based on the Vagrant tool[1], which permits us to create reproducible development environments. According to the Vagrant homepage, *"Vagrant is a tool for building complete development environments. With an easy-to-use workflow and focus on automation"*. There are three ways to use Vagrant: with a virtual machine, a Cloud provider, or with VMware.

### 9.3.2 First Prototype (with VirtualBox)

In order to run Renew and all required software on the host machine, configuration using a *Vagrantfile* is required. The latter permits the provision the host machine(s) with additional softwares (Renew).

Figure. 9.2 shows the steps to follow for the execution of a workflow (Petri net). Initially, workflows are modeled/specified by Petri nets using Renew and have .rnw extension. It should be noted that, for now, there is a focus on simple simulation/execution of workflow nets in the Cloud. Workflow management aspects are currently considered in the background. For example, human interaction with the workflow, e.g. a user executing a task, is currently only simulated by the system. Later on, it is possible to incorporate a WfMS in the Cloud which would support these kinds of aspects. Workflow management within Renew implemented as a reference net (agent) system, which would be executed in the Cloud, is already possible [Wagner, 2009a]. For now, the vagrant machine is equipped with a Renew version without a graphical user interface, i.e that users are obligated to run the simulation with the command line. The correspondent console command is *startsimulation*. The syntax of the command is:
*startsimulation <net system > <primary net> [-i]*

- *net system*: The compiled net files (.sns files, Shadow Net System).
- *primary net*: The name of the net, of which a net instance shall be opened when the simulation starts. Using the regular GUI, this equals the selecting of a net before starting the simulation.
- *-i*: This must be set before starting the simulation (only for this prototype). Generally, -r is used, which means to run the whole simulation without steps. More information about this command can be found in [Kummer et al., 2013, p.106].

For testing purposes a simple net (primary net) is created and contains a single transition that prints a string on the screen. Since the reference net formalism allows using Java code, this is done simply by the instruction System.print.out("message") (see Figure. 9.3). Once the required files are prepared (.rnw and .sns), they are sent to the Vagrant machine. Required nets are either copied into the synced directory at local host or transferred using the *scp* command. There are three possibilities to start a simulation on the guest machine: (i) by a command line (using ssh) (ii) through a Web Gui (using

[1] https://www.vagrantup.com/

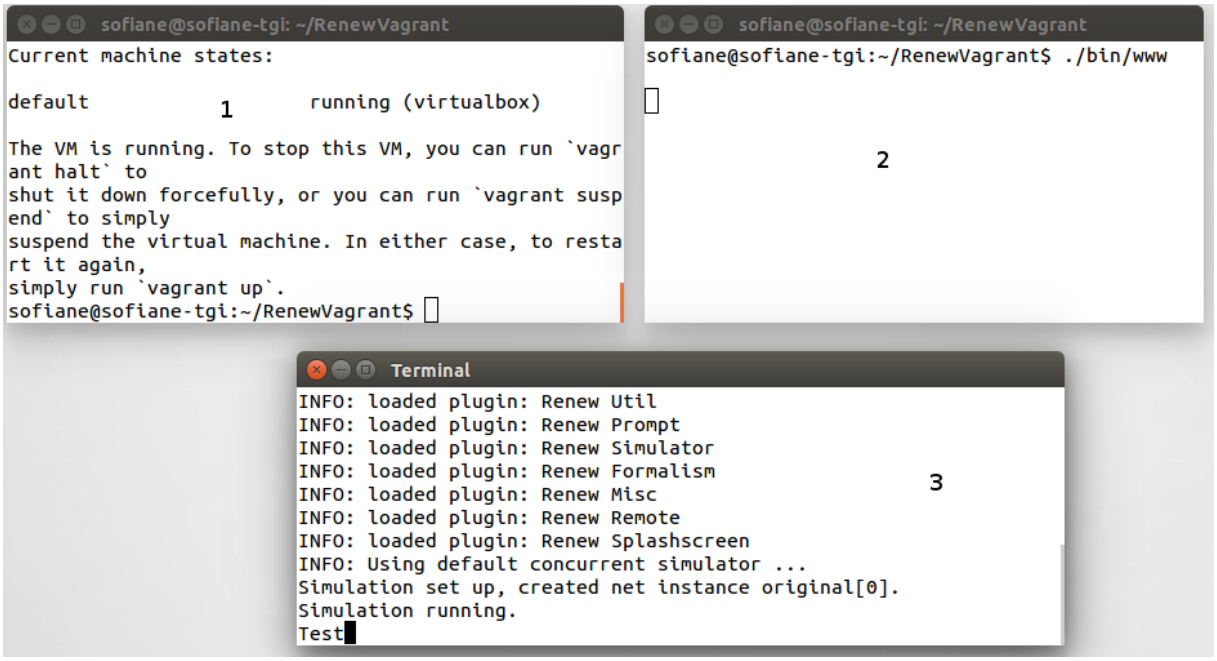



Figure 9.1: Run Simulation in a Vagrant Machine

NodeJS) (iii) from a reference net directly (inscribed to transitions). Figure. 9.1, shows the process of starting a vagrant machine and launching Renew and the simulation. Executing the command in 2, launches a new terminal and starts Renew and simulate the net on the guest machine. (1) The Vagrant machine should be up and running (2) The Web server (NodeJS) is started (3) Renew is launched and a simulation is started with the required nets.

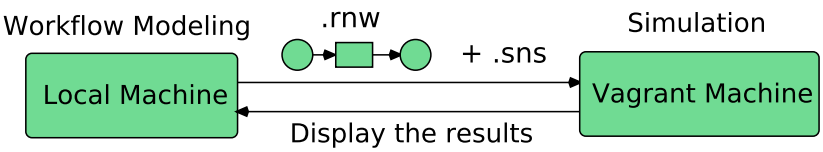



Figure 9.2: Remote Simulation with Vagrant

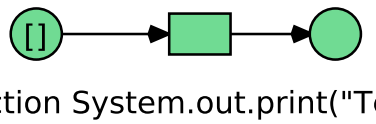



Figure 9.3: The Original Net (.rnw)

### 9.3.3 Second Prototype (with OpenStack)

The second version of the implementation is based on a concrete Cloud environment. Instead of being launched in virtual machine at local host, the instances are launched in the Cloud (see Figure. 9.5). *Vagrant* uses specific providers. The default one is *VirtualBox*[2]. Other built-in providers are VMWare[3], Docker[4] and Hyper-V[5]. When executing *vagrant up* a virtual machine is created on the local host. If only one VM is required then it is enough to work locally. Nevertheless, when the number of VMs grows there is an overload due to a lack of resources. The natural solution is to look for external resources, which are available in a Cloud. Due to financial and technical constraints, free OpenStack based Cloud platforms are used as testbed. *OpenStack* is an open source software for creating private and public Clouds. It is installed on a CentOS Linux operating system. As a result of the plug-in architecture that *Vagrant* is based on, it is possible to connect to different Cloud providers and launch the instances. This is performed by a plug-in called *vagrant-openstack-provider*[6]. This plug-in permits to control and provision machines within an OpenStack Cloud. Other features are for instance: Create and boot OpenStack instances, SSH into the instances and suspend and resume instances. The principles for running Renew simulation in the Cloud are almost the same as presented in the previous section. The difference is at the configuration level, which is realized by the *Vagrantfile*. A minimal configuration consists of the following:

```
require 'vagrant-openstack-provider'
Vagrant.configure('2') do |config|
  config.vm.box        = 'openstack'
  config.ssh.username = 'stack'
  config.vm.provider :openstack do |os|
  os.openstack\_auth\_url = 'http://keystone-server.net/
  v2.0/tokens'
  os.username            = 'openstackUser'
  os.password            = 'openstackPassword'
  os.tenant\_name         = 'myTenant'
  os.flavor              = 'm1.small'
  os.image               = 'ubuntu'
  os.floating\_ip\_pool   = 'publicNetwork'
  end
end
```

The configuration presented above concerns only the credentials and the image used to boot the instances. The important next step is to configure these instances to be able to support Renew simulations. The configuration is performed exactly in the same way

[2] www.virtualbox.org
[3] www.vmware.com
[4] www.docker.com
[5] www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx
[6] https://github.com/ggiamarchi/vagrant-openstack-provider

as working with virtual machines (VirtualBox). Configuring of the instances plays an important role and directly affects the performance of the system. Although, the current work is based on a private OpenStack Cloud, the contribution can be easily integrated within other commercial Cloud providers like Amazon[7], Windows Azure[8] or HP[9]. With providers, Cloud consumers can configure their instances based on a pay-as-you-go model. Resources provided by commercial Cloud providers are not free, which can negatively affect the choice of the Cloud consumers. With respect to the application requirements, there are different types of instances which depend on the Cloud provider. Instance types describe the computing, memory and storage capacity of the instances that Cloud consumers use for hosting (computing) their applications. Therefore, the requirements for the applications should be clearly specified as QoS parameters. This issue has been already addressed in [Bendoukha and Cabac, 2013]. QoS parameters can be specified as inputs to the transitions. For example, with OpenStack these are called by names such as "*m1.large*" or "*m1.tiny*". Figure. 9.4 shows the characteristics of T2 instances.

| Model | vCPU | CPU Credits / hour | Mem (GiB) | |
|---|---|---|---|---|
| t2.micro | 1 | 6 | 1 | E |
| t2.small | 1 | 12 | 2 | E |
| t2.medium | 2 | 24 | 4 | E |

Figure 9.4: Amazon T2 Instance Characteristics

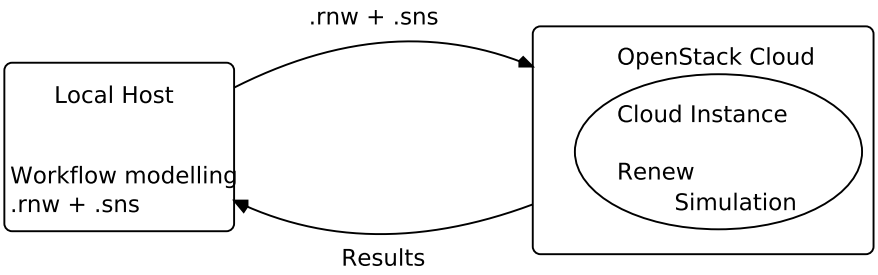



Figure 9.5: Renew Simulation in the OpenStack Cloud

[7] http://aws.amazon.com/
[8] http://azure.microsoft.com
[9] http://www.hpcloud.com/

## 9.4 Interfaces

The previous section describes how to enable Renew simulations in a Cloud environment. The basic technical realization bundles up and simply executes a workflow net system and its shadow net representation. In order to practically utilize this execution one needs to define an interface for it. Different possible interfaces are examined and can be grouped into three categories: Simple, Simulation, Advanced. These categories will be discussed in Sections 9.4.1 through 9.4.3.

Prototypes for the simple interfaces already exist. More features for these interfaces as well as the simulation and advanced interfaces are currently under development. They will be discussed on a conceptual level here.

Note that the method of calling a simulation as a Cloud functionality has been already discussed in the previous section. Generally it can be called either via the console, a Web interface or directly within a (local) running net system. If an interface restricts these possibilities it will be shortly addressed.

### 9.4.1 Simple Interfaces

Simple interfaces offer basic, yet versatile functionality that can later be utilized in more complex settings. The input for simple interfaces remains the workflow system and its shadow net representation. The output options vary, but essentially any results obtained are returned as simple data or objects. Simple interfaces do not support any kind of intelligence or autonomy. They are simply called when needed and report back the predefined results.

*Console Interface:* This interface uses either the internal Renew console or the general system console as the output medium. Consequently it already directly works with reference nets. By simply inscribing
a `System.out.println(textVariable)` to any transition of the net system being executed in the Cloud, the String representation of the object *textVariable* is printed on the console. Figure. 9.1 already shows a working prototype using such a console interface.

For very simple use cases (e.g. testing a certain outcome of the net system) this is already sufficient, but in most cases any obtained result should automatically be made available to the caller in a more usable way. This can be realized by reading any output in the console and combining these outputs into a result object that is passed back when the execution has been completed. Accordingly, more complex use cases and computations can also be supported even with this very simple interface. One problem with this approach is that it is not standardized or regulated by the modeling approach. This is a general problem that will be discussed in Section 9.5.

*Synchronous Channel Interface:* Realizing the interface through synchronous channels is another way of providing a simple interface. Synchronous channels, in general, are a mechanism to allow data and object transfer between net instances. They were first introduced in [Christensen and Hansen, 1994] and are fundamental to the reference net formalism. Within the Cloud context, synchronous channels allow for data objects created and modified during the execution of a workflow to be transferred back to its initiator or even directly into other running (local or remote) net systems. Consequently,

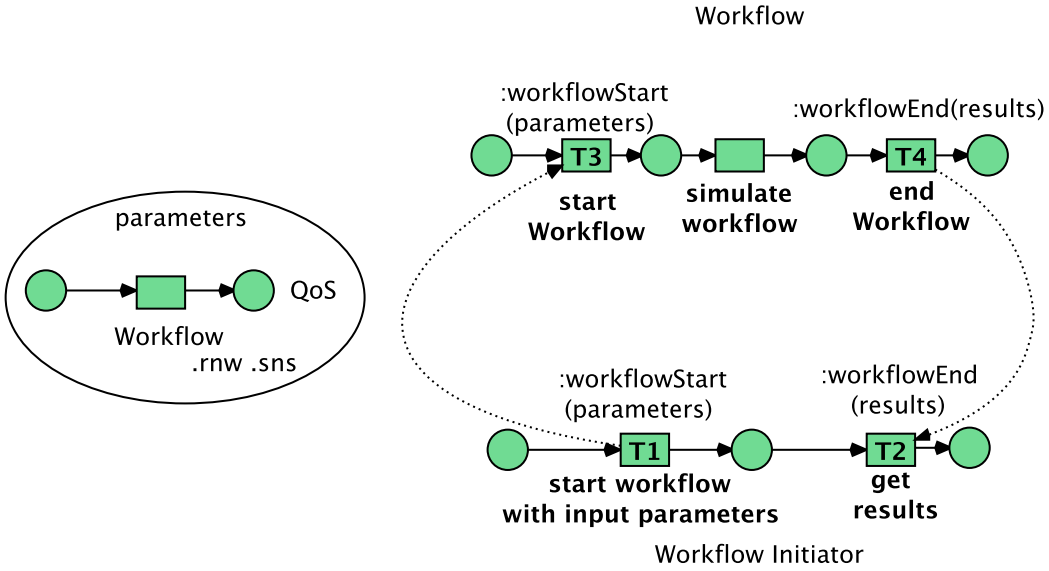



Figure 9.6: Synchronous Channels Interface

the full potential is realized when the Cloud call is incorporated into a net system. There are a number of ways in which synchronous channels can be incorporated into an interface for Cloud-based workflows. The simplest way is to explicitly inscribe an output channel to a transition in the net. When this transition fires, the synchronous channel is called and the specified data object is transferred to the Cloud call initiator. By extending this to multiple transitions one can realize a kind of continuous feedback for the initiator. Whenever a transition inscribed with the synchronous channel would fire a result would be send to the initiator.

Figure. 9.6 illustrates the approach mentioned above. There are two main nets: *Workflow Initiator* and *Workflow*. The Workflow Initiator manages the workflow locally and is responsible for the communication with the Cloud provider. On the other side, the workflow is executed in the Cloud. After modeling the workflows, the model is saved in Renew (.rnw) and Shadow net (.sns) files. These files are required for the workflow to be executed. The communication between both nets is possible through synchronous channels. For instance, T1 and T4 are for sending data; T2 and T3 are for receiving data. Furthermore, all the parameters can be put into a place instead of synchronous channels.

There are two main problems when using synchronous channels. First of all, similar to the console interface, this interface is not structured. Careless modelers may set output channels to incorrect transitions, which may cause results to be invalid. Another issue is related to the continuous update mechanism. If (possibly partial) results are transferred back to the initiator at multiple times, it may be difficult to work with these results. Depending on the net a modeler would have to explicitly build against that specific interface in order to aggregate the results into a valid composition. For

this simple interface it would be cumbersome and inefficient. This is one of the issues addressed by the advanced interfaces described in Section 9.4.3.

Up until now, synchronous channels have only been discussed for output scenarios. Incorporating synchronous channels for the input of the Cloud-based workflows is also possible. In the simplest option this would only be used to incorporate initial input data. This would not change the basic functionality all too much, as initial data can easily be supplied via the console or simply as the initial marking of the workflows. Changing the initial marking would make it easier. If called from a running net system the Cloud workflow could be initiated with runtime information. A synchronous input channel would simply pass the data object directly into the workflow in the Cloud. Without synchronous channels a new net system with the specified initial marking would have to be created or the console call would have to be tailored to the runtime information.

It is also possible to transfer data into the running Cloud workflow. This would require the initiator to maintain a connection with the Cloud system. This is mostly feasible when the Cloud call is initiated by a running net system which would continue with its own execution and provide additional data to the Cloud net system at some later point. Certain transitions in the Cloud net system could then be inscribed with an input channel over which this additional data could be received. Ensuring the correct connection and synchronization between local and Cloud net systems is the main challenge in this context.

Using synchronous channels in the proposed ways has some disadvantages. Without any restrictions to modeling the placement of input and output in the net would affect any verification of workflow correctness or other Petri net properties. This is discussed further in Section 9.5.

### 9.4.2 Simulation Interfaces

The simulation interfaces are not so much interfaces, as they are a utilization of Renew in a Cloud environment. Instead of executing a net system remotely for some direct usage these interfaces execute the net system a large number of times. The information about these simulation runs is then reported back to the initiator. This constitutes the output of these interfaces. The input consists, in addition of the net system and shadow net representation, of simulation parameters (e.g. number of simulation runs). The advantage of running these simulations in a Cloud environment is that it frees up the modelers local machine.

**Result Simulation** One possibility is to run a set of simulations and have the system report back the results of each run. With the same initial marking different simulations may still produce different results. This could be due to race conditions, non-deterministic behavior, etc. With these results the modeler could validate assumptions about the net system or determine possible error sources.

This kind of simulation could be extended by enabling variable initial markings. Simulating a net system with differing parameters might influence the results and help modelers even more.

**Timed Simulation** Another possibility is to run a set of simulations and compare the time it takes to complete them. This kind of simulation is more useful for testing the performance or new features in the runtime environment (Renew). Running the simulation with new features enabled and comparing the results obtained without them can yield information about new algorithms.

Focusing more on the performance of the net system, it might be of interest to the modeler to determine the impact of different initial markings. Varying over the initial marking of the net system could then help modelers determine performance bottlenecks. When using (reference) Petri nets for processes in practical software engineering within the Paose (Petri net-based, Agent- and Organization-oriented Software Engineering [Cabac, 2007]) development approach for example, such simulations and their results become especially useful and interesting.

### 9.4.3 Advanced Interfaces

The advanced interfaces go beyond simple call interfaces similar to the ones discussed in Section 9.4.1. They utilize these simple interfaces but add another layer of abstraction to them. This leads to additional characteristics like certain degrees of intelligence and autonomy. They can also feature mechanisms to manage and store known net systems so that they may even serve as a kind of directory service. They can also aggregate results, enforce quality of service concerns or choose the best from a set of results. Consequently, no general statements about input and output can be made.

**Agent Interface** Using agents for an advanced interface to the Cloud execution of Petri net systems has a number of intrinsic advantages. Agents possess autonomy and a certain degree of intelligence. Reactive and proactive agent behavior can also be utilized.

In an advanced interface an agent would serve as a kind of gateway between the local net systems and the Cloud net systems. For the Mulan and Capa agents this would expand upon the ideas introduced by the WebGateway agent [Betz et al., 2014] towards Cloud calls. The WebGateway agent serves as a kind of bridge between the net execution of a Renew environment and the Web environment. Agents in Renew can then offer their functionality as Web services and also access remote Web services.

For the Cloud context, agents would serve in a similar fashion. The idea is illustrated in Figure. 9.7. Some agents would be responsible for the net systems. They would take on the role of the initiator. They have the possibility of autonomously or be controlled by a human user via some kind of user interface.

These agents would control and/or create the workflows which should be executed in the Cloud. They would send requests and data to the gateway agent[10]. The gateway agent would then use a simple interface (see above) in its internal functionality to initiate the workflow in the Cloud on behalf of the other agents. Any result obtained in

[10] Alternatively the workflows could be stored in a database known to all agents. In that case the initiator agents would simply send requests and identifiers of the workflows to the gateway agent.

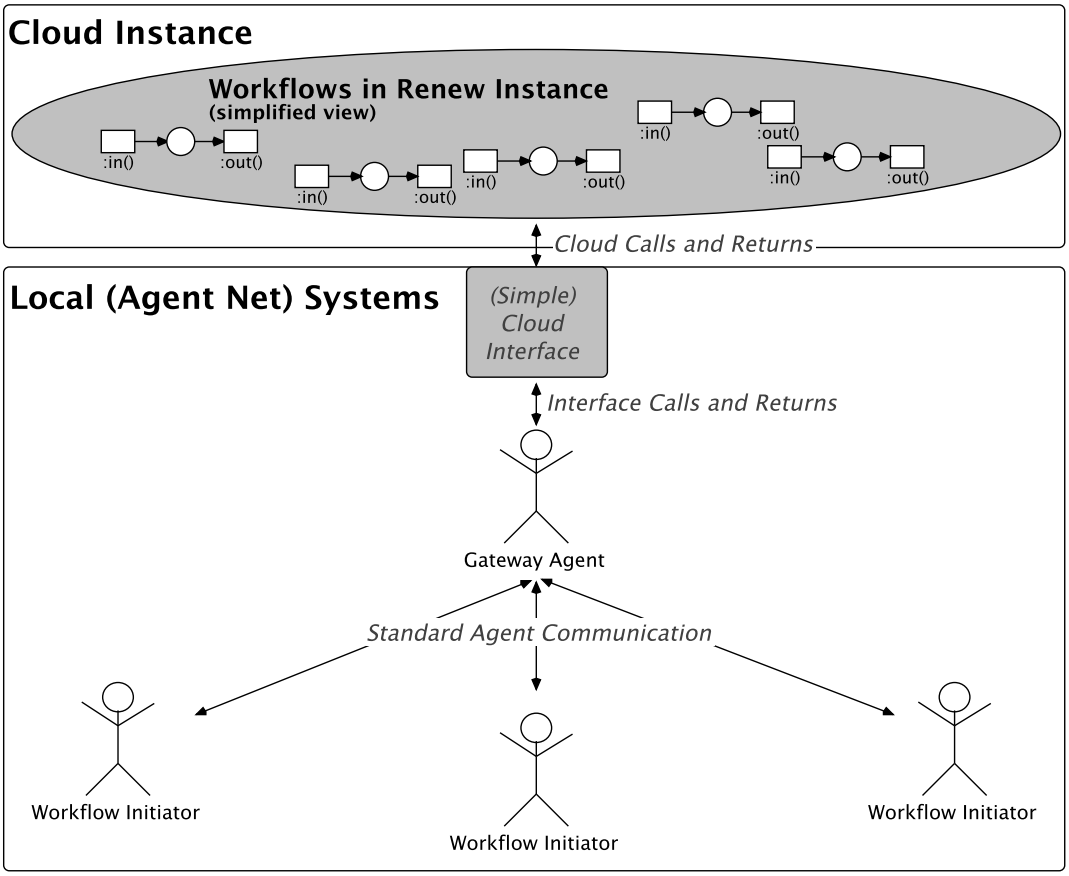



Figure 9.7: Agent Interface Illustration

the Cloud would be sent back to the gateway agent which would then forward it to the other agents.

At this point the characteristics and advantages of software agents can be utilized. In the following, some ideas are covered of how, starting from the relatively simple approach described above, this can be done.

The gateway agent can aggregate the results of the Cloud calls into more informative composite results. Partial results could be incorporated into the workflows with standardized instructions for the gateway agents to combine them after the execution has been completed. The gateway agent can also instantiate the workflow multiple times and choose the best (or fastest) result. Of course, the gateway agent has to be equipped with mechanisms to aggregate or assess results in these fashions. This is, however, simply a question for the technical implementation and not a conceptual one. Aggregation of results is especially interesting for simulation purposes. The gateway agent could automatically create composite results for modelers to inspect. It could also automatically vary over the initial parameters based on the initial results (e.g. to validate results or test certain outliers data).

The gateway agent can also react to errors or other problems occurring during the execution in the Cloud. If the Cloud execution returns an error, the gateway agent can

retry the instantiation. If the error was caused by the call it can also adapt the call (e.g. if input parameters had incorrect types like a string representation of an integer value). This would happen transparently to the initiator of the call which would only have to be invoked if the gateway agent was unable to find a solution to the problem.

Using proactive behavior the agent can also support the execution of Cloud workflows. For example, it could restart workflows if the returned result strongly deviated from expected results. It could also prepare or even already initiate recurring net executions.

The gateway agent can also handle quality of service (QoS) concerns. As stated in 9.3.1, QoS are specified as parameters either in transitions or places. The second scenario is the more appropriate since it uses synchronous channels. In this situation, in addition to the workflow model (and its related files .rnw and .sns) modelers also include QoS parameters. In this work, there is a focus on time and budget, however modelers can include other constraints. The gateway agent can consequently play another role, which is Cloud brokering. Brokering means that the agent looks for the suitable Cloud provider to execute the workflow based on its requirements. This can be useful when working with multiple Clouds.

One disadvantage of using a gateway agent for the Cloud is that it centralizes the communication. This decouples the communication aspects from the individual agents, but gives the system a single point of failure. Only one agent in the system, the gateway agent, possesses the functionality and mechanisms to invoke Cloud systems. This makes other agents more simple and possibly more efficient to execute, but if the gateway agent fails communication with the Cloud, then it is lost. This could be remedied by implementing a solution with multiple gateway agents and distributing the functionality. If one gateway agent failed others could take its place.

**Entity Interface** The term entity describes a hybrid construct between an agent and a workflow. Depending on the runtime needs they can act as an agent (e.g. for communication), a workflow (e.g. for task deployment and execution) or something between the two (e.g. as a mobile process). Entities and modeling with entities is currently ongoing research. The Cloud context enhances the capabilities of entities in many regards.

From the interface point of view an entity possesses all the characteristics of agents and has access to the entire functionality described in the previous paragraph for an advanced interface provided through an agent. This interface is extended even more because of the additional possibilities gained through the workflow properties of an entity. Entities are, in one perspective, a (workflow) process. This automatically entails a certain behavior-centric structure and purpose to the modeling.

By structuring the calls and instantiations of the Cloud net systems as a process itself the modeler is directly supported. While anything can be achieved through regular, less-rigidly structured modeling, restricting the modeler into such a process perspective is still beneficial. Considering process order, task subdivisions, processing of partial results and other aspects of a process are direct requirements in this perspective. Consequently they are essential to the modeler here. But that means that these aspects,

which range from helpful to essential, can also not be ignored or omitted. This is what the entities add on a conceptual level to the advanced interface of agents.

### 9.4.4 Advanced Features

Remote or distributed simulations with Renew is already tackled using RMI (Remote Method Invocation) [Kummer et al., 2013, p.22] [Simon, 2014]. The objective of remote simulation within Vagrant machines is not to be an alternative to RMI, but instead is targeted to a specific type of workflows. These workflows are characterized to have long running tasks with less external services. The simulation step is not provided at the moment, which means that the execution of specific transitions is not possible.

Until now, the results are displayed only in a terminal, which is not convenient. This is resolved, by creating a Web server based on NodeJS. This Web server is for the communication between the client (host machine) and the Vagrant machine. This allows for more automation of the simulation process. Figure. 9.8 shows the Web GUI, which allows the users to automatically:

- create a Vagrant machine
- upload the required nets (.rnw and .sns)
- run the simulation
- display the results

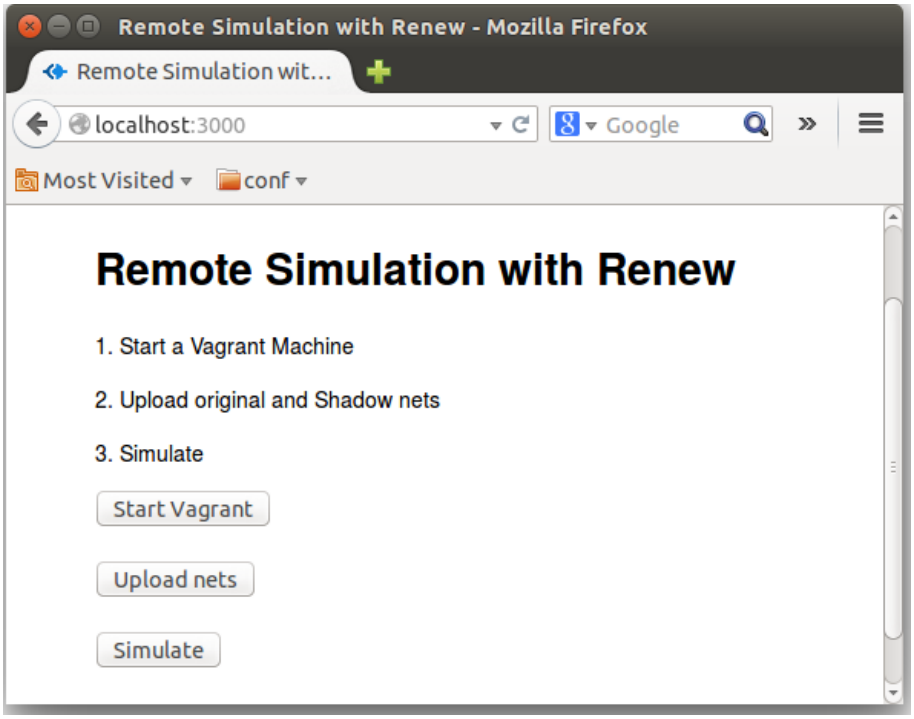



Figure 9.8: Web Gui for Vagrant-based Simulation

The future use of this solution is to be able to provide development environments equipped with the Paose tools: Renew and the Mulan/Capa framework. During the yearly teaching project (AOSE), more then 15 students develop agent-based applications. On this project, the objective is that all the students have an identical development environments, which can be a big challenge. On the top of that, some students use Mac while others use Windows or Linux, which means that errors can occur during the installation of the required tools. Vagrant manages all this for the students so everyone can focus on programming rather then their development environments.

## 9.5 Discussion

One issue that was raised in Section 9.4 concerned the restrictions on modeling and the placement of input and output in a net for the interfaces. If that placement is unrestricted it may be error-prone and puts the responsibility solely on the modeler without any support. An effort could be made to restrict input and output to the initial and exit places of the workflow. This would ensure only full results are returned to the caller and make it easier to verify workflow net properties. However, there are cases in which partial results (e.g. status updates) *during* the execution of a workflow net are desirable. The restriction would preclude this. A compromise would be to allow simple status reports from anywhere in the net (e.g. via the console), but only complete results from the final place or transition of the workflow (e.g. via synchronous channel). Only these complete results would then be made available for further operations in the workflow initiator.

Without any restrictions it would also be impossible to make any statements about the correctness of the executed workflows. For practical purposes allowing input into already running workflows and arbitrary input/output locations might be helpful to some use-cases. However, from a verification and validation point-of-view these mechanisms are problematic. Incorporating concepts like workflow correctness into the Cloud calls and interfaces is currently ongoing work but outside the scope of this thesis.

The question of restrictions raises another interesting point. This work is focused on the execution of workflow nets. Arbitrary workflow net systems can be executed in the Cloud. That includes scientific and inter-organizational workflows.

From a technical standpoint though, it is possible to execute any net system in the Cloud. The only precondition is that a plug-in for the net formalism in question is provided for the Renew instance running in the Cloud. Renew plug-ins for many formalisms already exist (e.g. P/T nets, nets supporting time annotations) and more can be added.

When allowing arbitrary net systems without restrictions to the interface or without any structured modeling these arbitrary net systems might pose challenging to modelers in terms of efficiency and manageability. For this reason it is advisable to use structured modeling paradigms, like agents or entities, for the Cloud net systems as well. In the following paragraphs it is examined how this would affect the advanced interfaces described in the previous section.

By executing the agent interface within the Cloud (as opposed to outside the Cloud as described in Section 9.4.3) the communication can be simplified. In this scenario the net system executed in the Cloud is a Capa agent platform with a running gateway agent. The gateway agent is accessible for other agents via the standardized FIPA compliant asynchronous message communication supported in Capa. This would "move" the interface from the local execution into the Cloud, since it does not matter where the gateway is executed to other agents. They communicate with it in the same way as any other local or remote agent. This would lead to efficiency gains as the gateway agent could access resources in the Cloud environment directly. The technical capabilities of the gateway agent would also be improved. Other properties of the interface would largely remain the same.

The entity interface would benefit in the same way as the agent interface. In addition, it would also affect the modeling abstraction of the entity, as it could be considered a (workflow) process in the Cloud executing other (workflow) processes. This is especially interesting in the inter-organizational workflow setting.The entity in the Cloud could be considered as the overall inter-organizational workflow while the workflows it controls are the subworkflows for each involved organization.

## 9.6 Conclusion

This chapter tackles the first aspect of the relationship between Cloud computing and workflow concepts. It concerns the features that Cloud providers can offer for the execution of complex (scientific) workflows. Concretely, workflows are specified by Petri nets, especially reference nets. Thus mechanisms have been investigated to show how Cloud instances can support the simulation/execution of Petri net models. The current implementation supports the OpenStack Cloud, but can be extended to other Cloud providers. The use of OpenStack is related to the ICWorkflow plug-in (see Chapter 11). The latter provides ICNets (see Chapter 10) with means to invoke Cloud services deployed in OpenStack Cloud. There are several ways to improve the current work. For example, the transfer of data to the Cloud can be performed easily through Web interfaces rather than with command line. The research on the notion of interfaces is still at the early stage and can be a subject of further research. The agent aspect of the interfaces proposed in Section 9.4 can be realized following the work of [Wagner, 2010, Reese, 2010]. The latter proposes a strong approach for the integration between workflows and agents. The following chapter (see Chapter 10) addresses the second research on the integration between workflows and Clouds. It consists of the introduction of new modeling/implementation techniques to support Cloud-based workflow management.

# Chapter 10

# Workflow Concepts for (Inter-) Cloud Computing

In this chapter, two essential contributions of this dissertation are presented. The CTT (see Section 10.2) and the Inter-Cloud Nets (ICNets) (see Section 10.3). These introduced concepts cover both the conceptual as well as the technical aspect of building Cloud-based workflows. Furthermore, these contributions do not only concern single-Cloud systems but also enable the integration of multiple Clouds into one architecture.

## 10.1 Introduction

The previous chapter concerns mainly the use of Cloud resources for the execution of workflows. The objective is to take advantage of the Cloud technology in order to enhance the performance of the system. In the current work, mechanisms are provided to move applications to the Cloud. It is now possible that processes modeled with Petri nets by Renew are deployed and executed in one of the famous Cloud providers (AWS and Azure) or in open-source Cloud platforms such as Openstack. The latter is the chosen platform for the deployment of Renew processes.

Nevertheless, the technical side is not enough to meet a complete integration between Clouds and workflows. For many simple Cloud consumers building Cloud-based applications from scratch is arduous. Moreover, the focus on moving applications to the Cloud should be investigated in priority, and this starts with the modeling step. Thus in this chapter adapted modeling techniques based on reference nets are provided. The idea is to offer a set of Petri nets models to allow both the modeling of Cloud-based workflows and its management. With the Ctt for example, one can specify a workflow task that requires Cloud services with respect to QoS constraints. ICNets are predefined Petri nets structures that help both the design and the execution of workflows in an Inter-Cloud environment. ICNets can also exploit the features of Ctt for modeling workflows. Furthermore, a Cloud-based workflow management architecture (see Section 10.2.5) makes use of the two concepts mentioned above.

## 10.2 The Cloud Task Transition (Ctt)

In this section, the concept of the Ctt is presented. The idea follows the concept of the workflow transition from [Jacob et al., 2002]. It is based on high-level Petri nets as the main modeling language, using reference nets [Kummer, 2002] as the implementation basis. The basic technical matters of the tool support in Renew are described, leaving most of the application aspects aside.

### 10.2.1 Preliminaries

Cloud computing provides the technical means to allow for efficient scalability for inter-organizational or large intra-organizational implementation of distributed applications. At the same time, agents and multi-agent systems provide a conceptual underpinning of Cloud technology. They represent another distributed computing paradigm where several agents interact and are able to behave intelligently.

Despite the differences between these two areas of computing they can take benefits from each other. While Cloud computing offers a reliable and scalable infrastructure for the execution of multi-agent systems especially in terms of modeling and simulating, agents can be used to handle SLA (Service Level Agreement) negotiation, Cloud service discovery and composition, etc. Using Cloud technology as the technical implementation means, Ctt is introduced, which eases the modeling of complex inter-organizational business processes. Its role is to facilitate the specification of user's requirements and supports QoS management. Workflow modelers specify the requirements as parameters to the Ctt in form of tuples (S, Q, I), which corresponds respectively to the Cloud service (S) (can be storage or compute service), the QoS constraints (Q) (consisting of time and costs) and input data (I) (files in case of storage and scripts in case of compute service). Synchronous channels are used to make the connection with the WfMS, which controls the completion of the task. It either initiates the firing or cancels it and all input parameters are put back in the input places [Bendoukha and Cabac, 2013].

The main idea behind the Ctt is to introduce the possibility to even further abstract from details, concentrating on necessary contexts of a business activity. Whole interaction schemes, modeled by business interaction diagrams / agent interaction diagrams, are used as input parameters for a given business task. Based on an agent oriented style of modeling related agents (and roles), necessary artifacts as well as the normal data sets into the business activity are also fed. Incorporating Cloud interactions into the workflow system can be advantageous and can make things easier for users, modelers and administrators. To see how the Ctt is used in practice, a Cloud-based workflow architecture is introduced (see Section 10.2.5).

### 10.2.2 The Workflow Task Transition

The core of the workflow net formalism for Renew is the specialized task transition, seen in Figure. 10.4. In Renew each of such transitions corresponds to one task in a workflow. The task transition makes use of the shadow net layer, to hide technical information about the connection to a workflow management system from the user.

A task transition cannot fire automatically when it becomes activated. The firing is controlled by a workflow management system (WfMS) in which users can request it and then either confirm or cancel the task.

The upper part of that figure shows the transition as it is used in Renew. It is represented as a transition with thick vertical bars. The inscription triple defines the task, which is to be executed, the input parameters for to the task and the (optional) variable, which is the result of the task. This concise representation of a task in the workflow net is easily usable for workflow designers.

The lower part of the figure shows, how the transition is translated in the shadow net layer of Renew. What, in the graphical interface, appears to be a single transition is in fact, during execution, a complex net structure consisting of three transitions and a place. The places connecting to the task transition are in fact connected to the internal transition representing the request of the task. The synchronous channel realizes the interface to the WfMS. Only when the WfMS initiates the firing, can fire the task transition and the task becomes activated. Internally an object representing the task (*_WF_activity* in the figure) is put on the internal place. The WfMS then (through user input) controls the rest of the task. It either is canceled by the user and all input data is put back onto the input places (rollback) or it is confirmed in which case the variables on outgoing arcs from the task transition are filled with the according values.

### 10.2.3 Refinements

Petri nets can model different activities in a distributed system; a transition to model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on. Therefore, some refinements are proposed with respect to the original workflow task transition. The refinements are depicted in Figure. 10.2. New is the integration of QoS management and the communication with multiple Cloud providers. The input places of the Cloud transition model the pre-conditions of an event, the input data for the computational task. The output places of the transition model the post-conditions associated with an event, the results of the computational task. The approach uses workflow nets and draws on the workflow task transition. The idea is to handle user requests formulated as tasks and parameters. These requests will be treated in a transparent manner i.e. that technical information are hidden using the shadow net layer. Clearly, through the Cloud transition users can specify which Cloud product or service to use. A service can be a compute or a storage. The parameters are per example authentication keys or file's path etc. The WfMS will then either accept the request and make the connection to the specified Cloud services according to user inputs or reject it. The main purpose behind the proposed refinements is to allow users to automatically execute workflows on distributed resources (see Figure. 10.1). This will permit the connection to different infrastructure (Cloud, SOA, grid, cluster) in a transparent manner. The Ctt is suited to make workflow more flexible and dynamic, from build time to run time. Large scale workflows utilize distributed resources in order to access, manage and process large amounts of data. These resources are often limited in supply and are shared among many competing users. Workflow

architectures need to adapt into the Cloud in order to leverage the benefits of Cloud services.

Figure. 10.1 shows that *Cloud Workflow* may require services for its execution from Amazon Web services. That is in the case if the whole workflow is executed entirely in one Cloud. Nevertheless, in such a distributed dynamic environment, (requirements) resources or services could be not delivered due to a failure, unavailability or saturation. Hence, workflows require services from other Clouds. Therefore, coordination is beyond the scope of the workflow, but involves the coordination of Cloud services.

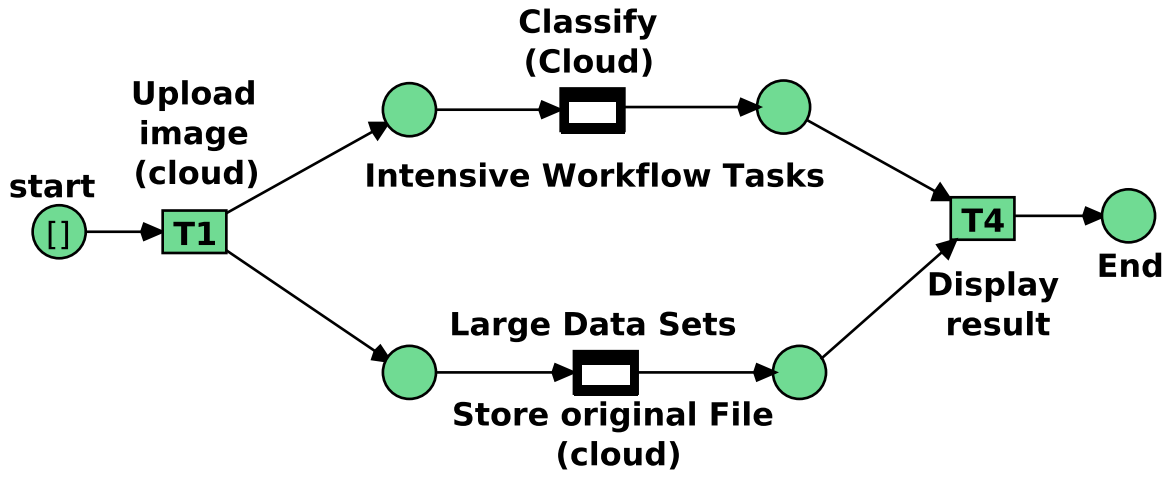



Figure 10.1: A Simple Image Processing Workflow Using Ctt

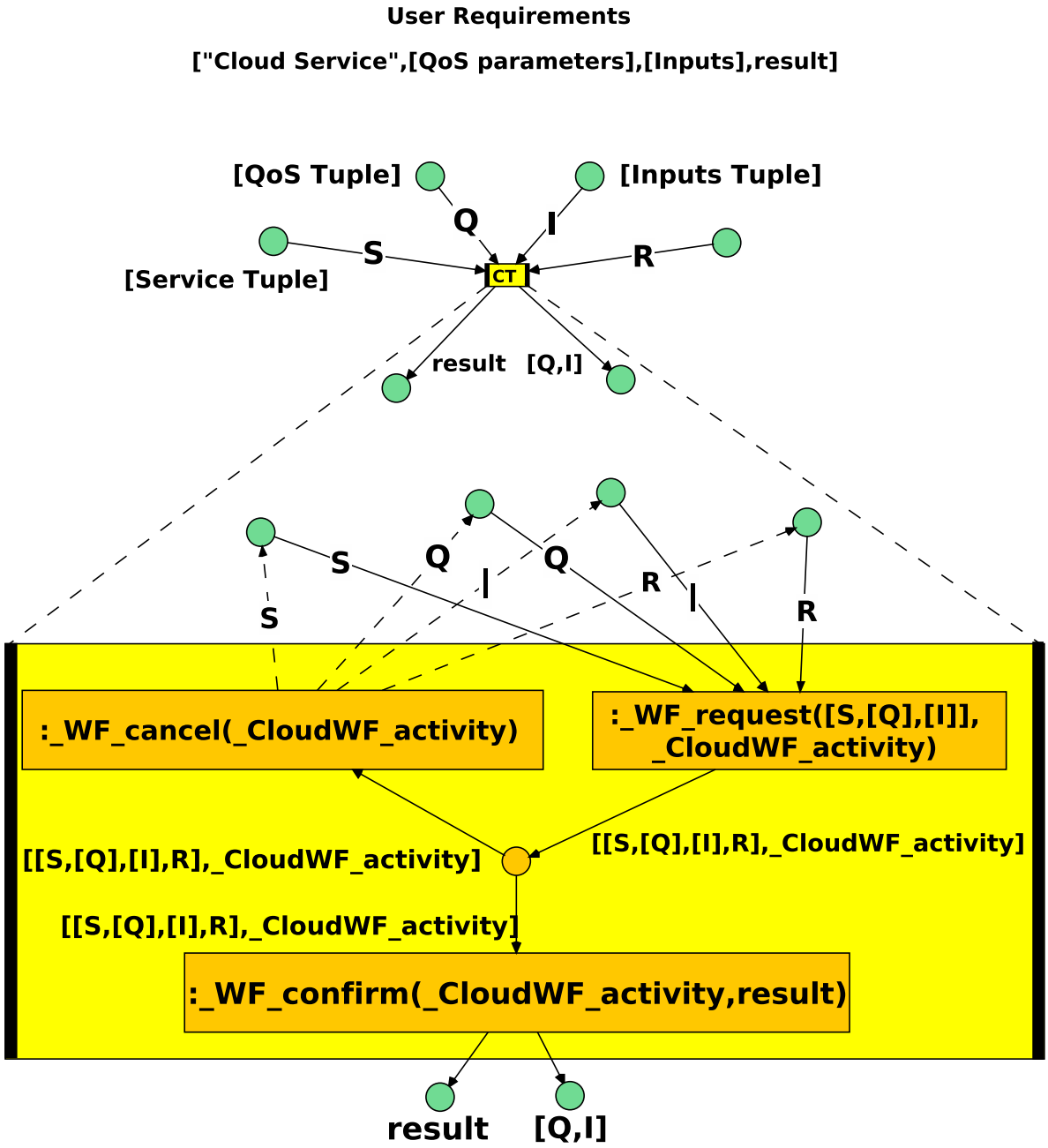



Figure 10.2: The Cloud Task Transition Semantic (modified from [Wagner, 2009b])

Connections to Cloud services are made possible through Cloud application programming interfaces (APIs). A Cloud API determines the vocabulary / taxonomy that a programmer needs to employ while using a particular set of Cloud services. Accordingly, APIs are linked to conditions of use. Therefore, a classification of APIs has been proposed by [Cohen, 2009] which stated that there are three kinds of APIs:

1. Blind APIs: Amazon Web Services

2. Closed APIs: Google App Engine

3. OpenCloud APIs: Rackspace, Gogrid, sun

Cloud APIs allow users (software) to request data and computations from one or more services through an interface. For example the Software-as-a-Service, it considers the provision of software not as a product that the user installs internally on its servers, but as an application remotely accessible as a service. Users do not pay for owning the software itself but rather for using it. They use it either directly through an available interface or via a provided API (often made through Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) Web services ).

By now, most developers are using a REST approach because it's easy to use and background knowledge about WSDL, CORBA, or RMI isn't required. They can easily be called from Java, PHP, Ruby, Python, C#, or even a shell script. Petri nets are an adequate modeling technique for Web services behavior and can be used as a composition model and language for RESTful Web services [Alarcón et al., 2010] [Decker et al., 2008].

### 10.2.4 Integration Issues

The technical integration into the net formalism for Renew (see [Jacob et al., 2002]) is possible. Since workflow nets are more or less regular reference nets making use of the task transition. Thus, Ctt can be added into this as well. The Ctt is also available at the main menu of Renew at the bottom in form of a Cloud. Handling complex input and output of Ctt is no problem, since the underlying reference net formalism can manage any Java object.

The integration of the Ctt into the current Renew WfMS can be realized. Currently, the internal transitions of the task transition communicate with the workflow engine directly via synchronous channels. This mechanism can be adapted to also handle the Ctt. However, this will cause considerable efforts to implement a generic solution for general Cloud settings. Also the integration into an agent based environment causes a certain technical overhead. The technological requirements and their solutions will require considerable work. It is expected that performance will be a major issue here for practical application in daily execution environments.

In addition, the technical aspect must be taken into account as well, which means the integration of the Ctt applications into a user interface. Again it should be possible with the current WfMS to find some good implementations. However, it will require some adjustments in the user agents and other related parts in order to support additional GUI elements. With these additional interface handling a smooth integration into the agent-based WfMS shall be possible.

As the short history of Cloud computing shows, Cloud services may be unavailable for a short, or even for an extended period of time. Such an interruption of service is likely to impact negatively the organization and possibly diminish, or cancel completely, the benefits of utility computing for that organization. QoS (Quality-of-Service) requirements such as time limit (deadline) and expenditure limit (budget) for workflow execution also need to be managed by the WfMS. The WfMS should be able to identify

and handle failures and support reliable execution in the presence of concurrency and failures. These are the short-term technical working steps in future. A solution is expected by the intensive use of intelligent agent application. Agents should be supporting environments to take over the burden of the provision of proper services.

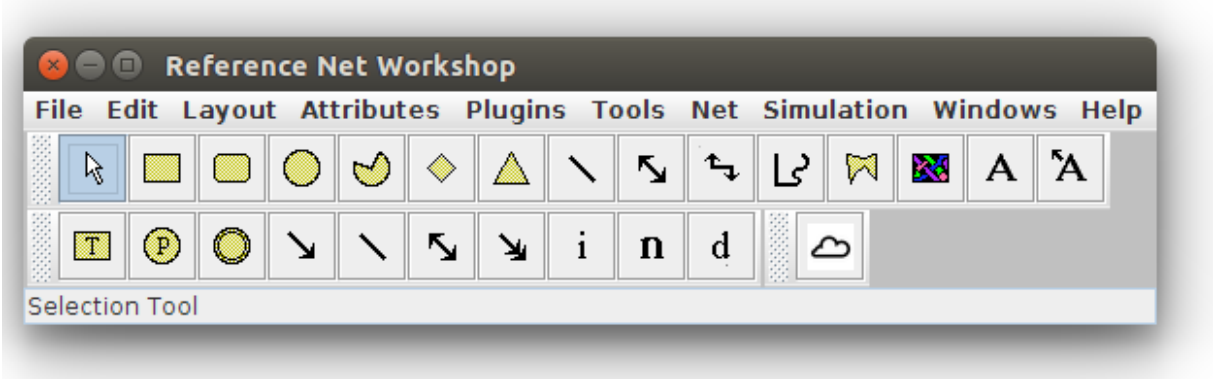



Figure 10.3: The Cloud Task Transition in Renew

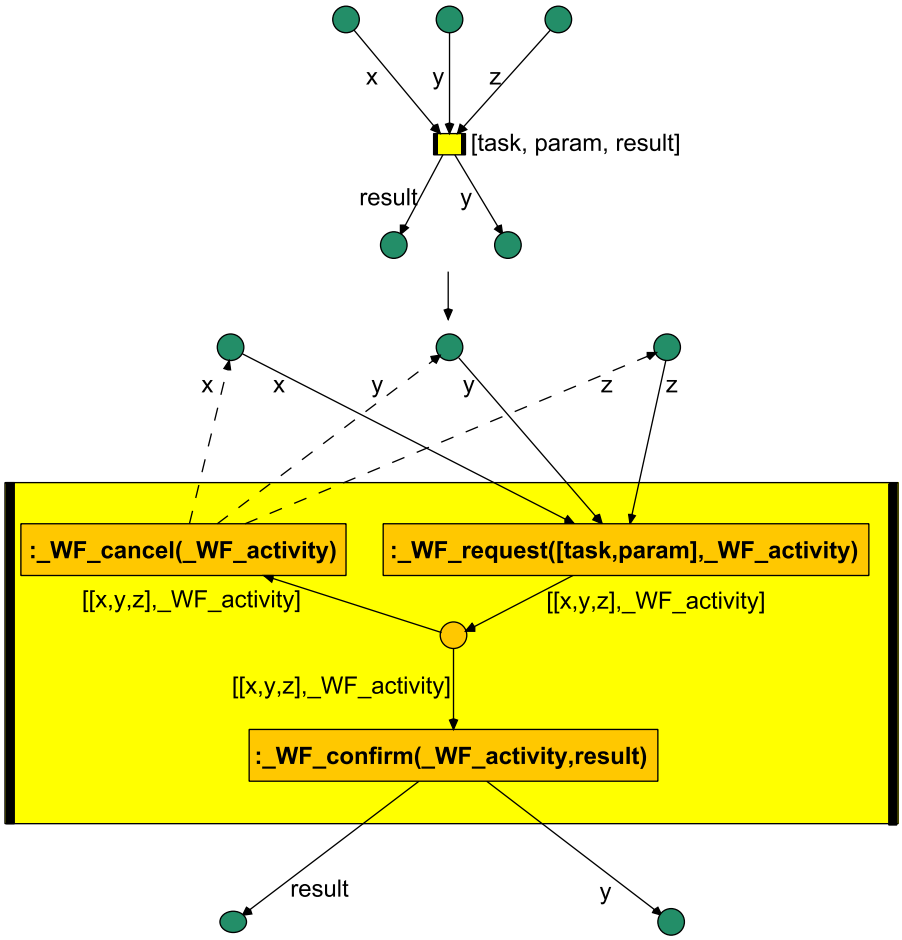



Figure 10.4: The Workflow Task Transition (from [Wagner, 2009b], originally modified from [Jacob et al., 2002])

### 10.2.5 Generic Cloud-based Workflow Architecture

Many Cloud applications require the completion of multiple interdependent tasks; the description of a complex activity involving such an ensemble of tasks is known

as a workflow. Existing workflow architectures need to be adapted for the Cloud and workflow management systems should be integrated with Cloud infrastructure and resources [Pandey et al., 2011a].

According to [Wu et al., 2013], the most important aspects that differentiate a Cloud workflow system from the conventional one is the market-oriented business model. Furthermore, the authors state that the role of a Cloud workflow system, is to facilitate the automation of user submitted workflow applications where the tasks have precedence relationships defined by graph based modeling tools such as DAG (directed acyclic graph) and Petri Nets or language-based modelling tools such as XPDL (XML Process Definition Language). Complex workflow tasks need in some cases to be mapped to distributed resources and involves cooperation among several partners. Workflow management is critical to a successful long-term Cloud computing strategy. The notion of inter-organizational workflow still needs conceptual and technical support especially in complex and dynamic environments like Clouds. New ways to tackle this problem have to be found. Therefore, existing workflow architectures need to be adapted for the Cloud and workflow management systems (WfMS) should be integrated with Cloud infrastructure and resources [Pandey et al., 2011b].

As a solution, the Inter-Cloud Workflow Petri Nets (ICWPN) is introduced, which is an approach for enabling workflows in an (Inter)-Cloud environment. A specialized CTT is introduced to facilitate the connection to the Cloud and to support Quality of Service (QoS) management [Bendoukha and Wagner, 2012]. The Ctt (see Figure. 10.5) is based on the Workflow Task Transition [Jacob et al., 2002], which is the core of the workflow net formalism in Renew. Workflow modelers specify their requirements in form of tuples (S, Q, I) as parameters to the Ctt. It corresponds respectively to the :

- Cloud service (S) that they want to use (it can be a storage or a compute service)
- QoS constraints (Q) consisting of deadlines or costs and input data (I) consisting either of required files in case of a storage or scripts if they want to execute their codes on the Cloud

Synchronous channels are used to make the connection with the WfMS, which controls the completion of the task. It either initiates the firing or cancels it and all input parameters are put back onto the input places.

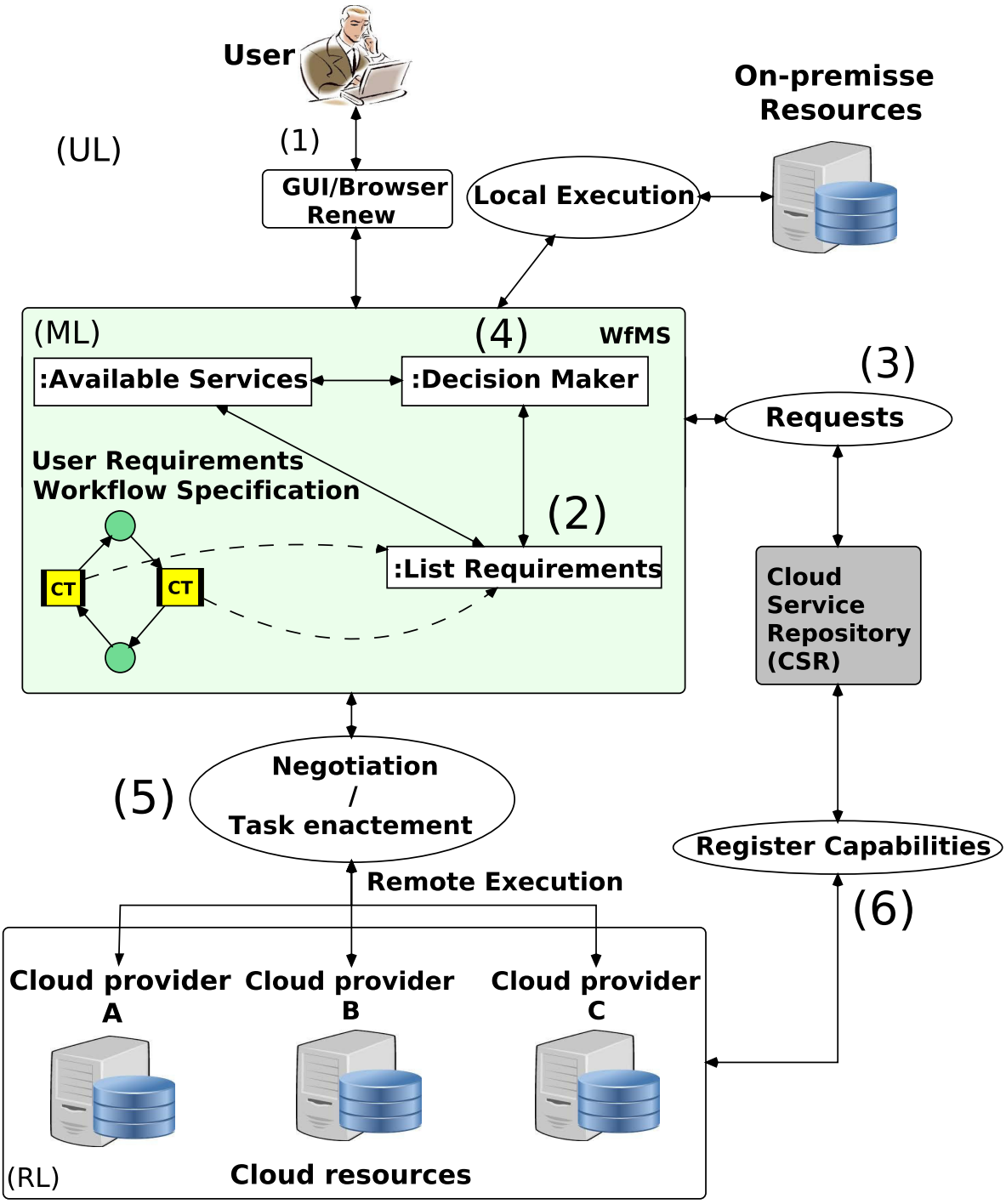



Figure 10.5: General Cloud Workflow Architecture(from [Bendoukha et al., 2013])

These steps are reproduced in Figure. 12.1, which represents another Cloud-based workflow architecture that emphasizes on the use of agent concepts and techniques especially the Mulan/Capa framework.

### 10.2.6 Scenario

The scenario shows clearly how the Ctt is used in practice, in order to model and execute Cloud-based workflows. Assuming that one is working under workflow configuration[1]. As indicated in the Figure. 10.5, complex workflow tasks are modeled with the Ctt, the parameters that should be transmitted are mentioned above. First of all, before explaining in detail, it is important to notice that the architecture is composed of three layers from top to bottom: *user applications layer* (UL), *middle-ware layer* (ML) and the *resource layer* (RL), which consists mainly of Cloud services.

In the proposed approach the process of executing an application in an Inter-Cloud environment is composed of 6 phases which are: (1) Users use the offered modeling tools consisting mostly of Renew and the introduced Ctt to specify the requirements (Cloud services, QoS constraints, specific input data) for their applications using Petri nets models. (2) A list of requirements is created consisting of required services as well as their related QoS constraints. (3) Make a request to the *Cloud Service Repository* (CSR) which is accessible by the WfMS to achieve workflow tasks (4) Based on the above steps (2-3) a decision is made by the *Decision Maker* who determines whether the workflow tasks will be executed locally or using Cloud resources. (5) After that the workflow tasks are mapped to the adequate resources. (6) When the workflow is deployed, information about Cloud providers and the state of their services are constantly updated.

### 10.2.7 Ctt and Paose

Like the other contributions in this dissertation the objective is always to provide means for the development of agent-based applications following the Paose approach. The use of Ctt in such context will be discussed in Chapter 13.

## 10.3 Inter-Cloud Nets

The objective of this section is to propose a new Petri net model that cover some workflow management functionalities: modeling, deployment and execution. The solution is named ICNets and consists of using Petri nets especially reference nets to model Cloud-based workflows. Furthermore, the aim is to propose a formal specification to ICNets by defining all the elements and the relation between them.

In order to give an insight to ICNets, a scenario is presented, where Cloud consumers/developers are confronted to interoperability/portability and vendor lock-in issues during the development of their applications. The scenario explains how ICNets can be helpful for modeling and executing workflow based on services provided by different Cloud providers. The idea of ICNets appeared when trying to design a Cloud-based workflow management system by using the reference nets. Such complex systems are difficult to model and to deploy in the Cloud. The problem is that Cloud-based

[1] the workflow configuration means that Renew is started with some additional plug-ins such as the workflow, WFNet and Access. Other external resources might be also used such as a Web server in case of using a browser and connexion to Cloud services.

workflows are more data-centric. Data need to be passed (specified by a entity or left abstract) to the tasks, which next move them to the correspondent Cloud.

### 10.3.1 Introduction

The Cloud technology is a recent computing paradigm, which has been defined by NIST [Peter Mell, 2011]: *"as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*. Nevertheless, complex applications require several services that a single Cloud is not able to satisfy. Unavailability of a Cloud service may lead to unsatisfied contracts established between Cloud providers and Cloud consumers. Therefore, Inter-Cloud computing is considered as the next natural evolution of Cloud computing [Toosi et al., 2014]. The main challenges for the emerging Inter-Cloud Computing identified up to now are application and data interoperability and portability, governance and management, metering and monitoring, as well as security.

The research on the Inter-Cloud is still at an early stage of development. A semi-automated process to guide consumers to work with different Clouds based on monitoring tools for the quality of services, is unfortunately not yet technically possible.

Hence, the construction of such complex systems remains a problem as soon as there are several independent partners involved in the design and the execution of these systems. Missing is the support of processes in this environments. For complex systems with independent partner expressive and powerful software systems have to be provided.

Workflow concepts are strong candidates to address the previous issues. Their advantage is the automation of processes and efficient coordination and collaboration among various entities composing the system. However, Workflow Management Systems (WfMS) usually do not address the special aspects of Cloud-based systems.

New concepts and constructs to overcome this problem are necessary. Therefore, this work provides a conceptual and technical solution for the modeling and design of complex systems in Cloud-like environments with a special emphasis on processes. The contribution consists of the introduction of the Inter-Cloud NETS (ICNets). ICNets are predefined Petri nets structures that allow modeling of Cloud-based workflows and the management system. New forms of Petri net *places/transitions* are introduced. The latter have specific functionalities, which consist of invoking Cloud services directly from the net. The idea of proposing a classification of Petri nets transition and places has been already discussed at the end of Chapter 3. The complexity of the (Inter-) Cloud environment is hidden by a gateway layer. Furthermore, Cloud developers and Cloud consumers will not be confronted to issues like model transformation (instantiating the system and the processes) or interoperability. This thesis concentrates on workflow modeling, deployment and Cloud service brokering than monitoring, security and migration of applications among Clouds. In order to illustrate their usability, *data* management operations have been implemented by ICNets. The operations include allocating as well as accessing containers and objects in the Cloud.

### 10.3.2 Towards a Formal Description for ICNets

The main objective of ICNets is to provide a separation between the model's element (places, transitions and arcs) in order to efficiently analyze the whole behavior as well as to detect faults quickly in the model. Figure. 10.3.2 gives an overview of ICNets. The principle of this approach is based on classification of places/transitions depending on their utility in the whole model. For example, it is assumed that a workflow task can be either compute ($T_{c1}$), storage ($T_{s1}$) or both. These tasks require (Cloud) resources (R), which can be available or not. Furthermore, both compute and storage tasks need inputs, which are data (for storage tasks) and script (for compute tasks). Following the reference nets principle, there is a system net, that communicates with different object nets through synchronous channels. For instance, firing $T_{s1}$ will create an instance of a net that will store the data in the Cloud. Data are passed as parameter to the $T_{s1}$. Before calling the storage net, a test should be performed to check whether the Cloud resource (R) is available or not. As shown below, there are also special kinds of places such as $P_{s1}$ and $T_{c1}$. These places can only communicate with specific transitions in the net. Decomposing a workflow (net) in such manner enhances debugging as well as analyzing the process and detects failures at earlier. Next, a formal description for ICNets is proposed.

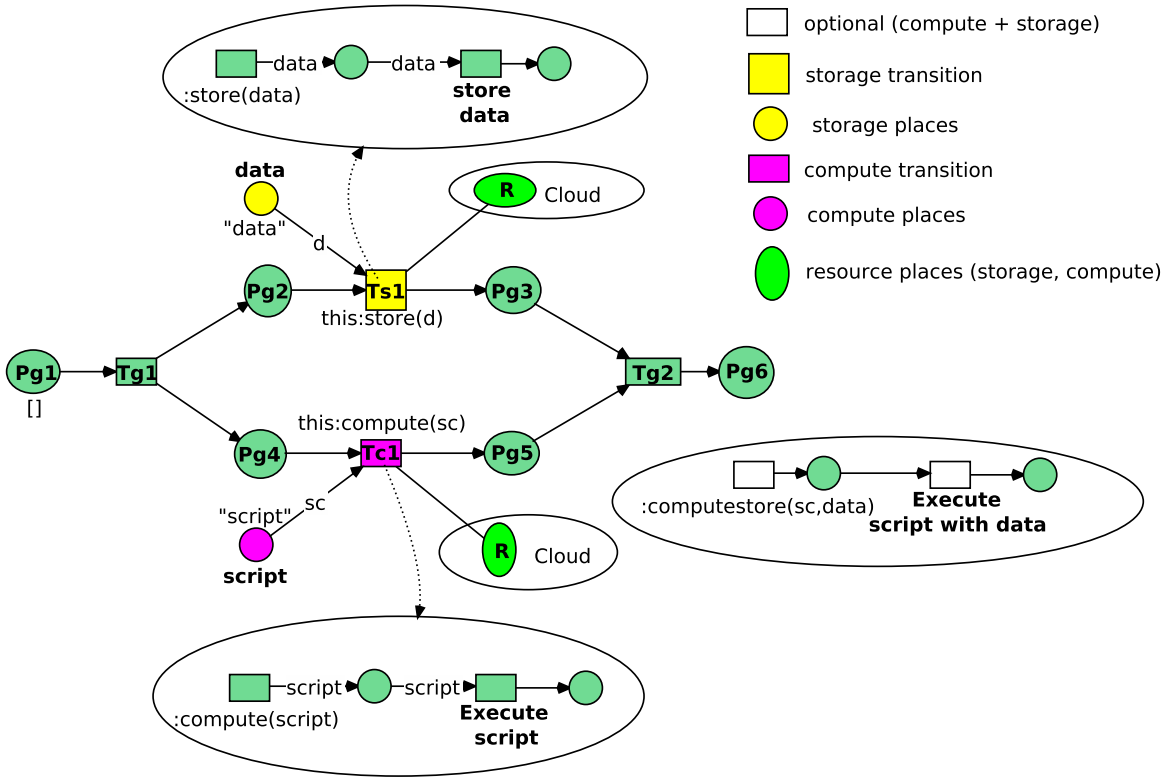



Figure 10.6: Inter-Cloud Nets

#### 10.3.2.1 Definitions

ICNets are based on reference and workflow nets and extend by special transitions and places.

**Definition (ICNET)** An Inter-Cloud NET ($ICNET$) is a 6-tuple, $PN = (P, T, A, C, R, CL)$, where:

- finite set $P$ of places and $T$ of transitions: $\mathrm{P} \cap \mathrm{T} = \emptyset$, $\mathrm{P} \cup \mathrm{T} \neq \emptyset$ and $\mathrm{P} = P_G \cup P_S \cup P_C$ where $P_G$ is a set of ordinary places; $P_S$ a set of storage places and $P_C$ a set of compute places.

  $\mathrm{T} = T_G \cup T_S \cup T_C$ where $P_G$ is a set of ordinary transitions; $P_S$ a set of storage transitions and $P_C$ a set of compute transitions.

  $P_S \neq \emptyset, P_C \neq \emptyset$.

- $\mathrm{A} \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation)
- C is a finite and non empty color set (types of tokens)
- R is a finite set of *storage* and *compute* resources $\mathrm{R} = \{r_1, r_2, ..., r_n\}$
- for each $T_S \in T$, there is a resource $R_i \in R$
- CL is a set of Clouds. Each $CL_i$ is a collection of storage and compute resources.

The technical aspect of ICNets is discussed in the following sections. Here, they are used in concrete use cases, where Cloud services from different Clouds are required for the completion of workflows. Some functionalities have been also implemented following the idea of ICNets. The scenario consists of storage in the Cloud and its related operations such as CRUD and container management.

### 10.3.3 ICNets Principles and Functionality

In this section ICNets and their role in the proposed architecture are presented. As their name indicates, ICNets are based on Petri nets. Petri nets are simple, graphical modeling formalism with a strong mathematical basis. Their efficiency for workflow modeling does not need to be proven [van der Aalst, 1998]. Nevertheless, traditional Petri nets only allow modeling control flow[2]. Furthermore, the resulted workflow model usually needs a translation to an intermediary (executable) model in order to be instantiated and executed by a workflow engine. For this, a special kind of Petri nets called reference nets [Kummer, 2002] are used, which combine the *nets-in-nets* paradigm [Valk, 1998] and *synchronous channels* from [Christensen and Hansen, 1994]. Reference nets play an important role in this work. Firing a transition can also create a new instance of a subnet in such a way that a reference to the new net will be put as a token into a place. This allows for a specific, hierarchical nesting of networks, which

[2]A workflow schema is a combination of three essential dimensions: control flow, data flow, and resource flow.

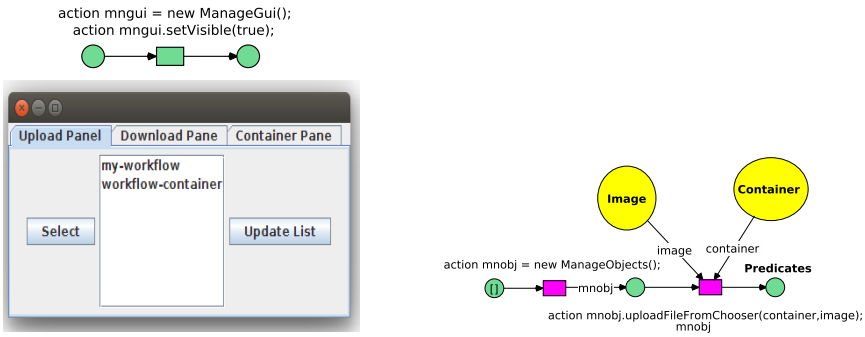
is helpful for building complex systems in these formalisms. This feature is exploited for the implementation of the architecture presented in the following section. The components are in form of Petri nets models communicating with each other through synchronous channels.

With respect to traditional *Petri nets* or even reference nets, new kind of *place/transition* are introduced. The new *places/transition* can be distinguished by their color. Their principle role is to separate between ordinary activities and those requiring Cloud services for their execution. For the latter, a whole mechanism has been implemented to allow the interaction with different Cloud providers. In general, there are two categories of places/transitions: *storage places/transitions* and *compute places/transitions*. Moreover, there are also special arcs to rely these new elements. These arcs allow an appropriate binding between the places and the transitions. For example a *storage place* can only be bonded with a *storage transition*. There is also the notion of *Type* for the information passed to the transitions. *Storage places* contain mostly data to be stored in the Cloud and the *compute places* hold script file to be executed in the Cloud. The first category concerns all workflow tasks making use of Cloud-based storage services and the second one all tasks related to computing activities. These new *places/transitions* are clearly identified in Figure. 10.7. In that example, the *storage places/transitions* are highlighted with yellow color and blue color respectively.

In order to illustrate the features of ICNets, some workflows dealing with storage in the Cloud have been implemented. These workflows consist on performing some data transfer and management on an OpenStack Cloud. With respect to the motivating example from Section 1.1 the data handled within the operations are satellite images. These operations (models) can be *reused* on-demand and be part of another workflow. Moreover, they can communicate with each other, instantiate new nets through *synchronous channels*[3]. A *CloudGateway* has been also implemented to avoid interoperability issues and to invoke Cloud services from different Cloud providers. During the modeling phase, developers just need to specify the Cloud provider that they require. The rest is performed by the *CloudGateway*. Each operation has a number of parameters. In the context of image processing, the important inputs are: the image and the container where the image will store in. Figure. 10.7 shows some of the operations modeled by the ICNets. In the proposed approach, all the behavior is captured by reference nets. The latter should be executed/simulated to perform the Cloud operations. For example, executing the first net (Figure. 10.7a) triggers the display of the GUI, thus Cloud consumers can select the image to upload/retrieve and also to specify in which container the data will be stored. After the specification of these two parameters, their values are passed to the correspondent net. In this figure, the values are required for the upload and the retrieve nets (Figure. 10.7c and Figure. 10.7b respectively).

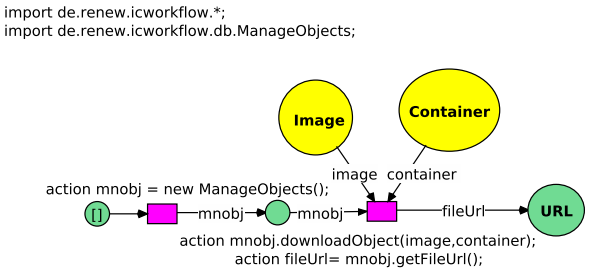
For now, the operations specified by ICNets concerned only the Cloud consumers. They allow them to model the workflow and to specify the parameters. Nevertheless,

[3]Synchronous channel inscriptions consist of two types of inscriptions, *up-links* and *down-links*. Up-links are used in object nets while down-links are used in system nets. They consist of at least two transitions where one of the transitions is seen as the initiator of the communication having a down-link inscription.




(a) GUI Net

(b) Create/Upload Net




(c) Retrieve Net

Figure 10.7: Cloud Operations with ICNets

ICNets have another functionality, which is the design and the implementation of the Cloud-based system, that manages the workflows. This will be illustrated in the following section.

In other words the ICNets serve to:

- specify/implement internal behavior of the components composing the architecture
- model the control and data flow of the workflow
- make it easier for modelers (developers) to orchestrate (invoke) composite Cloud services from different Cloud providers
- avoid vendor lock-in
- make the design of complex Inter-Cloud applications simple
- use well-founded modeling techniques (Petri nets)
- reduce the gap between modeling and implementation
- automate the discovery and selection of Cloud providers

### 10.3.4 Related Work

In terms of specification of Cloud-based applications and systems, the Cloud Modeling Language (CLOUDML) is a tool-supported domain-specific language (DSL) [Ferry et al., 2013]. It relies on model-driven techniques and methods and allows developers to model the provisioning and deployment of a multi-Cloud application at two levels of abstraction: (i) the Cloud Provider-Independent Model (CPIM), which specifies the provisioning and deployment of a multi-Cloud application in a Cloud provider-agnostic way; (ii) the Cloud Provider-Specific Model (CPSM), which refines the CPIM and specifies the provisioning and deployment of a multi-Cloud application in a Cloud provider-specific way [Ferry et al., 2013]. The strength of CloudML is that the description of the deployment is causally connected to the real running system: any change on the description can be enacted on demand on the system, and, conversely, any change occurring in the system is automatically reflected in the model.

The Cloud Application Modeling Language (CAML) [Bergmayr et al., 2014] enables representing Cloud-based deployment topologies directly in the Unified Modeling Language (UML) and refining them with Cloud offerings captured by dedicated UML profiles. In general, the purpose of CAML is (i) to enable the representation of models from the reverse engineering perspective plus the forward engineering perspective and (ii) to provide guidance in terms of optimization patterns for turning Cloud independent models into Cloud specific models from which Cloud optimized application code can be generated as a prerequisite for the deployment on the selected Cloud environment.

The EU-funded mOSAIC[4] project [Martino et al., 2011] proposes a complementary solution based on software agents and semantic data processing. It allows transparent and simple access to heterogeneous Cloud resources and to avoid vendor lock-in. It fulfills this goal by its Cloud ontology that describes services and their interfaces. Moreover, a unified cross platform API that is platform and language independent is provided by mOSAIC. The mOSAIC platform is targeted mainly at Cloud application developers. The mOSAIC approach is based on a *Cloud Agency* gathering client and provider agents in a brokerage process working with service level agreements [Aversa et al., 2010]. It is used as a Multi-Cloud resource management middle-ware, it plays the role of run-time environment in the model-driven engineering project named MODAClouds [Ardagna et al., 2012]. MODAClouds, a MOdel-Driven Approach for the design and execution of applications on multiple Clouds that aims at supporting system developers and operators in exploiting multiple Clouds and in migrating their applications from Cloud to Cloud as needed. To do so, MODAClouds proposes an advanced quality-driven design, development and operation method based on the Model-Driven Development (MDD) paradigm.

## 10.4 Conclusion

This chapter represents an essential contribution of this dissertation. It tackles an important issue in distributed systems, concretely Inter-Cloud environments. The

[4] http://www.mosaic-fp7.eu/

issue concerns the design and implementation of Cloud-based workflows, that is why chapter focuses more on modeling level. New modeling techniques are introduced. They allows for more flexibility and easiness in the specification of workflow tasks requiring Cloud services. First, the concept of a Cloud Task Transition is introduced (see Section 10.2). The role of the Ctt is to help *modelers* to use the power of Petri net formalisms and their high level variants to specify the flow and the dependencies of workflow tasks. Moreover, the Ctt can be also used in agent-oriented software development (see Chapter 13).

In the second part of the chapter the concept of ICNets (see Section 10.3) is presented. ICNets are predefined Petri net models, whose role is twofold. On the one hand, they serve to specify workflows for the Inter-Cloud. They offer more exact abstraction then the Ctt, since they target the use of multiple Clouds rather then following a singe Cloud strategy. On the oder hand, ICNets offer the ability to design Cloud-based WfMSs. The following chapters resumes Part II, gives a summary and evaluates the results obtained until now.

# Chapter 11

# The Inter-Cloud Workflow (ICWorkflow)

This chapter tackles two important issues: the development and the deployment of workflows in Inter-Cloud environments. Therefore, a framework named *Inter-Cloud Workflow* (ICWorkflow) is developed and integrated as a plug-in in Renew.

Several contributions of this dissertation are based on this framework. ICWorkflow is responsible for operations running in the Cloud. This includes: authentication, access to Cloud services (compute and storage), deployment and execution of workflows.

## 11.1 Introduction

Cloud technology is a recent computing paradigm, which has been defined by NIST: *"as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [Peter Mell, 2011].

Nevertheless, complex scientific applications require several services that a single Cloud is not able to satisfy. Unavailability of a Cloud service for example may lead to unsatisfied contracts established between Cloud providers and Cloud consumers. Therefore, Inter-Cloud computing is considered as the next natural evolution of Cloud computing [Toosi et al., 2014]. However, building applications in such environments faces several obstacles, which are portability and interoperability. In this work there is more emphasis on client-centric interoperability following the classification made by [Toosi et al., 2014] (Multi-Cloud and Aggregated Service by Broker scenarios) (see Figure.11.1a and Figure. 11.1b).

In fact, connections to Cloud services are made possible through Cloud application programming interfaces (APIs). A Cloud API determines the vocabulary / taxonomy that a programmer needs employ when using a particular set of Cloud services. Accordingly, APIs are linked to conditions of use. Cloud APIs allow users (software) to request data and computations from one or more services through an interface. They use it either directly through an available interface or via a provided API (often made through

Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) Web services). By now, most developers are using a REST approach because it's easy to use and background knowledge about WSDL, CORBA, or RMI isn't required. They can easily be called from Java, PHP, Ruby, Python, C#, or even a shell script. Therefore, a framework called ICWorkflow has been implemented in order to enable workflow creation, deployment and execution in Inter-Cloud environments. Workflows are specified by Petri nets. One of the reasons of using Petri nets is that they are an adequate modeling technique for Web services behavior and can be used as a composition model and language for RESTful Web services [Alarcón et al., 2010] [Decker et al., 2008].

The ICWorkflow framework provides the required layer for the ICNets (Section 10.3) and and can be combined with the results obtained in Chapter 9. This framework covers several steps when modeling and executing workflows in the Cloud. For example creating customized Cloud instances and launching Renew simulations. Furthermore, the framework is also intended to implement workflow management systems for Cloud-like environments. This chapter aims to provide a solid understanding of the ICWorkflow framework. The following sections focus more on architectural properties of the framework as well as the technologies used to enable the Inter-Cloud communication. Section 11.2 provides a classification of different approaches to build Inter-Cloud environments. In Section 11.3 the underlying technologies, which the current work is based on are presented. The ICWorkflow framework is introduced in Section 11.4. In Section 11.5 an approach is proposed for the management of workflows in Inter-Cloud environments. A short introduction about ICWorkflow in Paose is given.

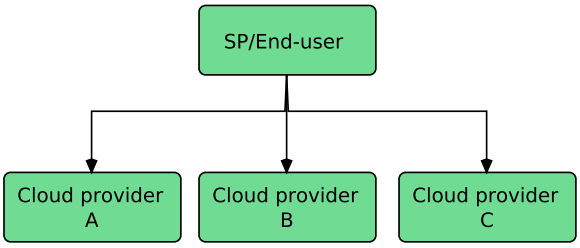
## 11.2 Design of Inter-Cloud Systems: A Classification

There are several approaches on how to build an Inter-Cloud environment. The most important challenge facing Cloud developers and also customers is the interaction with multiple Clouds. Grozev et al. [Grozev and Buyya, 2012a] have realized a review of these approaches and provided a taxonomy of Inter-Cloud architectures and brokering mechanisms. According to [Grozev and Buyya, 2012a], Inter-Cloud is classified as (see Figure. 11.2):

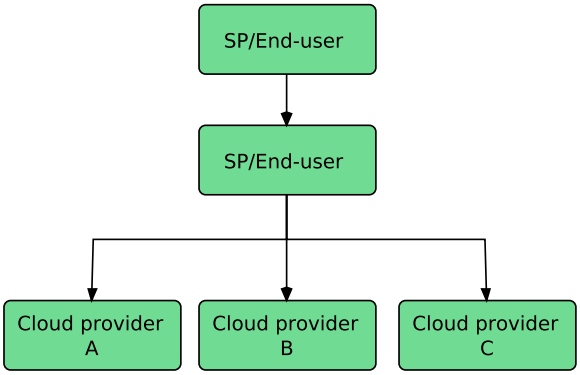
- *Volunteer federation*: as stated above, this is a result of a voluntarily cooperation between a group of Cloud providers. This targets more governmental Clouds or private Cloud portfolios.
- *Independent*: this occurs when an application decide in its own and without the implication of a Cloud provider to make use of multiple Clouds.

The volunteer federation is classified into two architectural categories:

- *Centralized*: this is similar to a broker, which is a central entity, responsible of the allocation of resources. It can be seen as a repository where Cloud providers register their capabilities.




(a) Multi-Cloud Scenario




(b) Aggregated Service Broker Scenario

Figure 11.1: Inter-Cloud Scenarios (Adapted from [Toosi et al., 2014])

- *Peer-to-Peer*: in this architecture, there is not mediator the communication between the Clouds is made directly.

Independent Inter-Cloud are also classified into two categories:

- *Services*: it means that there is a service (hosted externally or in-house by Cloud clients) responsible for application provisioning. Most of these services include brokering features and provisioning process is SLA-based containing predefined attributes.
- *Libraries*: Except the provisioning step, communicating with different Clouds need special Inter-Cloud libraries that facilitate the access and the interaction in a uniform way without the need to use specific Cloud provider APIs.

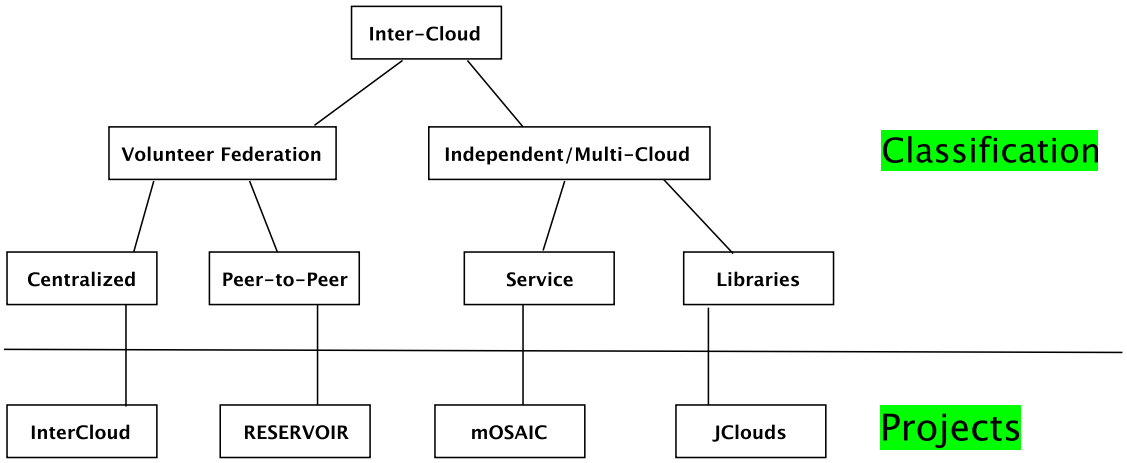



Figure 11.2: Architectural Classification of Inter-Cloud (From [Grozev and Buyya, 2012a])

Figure. 11.3, represents the correspondent architectures taken from the taxonomy (see Figure. 11.2).

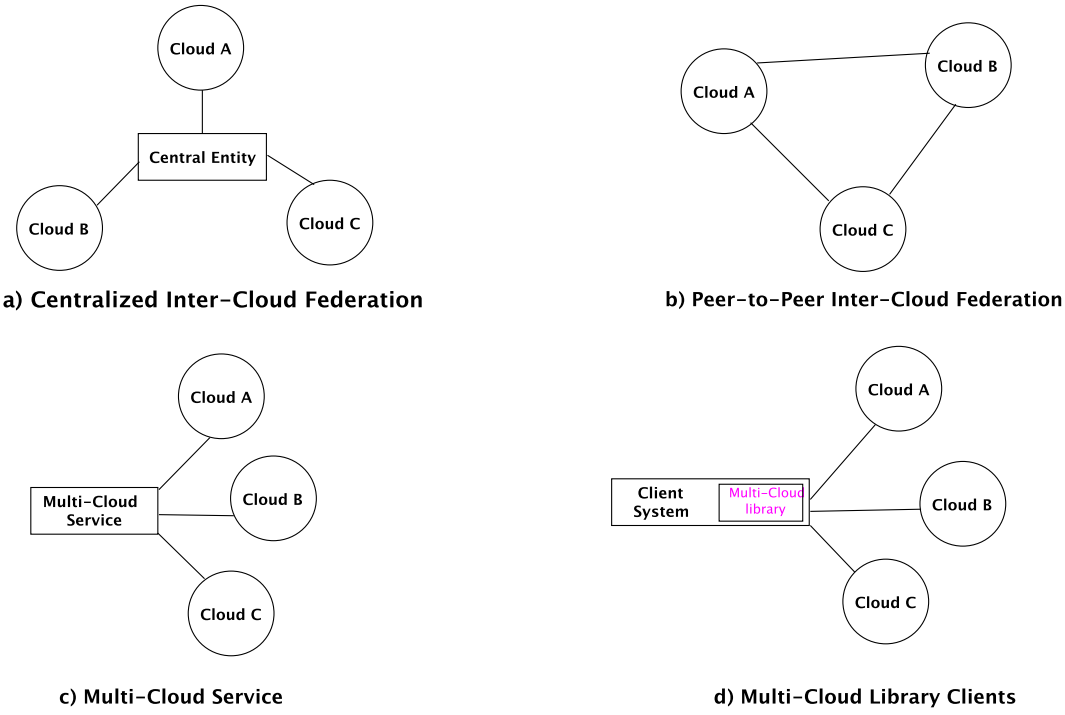



Figure 11.3: Inter-Cloud Development Architectures (Adapted from [Grozev and Buyya, 2012a])

## Inter-Cloud System Architectures

For some time now the term Inter-Cloud is more often addressed. In this section a state-of-the-art is given of the actual architectures for designing Inter-Cloud-based systems. The research concerning this topic stills in his infancy.

Several researchers have already dealt with the issue. In the article [Buyya et al., 2010] by Buyya et al., an architecture of federated Clouds named Inter-Cloud to allow scaling applications through multiple Cloud providers and the possible interactions are described. In this approach, two main components are provided (see Figure. 11.4):

- Cloud Exchange (CEx): The core of this model is the *Cloud Exchange* which makes it possible the distribution of the executions on the Cloud resources. It is a centralized server that aggregates information from various Cloud brokers and looks for the available services. This is the market making component of the architecture. It offers services that allow providers to find each other in order to directly trade Cloud assets, as well as allowing parties to register and run auctions.. In the former case, CloudExchange acts as a directory service for the federation. Therefore, the CloudExchange implements a Web service-based interface that allows data centers to join and leave the federation; to publish resources they want to sell; to register their resource requirements; to query resources and to consult the status of a running auction.

- Cloud Coordinator (CC): It controls the interaction between the Cloud Exchange and the Cloud Broker. This component manages domain-specific issues related to the federation. This component is present on each party that wants to join the federation. CloudCoordinator has front-end (elements that interact with federation) components as well as back-end components (elements that interact with the datacenters). Front-end components interact with the Cloud Exchange and with other coordinators. The former allows data centers to announce their offers and requirements, whereas the latter allows the Coordinator to learn about the current state of the datacenter to decide whether actions from the federation are required or not. Therefore, wherever the Coordinator detects that extra resources are required by the data center, it triggers the process of discovery of potential providers.

The *Cloud Broker* negotiates with the *Cloud Exchange* to find the appropriate cloud provider for the to meet the user requirements (SLA, QoS). What is missing in this model is billing and security issues.

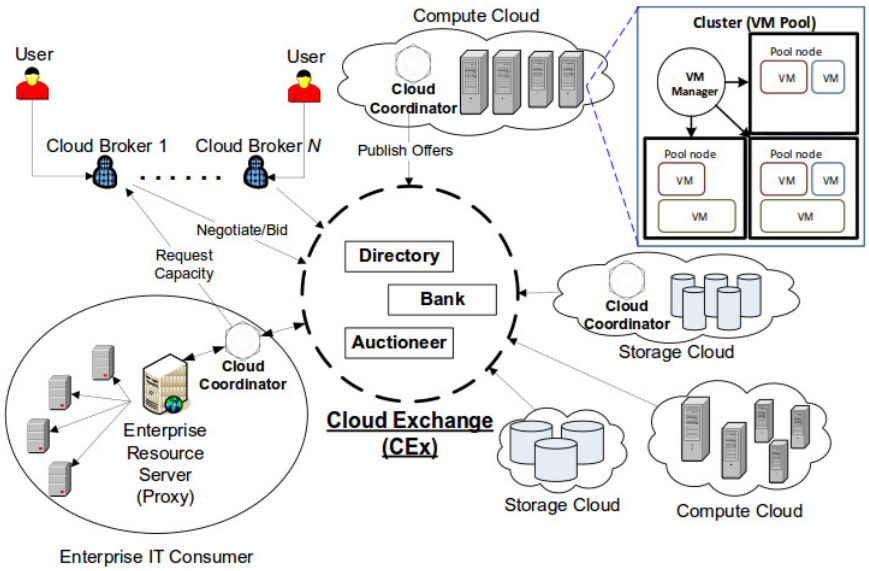



Figure 11.4: Federated Network of Clouds (from [Buyya et al., 2010])

Security plays an important role in such heterogeneous environment where multiple partners collaborate in a joint software development or for information exchange. A Cloud information gateway to control information exchanged between involved partners[1] is proposed by [Tsuda et al., 2012].

## 11.3 Testbed and Related Technologies

Before presenting the ICWorkflow framework and its functionalities, it is important to give a short overview about the technologies taking part to its development as well as the environment that is targeted to work on it. First, a brief introduction is given to present *OpenStack* and *TryStack*, which are the execution platforms of Cloud-based workflows. Second, the *JClouds* toolkit is presented, which is used to implement communication and access mechanisms to connect to the Cloud. The decision to build a private Cloud at the TGI group or to use a free Cloud platform was not random. It was based on different parameters. The decision has been influenced principally by the financial aspect. In fact, the current work is not a part of a supported project, which has an important impact on the selection of a Cloud providers. Working with commercial Cloud providers like Amazon or Windows Azure is very difficult since their services are not free of charge. However, this has no effect on the usability or the credibility of the results obtained in this dissertation. Although the tests have been performed on OpenStack-based Cloud, working with other Cloud providers is not complicated. The ICWorkflow framework itself is designed originally to work with different Cloud providers.

[1] *Tsuda et al.* [Tsuda et al., 2012] define partner cloud as: *using one or more cloud services for cooperation between different organizations"*

### 11.3.1 OpenStack

In order to test the tools and prototypes that are implemented and presented in this dissertation, the OpenStack framework[2] has been selected. Openstack is the facto reference for private Clouds. Figure. 11.5 provides a conceptual architecture of a typical OpenStack environment. Different services cooperate with each other in order to launch a virtual machine or an instance. The OpenStack services can be summarized as shown in Table. 11.1. In the following a brief overview of the most important services [Foundation, 2015].

Table 11.1: OpenStack Services

| **Service Name** | **Description** |
|---|---|
| Horizon | Web-based dashboard |
| Cinder | Block storage |
| Glance | Image management and deployment |
| Nova | Compute virtualization |
| Neutron | postscript |
| Heat | Cloud applications orchestration |
| Keystone | Authentication between Cloud services |

- *Horizon* (OpenStack Dashboard): provides a Web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
- *Cinder* (OpenStack Block Storage): provides persistent block storage to running instances. Cloud users can manage their storage requirements through the dashboard. The system provides interfaces to create, attach, and detach block devices from/to servers.
- *Glance* (OpenStack Image Service): provides support to store and retrieve VM images. Specifically OpenStack Compute makes use of this during instance provisioning.
- *Nova* (OpenStack Compute): manages the life-cycle of compute instances in an OpenStack environment. Written in Python, it creates an abstraction layer for virtualizing commodity server resources such as CPU, RAM, network adapters, and hard drives, with functions to improve utilization and automation.
- *Swift* (OpenStack Object Storage): a distributed storage system primarily for static data, such as VM images, backups, and archives. It stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API.

[2] https://www.openstack.org/

- *Neutron* (OpenStack Networking): provides an API for users to define networks and the attachments into them. Floating IP addresses allow users to assign (and reassign) fixed external IP addresses to the VMs.
- *Heat* (OpenStack Orchestration): orchestrates multiple composite Cloud applications, through both an OpenStack-native REST API. It allows developers to define application deployment patterns that orchestrate composite Cloud applications
- *Ceilometer* (OpenStack Metering): is a mechanism for centralized collection of metering and monitoring data. It delivers a single point of contact for billing systems to obtain all the usage information they need across the suite of OpenStack components.
- *Keystone* (OpenStack Authentication): provides a catalog of endpoints (API access points) of other services.

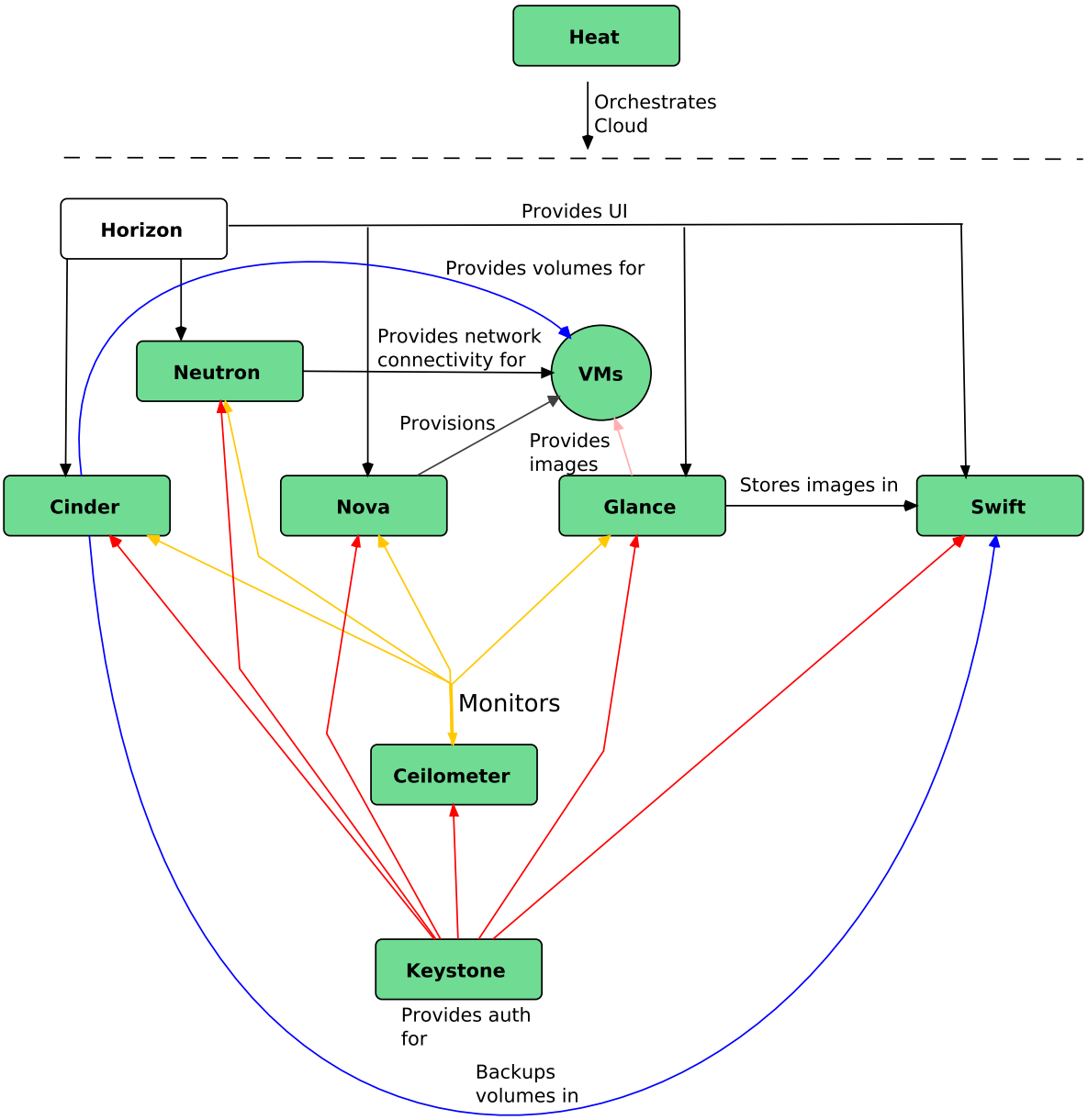



Figure 11.5: Conceptual Architecture of OpenStack (from [Foundation, 2015])

The OpenStack-based private Cloud at the TGI group is mainly dedicated to provide storage services. These services have been used to enable CRUD operations modeled and implemented by ICNets (see Chapter 10). Furthermore, the private Cloud is also useful to provide instances for running Renew simulations as discussed in Chapter 9. Due to some technical issues, some operations can not be performed with the local

Cloud. Therefore, another Cloud provider is used for testing purpose, which is called TryStack[3]. It will be briefly presented in the next section.

### 11.3.2 TryStack

Few years ago, building private Clouds was not obvious and required mostly commercial softwares. There was less support for Cloud developers and administrator. Thus, in order to test the implementations of this thesis the alternative is use services from existing (free) Cloud providers[4]. TryStack is a testing sandbox running OpenStack that developers using OpenStack in their applications could use to test their implementations. It is noteworthy to note that TryStack is only for testing purposes and not intended for production. Instances are deleted after 24h. This has no effect on the ICWorkflow framework. For future users that want to use ICWorkflow in concrete Cloud system, they need either to have their own Cloud (the case treated in this work) or to get payable instances from commercial Cloud providers (Amazon, Windows Azure and GAE). It depends on the nature of the applications that they want to perform on the Cloud. In this work, TryStack has been used to create Cloud instances. These instances served to perform Renew simulations in the Cloud [Bendoukha and Wagner, 2015, Bendoukha et al., 2015a]. Except the restrictions for the validity of the instances, TryStack provides almost the same fnctionality as OpenStack in term of networking, compute and storage.

### 11.3.3 Inter-Cloud Toolkits

Enabling an Inter-Cloud environment requires specific development tools. Programming directly against the Cloud is not trivial and needs a strong understanding of the Cloud platform and its APIs. One of the main problems is the vendor-lock in, which is diminishing the freedom of Cloud consumers taking advantage from the Cloud technology. The reason lies mostly at the interaction layer. There are still many challenging issues especially concerning portability and interoperability [Martino et al., 2015, Petcu, 2011]. Therefore, an additional abstraction layer should be provided in order to ease the access to the Cloud and give developers the freedom to work with different Cloud providers. There are different projects focusing on this issue such like: JClouds [JClouds, 2015], $\sigma$-Cloud [CloudSigma, 2015], LibCloud [LibCloud, 2015] and DeltaCloud [DeltaCloud, 2015]. In this work, JClouds has been chosen for implementing access and other operations for the Inter-Cloud. Since JClouds is written Java, this makes it easy the integration with Renew for the development of Cloud-based applications. The library support several known Cloud providers like Amazon, Azure and OpenStack. JClouds is an open source Java library that introduces abstractions aiming at the portability of applications and supports several Cloud providers including AWS, Azure and OpenStack. The resources that one can manage with JClouds are classified in three categories:

[3] `http://trystack.org/`

[4] There actually only few free Cloud services. There are many disadvantages and drawbacks. There are always limitations and can not fit the requirements.

- *ComputeService*: The JClouds API offers the ability to manage instances in the Cloud. One can start multiple machines at once. It also provides configuration possibilities to customize instances like: the *Image* used to create the computing node, he Hardware on which the instance will run, comprehensive of CPU speed, available RAM, and disk space; the Location where the machine will run [Martino et al., 2015].
- *BlobStore*: JClouds provides means for managing key-value storages from different Cloud providers. It also offers a Map view of the container to access the data.
- *LoadBalancer*: JClouds offers a common interface to configure load balancers. This functionality is still under development until now.

Concerning the ICWorkflow, JClouds has been mainly used to provide access to the Cloud. The other functionality, which are described in Section 11.4 are not based on the library.

## 11.4 The ICWorkflow Components

In this section ICWorkflow functionalities are presented. As stated above the objective is to provide means for the deployment and execution of Cloud-based applications. Besides the applications the objective is also to makes it easy the management of a Cloud environment. All the work focuses on the enabling of the interactions between Petri net processes and Cloud services. ICWorkflow is the underlying technology behind the ICNets introduced in Chapter 10. The framework is built up from many different components. The architecture is depicted in Figure. 11.6 and the components are: *IC-Administration*, *IC-Persistence*, *IC-GUI*, *IC-Instances* and *IC-DB*. Most of the description is based on the OpenStack terminology since it is the chosen platform to build the testbed (see Section 11.3). In the following a description of each of these components:

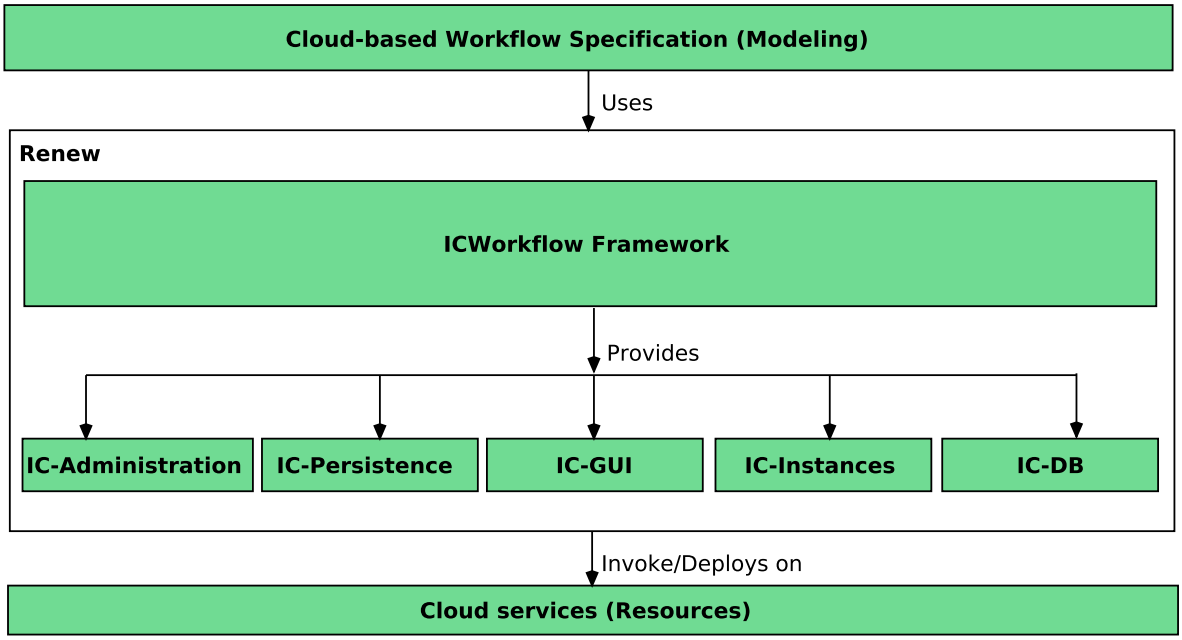



Figure 11.6: The ICWorkflow Framework

- *IC-Administration*: it sets and holds all required information for authenticating Cloud customers and configuring different *contexts*. A context defines the service that will be used and represents a specific connection to a particular provider. In case of an OpenStack Cloud, a context can be a *Nova*, a *Swift* or a BlobStore service for example. The application creating the context should be first authenticated connected. Generally, in most known Cloud providers such as Amazon, applications have to provide *credentials*, which consists mainly of the *Access Key ID*, the *Secret Access Key* and the *Endpoint address*.

- *IC-Instances*: the role of this component is to provide information about the instances running within a context. Most important information concerning a Cloud instance is its state. The possible statues of a Cloud instance (or a server) are: *Building*, *Active*, *Stopped*, *Suspended*, *Paused* and *Deleted*. The status of an instance is stored in local data base by the IC-DB component and can be at anytime retrieved and communicated to the user. Moreover, through the IC-Instances component is possible to provide information about running servers for a specific provider.

- *IC-DB*: this component is the most essential in the ICWorfklow framework. It allows two different modes of storage: *Object storage* and *Block storage* as well as the ephemeral[5] storage. In order to store data (objects) containers/directories must be created to hold the objects. The IC-DB component implements some

[5]The *ephemeral storage* as defined by `http://docs.openstack.org/openstack-ops/content/storage_decision.html` The disks associated with VMs are "ephemeral," meaning that (from the user's point of view) they effectively disappear when a virtual machine is terminated.

operations for the management of containers in OpenStack Cloud. For example creating, deleting or updating containers or listing container details. Furthermore, IC-DB allows CRUD operations on objects such like upload, download and delete objects from the Cloud. This function is performed later by ICNets.

- *IC-Persistence*: as stated above, the status of a Cloud instance is stored in local database in order to be communicated to other entities. Concretely, this concerns the entities described in Section 11.5. This functionality is performed by local (NoSQL) database named mongodb[6]. The information about a Cloud instance is useful for example prior deploying application in it. For instance, the *CloudBroker* presented in see Section 11.5 requires the status of the instance when selecting available Clouds.

- *IC-GUI*: the role of IC-GUI is to provide easy to use graphical user interfaces to Cloud customers. This enhances the usability and the accessibility of the ICWorkflow and respectively the Cloud in general. For example handling objects with command line is arduous for unexperienced users. IC-GUI offers forms for downloading and uploading objects in the (OpenStack) Cloud.

Some examples describing the use of these components with an OpenStack Cloud are available in Appendix B.1. It is noteworthy to note that all the features of the ICWorkflow framework have been exclusively performed on private OpenStack based Clouds. However, the ICWorkflow is designed to work with different Cloud providers. Extending the framework to support other Cloud providers such as Amazon (EC2 or S3) or Microsoft Azure is not complicated.

## 11.5 Cloud-based Workflow Management with ICNets and ICWorkflow

In this section another important feature of ICNets is presented. It concerns the design and the implementation of the system that manages workflows in an Inter-Cloud environment. An architecture is proposed to describe the entities responsible of workflow submission, deployment and execution.

The proposed architecture is depicted in Figure. 11.7. The components of the architecture are: *CloudBroker*, *CloudPersister*, *CloudRegister*, *WorkflowGUI* and the *CloudGateway*. The components and their roles will be explained gradually the architecture is described. As mentioned in the introduction (see Chapter 1) there is an emphasis on process management, (Cloud) service discovery and matching than security or monitoring issues. Before going further, an important information about the architecture needs to be taken into account. Most of the components of the architecture and their internal behavior (control flow) are modeled by reference nets. These nets execute tasks only after being instantiated (simulated). The components can communicate with each other, instantiate other components and transmit variables using *synchronous channels*.

[6] https://www.mongodb.org/

Workflow holders start by modeling their workflows using Renew[7]. Modeling a workflow consists of specifying clearly the control and data flow of the tasks. Considering the example presented in Section 1.1, the image path as well as the the service needed by the task (storage or compute) need to be communicated. These concrete information can be specified either directly in the *places* as *Strings* or through the *WorkflowGUI*, which provides forms (see Section 10.3.3). Such forms can also be provided and handled by the *Workflow* and the *WFNet* plug-ins. These plug-ins are highlighted with different color only to indicate that they are not critical for the specification of the workflow but bring more facility during the specification phase.

The next step is to invoke the *CloudBroker* with the given parameters. A request consists to mention the required Cloud service and optionally the name of the Cloud provider. The role of the *CloudBroker* is to check for available Cloud providers (and their actual status), that fit the requirements of workflow tasks. For this purpose, it invokes the *CloudPersister* and gets the list of available Cloud providers. This list is communicated to the workflow holder in order to approve the offer or to reject it. This mechanism can be provided by the *WFNet* plug-in especially the *Cloud Task Transition* introduced in Chapter 10.

The *CloudPersister* consists of a central (or distributed) data base for persistence) that contains useful information about the Cloud providers such as: Id, Name as well as instance and availability status. These information are obtained by the *CloudRegister*. In opposite to other approaches [Buyya et al., 2010], this is the role of the *CloudRegister* to communicate with the Cloud providers and request information about their status.

The *CloudGateway* is an essential component in the architecture. All the communication with the Cloud providers is performed through this component. Its role is to provide an unified layer to communicate with different Cloud providers regardless of their specifications and APIs. Therefore, advantage is taken from current technologies and techniques to hide complexity of an Inter-Cloud ecosystem. An interface is provided to support several well-known Cloud providers (Amazon AWS, Openstack, Windows Azure). The *CloudGateway* serves as a gateway for both *CloudRegister* and *CloudBroker*. It allows the *CloudBroker* to check the availability of a specific Cloud provider and permits to the *CloudBroker* to perform operations on different Clouds. To illustrate this, some Cloud-based operations have been performed on a private Cloud managed by *OpenStack* (see Section 10.3.3).

Figure. 11.8 shows the Petri net variant of the life-cycle of Cloud-based workflow and the participating components. The life-cycle covers the submission of the workflow, the selection of appropriate Cloud providers and the execution of the tasks.

As already mentioned several tasks can be performed by ICNets. For instance task T4 is responsible of contacting the Cloud provider and get the status of the Cloud instance. Like in the example presented in Section. 10.3 the process is coded in Java methods that can be instantiated from the transitions. It is important to notice here that this model does not reflect the behavior of each component. Although tasks which are performed by the components is represented by one single transition, the concrete

[7] Renew (`www.renew.de`) is an editor and simulation tool for different kind of Petri nets especially reference nets.

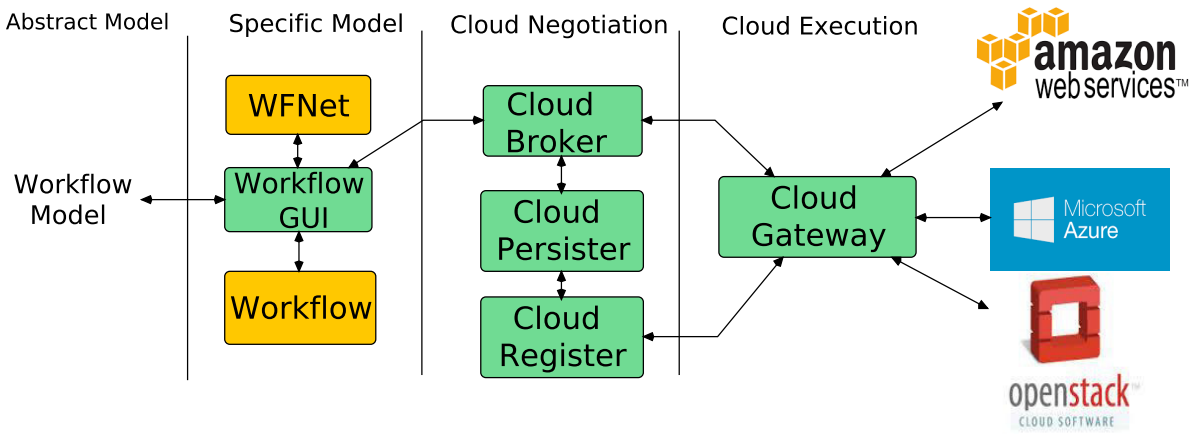



Figure 11.7: Cloud-based Workflow Management Architecture

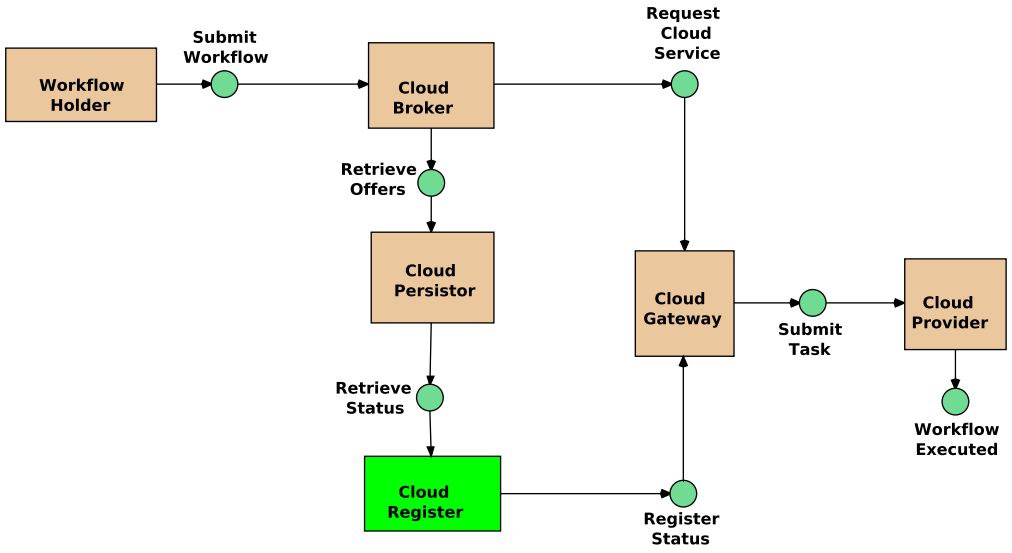



Figure 11.8: Workflow Life-cycle by Petri Nets

behavior is captured by another net model that should triggered from this transition. This is allowed by using synchronous channels, which permit the instantiation of net models (see Section 10.3).

## 11.6 ICWorkflow within the Paose Approach

The architecture presented in Section 11.5 can be designed and implemented following the Paose approach (see Chapter 6). It means adopting the agent metaphor for developing Cloud applications and their management. Agents can provide techniques and methodologies, which are adapted to dynamic behaviors of the Cloud environment. Functionality such as negotiating user access, automating the resource and service

discovery, and composition of cloud services can be performed by agents. The ICWorkflow can intervene at different level during the development process. For instance to implement agent-internal behavior.

As a proof of concept, some functionality of the ICWorkflow have been integrated within multi-agent applications. These applications are included in the Mulan/Capa framework and their development follows the Paose approach. Implemented functions can intervene at different phases when developing agent-based applications. Concretely, storage transactions in the Cloud are now possible to use in some steps of the Paose approach. For example within *protocol nets* or *decision components* [Cabac, 2010]. The use case concerns the PersistenceOntology plug-in from Mulan. This work will be described in detail in Chapter 13.

## 11.7 Conclusion

In this chapter the ICWorkflow framework is presented. It is integrated as plug-in Renew and it provides means to facilitate the interaction with multiple Clouds. It enables access, deployment and execution of workflows, which are specified by reference nets. Functionalities of ICWorkflow can be also used to implement Cloud-based WfMSs. As proof of concept CRUD operations have been implemented. It permits to store different kind of data in the Cloud. As Cloud provider, a private Cloud based on the OpenStack framework is made available at the TGI group. Furthermore, the ICWorkflow can be also used within an agent-based development approach such the Paose approach.

# Summary

This part presents solutions for the integration between Clouds and workflows. It argues that the integration necessitates a precise identification of the role of both Cloud technology and workflow concepts. There are two main topics that are investigated: *Cloud for workflows* and *workflow concepts for Cloud computing*. Thus, the relation between workflows and Cloud computing can be addressed in different ways. On the one hand, there is Cloud for workflow which consists of using Cloud resources to execute complex workflow tasks especially scientific workflows [Hoffa et al., 2008] [Juve et al., 2009]. Such kind of workflows are more resource-centric and focus on the computational tasks. On the other hand, designing Cloud ecosystems need structured and mature workflow concepts and high-level languages to handle issues like managing complex task and data dependencies.

This part covers both scenarios, mentioned above by providing new concepts and techniques for modeling (Ctt and ICNets), deployment and execution of Cloud-based workflows. The proposed approaches are based on a concrete study of the current issues facing the migration to the Cloud. The approach supports both modeling and execution. This is due to the fact that a special kind of high level Petri nets are used (reference nets). They allows to specify workflow models and use Java code as an inscription language. This feature is a large advantage, because it eliminates the gap between the conception and implementation phase.

Results obtained here affect directly Part III, since they represent the conceptual as well as the technical basis for the design and the implementation of the agent-based architecture introduced in Chapter 12.

The following part will discuss different possibilities to deploy the contribution obtained in Part II within agent-based approaches. Concerned are RenewGrass plug-in, Ctt, ICNets and ICWorkflow. Discussion about the usability of agents for distributed systems in general and for (Inter-) Cloud particularly and evaluation of the prototypes.

# Part III

# The Drop-Engine

# Table of Contents

# Introduction

This part is one of the central results of the thesis. It shows how agent concepts can be integrated into development process in order to design and implement a WfMS for the Cloud. Based on the Renew and the Mulan/Capa framework, the necessary workflow management functionality is provided for Cloud applications.

The Drop-Engine is the name given to the research discussed in this Part. The development of the Drop-Engine is presented as an example for a general process execution engine that is specifically developed for Cloud environments. What represents a major contribution of this thesis.

This part is complementary to Part II. It provides extensions to the contributions already presented. The extensions consist principally of integrating the agent metaphor to support Cloud-based workflow management systems. Approaches and architectures are proposed to achieve this goal. Whereas Chapter 12 tackles conceptually the relation between Cloud computing and agents, Chapter 13 shows how agents can be used with Ctt, ICNets and RenewGrass and ICWorkflow.

# Chapter 12

# Agent-Based (Cloud) Workflow Management

This chapter investigates the relation between Clouds and agents. The aim is to show how these two different *computing paradigms* can benefit from each other. Both conceptual and technical basis of the integration have been presented in Part II. The focus here is more on the advantage that one can benefit from the coupling of Cloud technology and workflow concepts. It has been demonstrated that Cloud computing provides workflows with large number of powerful resources while workflow concepts offer techniques and tools for modeling and deployment of applications in the Cloud. Beside workflow and Clouds, another aspect of the integration is the integration between workflows and agents, which is an important topic. The latter has been already investigated by Christine Reese [Reese, 2010] and Thomas Wagner [Wagner, 2010, Wagner and Moldt, 2015b]. Thus, the contributions of this thesis are strong candidates to enhance the above cited works.

## 12.1 Introduction

As mentioned in the introduction (see Chapter 1) the main objective of the dissertation is to provide new and adapted concepts, techniques and tools to support agent-based application development. The following sections give a state-of-the-art introduction to the relation between agents and Cloud computing and also workflows. The contribution of this dissertation to enable such an integration is discussed in Chapter 13.

## 12.2 Agent Paradigm for the (Inter-) Cloud

For this dissertation the relationship of the notions *agent* and *Cloud* need to be described. The most relevant references in this area are [Talia, 2012, Sim, 2012, Aversa et al., 2010, M. Spata, 2011a, I. Foster, 2004, Jander and Lamersdorf, 2013, Braubach et al., 2011]. Despite the differences between multi-agent systems and Cloud technology they can take benefits from each other. While Cloud computing offers a reliable and

scalable infrastructure for the execution of multi-agent systems especially in terms of modeling and simulating, agents can be used to handle Service Level Agreement (SLA) negotiation, Cloud service discovery and composition, etc.

According to [Talia, 2012], some of the important features that agent paradigm can bring to Cloud technology are intelligence, autonomy and adaptability at many levels during the application life-cycle and in order to improve Cloud resource and service management and discovery, SLA negotiation and service composition. In [Aversa et al., 2010], the notion of Cloud agency is presented. Authors used a mobile agent platform permitting to dynamically add and configure services on the virtual clusters offered by the Cloud. Multi-agent systems were also used to construct a Cloud computing federation mechanism to permit portability and interoperability between different Grid/Cloud computing platforms (see [M. Spata, 2011a] and [Foster and Kesselman, 2004]).

Much interesting work has been devoted to investigate the possibility to integrate agent, workflow concepts and Cloud computing. For example, Pandey et al. [Pandey et al., 2011b] present a high-level architecture of a workflow management system for developing distributed applications on the Cloud. Key components of the presented architecture are: A *Market-Maker broker* and a *workflow engine* to schedule workflow tasks to the resources based on the QoS constraints. Liu et al. [Liu et al., 2012] outline three key issues in the design of Cloud workflow systems: *system architecture* that decides how the system components are organized and how they interface with each other, *system functionality* that realizes the basic workflow system's functionality and manages the Cloud resources, and finally *QoS* management. In [Liu et al., 2010], *SwinDeW-C*: a peer-to-peer workflow management system for Cloud is proposed.

Concerning the Inter-Cloud, Buyya et al. [Buyya et al., 2010] present the notion of federated Cloud (Inter-Cloud) that facilitates scalable provisioning of services under variable conditions. In [Grozev and Buyya, 2012b], the authors provide a classification of Inter-Cloud delivery models, which are *federated Cloud* and *multi-cloud*. The EU-funded RESERVOIR[1] project [Rochwerger et al., 2009] is the first initiative intending to provide open source technology to enable deployment and management of complex services across different administrative domains. The EU-funded mOSAIC[2] project [Martino et al., 2011] proposes a complementary solution based on software agents and semantic data processing. The mOSAIC approach is based on a *Cloud Agency* gathering client and provider agents in a brokerage process working with service level agreements. It is used as a Multi-Cloud resource management middle-ware, it plays the role of run-time environment in the model-driven engineering project named MODAClouds [Ardagna et al., 2012].

In [Zhang and Zhang, 2009], the Mobile Agent Based Open Cloud Computing Federation (MAB-
OCCF) is presented, where data and code are transferred from one device to another via mobile agents. Each mobile agent is executed in a virtual machine called Mobile Agent Place (MAP), and the mobile agents are able to move between MAPs, and also to

[1] http://www.reservoir-fp7.eu/
[2] http://www.mosaic-fp7.eu/

communicate and negotiate with each other, realizing portability among heterogeneous Cloud computing service providers. In [Sim, 2012], the concept of agent-based Cloud computing is introduced. This concept is introduced to aid the development of software tools for service operations in the Cloud using agent-based cooperative techniques. WADE (Workflow and Agent Development Environment) [Caire et al., 2008] is a domain independent platform built on top of JADE[3], it allows to develop distributed and decentralized applications based on the agent paradigm and the workflow metaphor.

## 12.3 Agents as a Workflow Management Infrastructure

In this section the objective is to give an overview about the contribution of this work for Cloud-based workflow management. The approach is based on several concepts, techniques and tools. An architecture is proposed and described to provide more clarity on the objectives. Before that, some important related work, which is related to the approach is outlined.

### 12.3.1 The Architecture

The architecture is depicted in Figure. 12.1. It includes three basic layers from top to bottom:

- *The User applications layer* (UL): permits both managing users (access to the system) and monitoring deployed workflows,
- *The Middleware layer* (ML): composed mainly of the workflow engine as well as the task dispatcher module (see step 4).
- *The Resource layer (Cloud infrastructure)* (RL): This layer represents the resources used to excute the workflow tasks. They can be either compute or storage services. This depends on the workflow requirements.

[3] http://jade.tilab.com/

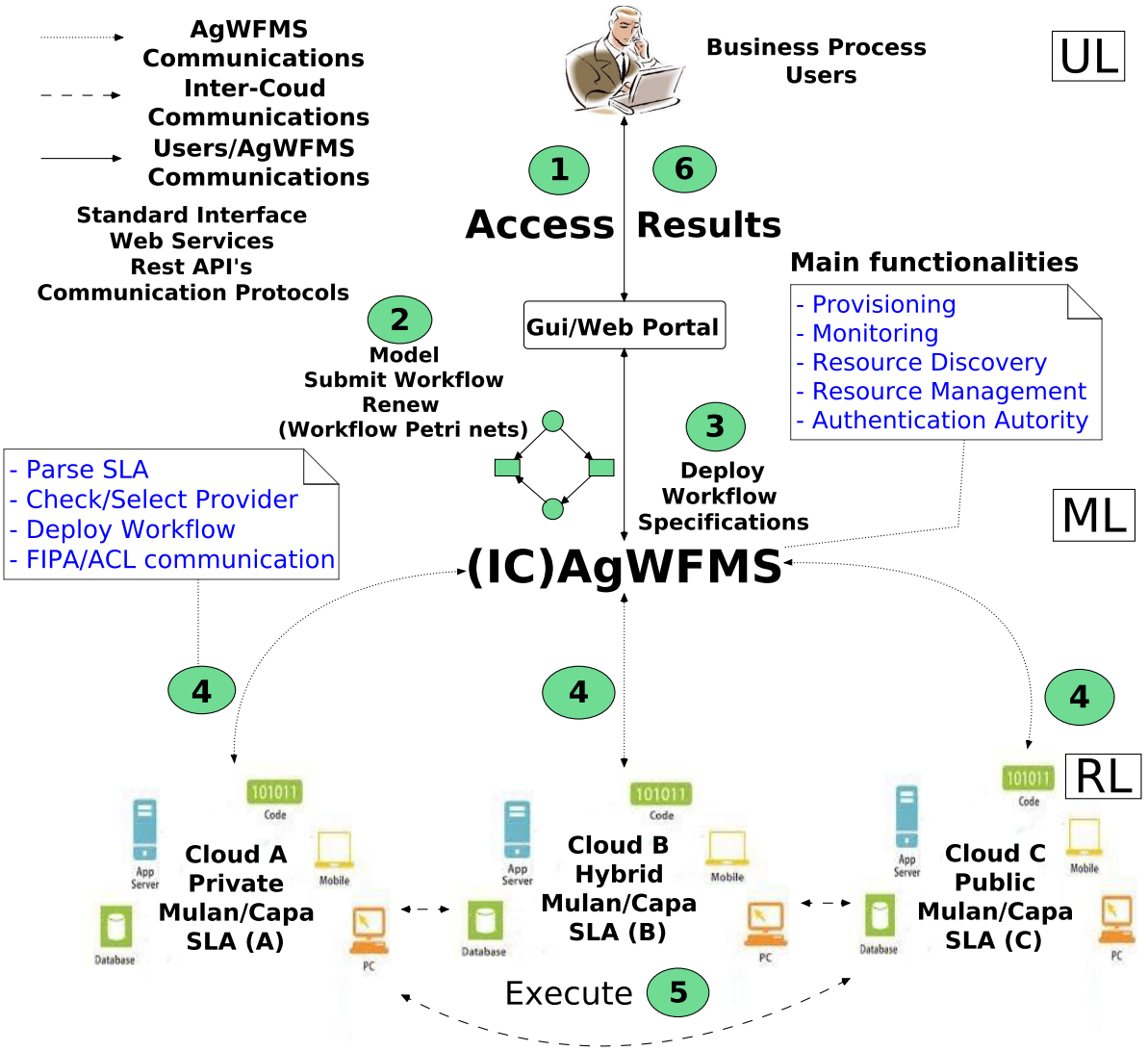



Figure 12.1: Workflow Management for the Inter-Cloud (From [Bendoukha et al., 2013])

Note that in this section the focus is on the system architecture, leaving most of Cloud management services (pricing, accounting and virtualization) and other functional modules aside. As shown in Figure. 12.1, managing workflows can be broken down into a series of steps and carried out by several components. These steps are as follows:

1. *Access* the user application provided by software agents. These agents provide some kind of (graphical) interface for a user to work with. Any authentication authority is also enforced at this point.

2. *Model* the workflow using the provided/supported tools. These tools may be incorporated into the agents or their environment, but it is also possible to use

external tools for this. The result of this step is a workflow definition in a specific format supported by the IC-AgWfMS.

3. *Deploy* the workflow definition to the IC-AgWfMS. This step includes saving the definition into a storage available to the IC-AgWfMS (e.g. a database or Cloud service), possibly transforming the definition into an internal format, and performing any checks on the definition (e.g. file readability, consistency).

4. Begin *enacting* the workflow. This includes a number of steps in the IC-AgWfMS:

   (a) *Analyze* the requirements of the workflow. The IC-AgWfMS needs to gather the information about what is needed for the workflow to be completed (e.g. which resources for task execution, what technical basis)

   (b) *Check* available platforms and retrieve their SLAs (*Discovery phase*). For this, the IC-AgWfMS gathers the SLAs from the available platforms. These are then parsed in order to be compared.

   (c) *Select* the platforms capable of executing the workflow tasks (*Match-making phase*). The deciding factor for this is the SLA. The SLA/platform, which is best suited for the specific requirements of the workflow, is chosen.

   (d) *Instantiate* the workflow and distribute the tasks on the selected platforms. The workflow definition and any necessary input information are sent to the selected platform, which handles instantiation and management itself.

5. *Execute* the workflow. This is done by the resources connected to the selected platform (*Execution phase*). Once all tasks[4] of the workflow have been successfully executed it can be finished. The platform then terminates it and informs the IC-AgWfMS.

6. *Notify* the original modeler of the successful completion of the workflow.

The central component in the architecture is the IC-AgWfMS because most of the work is done at this level, it has the ability to analyze the application requirements and find the best provider that fits these requirements. Of course, applications are in form of workflows modeled with Petri nets. So the system has to go through the net and list the Cloud services needed by each transition.

To achieve better Quality of Service (QoS), reliability, and flexibility step (4) plays an important role in the system. It has the responsibility of facilitating resource matching and sharing. It involves resource planning and delivery via the use of schedulers and load balancers (which are out of scope of this thesis). In this approach an SLA-based mechanism for Cloud service delivery can be used. The advantage of a SLA-based approach is that it is more transparent to Inter-Cloud clients. The latter have no control on the provisioned resources. Therefore, there is need of a certain level of trust between Inter-Cloud clients and the Inter-Cloud providers. As shown in Figure. 12.1, a

[4]A task is the central concept in workflow modeling. It is a unit of work to be performed on the Cloud.

centralized Inter-Cloud administration is shown. It means that the IC-AgWfMS acts as a central entity to mediate between Inter-Cloud clients and the Inter-Cloud providers.

An Inter-Cloud is clearly a distributed environment composed of multiple heterogeneous partners. Collaborating workflow concepts and agent capabilities enhance the mapping of workflows onto Cloud resources. As stated in [GICTF, 2010], one of the Inter-Cloud use cases is disaster recovery or large scale failure. In this case, there is an eminent need to move the execution of the workflow to another Cloud provider. The idea is to encapsulate the workflow and its related information (specification and data) into an agent with mobility capabilities to recover the workflow again and execute it on another platform. This solution can be time-consuming due to the fact that switching to another Cloud provider must follow all the steps (starting with (4)) described above.

For a better cooperation between implied partners, Cloud platforms are also considered as agent platforms. Thus, interactions between the Clouds is assured through agent communication languages like FIPA/ACL.

### 12.3.2 Workflow/ Agent Integration

An area that is also investigated is the integration of agents and workflows on a conceptual level (see [Wagner, 2012, Wagner and Moldt, 2011, Wagner and Moldt, 2015a]). This work is motivated by the desire to combine the inherent strengths of both the individual agent and workflow concepts, in order to balance out their weaknesses. Agents will benefit from the behavioral view provided by workflows, while workflows will gain support for the structural view associated with agents. The link between these views will be established through the shared ontology. The resulting benefits include the transfer of the individual concepts properties to the respective other one. This means that workflows can exhibit agent properties, like intelligence and mobility, while agents can make use of workflow aspects, such as task atomicity. The integration efforts though do not stop at these partial integrations, but combine agents and workflows on every level to achieve a full integration [Bendoukha et al., 2013].

The full integration offers both agents and workflows on the same abstraction level for modeling. Practically the integration takes place on the lowest level, which combines the basic agent actions (send message, receive message, internal action) and the operations of tasks (accept/request workitem, complete activity, abort activity). The necessary adaption to agent and workflow management cascade throughout the system architecture. This allows modelers to incorporate and combine any aspects of the two individual concepts on the different abstraction levels. While the integration efforts are, in general, decoupled from the current work on Clouds and the IC-AgWfMS, the two areas of research benefit from one another. The introduction of Cloud services into the integration offers improved scalability and flexibility, especially for the workflow aspects. Coupling workflow concepts and agent features will provide solutions to overcome issues related to moving business processes to the Cloud (Inter-Cloud). Generally, agents improve WfMSs by adding a certain level of intelligence and mobility [Bendoukha et al., 2013].

## 12.4 Conclusion

This short chapter presents the features that agents can provide to support building complex application based on Cloud resources. It also provide agent-oriented architecture that cover workflow life-cycle based on Cloud resources. The following chapter treats the topic with concrete examples. The latter are extensions to the works presented in Part II.

# Chapter 13

# Prototypes

This chapter shows how the results presented previously in Part II are applicable within agent-oriented approaches. The contributions of this chapter differ from abstract/conceptual to technical. The chapter is an essential outcome of this dissertation since it shows the concrete usability of Ctt, RenewGrass, ICNets and ICWorkflow in building agent-based application following the Paose approach.

## 13.1 Introduction

The essential objective of this dissertation as described in Chapter 1 is to provide techniques and tools for Cloud application developers. The contribution covers specification as well as deployment and execution of those applications in Cloud-like environments. If the previous part covered the relation between workflows and Clouds, the current chapter integrates the agent paradigm as a complementary layer. Chapter 12 already presented the benefits that can Clouds and agents take from each other. Here it is about concrete examples of the integration between Clouds and agents. It concerns mainly the concepts, techniques and tools presented in Part II. Some solutions here are conceptual and have been not implemented while others are fully functional with a real Cloud system. In this chapter it is presented how Ctt, ICNets, ICWorkflow and RenewGrass are deployed following an agent-based approaches and systems. Principally, some concepts and techniques from the Paose approach such as the Mulan/Capa framework are used. For instance some functionality of ICNets are integrated with the persistence ontology from Mulan. Section 13.2 presents a conceptual integration of the Ctt for agent interactions. Section 13.3 shows a concrete example of the usability of ICNets for Cloud-based workflow management systems Section 13.4 concerns the RenewGrass plug-in and its conceptual deployment in Cloud-like environments following agent concepts.

## 13.2 Ctt for Agents

In this section the usability of Ctt within an agent-based approach is introduced. It concerns the use of Ctt as platform covering the whole business process execution.

### 13.2.1 The Cloud Task Transition for Agent Interactions

For the correct execution of the (business process) workflow there needs to be a frame. The aim is to offer a frame which covers the whole business process. To do so, a virtual platform for each respective business process is provided. On this platform there are descriptions for business interaction patterns that are necessary to perform a business process. Business interaction patterns consist of roles that interact. Overall each process has to conform to the workflow specification which at the most abstract level defined as an AIP.

In the context of the Paose approach all required participants in the (workflow) process are agent-like entities acting within the platform. The Figure. 13.2 demonstrates this where the Petri net for the producer/consumer AIP is modeled. This is just a simple example to show the way how agent interactions are modeled using Petri nets. Usually there are several agents (and roles) performing required tasks in a complex process.

While those parts of the AIP that belong to a business process of a certain role are modeled by an agent protocol net, other parts are modeled in knowledge bases or so called decision components. All relevant parts that form the whole setting of a specific business process are combined within a set of different nets. These nets are used as parameters of the Ctt. Figure. 13.1 shows the principle idea.

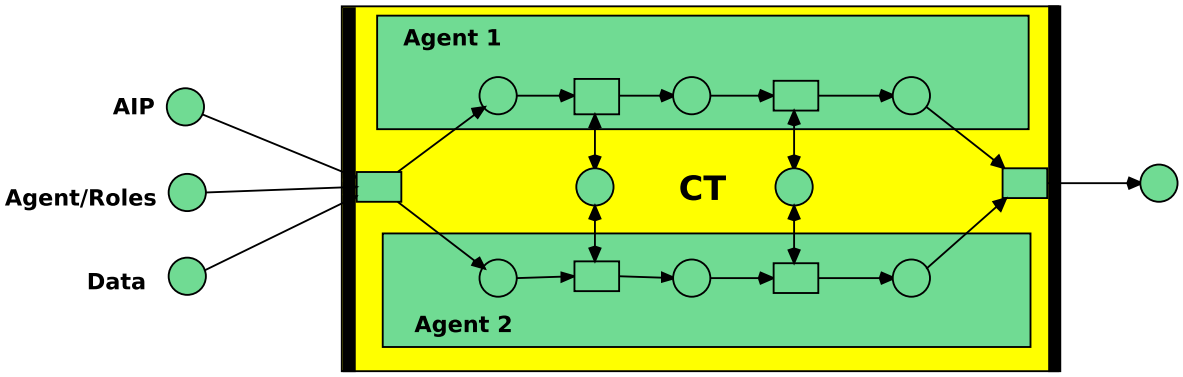



Figure 13.1: The Cloud Task Transition in Agent-based Software Development

The implementation of this idea follows the workflow transition concept (see Chapter 10), indicated by the two strong bars at each side of the refined transition.

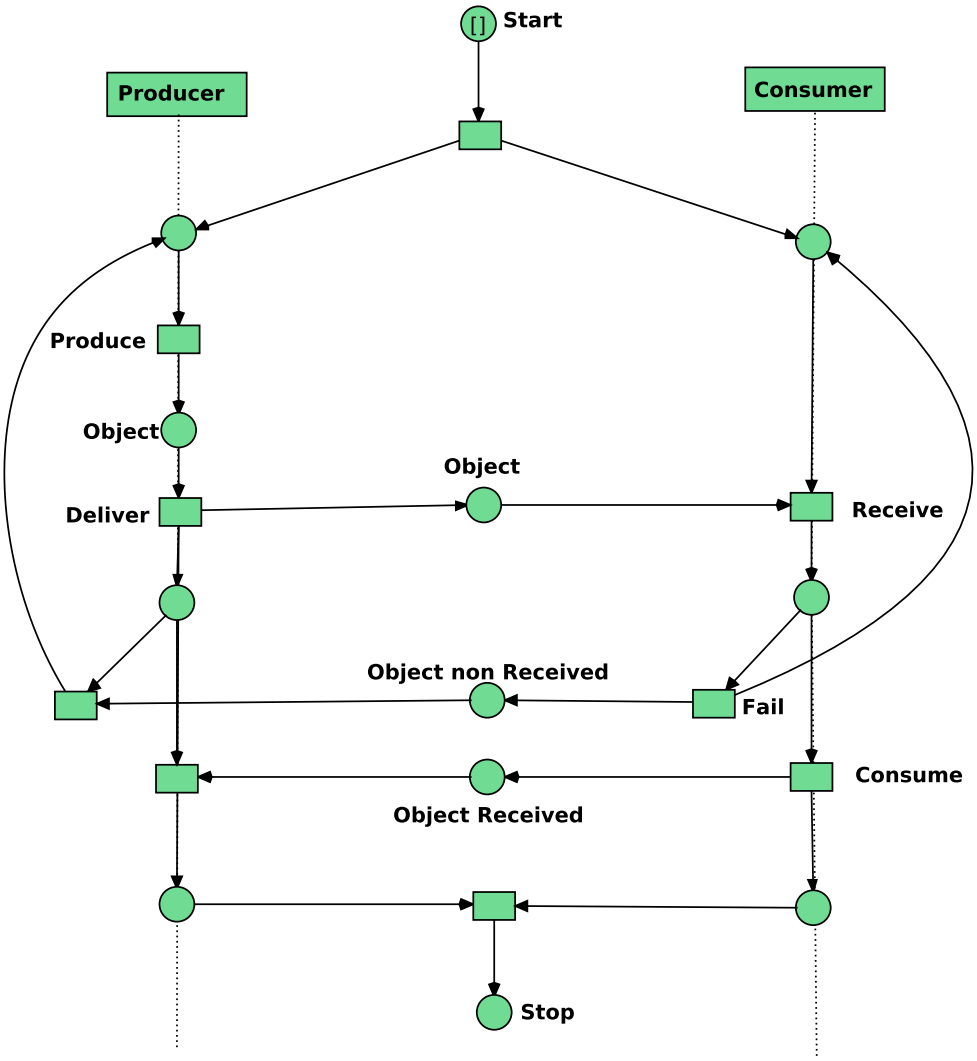



Figure 13.2: Reference Net for Producer/Consumer AIP

However, special types of parameters will be used: AIPs, roles and data. The internal structure in Figure. 13.1 is only a hint about the automatically generated shadow net instance in Renew. The shadow net is generated to hide the real set of nets that can really execute the specific business process instance. Here it reflects the separation into two different threads which are somehow synchronized. In general a set of agents gets a certain set of nets assigned and executed them. Depending on the availability of the types and specific data some nets can be typed in advance and some need to be generated at run-time via reflection mechanisms of Java.

Each role is assigned to the respective agents. The data set is partitioned according to the roles and the ontology. The agents execute then the assigned AIP fragments that belong to their assigned role. Cloud transitions as the platform of all these agents (components) regulate, control and monitor the overall execution of the specific business process instance.

### 13.2.2 Related Work

Efficient inter-organizational business processes require the implication of multiple partner interchange and inter-organizational process management. From a process-oriented perspective, modeling, analysis, and automated execution of distributed workflows gain more and more importance. Cloud computing offers a shared infrastructure for improving business processes. Organizations and their partners can share data and applications in the Cloud. In [Buijs et al., 2012], the term *Shared Business Process Management Infrastructure* (SBPMI) was introduced to support different process variations through configuration utilizing configurable process models. In [Aversa et al., 2010], the notion of Cloud agency is presented. Authors use a mobile agent platform permitting to dynamically add and configure services on the virtual clusters offered from the Cloud. Multi-agent systems have been used to construct a Cloud computing federation mechanism to permit portability and interoperability between different Grid/Cloud computing platforms (see [M. Spata, 2011b] and [I. Foster, 2004]).

This work follows the Paose approach that allows the development of multi-agent applications on the basis of the Mulan/Capa (Multi-Agent Nets, see [Rölke, 2004]; Concurrent Agent Platform Architecture, see [Duvigneau et al., 2003]) framework. Paose follows a multi-dimensional perspective dividing the whole setting into models for interactions, roles and ontology. Agents' interactions are modeled on an abstract level using AIP (Agent Interaction Protocol Diagrams) [Cabac, 2010]. Agent Interaction Protocol Diagrams are defined by the FIPA (Foundation for Intelligent Physical Agents) in the AUML (Agent Unified Modeling Language). The advantage of modeling in AUML is its intuitive graphical representation of the processes/scenarios. Using high-level Petri nets for the operational semantics a plug-in is provided to generate reference net models directly from the AIPs.

## 13.3 Integration within Agent-based Approach

In this section the usability of the ICNets and ICWorkflow in an agent-based approach is presented. ICNets already introduced in Chapter 10 allow for the development of (Inter) Cloud-based workflows and their management with ease. In this chapter only the first feature of ICNets has been presented. It consists of the specification and the execution of workflows in an Inter-Cloud environment. CRUD operations have been implemented based on the introduced concepts.

### ICNets and ICWorkflow for PersistentOntology

Here another feature of ICNets is presented and concerns the integration of the agent flavor. As already mentioned CRUD operations have been implemented in form of Petri nets-based models (see Chapter 10). This is performed based on the ICWorkflow plug-in. In order to integrate the current work within an agent-based approach there is no better then following Paose approach. Paose supports the development of MAS through the Mulan/Capa. With respect to storage capability Mulan/Capa framework provides

persistence for ontologies. Ontologies define a common understanding of knowledge-entities, which they store in a knowledge base. The idea is to provide agent service for storing, querying, updating and deleting of persistable concepts. This functionality is performed by two agent roles, which are *Depositer* and *Persister*. The *Depositer* role is assigned to each agent wishing to use persistence agent. Figure. 13.3 shows a Use-Case that was used for the coarse design. It illustrates the two roles Depositor and Persister who take part in the four interactions store, query, update and delete. The communication between two agents is in form of protocols based on a special Petri net formalism combining Java and Petri nets under reference semantics Same applies to the agents internal reasoning, which is carried out in so called decision components (DCs), also Java reference nets. The access to the database is wrapped by a helper class which implements the DBHelper interface. Technically, there are two implementations, db4o[1] and recently mongodb[2]. Hence, this is the where ICNets and ICWorkflow can intervene. The idea is that rather using local databases mechanisms are implemented to allow the use a Cloud-based storage service. This allows to perform CRUD operations in the Cloud. Thus the objective is to investigate how the *PersistentOntology* can benefit from ICNets and the ICWorkflow plug-in in Renew. This opens new perspectives for the development of Cloud-based applications based on the Mulan/Capa framework. Figure. 13.4 shows the AIP diagram of the interaction *store* between *Depositor* and *Persister*. The implementations of the interface are used by the Persister Decision Component (Persister_DC), the agent internal persistence. The agent communicative persistence interface is represented by the four interactions store, query, update and delete between *Depositor* and *Persister*. The user interface can be provided by the agent with the Depositor role or by another database access layer. Stored objects can be viewed at the OpenStack dashboard (see Figure. 13.6). On the right side one can see the containers and on the right the corresponding objects.

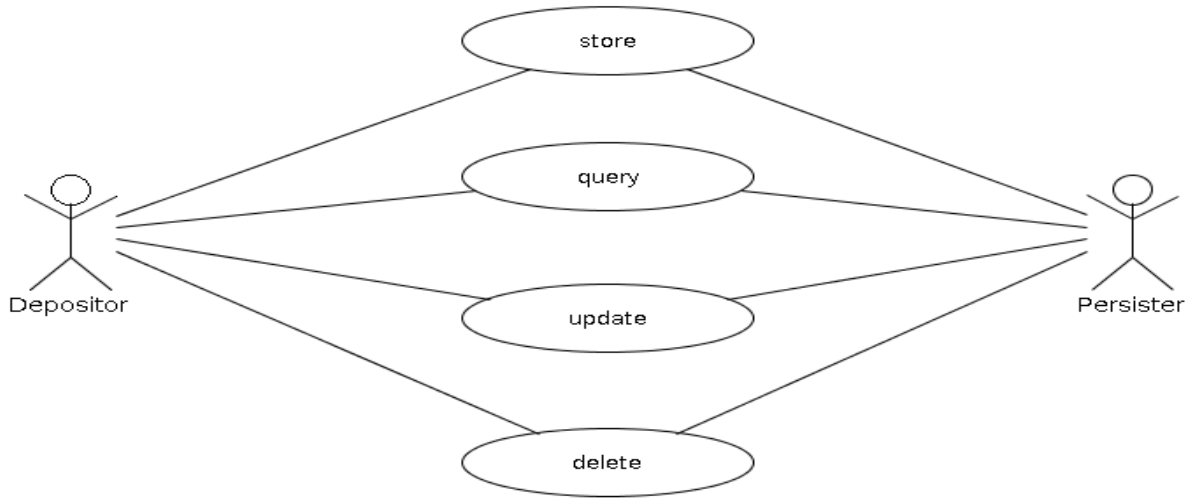



Figure 13.3: Persistent Ontology Use Case

[1] www.db4o.com
[2] www.mongodb.org

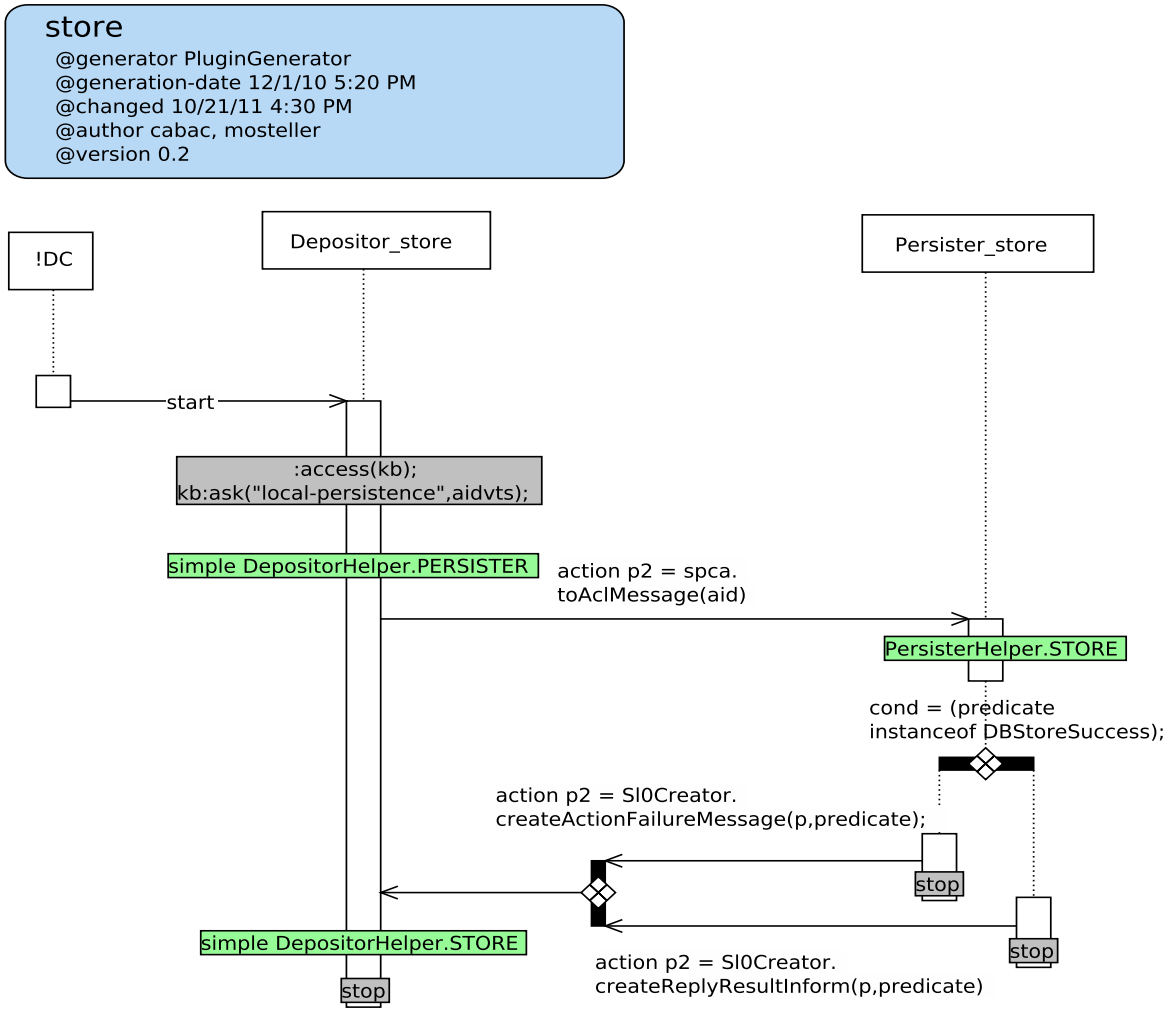



Figure 13.4: Store AIP (Figure taken from internal project; authors are L. Cabac and D. Mosteller)

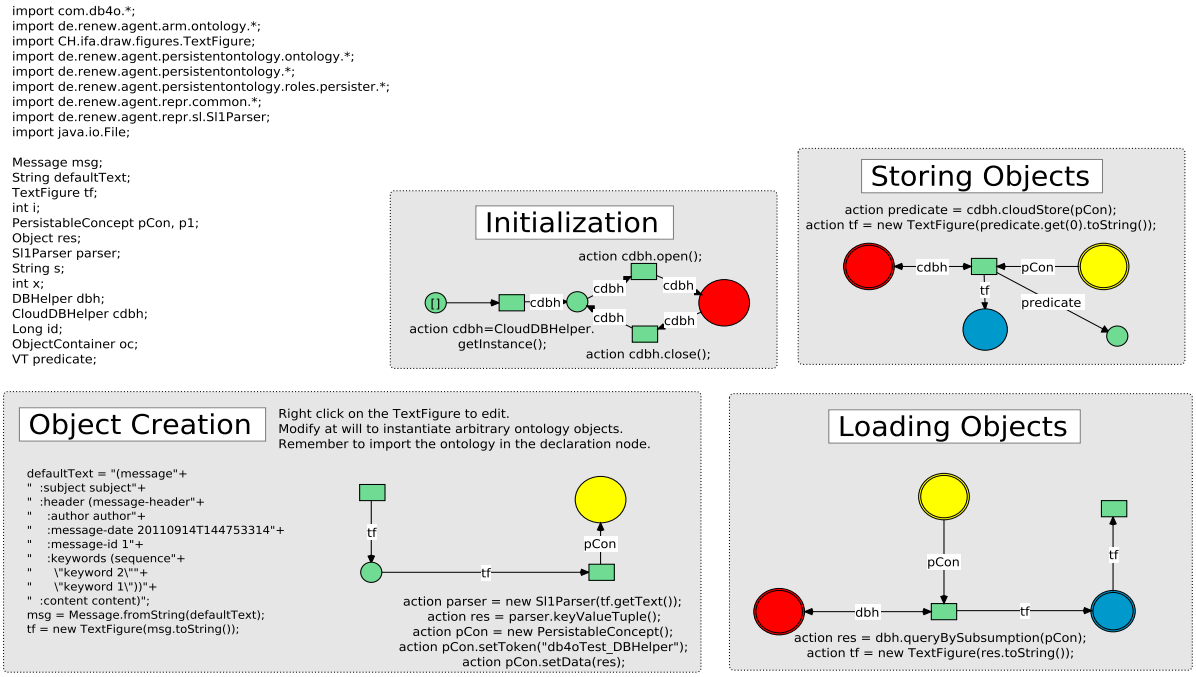



Figure 13.5: A Test Net for Storage in the Cloud

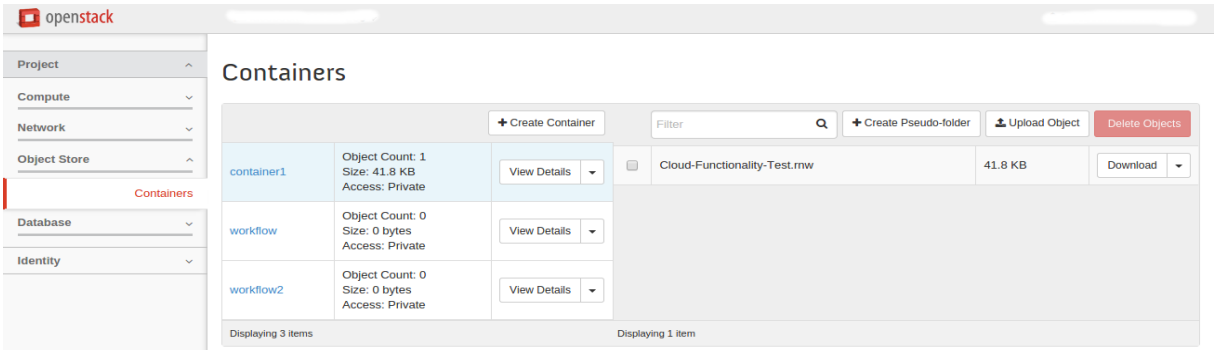



Figure 13.6: OpenStack Dashboard

The previous work has been achieved in order to prove the applicability of ICNets and ICWorkflow within the Mulan/Capa framework. Nevertheless, this work does not really reflect the real feature that these contributions can offer for complex workflows. In fact, the role of the current work is to allow scientists to transfer large amount of data to the Cloud. This can be applicable to the RenewGrass plug-in (see Chapter 8 and Section 13.4). A use case could be the processing of satellite imagery, which are characterized to be data and compute intensive. There is no difficulty to perform transfer big amount of data, ICNets handle different types of data. ICNets and ICWorkflow could be applicable during the AOSE teaching project. During the project, students work collaboratively to develop agent-based applications following the Paose approach. Thus as future work, functionality of ICNets or ICWorkflow can be used as means

to develop Cloud-based application following Paose approach. Agents will have the ability to interact with the Cloud in order to satisfy the requirements of workflow tasks.

## 13.4 RenewGrass in the Cloud

RenewGrass has been successfully tested in a local environment. All the required components are hosted on-premise (Renew, Grass GIS and the data). Nevertheless, as soon as the number of tasks increases and the size of data become large, there is computing and storage issues. This is due to insufficient resources on the local site. This section presents a vision to deploy the current implementation of the RenewGrass tool onto the Cloud services. First, different possibilities to Cloud-enable an application in general are shortly illustrated. These possibilities are formulated in form of patterns. The illustration is based on the work of [Han et al., 2010, Strauch et al., 2013, Anstett et al., 2009]. They investigate and strive to answer questions like: Where to enact the processes? Where to execute the activities? and Where to store the data? Thus the entities taken into account are: (1) the process engine (responsible for the execution and the monitoring of the activities) (2) the activities that need to be executed by the workflow and (3) the data. Next, an architecture is proposed to introduce a new pattern and an appropriate methodology to enable remote execution in the Cloud.

### 13.4.1 Migration Patterns

As mentioned in Chapter 7 moving an existing application to the Cloud should be based on a solid strategy. Providing the business management system (or WfMS) or a part in the Cloud raises a series of concerns about ensuring the security of the data and the performance of the system. For example, Cloud users could lose control on their own data in case of a fully Cloud-based solution. Some activities, which are not compute-intensive can be executed on-premise rather than moving them to the Cloud. Unfortunately, this transfer can be time and cost-consuming because of the pay-per-use model and the nature of the workflow tasks.

In the following, the patterns from [Han et al., 2010, Anstett et al., 2009] are shortly introduced. The first pattern designs the traditional scenario where all the components of the workflow system are hosted at the user-side (on-premise). The second scenario represents a case when users already have a workflow engine but the application contains compute or data-intensive activities, so they are moved to the Cloud for acquiring more capabilities and better performance. The third case designs a situation, where the end-users do not have a workflow engine, so they use a Cloud-based workflow engine, which is provided on-demand. In that case, workflow designers can specify transfer requirements of activity execution and data storage, for example, sensitive data and non-compute-intensive activities can be hosted on-premise, and compute-intensive activities and non-sensitive data can be moved to the Cloud [Han et al., 2010]. The last scenario presents a situation where all components are hosted in a Cloud and accessed probably from a Web interface. The advantage is that users do not need to install and configure any software on the user side. To make an analogy with the

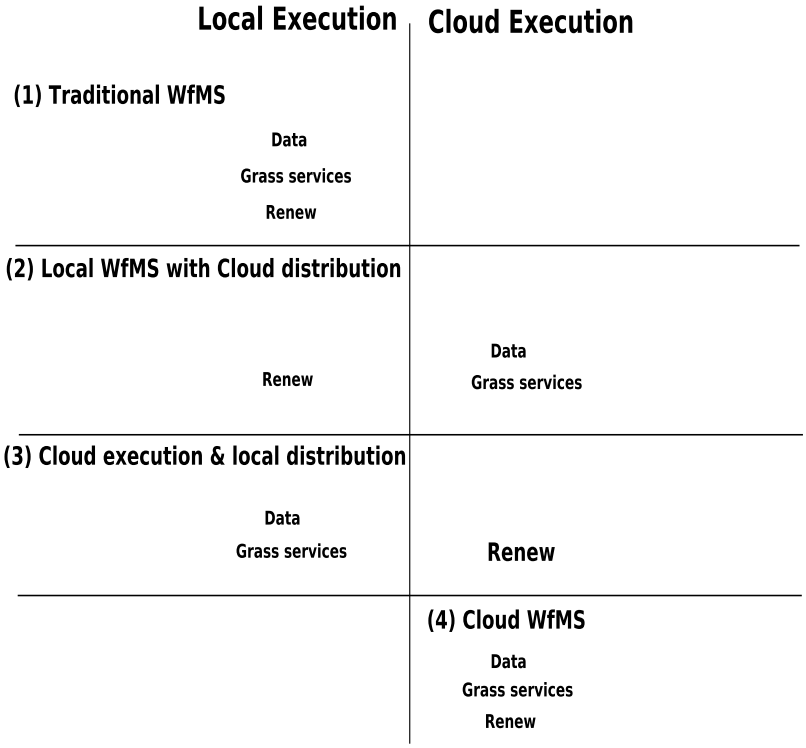



Figure 13.7: Patterns for Cloud-based Workflow Systems
(Adapted from [Han et al., 2010])

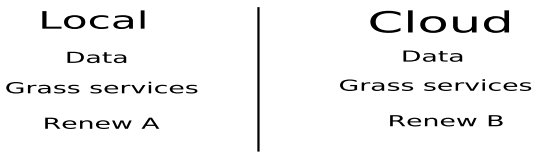



Figure 13.8: A Pattern Designing Multiple Process Engines Integration

elements of the proposed approach, Renew is the process engine, the activities are the geoprocessing tasks (performed by the Grass services), which are related to the satellite images (data). The latter (data and Grass GIS) can be either on-premise or hosted in the Cloud. Based on these elements and the illustration presented above, Figure. 13.7 presents an overview of the diverse approaches (patterns) to design the workflow system based on the Cloud technology.

Both [Han et al., 2010] and [Anstett et al., 2009] do not address all possible situations. For instance, the following situation has been not addressed: the process engine is available on the user side but due to circumstances (internal failure, not sufficient compute or storage resources), remote process engines need to be integrated and remotely invoked. Figure. 13.8 shows approximately how this scenario looks like. The solution consists on transferring the data and executing the process by another process engine. This is discussed in the following sections.

### 13.4.2 Architecture

Figure. 13.9 shows the architecture to integrate the current implementation into a Cloud system. While RenewGrass is already implemented and successfully integrated in Renew, most efforts are dedicated now to move the execution of the geoprocessing tasks to the Cloud and the provision of an interface to invoke these services directly from workflow models.

In this work, an agent-based approach is followed, i.e., many components functionalities are performed by special agents. In summary, the role of each agent used in the approach is described below.

1. *Workflow Holder Agent*: The Workflow holder is the entity that specifies the workflow and in consequence holds the generated Petri net models. This entity can be either human or a software component. The specification of the image processing workflow is performed using RenewGrass, which provides a modeling palette or downright predefined modeling blocks.

2. *Cloud Portal Agent*: provides the *Workflow Holder Agent* a Web portal as a primary interface to the whole system. It contains two components: *Cloud Manager* and *Workflow Submission Interface*. The latter provides a Web interface to the workflow holders to upload all necessary files to execute the workflow. This includes the workflow specification (Renew formats[3]), input files (images). It also serves getting notifications from the *Cloud Broker Agent* about the status of the workflow or the availability of the Cloud provider. The role of the *Cloud Manager* is to control the Cloud instances (start and stop or suspend).

3. *Cloud Broker Agent*: It is a critical component of the architecture, since it is responsible of (i) the evaluation and selection of the Cloud providers that fits the workflow's requirements (e.g., data volume and computing intensities) and (ii) mapping the workflow tasks.Both activities require information about the Cloud provider, which are available and provided by the Cloud Repository Agent.

4. *Cloud Repository Agent*: The Cloud repository register the information about the Cloud providers and the state of their services. These information are saved in a database and are constantly updated, since they are required by the Cloud Broker Agent. To avoid failure scenarios (repository down, loss of data),distributed databases can be used, which allows high availability and fault-tolerant persistence.

5. *Cloud Provider Agent*: The role of this agent is to control the instances and to manage the execution of the tasks. Regularly, the Cloud providers need to update their status and send it to the Cloud Repository Agent. The status concerns both the instance and the services (Grass services).

Concerning the *Cloud Broker Agent*, the evaluation and the selection of the Cloud providers are critical processes for the *Workflow Holders*. In Cloud computing there

[3]Renew supports various file formats saving (XML, .rnw, .sns, etc.)

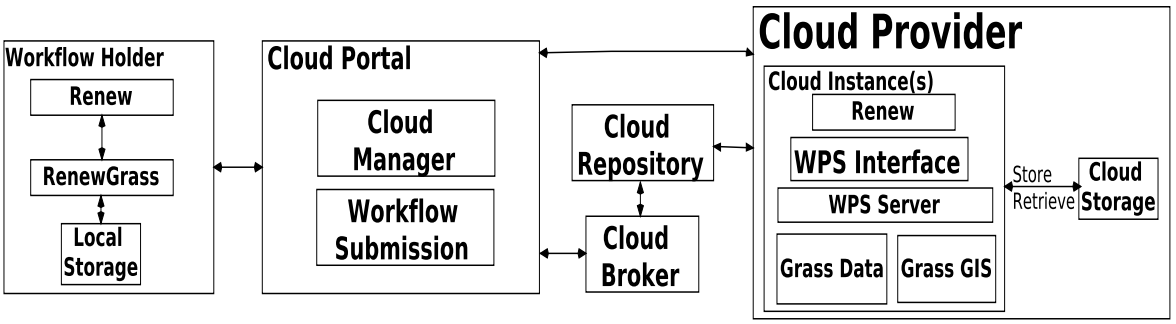



Figure 13.9: The Architecture of the Cloud-based Workflow System

are various factors impacting the Cloud provider evaluation and selection [Haddad, 2011, Lee Badger and Voas, 2011] such as: computational capacity, IT security and privacy, reliability and trustworthiness, customization degree and flexibility/scalability, manageability/usability and customer service, geolocations of Cloud infrastructures. In the current work, brokering factors are limited to the computational capacity and the customization degree.

### 13.4.3 Cloud Configuration

Concerning the customization degree, there are some requirements for a successful deployment onto the Cloud. In general, a common procedure to deploy applications onto Cloud services consists of these two main steps:

1. *set up the environment*: this mainly consists of the provision of Cloud instances[4] and the configuration the required softwares properly. Essential are the environment variables, which differ from the local implementation such as JAVA configuration for the Web server.

2. *deploy the application*: it consists of the customization of the Cloud instance with the appropriate softwares. For this work, Renew and the Grass GIS should be correctly and properly configured, especially the database and the installation path.

Furthermore, the Grass commands can be invoked in different ways. Either through a wrapper like in the original implementation of RenewGrass or provided as Web services. For the latter, a Web-based approach is followed with respect to the Open Geospatial Consortium (OGC) Web Processing Service (WPS) interface specification. Thus the Grass GIS functionalities are provided as Web services instead of desktop

[4]The Cloud instances should be in priori customized, i.e, they need to have the Grass GIS as back-end, the WPS server and Renew. It is assumed that the Cloud storage is a service, which is configured by the Cloud provider itself.

application. To achieve this, the 52North[5] as a WPS server as well as the wps-grass-bridge[6] are good candidates.

### 13.4.4 Execution Scenario

Considering the proposed architecture and the agent roles described above, a typical deployment scenario is broken into the following steps:

1. Workflow holders specify their image processing workflows (data and control-flow) using Petri nets for example the NDVI workflow.
2. They send a request to the *Cloud Broker* via the *Cloud Portal*.
3. The *Cloud Broker* checks for available Cloud providers, which provide geoprocessing tools (Grass GIS). This information is retrieved from the *Cloud Repository*.
4. The *Cloud Broker* sends a list to the *Workflow Holder* (through the Cloud Portal) to accept or to reject the offer.
5. If the offer is accepted, the *Workflow Holder* submits the workflow specification (.rnw + .sns) to the selected *Cloud Provider*.
6. Launch a customized Cloud instance with Renew and Grass GIS running in the background.
7. After simulation/execution of the workflow, results (in the prototype it consists of calculating the NDVI value) are transmitted to the *Workflow Holder* through the *Cloud Portal*.

Rejecting an offer does not conclude the execution process immediately. Since the list transmitted by the *Cloud Broker* is updated constantly, it might be that new Cloud providers are available and fits the requirements. Therefore, from step (3), the process is iterative until the satisfaction of the *Workflow Holder*. Regarding step (5) and (6), Renew supports starting a simulation from the command line. This is possible by using the command *startsimulation (net system) (primary net) [-i]*. The parameters to this command have the following meaning:

- *net system*: The .sns file.
- *primary net*: The name of the net, of which a net instance shall be opened when the simulation starts.
- -i: If you set this optional flag, then the simulation is initialized only, that is, the primary net instance is opened, but the simulation is not started automatically.

[5] http://52north.org/
[6] https://code.google.com/p/wps-grass-bridge/

## 13.5 Conclusion

While Part II addressed exclusively the relation between (Inter) -Cloud computing and workflow concepts this chapter adds the agent flavor to the whole system. How agents participate building complex workflows and their management is the question that is addressed within the previous sections. Following the Paose approach agents can offer a several advantages in term of process modeling, service composition and monitoring. The contributions presented here provides concepts and tools for Cloud developers to achieve tasks in the Cloud. Ctt for agents is a conceptual solution to encapsulate the whole process within a special transition. The latter takes care of the good execution of the process. It extends the original idea of Ctt introduced in Chapter 10. ICNets are described in Chapter 10 as Petri net-based models to enable interaction with multiple Clouds. Technically based on the ICWorkflow plug-in ICNets make it easy implementation through modeling of Cloud-based workflows. Moreover, an approach is proposed to deploy RenewGrass plug-in (see Chapter 8) into the Cloud. At the end of chapter the whole work is resumed in one concept called the Drop-Engine. The Drop-Engine is the name given to the all contributions presented in this thesis. It provides a system to manage processes in general in the Cloud. The management covers several phases of the life-cycle of a workflow: modeling, implementation, deployment and execution.

# Summary

This part introduced the concept of Drop-Engine. The Drop-Engine is a vision that consists of providing techniques and tools for the management of processes in-premise and on the Cloud. The principle features are workflow specification by Petri nets (ICNets, RenewGrass), deployment and execution by dedicated and implemented plug-ins (Chapter 9). Chapter 12 investigated the relation between agents and Clouds. It shows how Clouds and agents can benefit from each other. Section 12.3 presents an architecture that describes the process of managing workflows in Inter-Cloud environments. Chapter 13 explains how the contributions like: the Ctt, ICNets, ICWorkflow and RenewGrass could be deployed within an agent-based approaches and systems.

# Part IV

# Conclusion

This part summarizes the research work on managing workflows in an (Inter-) Cloud environment presented in this thesis and outlines the main contributions. Furthermore, it presents and discusses the open research questions and problems that still to be investigated and outlines a list of future research works.

# Chapter 14

# Summary and Outlook

This thesis addressed Cloud-based workflows from a conceptual and technical viewpoint. It was shown that workflow concepts coupled with agent paradigms are suitable to overcome many challenging obstacles that face distributed application development like heterogeneity, collaboration and coordination. The proposed approaches covered the application development life-cycle at different levels. This work provides concepts and techniques for the modeling of Cloud-based workflows as well as the execution of tasks in concrete testbeds. In the following, each Chapter is reflected and the obtained results are resumed again and discussed as well as the possibilities for future extensions/improvements are presented.

## 14.1 Summary

Chapter 1 provides an introduction to the topic. It describes the motivation behind the integration of (Inter-) Cloud computing, workflow management and multi-agent systems. The objectives of the thesis as well as the work road-map are outlined. It is mentioned that, despite the attractive features that the Cloud technology offers, much work still has to be devoted to the management, the control of processes and resources. The outcome of this work is in form of modeling techniques, methods, tools and architectural design, which support definition, deployment and monitoring of distributed workflows within Cloud environments.

The thesis is composed of three main parts: Foundations (Part I), workflow management in the Cloud (Part II) and the Drop-Engine (Part III). In Part I, the basic concepts, techniques and tools, which are covered by the thesis are discussed. This part deals with the main topics: SOC (Chapter 2) and Grid/Cloud computing (Chapter 3), process management and workflows (Chapter 4), workflow modeling (Chapter 5) and MAS/Paose (Chapter 6). Chapter 2 presents the notion of SOC and its paradigms, to provide the background of SOA principles, Web services and service composition. The reason to introduce SOA principles is related to the implementations provided by this thesis. For instance, ICWorkflow (see Chapter 11) uses Web services in order to read Cloud Web services. Chapter 3 is about Cloud computing. It answers questions like: What Cloud computing is? and How can Cloud computing be developed and deployed?

Existing Cloud standards and formal aspects of Cloud computing are outlined. Moreover, Inter-Cloud Computing is introduced and an overview about grid computing is given. It does not cover all the concepts but concentrates on the main ones: definitions, Grid resources and Grid topologies.

Chapter 4 concerns process management and workflows. It starts by defining the workflow concepts based mainly on the terminology provided by the WfMC: process, business process, business workflow and scientific workflow. Afterwards, the notion of workflow management system is tackled. Mainly, the workflow reference model from the WfMC is described.

Much work of the thesis is devoted to modeling techniques. Therefore, Chapter 5 deals exactly with that topic. It is composed of two main sections. First, an introduction is given to differentiate between script-like and graph-based workflow description languages. After that, some existing workflow specification languages are presented such as BPEL, BPMN and Petri nets. The latter are explained in details to provide the basic concepts of Petri nets as well as the advanced features of reference nets, which are the main modeling technique in this thesis.

Chapter 6 provides an overview about agents and multi-agent systems as well as Mulan/Capa/Paose. It describes the specifications of the Foundation for Intelligent Physical Agents (FIPA) concerning the agents management (Agent Management System (AMS), Directory Facilitator (DF) and the Message Transport System (MTS)) and the FIPA agent communication language (FIPA ACL). Since the Mulan/Capa framework is used throughout this thesis as the underlying conceptual and technical basis all relevant parts of the framework are described. Paose as the overall embedding approach its fundamental contributions are described. This provides the means to address the improvements made by this thesis. First, the four layers of Mulan are presented namely the infrastructure, the agent platform, Mulan agents and Mulan protocols. Capa is also described, which is an extension of the Mulan architecture. The second part of the chapter concerns the Paose approach and its six steps: Requirements analysis, coarse design, ontology implementation, role implementation, interaction implementation and integration. The notion of team organization is presented, which is a central concept of the Paose approach. Chapter 6 concludes Part I.

Part II covers the main contributions and investigates the integration of workflow concepts and Clouds. Chapter 7 discusses and presents different scenarios and approaches concerning moving the execution of processes to the Cloud. Based on the work of [Anstett et al., 2009, Han et al., 2010, Strauch et al., 2013] questions like: Where to store the data? Where to execute the activities? and Where to enact processes? are investigated. Furthermore, different execution scenarios, which are covered by different implementations are presented. Essentially, RenewGrass is considered as a use case. In Chapter 8 RenewGrass, a new plug-in developed throughout this thesis for Renew, is presented. It allows the development of image processing workflows based on the Grass GIS. The development signifies also modeling of those workflows mainly based on Petri nets and specifically reference nets. By RenewGrass, Renew is extended and is now able to support the processing of satellite imagery in terms of modeling and implementation. Chapter 9 presents techniques and tools for the

deployment and execution of Petri nets based workflows in remote resources, mainly in the Cloud. Technically, it concerns essentially the deployment of Renew on multiple virtual machines and to run processes modeled and implemented by Petri nets. Based on the insights gained by building the prototypes, it is investigated and explained how Renew simulations can be supported by Cloud instances. Different mechanisms are made available to automatically deploy and run Renew processes in the Cloud. The mechanisms concern the automatic configuration of Cloud instances, the transfer of processes to the Cloud instance and launching of Renew simulations. Concerning the infrastructure, a private Cloud has been deployed at the TGI group, based on the OpenStack framework. Based on the Vagrant tool[1] the provision of Cloud instances by Renew and related developed plug-ins is now possible. Chapter 10 presents two essential contributions, which are the Cloud Task Transition (Ctt) (see Section 10.2) and the Inter-Cloud Nets (ICNets) (see Section 10.3). On the one hand, the Ctt is a concept that consists of providing a new type of Petri net transitions. This kind of transitions is suitable for Cloud interactions. The idea is that Petri net based processes are enabled to invoke services provided by the Cloud without being confronted to the complexity of Cloud APIs. The introduced concepts cover both the conceptual as well as the technical aspect of building Cloud-based workflows. On the other hand, ICNets are pre-defined Petri net models that serve to specify Cloud-based processes as well as the management of Cloud-based workflows. The main objective of ICNets is to provide a separation between the model's element (places, transitions and arcs) in order to efficiently analyze the whole behavior as well as to detect faults quickly in the model. Chapter 11 presents the ICWorkflow plug-in. The latter is an important contribution of this dissertation since it represents the technical basis to many presented concepts and approaches. It provides means for modeling and executing workflow based on multi Cloud resources. ICWorkflow is composed of different components, which facilitate the management of the Cloud infrastructure as well as the applications. These components have several functionalities that cover user management, Cloud instance administration and most importantly the interoperability among Clouds. In order to improve the interoperability, ICWorkflow provides a layer that allows deploying and running processes on multiple Cloud platforms. This has benefit, like avoiding vendor lock-in or improving the scalability. ICWorkflow is the technical basis of ICNets. Tasks performed by ICNets or Ctt are implemented by functionality from the ICWorkflow plug-in. Mostly, these functionalities are added as Java inscriptions over the Petri net transitions.

Part III presents an extension to the previous contributions. While Part II deals with the integration between Clouds and workflows, Part III adds a supplementary layer to integrate agent concepts in the management system. This is in order to bring intelligence, autonomy and better communication in Cloud-like environments. Chapter 12 shows first how Clouds and agents as two computing paradigms can benefit from each other. A three layer architecture is proposed and described to provide more clarity on the objectives. Next, in Chapter 13 a concrete example of the integration between

[1] https://www.vagrantup.com/

Clouds, workflows and agents are presented. Concretely, it shows how Ctt, ICNets, ICWorkflow and RenewGrass can be applied and deployed in Cloud environment.

## 14.2 Future Directions

Different contributions covering workflow modeling, deployment and execution of workflows in (Inter-) Cloud like environments have been presented. Nevertheless, there are still open research issues that could be investigated more in order to improve the current work. In the following some of these issues are outlined.

### 14.2.1 Validation on Proprietary Cloud Providers

For now, all tests have been performed locally on a single Cloud platform based on OpenStack. Regarding the validation of the results, current work should be tested on different Cloud infrastructures. The ICWorkflow plug-in (see Chapter 11) already provides a supplementary layer to work with multiple Cloud providers. However, some refinements are required in order to get a fully working Inter-Cloud environment. For instance, the IC-Administration and IC-Instances need to be adapted to a specific Cloud provider. Deploying on different Cloud providers has many advantages and permits to evaluate the real performance of the applications on different platforms. Concretely, Renew simulations can be performed on multiple Cloud instances from different providers and can then be evaluated in terms of availability and responsiveness.

### 14.2.2 Extending the Use Cases and Building Large Applications

The use cases used in this thesis are mostly for testing purpose. This is due to the type of the Cloud infrastructure deployed at the TGI group. The All-in-one (AIO) built OpenStack Cloud does not allow for developing complex applications and is not devoted to production deployments. Applications that handle huge amount of data or necessitate high computing power require a realistic Cloud environment as can be provided by Amazon or Windows Azure. Thus, the features provided by ICWorkflow and RenewGrass can not be fully reflected. The NDVI and EVI prototypes (see Chapter 8) for instance, which are implemented by RenewGrass are considered as proof of concept only. More complex workflows could be implemented and deployed in the Cloud. An example of complex (workflow) applications would be the processing of high-resolution satellite imagery provided by the NASA Mars Exploration Rovers (MER)[2]. For this kind of application, a large volume of data needs to be stored and processed across multiple machines. Hence RenewGrass and ICWorkflow are suitable to the development of those complex applications. RenewGrass provides means for modeling and implementing image processing workflows by Petri nets, whereas ICWorkflow offers among others mechanisms to configure Cloud instances and run applications on different Cloud providers.

[2] https://aws.amazon.com/swf/testimonials/swfnasa/

### 14.2.3 Integration within Paose Approach

Integrating the results presented in this thesis within a concrete agent-based approach is interesting but also a challenging task. A good opportunity would be to integrate contributions like ICNets, Ctt, RenewGrass and ICWorkflow directly (in modeling frameworks) within the Paose approach and use them e.g. in the AOSE teaching project. The idea is to develop Cloud-based applications following an agent metaphor. A team of students (developers) work together to implement different kinds of Cloud-based applications. As an application candidate the architecture proposed in Chapter 13 (Section 13.4.2) could be implemented following the Paose approach and and Mulan/Capa framework. The architecture is composed of different components performing different roles: Cloud brokering, Cloud storage and image processing. These components can be considered as agents, which have to communicate with each other in order to achieve a certain goal. By providing the results from this thesis, the development process of Cloud-based application is more easier. Students are not obligated to worry about issues concerning the Cloud infrastructure or the configuration of Cloud instances.

### 14.2.4 Security and Privacy

Security of Cloud-based applications is a major concern. Cloud customers have fear to expose their sensitive tasks and the corresponding data. The works in Chapter 8 and Chapter 9 address the issue, but do not provide sufficient means for securing the data. RenewGrass handles in general satellite imagery, which are sensitive. Additional mechanisms should be developed in order to ensure the security of the data transfered to the Cloud. For now, the idea is to exploit the work from Chapter 9, which provides means to move Renew nets to Cloud instances in order to be simulated/executed. Unfortunately, the transfer of data is performed separately and not during the creation of the instance. This means that this activity is left for the user. To overcome this, an additional layer could be added to manage the security of the data. One solution is investigated and consists of integrating additional authentication middleware to NodeJS like PassportJS[3].

By the support of Inter-Cloud computing through the solutions presented in this thesis each participant can encapsulate relevant parts of the application. When interacting with others, complete separation is hardly reachable. The concepts of autonomous agents is helpful to support the autonomy of local Cloud management systems and hence the autonomous process management. Nevertheless further improvements with respect to security and privacy are necessary and possible.

### 14.2.5 Integration between Agents and Workflows

The extensions for the Petri net modeling technique allow the direct support of Cloud-based workflow tools. In the forthcoming thesis of Wagner [Wagner, 2016] the tight integration of agent and workflows concepts are addressed. This will provide a major

[3] http://passportjs.org/

improvement for this thesis. The supporting modeling techniques and the tool support will ease the currently very complex handling of the agent/workflow integration in Inter-Cloud environments, since it covers especially inter-organizational process management. At this time it will profit from the results of this thesis by using Cloud technology in the Paose context.

# Acronyms

**Capa** Concurrent Agent Platform Architecture. 25

**Mulan** Multi-Agent Nets. 22

**Paose** Petri net-based, Agent- and Organization-oriented Software Engineering. 22

**Renew** Reference Nets Workshop. 22

**Rest** Representational State Transfer. 37

**Amazon EC2** Amazon Elastic Compute Cloud. 61

**BPEL** Business Process Execution Language. 40

**CCUCDG** Cloud Computing Use Case Discussion Group. 58

**CORBA** Common Object Request Broker Architecture. 38

**CPU** Central Processing Unit. 47

**CTT** Cloud Task Transition. 135, 175, 183

**DE** Drop-Engine. 26

**DMTF** Distributed Management Task Force. 55

**FIPA** Foundation for Intelligent Physical Agents. 25

**HTTP** Hypertext Transfer Protocol. 37

**IaaS** Infrastructure as a Service. 60

**ICWPN** Inter-Cloud Workflow Petri Nets. 183

**MAS** Multi-Agent Systems. 22

**MIME** Multipurpose Internet Mail Extensions. 37

**NIST** National Institute of Standards and Technology. 51

**PaaS** Platform as a Service. 60

**PHP** Hypertext Preprocessor. 38

**QoS** Quality of Service. 52

**RMI** Remote Method Invocation. 38

**SaaS** Software as a Service. 60

**SLA** Service Level Agreement. 52

**SOAP** Simple Object Access Protocol. 36

**SOC** Service Oriented Computing. 21, 33

**SWF** Simple Workflow Service. 94

**TGI** Theoretische Grundlagen der Informatik. 22

**UDDI** Universal Description, Discovery and Integration. 37

**UML** Unified Modeling Language. 25

**URIs** Unified Resource Identifiers. 37

**WfMSs** Workflow Management Systems. 22

**WSDL** Web Services Description Language. 37

**XML** Extensible Markup Language. 35, 36

# Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg ———————— Unterschrift ————————

# Part V

# Appendix

# Appendix A

# INSTALL GRASS 6.4 from SOURCE

Appendix A lists the required steps to compile the Grass GIS (version 6) from source. Installing from the binaries is not sufficient to use the plug-in properly, this comes from the fact that the wrapper will not be able to access the *bin* directory, which contains all the Grass GIS modules. It also explains how to set up the required variables as well as how to organize the data in a Grass database.

## A.1 Compile and Install

**Assumption**: the following steps are for Ubuntu systems. First one needs to download a copy of the source code. For this, svn (a revision control system) needs to be installed.

```
$ sudo apt-get install svn
```

After a successful install of svn run:

```
$ sudo checkout https://svn.osgeo.org/grass/grass/branches/
  release\\branch_6_4 grass64_release
```

After downloading the code, it should be compiled. There are many dependencies, which are necessary for Grass compilation for example: PROJ.4, GEOS and GDAL.

Grass needs to be configured before running (make and make install) by executing the *configure* command. This generates a file with all the variables (can/should be adjusted according to specific needs) that they will be used in the compilation phase.

```
CFLAGS="-O2 -Wall" LDFLAGS="-s" ./configure \
--enable-largefile=yes \
--with-nls \
--with-cxx \
--with-proj-share=/usr/local/share/proj/ \
--with-geos=/usr/lo\\cal/bin/geos-config \
--with-readline \
--with-python=yes \
--with-wxwidgets \
--with-cairo \
--with-opengl-libs=/usr/include/GL \
--with-motif \
```

```
--with-tcltk-includes="/us\\r/include/tcl8.5" \
--with-ffmpeg=yes --with-ffmpeg-includes="/usr/include/li
bavcodec /usr/include/libavformat /usr/include/libswscale
 /usr/include/libavutil" \
--with-freetype=yes --with-freetype-includes="/usr/includ
e/freetype2/" \
--with-postgres=yes --with-postgres-includes="/usr/includ
e/postgresql" \
--with-sqlite=yes \
--with-mysql=yes --with-mysql-includes="/usr/include/mys
ql" \
--with-odbc=no
```

After ./configure ends successfully run:

```
$ make

$ make install
```

To run Grass (wich is not necessary, since it will be called directly from the nets) simply use this command *grass64*.

## A.2 Sample Data and Project Structure

Grass GIS has a location struction which needs to be respected. A LOCATION is simply a set of directories which contains the GRASS data of a project [1] . To create the GRASS database with sample data:

1. Find a place on your disk where you have write access and that has enough disk space to hold your spatial data.
2. Create a subdirectory that will hold the general GRASS database (/data/grassdata or /home/yourlogin/grassdata).

## A.3 Set up the Variables

The path of the Grass installation should mentioned as well as the sample data. Otherwise, Renew can neither call the Grass commands nor to display information about the images. Therefore, two variables need to be set in the plugin.cfg and more better in the *.renew.properties* at the home directory (or in config file in dist directory of Renew) as follows:

*de.renew.renewgrass.gisbase="PathToGrassRelease".*

*de.renew.renewgrass.mapset="PathTosample data".*

These variables are necessary for the classes ModuleSupporter.java, GrassRunner.java and GrassModuleRunnerWithScript.java. The complete documentation for RenewGrass can be found on `www.paose.net`.

[1] `http://grass.osgeo.org/grass64/manuals/helptext.html`.

# Appendix B

# Integrate JClouds in Renew

The *jclouds* toolkit is integrated successfully in Renew and provides interfaces to interact with different cloud providers. Appendix B lists the steps to integrate *jclouds* in Renew. The installation is performed using *Ant*, since it is the building tool used in Renew. The *jclouds* is essential for the ICWorkflow plug-in discussed in chapter 11. First, the structure of the ICWorkflow is presented in order to understand building process. Next, this appendix explains how to make the *jclouds* jars recognized by Renew.

## B.1 The ICWorkflow Plug-in

The ICWorkflow is a plug-in for Renew. All the operations that require cloud services are performed through this plug-in. These operations can be storage, compute or cloud instance management. Figure. B.1 and Figure. B.2 show the structure of the plug-in.

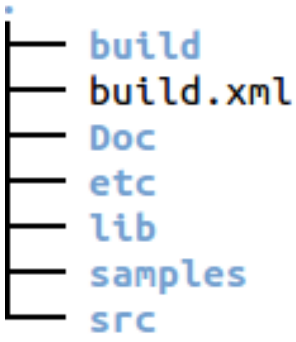



Figure B.1: The Structure of the Workflow Plug-in

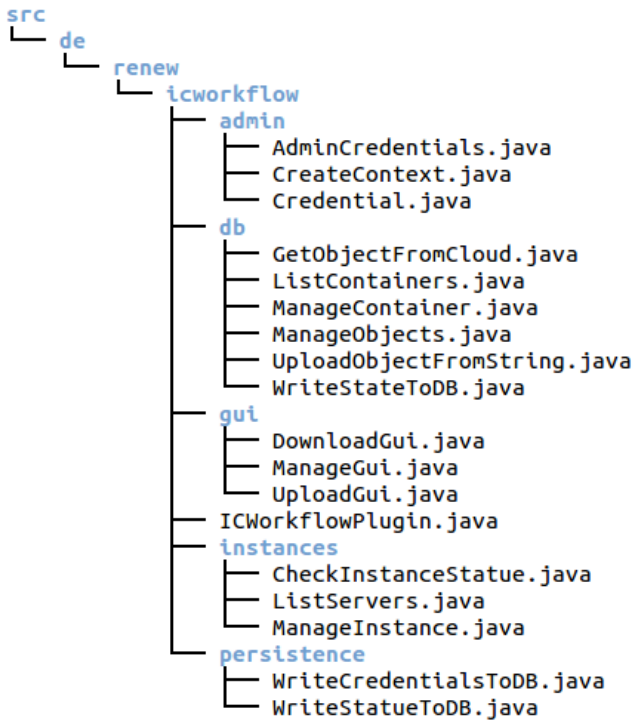



Figure B.2: The Structure of the Source Directory

The *lib* directory contain the downloaded *jclouds* jars as well as the *mongodb*[1]. These jars will be copied to the plug-in directory in Renew.

## B.2 How to Get the JClouds Jars

The ICWorkflow plug-in needs a lot of dependencies to work. Even if the download of these dependencies can be in priori configured. In the following, it is explained how to download/include all the jclouds library. In the Build.xml of the Workflow plug-in (B.2), there are three important targets:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<project name="ICWorkflow" default="dist" basedir="." xmlns:artifact="
    urn:maven-artifact-ant">



    <target name="dist"
        depends="init, compile, jar, lib, copytodist.dir, copytodist.jar"
        description="Builds the ICWorkflow plug-in completely. Prepend '
            clean' to build from scratch. (default)">
    </target>
```

[1]The mongodb is required for NodeJS web application. In this thesis, the mongodb is used to store the cloud provider status.

```
    <target name="lib"
   depends="init,computeFilenames"
   description="copies libraries into the plugin directory.">
   <mkdir dir="${dir.renew.dist}"/>
   <copy todir="${dir.dist.plugin}/lib">
   <fileset dir="${dir.lib}">
   <include name="**/*.jar"/>
   </fileset>
   </copy>
    </target>

<!-- Set property value depends on check result -->
     <target name="check-dir">
      <available property="p" file="lib" type="dir"/>
       </target>
     <!-- Create dir 'lib' if doesn't exist -->
     <target name="create-lib-dir" depends="check-dir" unless="p">
       <echo message="Create a Directory for the JClouds jars"/>
       <mkdir dir="lib"/>
     </target>


  <target name="mongodb" depends="create-lib-dir"
  description="Download the mongodb jars and copy them to the /lib
      directory">

     <get
        src="https://github.com/downloads/mongodb/mongo-java-driver/
           mongo-2.10.1.jar"
        dest="lib"/>

  </target>
  <target name="jcloud" depends="initmvn,create-lib-dir"
  description="Download all the jclouds jars and copy them to the /lib
      directory">
  <artifact:dependencies filesetId="jclouds.fileset" versionsId="
      dependency.versions">
  <dependency groupId="org.apache.jclouds" artifactId="jclouds-all"
      version="1.8.0" />
   <dependency groupId="org.apache.jclouds.driver" artifactId="jclouds-
       slf4j" version="1.8.0" />
   <dependency groupId="ch.qos.logback" artifactId="logback-classic"
       version="[1.0.9,)" />
   <dependency groupId="org.apache.jclouds.labs" artifactId="openstack-
       swift" version="1.8.0">
   </dependency>
   </artifact:dependencies>
   <copy todir="lib" verbose="true">
   <fileset refid="jclouds.fileset"/>
   <mapper type="flatten" />
   </copy>
   </target>
   <get src="http://search.maven.org/remotecontent?filepath=org/apache/
       maven/maven-ant-tasks/2.1.3/maven-ant-tasks-2.1.3.jar"
```

```
                                dest="maven-ant-tasks.jar"/>

    </project>
```

- ICWorkflow: compiles the ICWorkflow example.
- jcloud: the role of this target is to download the jars and copy them to the lib directory of the workflow plug-in.
- lib: copy the downloaded jars to the generated jar in the plug-ins directory.

There is also another target called *create-lib-dir*, the role of this target is to check whether the jars already exist or not. There are two situations:

If the ICWorkflow is used for the first time (no jars and no lib directory at all), compile only with the ICWorkflow target is not enough and leads to many errors. In this case, the plug-in should be compiled by running ant jcloud and then ant lib.

If the ICWorkflow is already built, there is no need to compile it again with the jcloud and lib targets. Ant dist is enough.

# Appendix C

# Configure VMs and Run Renew with Vagrant

In this appendix important steps to deploy and run Petri net-based workflows in the Cloud. There are in general four steps:

1. create custom virtual machine
2. configure log-in credentials
3. upload required nets
4. run simulations

## C.1 Configure the instance

With Vagrant, configuration is performed with the *Vagrantfile*. Configuring the VM is composed of two requirements are: Renew and Java 1.6. It might be that other software are required but it depends on the characteristics of the VM.

Figure. C.1 presents the process of creating a customized Vagrant machine. For creating a Vagrant machine the vagrant tool first needs to be installed as well as the VirtualBox. For both Vagrant and VirtualBox, the installation is possible on the three famous operating systems: Linux, Windows and Mac. Next, a configuration file called *Vagrantfile* is mandatory to configure a Vagrant machine. It is a Ruby file used to configure Vagrant and to describe virtual machines required for a project as well as how to configure and provision these machines. Finally, the guest Vagrant host can be started using the command *vagrant up*. Since Renew requires Java 6 or later, this portion of code shows instructions that should be added in the Vagrantfile.

```
#Install Java 7 for 64bit machines
config.vm.provision :shell, inline: 'wget --no-check-certificate https
    ://githu$


#Download Renew and extract the files
```

```
  config.vm.provision :shell, inline: 'sudo apt-get install unzip && wget
      http:$

  #Delete the zip of Renew
  config.vm.provision :shell, inline: 'sudo rm renew2.4.1base.zip'
```

If users want to work with OpenStack, there is another portion of code that needs to be added. The most important information consist of credentials, image and ssh keys. The following code shows an example of configuration.

```
    config.ssh.private_key_path = '~/.ssh/id_rsa'


    config.vm.provider :openstack do |os|
      os.openstack_auth_url = 'http://xxx.xxx.xxx.x:5000/v2.0/tokens'
      os.username           = 'xxxxxxxxxxxx'
      os.password           = 'xxxxxxxxxxxx'
      os.tenant_name        = 'xxxxxxxxxxxxxx'
      os.flavor             = /m1.small/
      os.image              = /ubuntu14.04-LTS/
      os.floating_ip    = 'xxx.xxx.xxx.xxx'
      os.server_name = 'RenewCloud'
      os.security_groups =  ['default']
      os.keypair_name = 'keypair'
     config.ssh.private_key_path = '~/.ssh/id_rsa'

    end
```

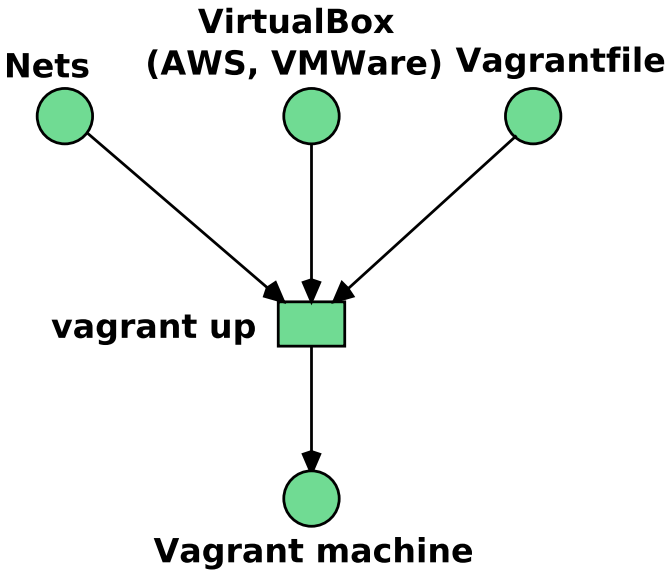



Figure C.1: Configuration of Vagrant Machine

## C.2 Run Simulations

To run simulation on the created instance there is a special script. In the same directory, where the Vagrantfile is created, a (NodeJS) Web server is installed. To start the instance one needs to execute the command npm start. Before starting the simulation the nets that need to be simulated should copied to the instance. This can be performed by the following command:

```
$ scp net net.rnw instance_username@floating_ip_adress:
```

There is a file called ssh.txt, which used to configure authentation information.

```
Host xxx.xxx.xxx.xxx
 HostName xxx.xxx.xxx.xxx
 User ubuntu
 Port 22
 UserKnownHostsFile /dev/null
 StrictHostKeyChecking no
 PasswordAuthentication no
 IdentityFile ~/.ssh/id_dsa.pub
 IdentitiesOnly yes
 LogLevel FATAL
```

# Appendix D

# Create a Cloud Environment using PackStack

This appendix concerns the creation of an all-in-one[1] private Cloud environment at the TGI group. This has been realized by using an open-source utility called *PackStack*[2]. Choosing an All-in-one solution is due to obstacles mentioned in Chapter 11. The private OpenStack Cloud is useful for "proof-of-concepts!" prototypes implemented in Chapter 9. The environment is created with CentOS 6.5 as operating system, RDO Havana and PackStack tool. Some of the steps are similar to those in TryStack (see Appendix E). The first step is to set set up SSH, because it will be used to access the Cloud environment.

1. Make RDO repositories available

```
$ yum install -y http://rdo.fedorapeople.org/rdo-release.rpm
```

2. Install PackStack

```
$ yum -y install openstack-packstack
```

3. Generate an answer file for configuration

```
$ packstack --gen-answer-file=FILE
```

4. Run PackStack

```
$ packstack --answer-file packstack-answers.txt
```

[1] All-in-one means that one machine is used at the same time as controller, compute and network node.

[2] https://wiki.openstack.org/wiki/Packstack

```
Welcome to Installer setup utility
Packstack changed given value  to required value /root/.ssh/id\_rsa
    .pub

Installing:
Clean Up...                                             [ DONE ]
Setting up ssh keys...                                  [ DONE ]
Discovering hosts' details...                           [ DONE ]
Adding pre install manifest entries...                  [ DONE ]
Adding MySQL manifest entries...                        [ DONE ]
Adding QPID manifest entries...                         [ DONE ]
Adding Keystone manifest entries...                     [ DONE ]
Adding Glance Keystone manifest entries...              [ DONE ]
Adding Glance manifest entries...                       [ DONE ]
Installing dependencies for Cinder...                   [ DONE ]
Adding Cinder Keystone manifest entries...              [ DONE ]
.
```

5. Post-Install Configuration

   (a) Fix Horizon (ALLOWED_HOSTS)

```
$ sed -i '/^ALLOWED_HOSTS/ s/=.*/= [ "*" ]/' \
  /etc/openstack-dashboard/local_settings
$ service httpd restart.
```

   (b) Source admin credentials (for management tasks)

```
$ . /root/keystonerc_admin
```

   (c) Create disk image

```
glance image-create \
  --copy-from http://download.cirros-cloud.net/0.3.1/cirros
      -0.3.1-x86_64-disk.img \
  --is-public true \
  --container-format bare \
  --disk-format qcow2 \
  --name cirros
```

   (d) Create external network

```
$ neutron net-create external --router:external=True
$ neutron subnet-create --disable-dhcp external 192.168.37.0/24
```

   (e) Create a flavor for testing. This flavor consumes minimal memory and disk so it is better than the default flavors for testing in constrained environments.

```
$ nova flavor-create m1.nano auto 128 1 1
```

6. Create non-admin users

```
$ keystone tenant-create --name demo
$ keystone user-create --name demo --tenant demo --pass demo
```

and store the credentials in /root/keystonerc_demo

```
export OS_USERNAME=demo
export OS_TENANT_NAME=demo
export OS_PASSWORD=demo
export OS_AUTH_URL=http://identity\_IP/v2.0/
export PS1='[\u@\h \W(keystone_demo)]\$ '
```

# Appendix E

# Configure Instances with TryStack

In this appendix the whole process of creating, configuring and managing an instance in TryStack is described. It concerns Chapter 9 which address the usability of Cloud resources for the execution of workflows. It is noteworthy that TryStack is only for testing the prototypes developed in the mentioned chapter. These steps are essential in order to make the instances support Renew simulations. All the following is performed using the dashboard provided by TryStack. Before going further users must be first registered.

## E.1 Create an Internal Network

These are the steps to create an internal network:

1. Go to Network > Networks and then click Create Network.
2. In Network tab, fill Network Name for example internal and then click Next.
3. In Subnet tab,
   (a) Fill Network Address with appropriate CIDR, for example 192.168.37.0/24.
   (b) Select IP Version with appropriate IP version, in this case IPv4.
   (c) Click Next.
4. In Subnet Details tab, fill DNS Name Servers with 8.8.8.8 (Google DNS) and then click Create.

## E.2 Create an Instance

1. Go to Compute > Instances and then click Launch Instance.
2. In Details tab,

(a) Fill Instance Name, for example Ubuntu 1.

(b) Select Flavor, for example m1.medium.

(c) Fill Instance Count with 1.

(d) Select Instance Boot Source with Boot from Image.

(e) Select Image Name with Ubuntu 14.04 amd64 (243.7 MB) if you want install Ubuntu 14.04 in your virtual machine.

3. In Access & Security tab,

   (a) Click [+] button of Key Pair to import key pair.

   (b) In Import Key Pair dialog,

      i. Fill Key Pair Name with your machine name (for example the content of id_rsa.pub in /.ssh).

      ii. Fill Public Key with your SSH public key. See description in Import Key Pair dialog box for more information. If you are using Windows, you can use Puttygen to generate key pair.

      iii. Click Import key pair.

   (c) In Security Groups, mark/check default.

4. In Networking tab (In Selected Networks, select network that have been created in Step 1, for example internal).

5. Click Launch.

## E.3 Create a Router

1. Go to Network > Routers and then click Create Router.

2. Fill Router Name for example router1 and then click Create router.

3. Click on your router name link, for example router1, Router Details page.

4. Click Set Gateway button in upper right:

   (a) Select External networks with external.

   (b) Then OK.

5. Click Add Interface button.

   (a) Select Subnet with the network that you have been created in Step 1.

   (b) Click Add interface.

6. To see the network topology go to Network > Network Topology.

## E.4 Configure Floating IP Address

1. Go to Compute > Instance.
2. In one of your instances, click More > Associate Floating IP.
3. In IP Address, click Plus [+].
4. Select Pool to external and then click Allocate IP.
5. Click Associate.
6. Now you will get an public IP, e.g. 128.136.179.125, for your instance. Repeat step 1-5, if you want setup floating IP to other instances.

## E.5 Configure Access & Security

1. Go to Compute > Access & Security and then open Security Groups tab.
2. In default row, click Manage Rules.
3. Click Add Rule, choose ALL ICMP rule to enable ping into your instance, and then click Add.
4. Click Add Rule, choose HTTP rule to open HTTP port (port 80), and then click Add.
5. Click Add Rule, choose SSH rule to open SSH port (port 22), and then click Add.
6. You can open other ports by creating new rules.

## E.6 Access the Instance

To access the instance one should ssh it. All the information resulted from the above steps are necessary when configuring the instance with Renew and other required softwares.

# Bibliography

[Aalst, 1999] Aalst, W. (1999). Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367.

[Alarcón et al., 2010] Alarcón, R., Wilde, E., and Bellido, J. (2010). Hypermedia-driven restful service composition. In Maximilien, E. M., Rossi, G., Yuan, S., Ludwig, H., and Fantinato, M., editors, *Service-Oriented Computing - ICSOC 2010 International Workshops, PAASC, WESOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers*, volume 6568 of *Lecture Notes in Computer Science*, pages 111–120.

[Alt et al., 2006] Alt, M., Gorlatch, S., Hoheisel, A., and Pohl, H.-W. (2006). A grid workflow language using high-level Petri nets. In Wyrzykowski, R., Dongarra, J., Meyer, N., and Wasniewski, J., editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *LNCS*, pages 715–722. Springer-Verlag.

[Altintas et al., 2004] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. In *16th International Conference on Scientific and Statistical Database Management, 2004. Proceedings.*, pages 423–424.

[Amazon, 2012] Amazon (2012). Amazon simple workflow service (swf). `https://aws.amazon.com/swf/`. Accessed: 2016-5-20.

[Anstett et al., 2009] Anstett, T., Leymann, F., Mietzner, R., and Strauch, S. (2009). Towards BPEL in the cloud: Exploiting different delivery models for the execution of business processes. In *Proceedings of the International Workshop on Cloud Services (IWCS 2009) in conjunction with the 7th IEEE International Conference on Web Services (ICWS 2009), Los Angeles, CA, USA, July 10, 2009*, Artikel in Tagungsband, pages 670–677. IEEE.

[Ardagna et al., 2012] Ardagna, D., Di Nitto, E., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.-S., Petcu, D., Gericke, A., and Sheridan, C. (2012). MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 50–56.

[Arjuna Agility, 2014] Arjuna Agility (2014). URL: http://www.arjuna.com/what-is-federation, [last accessed 04 June 2014].

[Armstrong, 2009] Armstrong, K. (2009). Modelbuilder: An introduction. In *Esri Southeast Regional User Group Conference, Held in Jacksonville (USA)*. Available at http://proceedings.esri.com/library/userconf/serug09/papers/tw/modelbuilder_introduction.pdf.

[Aversa et al., 2010] Aversa, R., Martino, B. D., Rak, M., and Venticinque, S. (2010). Cloud agency: A mobile agent based cloud system. In Barolli, L., Xhafa, F., Vitabile, S., and Hsu, H.-H., editors, *CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 15-18 February 2010*, pages 132–137. IEEE Computer Society.

[Barkhordarian et al., 2012] Barkhordarian, A., Demuth, F., Hamann, K., Hoang, M., Weichler, S., and Zaplata, S. (2012). Migratability of BPMN, 2.0 process instances. In Pallis, G., Jmaiel, M., Charfi, A., Graupner, S., Karabulut, Y., Guinea, S., Rosenberg, F., Sheng, Q., Pautasso, C., and Ben Mokhtar, S., editors, *Service-Oriented Computing - ICSOC 2011 Workshops*, volume 7221 of *Lecture Notes in Computer Science*, pages 66–75. Springer Berlin Heidelberg.

[Behrendt, 2014] Behrendt, M. (2014). IBM cloud computing reference architecture 4.0. Technical report, IBM.

[Bell et al., 2009] Bell, G., Hey, T., and Szalay, A. (2009). Beyond the data deluge. *Science*, 323(5919):1297–1298.

[Bendoukha et al., 2012a] Bendoukha, H., slimani, Y., and Benyettou, A. (2012a). Jasmin: A visual framework for managing applications in service-oriented grid systems. In *ICIW 2012, The seventh International Conference on Internet and Web Applications and Services*, pages 46–51. ThinkMind.

[Bendoukha et al., 2012b] Bendoukha, H., Slimani, Y., and Benyettou, A. (2012b). UML refinement for mapping UML activity diagrams into bpel specifications to compose service-oriented workflows. In Benlamri, R., editor, *NDT (2)*, volume 294 of *Communications in Computer and Information Science*, pages 537–548. Springer.

[Bendoukha, 2014] Bendoukha, S. (2014). Multi-agent approach for managing workflows in an inter-cloud environment. In Lomuscio, A., Nepal, S., Patrizi, F., Benatallah, B., and Brandić, I., editors, *Service-Oriented Computing – ICSOC 2013 Workshops*, volume 8377 of *Lecture Notes in Computer Science*, pages 535–542. Springer-Verlag.

[Bendoukha et al., 2015a] Bendoukha, S., Bendoukha, H., and Moldt, D. (2015a). ICNETS: Towards designing inter-cloud workflow management systems by Petri nets. In Barjis, J., Pergl, R., and Babkin, E., editors, *Enterprise and Organizational Modeling and Simulation*, volume 231 of *Lecture Notes in Business Information Processing*, pages 187–198. Springer International Publishing.

[Bendoukha and Cabac, 2013] Bendoukha, S. and Cabac, L. (2013). Cloud transition for QoS modeling of inter-organizational workflows. In Moldt, D., editor, *Modeling and Buisness Environments MODBE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 355–356. CEUR-WS.org.

[Bendoukha et al., 2015b] Bendoukha, S., Moldt, D., and Bendoukha, H. (2015b). Building cloud-based scientific workflows made easy: A remote sensing application. In Marcus, A., editor, *4th International Conference on Design, User Experience and Usability*, volume 9187 of *Lecture Notes in Computer Science*, pages 277–288. Springer-Verlag.

[Bendoukha et al., 2013] Bendoukha, S., Moldt, D., and Wagner, T. (2013). Enabling cooperation in an inter-cloud environment: An agent-based approach. In Morvan, F., Tjoa, A. M., and Wagner, R., editors, *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on Cloud Computing, Models and Services.*, pages 217–221.

[Bendoukha and Wagner, 2012] Bendoukha, S. and Wagner, T. (2012). Cloud transition: Integrating cloud calls into workflow Petri nets. In [Cabac et al., 2012], pages 215–216.

[Bendoukha and Wagner, 2015] Bendoukha, S. and Wagner, T. (2015). Improving performance of complex workflows: Investigating moving net execution to the cloud. In Moldt, D., Rölke, H., and Störrle, H., editors, *Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings*, volume 1372 of *CEUR Workshop Proceedings*, pages 171–189. CEUR-WS.org.

[Bergmayr et al., 2014] Bergmayr, A., Troya, J., Neubauer, P., Wimmer, M., and Kappel, G. (2014). UML-based cloud application modeling with libraries, profiles, and templates. In Paige, R. F., Cabot, J., Brambilla, M., Rose, L. M., and Hill, J. H., editors, *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, CloudMDE@MoDELS 2014, Valencia, Spain, September 30, 2014.*, volume 1242 of *CEUR Workshop Proceedings*, pages 56–65. CEUR-WS.org.

[Bernardinello and De Cindio, 1992] Bernardinello, L. and De Cindio, F. (1992). A survey of basic net models and modular net classes. In Rozenberg, G., editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351. Springer Berlin Heidelberg.

[Bernstein et al., 2009] Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., and Morrow, M. (2009). Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In Perry, M., Sasaki, H., Ehmann, M., Bellot, G. O., and Dini, O., editors, *ICIW*, pages 328–336. IEEE Computer Society.

[Betz et al., 2013] Betz, T., Cabac, L., Duvigneau, M., Wagner, T., and Wester-Ebbinghaus, M. (2013). Integrating Web services in Petri net-based agent applications. In Moldt, D. and Rölke, H., editors, *Petri Nets and Software Engineering.*

*International Workshop PNSE'13, Milano, Italia, June 2013. Proceedings*, volume 989 of *CEUR Workshop Proceedings*, pages 97–116. CEUR-WS.org.

[Betz et al., 2014] Betz, T., Cabac, L., Duvigneau, M., Wagner, T., and Wester-Ebbinghaus, M. (2014). Software engineering with petri nets: A web service and agent perspective. *T. Petri Nets and Other Models of Concurrency*, 9:41–61.

[Betz et al., 2011] Betz, T., Cabac, L., and Wester-Ebbinghaus, M. (2011). Gateway architecture for Web-based agent services. In Klügl, F. and Ossowski, S., editors, *Multiagent System Technologies*, volume 6973 of *Lecture Notes in Computer Science*, pages 165–172. Springer Berlin / Heidelberg.

[Binz et al., 2013] Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). Opentosca - A runtime for tosca-based cloud applications. In Basu, S., Pautasso, C., Zhang, L., and Fu, X., editors, *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*, volume 8274 of *Lecture Notes in Computer Science*, pages 692–695. Springer.

[Booth et al., 2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web Services Architecture, W3C Working Group Note. *World Wide Web Consortium, article available from: http://www. w3. org/TR/ws-arch*.

[Braubach et al., 2011] Braubach, L., Pokahr, A., and Jander, K. (2011). JadexCloud - an infrastructure for enterprise cloud applications. In Franziska Klügl, S. O., editor, *In Proceedings of Eighth German conference on Multi-Agent System TEchnologieS (MATES-2011)*, pages 3–15. Springer.

[Buijs et al., 2012] Buijs, J., van Dongen, B. F., and van der Aalst, W. (2012). Towards cross-organizational process mining in collections of process models and their executions. In Daniel, F., Barkaoui, K., and Dustdar, S., editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 2–13. Springer Berlin Heidelberg.

[Buyya et al., 2013] Buyya, R., Christian, V., and Selvi, S. T. (2013). *Mastering Cloud Computing: Foundations and Applications Programming*, chapter Introduction. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[Buyya et al., 2010] Buyya, R., Ranjan, R., and Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing ICA3PP 2010*, Germany. Springer. LNCS.

[Buyya et al., 2008] Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5–13.

[Buyya et al., 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comput. Syst.*, 25(6):599–616.

[Cabac, 2007] Cabac, L. (2007). Multi-agent system: A guiding metaphor for the organization of software development projects. In Petta, P., editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 1–12, Leipzig, Germany. Springer-Verlag.

[Cabac, 2010] Cabac, L. (2010). *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin.

[Cabac et al., 2012] Cabac, L., Duvigneau, M., and Moldt, D., editors (2012). *Petri Nets and Software Engineering. International Workshop PNSE'12, Hamburg, Germany, June 2012. Proceedings*, volume 851 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Cabac et al., 2006] Cabac, L., Duvigneau, M., and Rölke, H. (2006). Net components revisited. In Moldt, D., editor, *Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA'06*, number FBI-HH-B-272/06 in Report of the Department of Informatics, pages 87–102, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany. University of Hamburg, Department of Informatics.

[Cabac et al., 2016] Cabac, L., Haustermann, M., and Mosteller, D. (2016). Renew 2.5 - towards a comprehensive integrated development environment for petri net-based applications. In Kordon, F. and Moldt, D., editors, *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, volume 9698 of *Lecture Notes in Computer Science*, pages 101–112. Springer-Verlag.

[Caire et al., 2008] Caire, G., Gotta, D., and Banzi, M. (2008). WADE: a software platform to develop mission critical applications exploiting agents and workflows. In *Proceedings of the 7th international joint conference on AAMS*, pages 29–36. International Foundation for Autonomous Agents and Multiagent Systems.

[Caliz et al., 2011] Caliz, E., Umapathy, K., Sánchez-Ruíz, A., and Elfayoumy, S. (2011). Analyzing web service choreography specifications using colored Petri nets. In Jain, H., Sinha, A., and Vitharana, P., editors, *Service-Oriented Perspectives in Design Science Research*, volume 6629 of *Lecture Notes in Computer Science*, pages 412–426. Springer Berlin Heidelberg.

[Cardinale et al., 2013] Cardinale, Y., El Haddad, J., Manouvrier, M., and Rukoz, M. (2013). Web service composition based on Petri nets: Review and contribution. In Lacroix, Z., Ruckhaus, E., and Vidal, M.-E., editors, *Resource Discovery*, volume 8194 of *Lecture Notes in Computer Science*, pages 83–122. Springer Berlin Heidelberg.

[Carl, 2004] Carl, T. (2004). Entwicklung eines agentenbasierten verteilten Workflow-Management-Systems mit Referenznetzen. Diploma thesis, University of Hamburg, Department of Computer Science.

[Celesti et al., 2010] Celesti, A., Tusa, F., Villari, M., and Puliafito, A. (2010). How to enhance cloud architectures to enable cross-federation. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 337–345.

[Chinnici et al., 2012] Chinnici, R., Moreau, J.-J., Ryman, C. A., and Weerawarana, S. (2012). Web Services Description Language Version 2.0 (wsdl). URL:`http://www.w3.org/TR/wsdl20/`.

[Christensen and Hansen, 1994] Christensen, S. and Hansen, N. D. (1994). Coloured Petri nets extended with channels for synchronous communication. In Valette, R., editor, *Lecture Notes in Computer Science; Application and Theory of Petri Nets 1994, Proceedings 15th International Conference, Zaragoza, Spain*, volume 815, pages 159–178. Springer-Verlag.

[CloudSigma, 2015] CloudSigma (2015). URL:`http://www.cloudsigma.com/` [last accessed 11.3.2015].

[Cohen, 2009] Cohen, R. (2009). What is an opencloud api? URL::http://www.elasticvapor.com/2009/09/what-is-opencloud-api.html.

[Dadam and Reichert, 2009] Dadam, P. and Reichert, M. (2009). The adept project: a decade of research and development for robust and flexible process support. *Computer Science - Research and Development*, 23(2):81–97.

[Decker et al., 2008] Decker, G., Lüders, A., Overdick, H., Schlichting, K., and Weske, M. (2008). RESTful Petri net execution. In Bruni, R. and Wolf, K., editors, *WS-FM*, volume 5387 of *Lecture Notes in Computer Science*, pages 73–87. Springer.

[Deelman et al., 2005] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., and Katz, D. S. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237.

[DeltaCloud, 2015] DeltaCloud (2015). URL:`https://deltacloud.apache.org/` [last accessed 11.3.2015].

[DMTF, 2009] DMTF (2009). Interoperable clouds-a white paper from the open cloud standards incubator 1.0. Technical report, Distributed management Task Force, Inc. URL: `http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf` (Last access: 05-03-2013).

[DMTF, 2010a] DMTF (2010a). Architecture for Managing Clouds-a white paper from the open cloud standards incubator 1.0. Technical report, Distributed management Task Force, Inc. URL: `http://dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf`. (Last access: 2013-05-03).

[DMTF, 2010b] DMTF (2010b). Use Cases and Interactions for Managing Clouds- a white paper from the open cloud standards incubator 1.0. Technical report, Distributed management Task Force, Inc. URL: `http://dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf`. (Last access: 2013-05-03).

[DropBox, 2012] DropBox (2012). URL:`https://www.dropbox.com`.

[Duvigneau, 2002] Duvigneau, M. (2002). Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Duvigneau, 2010] Duvigneau, M. (2010). *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen*, volume 4 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin.

[Duvigneau et al., 2003] Duvigneau, M., Moldt, D., and Rölke, H. (2003). Concurrent architecture for a multi-agent platform. In Giunchiglia, F., Odell, J., and Weiß, G., editors, *Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 59–72, Berlin Heidelberg New York. Springer-Verlag.

[Emmerich et al., 2005] Emmerich, W., Butchart, B., Chen, L., Wassermann, B., and Price, S. (2005). Grid service orchestration using the business process execution language (BPEL). *Journal of Grid Computing*, 3(3-4):283–304.

[Erl, 2005] Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Erl et al., 2012] Erl, T., Carlyle, B., Pautasso, C., and Balasubramanian, R. (2012). *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*. Prentice Hall.

[Fahringer et al., 2005a] Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., and Wieczorek, M. (2005a). Askalon: a grid application development and computing environment. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 10 pp.–.

[Fahringer et al., 2005b] Fahringer, T., Qin, J., and Hainzer, S. (2005b). Specification of grid workflow applications with AGWL: an abstract grid workflow language. In *CCGRID*, pages 676–685. IEEE Computer Society.

[Ferber, 1999] Ferber, J. (1999). *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*. Addison Wesley.

[Ferry et al., 2013] Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 887–894.

[Fielding, 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, IRVINE.

[FIPA, 2002a] FIPA (2002a). Fipa abstract architecture specification. available from URL:http://fipa.org/specs/fipa00001/.

[FIPA, 2002b] FIPA (2002b). FIPA communicative act library specification. available from URL:http://fipa.org/specs/fipa00037/.

[FIPA, 2002c] FIPA (2002c). FIPA message structure specification. available from URL:http://fipa.org/specs/fipa00061/.

[FIPA, 2002d] FIPA (2002d). FIPA sl content language specification. available from URL:http://www.fipa.org/specs/fipa00008/.

[FIPA, 2004] FIPA (2004). Fipa agent management specification. available from URL:http://www.fipa.org/specs/fipa00023/index.html.

[Foster, 2002] Foster, I. (2002). What is the Grid? - a three point checklist. *GRIDtoday*, 1(6).

[Foster and Kesselman, 2004] Foster, I. and Kesselman, C. (2004). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid. *International Journal of High Performance Computer Applications*, 15(3):200–222.

[Foster et al., 2008] Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10.

[Foundation, 2015] Foundation, O. (2015). Openstack installation guide for ubuntu 14.0. Technical report, OpenStack Foundation. http://docs.openstack.org/juno/install-guide/install/apt/openstack-install-guide-apt-juno.pdf.

[Francisco et al., 2006] Francisco, C., Krämer, B. J., and Mike, P. (2006). Service oriented computing (soc), 15.-18. november 2005. In Curbera, F., Krämer, B. J., and Papazoglou, M. P., editors, *Service Oriented Computing*, volume 05462 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[Franklin and Graesser, 1997] Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, UK. Springer-Verlag.

[GICTF, 2010] GICTF (2010). Use Cases and Functional Requirements for Inter-Cloud Computing. Technical report, Global Inter-Cloud Technology Forum.

[Griphyn, 2010] Griphyn (2010). Griphyn: Grid physics network project. URL:http://www.griphyn.org.

[Group, 2010] Group, C. C. U. C. D. (July 2010). Cloud Computing Use Cases-White Paper. Technical report, Cloud Computing Use Case Discussion Group. http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf.

[Grozev and Buyya, 2012a] Grozev, N. and Buyya, R. (2012a). Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey. *Software: Practice and Experience*.

[Grozev and Buyya, 2012b] Grozev, N. and Buyya, R. (2012b). Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*.

[Haddad, 2011] Haddad, C. (2011). Selecting a Cloud Platform : A Platform as a Service Scorecard. Technical report, WSO2. URL: http://wso2.com/download/wso2-whitepaper-selecting-a-cloud-platform.pdf[accessed December 2, 2014].

[Hamadi and Benatallah, 2003] Hamadi, R. and Benatallah, B. (2003). A Petri net-based model for Web service composition. In *Proceedings of the 14th Australasian Database Conference - Volume 17*, ADC '03, pages 191–200, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[Han et al., 2010] Han, Y.-B., Sun, J.-Y., Wang, G.-L., and Li, H.-F. (2010). A cloud-based bpm architecture with user-end distribution of non-compute-intensive activities and sensitive data. *Journal of Computer Science and Technology*, 25(6):1157–1167.

[Heitmann and Köhler-Bußmeier, 2011] Heitmann, F. and Köhler-Bußmeier, M. (2011). On defining conflict-freedom for object nets. In Muller, B. and Köhler-Bußmeier, M., editors, *Proceedings of the International Workshop on Logic, Agents, and Mobility (LAM 2011)*.

[Hoffa et al., 2008] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, G. B., and Good, J. (2008). On the use of cloud computing for scientific workflows. In *eScience*, pages 640–645. IEEE Computer Society.

[Hoheisel and Alt, 2007] Hoheisel, A. and Alt, M. (2007). Petri nets. In Taylor, I., Deelman, E., Gannon, D., and Shields, M., editors, *Workflows for e-Science Scientific Workflows for Grids*, pages 190–207. Springer London.

[Hollingsworth, 1995] Hollingsworth, D. (1995). Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition. Available online URL: http://www.wfmc.org/standards/model.htm.

[I. Foster, 2004] I. Foster, N. Jennings, C. K. (2004). Brain meets brawn: why grid and agents need each other. In *3rd International Conference on Autonomous Agents and Multi-Agent Systems*, pages 8–15.

[IBM, 2001] IBM (2001). Autonomic computing manifesto. URL: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[Jablonski Stefan (Hrsg.), 1997] Jablonski Stefan (Hrsg.), Böhm Markus (Hrsg.), S. W. H. (1997). *Workflow-Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie*. Heidelberg : dpunkt-Verlag,.

[Jacob et al., 2002] Jacob, T., Kummer, O., Moldt, D., and Ultes-Nitsche, U. (2002). Implementation of workflow systems using reference nets – security and operability aspects. In Jensen, K., editor, *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark. University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.

[Jander and Lamersdorf, 2013] Jander, K. and Lamersdorf, W. (2013). Jadex wfms: Distributed workflow management for private clouds. In *Networked Systems 2013*.

[JClouds, 2015] JClouds (2015). URL:https://jclouds.apache.org/ [last accessed 11.3.2015].

[Jennings, 2000] Jennings, N. R. (2000). On agent-based software engineering. *Artif. Intell.*, 117(2):277–296.

[Juric, 2006] Juric, M. B. (2006). *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing.

[Juve et al., 2009] Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B., and Maechling, P. (2009). Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59 –66.

[Keahey et al., 2009] Keahey, K., Tsugawa, M. O., Matsunaga, A. M., and Fortes, J. A. B. (2009). Sky computing. *IEEE Internet Computing*, 13(5):43–51.

[Keith and Burkhard, 2010] Keith, J. and Burkhard, N. (2010). The future of cloud computing, opportunities for the european cloud computing beyond. Technical report, European Commission. http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf.

[Kelly, 2007] Kelly, K. (2007). A cloudbook for the cloud. URL:http://www.kk.org/thetechnium/archives/2007/11/a_cloudbook_for.

[Kepler, 2012] Kepler (2012). The kepler project. URL:https://kepler-project.org/.

[Kesselman and Foster, 1998] Kesselman, C. and Foster, I. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

[Kleinrock, 2003] Kleinrock, L. (2003). An internet vision: The invisible global infrastructure. *Ad Hoc Networks*, 1(1):3–11.

[Koehler and Alonso, 2007] Koehler, J. and Alonso, G. (2007). Service-oriented computing - introduction to the special theme. *ERCIM News*, 2007(70).

[Köhler, 2004] Köhler, M. (2004). *Objektnetze: Definition und Eigenschaften*, volume 1 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin.

[Kummer, 2002] Kummer, O. (2002). *Referenznetze*. Logos Verlag, Berlin.

[Kummer et al., 2013] Kummer, O., Wienberg, F., Duvigneau, M., and Cabac, L. (2013). *Renew – User Guide (Release 2.4)*. University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg. Available at: `http://www.renew.de/`.

[Kuropka et al., 2006] Kuropka, D., Vossen, G., and Weske, M. (2006). Workflows in computation grids. In *GCC Workshops*, pages 296–301. IEEE Computer Society.

[Lawrence, 1997] Lawrence, P. (1997). *Workflow Handbook 1997*. John Wiley & Sons, Inc., New York, NY, USA.

[Lee Badger and Voas, 2011] Lee Badger, Tim Grance, R. P. and Voas, C. J. (2011). Recommendations of the national institute of standards and technology. Technical report, National Institute of Standards and Technology, Information Technology Laboratory. `http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf`.

[LibCloud, 2015] LibCloud (2015). URL:`https://libcloud.apache.org/` [last accessed 11.3.2015].

[Liu et al., 2011] Liu, F., Tong, J., Jian Mao, R. B., Messina, J., Badger, L., and Leaf, D. (2011). Nist cloud computing reference architecture. Technical report, National Institute of Standards and Technology, Information Technology Laboratory. `http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505`.

[Liu et al., 2010] Liu, X., Yuan, D., Zhang, G., Chen, J., and Yang, Y. (2010). Swindew-c: a peer-to-peer based cloud workflow system. *Handbook of Cloud Computing*, pages 309–332.

[Liu et al., 2012] Liu, X., Yuan, D., Zhang, G., Li, W., Cao, D., He, Q., Chen, J., and Yang, Y. (2012). Workflow systems in the cloud. In *The Design of Cloud Workflow Systems*, SpringerBriefs in Computer Science, pages 1–11. Springer-Verlag New York Inc.

[Loutas et al., 2010] Loutas, N., Peristeras, V., Bouras, T., Kamateri, E., Zeginis, D., and Tarabanis, K. (2010). Towards a reference architecture for semantically interoperable clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 143–150.

[Ludäscher et al., 2006] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. (2006). Scientific workflow management and the kepler system: Research articles. *Concurrency and Computation: Practice and Expererience*, 18(10):1039–1065.

[M. Spata, 2011a] M. Spata, S. R. (2011a). Virtual machine migration through an intelligent mobile agents system. *Journal of Convergence Information Technology*, 6(6).

[M. Spata, 2011b] M. Spata, S. R. (2011b). Virtual machine migration through an intelligent mobile agents system. *Journal of Convergence Information Technology*, 6(6).

[MacKenzie et al., 2006] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., and Metz, R. (2006). Reference model for service oriented architecture 1.0. available at URL:https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

[Marshak, 1994] Marshak, R. T. (1994). Workflow white paper - an overview of workflow software. In *Proceedings of the Workflow 1994*, pages 15–42, San Jose, US.

[Martino et al., 2011] Martino, B., Petcu, D., Cossu, R., Goncalves, P., Máhr, T., and Loichate, M. (2011). Building a mosaic of clouds. In Guarracino, M., Vivien, F., Träff, J., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Martino, B., and Alexander, M., editors, *Euro-Par 2010 Parallel Processing Workshops*, volume 6586 of *Lecture Notes in Computer Science*, pages 571–578. Springer Berlin Heidelberg.

[Martino et al., 2015] Martino, B. D., Cretella, G., and Esposito, A. (2015). *Cloud Portability and Interoperability - Issues and Current Trends*. Springer Briefs in Computer Science. Springer.

[MEDIA, 2008] MEDIA, S.-C. (2008). Twenty experts define cloud computing. URL:[http://cloudcomputing.sys-con.com/read/612375_p.htm.

[Mimoso, 2004] Mimoso, M. (2004). A defining moment for soa. Technical report, searchsoa.techtarget.com. Available online URL: http://searchsoa.techtarget.com/news/1017004/A-defining-moment-for-SOA.

[Moldt, 2006] Moldt, D. (2006). PAOSE: A way to develop distributed software systems based on Petri nets and agents. In Barjis, J., Ultes-Nitsche, U., and Augusto, J. C., editors, *Proceedings of The Fourth International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS'06), May 23-24, 2006 – Paphos, Cyprus 2006*, pages 1–2.

[Moldt et al., 2004] Moldt, D., Ortmann, J., and Offermann, S. (2004). A proposal for Petri net based web service application modeling. In Koch, N., Fraternali, P., and Wirsing, M., editors, *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, volume 3140 of *Lecture Notes in Computer Science*, pages 93–97. Springer-Verlag.

[Moldt and Rölke, 2003] Moldt, D. and Rölke, H. (2003). Pattern based workflow design using reference nets. In van der Aalst, W., Hofstede, A. t., and Weske, M., editors, *Proceedings of International Conference on Business Process Management, Eindhoven, NL*, volume 2678 of *Lecture Notes in Computer Science*, pages 246–260. Springer-Verlag.

[Nagavaram et al., 2011] Nagavaram, A., Agrawal, G., Freitas, M., Telu, K., Mehta, G., Mayani, R., and Deelman, E. (2011). A cloud-based dynamic workflow for mass spectrometry data analysis. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 47–54.

[Neteler et al., 2012] Neteler, M., Bowman, M. H., Landa, M., and Metz, M. (2012). {GRASS} gis: A multi-purpose open source {GIS}. *Environmental Modelling & Software*, 31(0):124 – 130.

[Nicholas Charles, 2007] Nicholas Charles, R. (2007). *Foundations of Process-Aware Information Systems*. PhD thesis, Faculty of Information Technology Queensland University of Technology Brisbane, Australia.

[Nuñez et al., 2011] Nuñez, D., Agudo, I., Drogkaris, P., and Gritzalis, S. (2011). Identity management challenges for intercloud applications. In *1st International Workshop on Security and Trust for Applications in Virtualised Environments (STAVE 2011)*, Crete (Greece).

[OASIS, 2007] OASIS (2007). Web services business process execution language. Technical report, OASIS.

[Oasis, 2012] Oasis (2012). Organization for the advancement of structured information standards). available at URL:`http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html`.

[OASIS, 2014] OASIS (2014). Amazon simple workflow service, developer guide. Technical report, AWS.

[Offermann, 2003] Offermann, S. (2003). Ein Kompositionsmodell für Mehrwertdienste auf Basis von Referenznetzen. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Oinn et al., 2004] Oinn, T., Addis, M., Ferris, J., Marvin, D., Carver, T., Pocock, M. R., and Wipat, A. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20:2004.

[OMG, 2013] OMG (2013). Business process model and notation (BPMN), version 2.0.2. Technical report, Object Management Group (OMG). `http://www.omg.org/spec/BPMN/2.0.2/PDF/`.

[OMG, 2015] OMG (2015). OMG unified modeling language specification. Technical report, Object Management Group (OMG).

[Pandey et al., 2011a] Pandey, S., Karunamoorthy, D., and Buyya, R. (2011a). *Workflow Engine for Clouds*, pages 321–344. John Wiley & Sons, Inc.

[Pandey et al., 2011b] Pandey, S., Karunamoorthy, D., and Buyya, R. (2011b). *Workflow Engine for Clouds*, pages 321–344. John Wiley & Sons, Inc.

[Papazoglou, 2012a] Papazoglou, M. P. (2012a). Cloud blueprints for integrating and managing cloud federations. In Heisel, M., editor, *Software Service and Application Engineering*, volume 7365 of *Lecture Notes in Computer Science*, pages 102–119. Springer.

[Papazoglou, 2012b] Papazoglou, M. P. (2012b). Cloud blueprints for integrating and managing cloud federations. In Heisel, M., editor, *Software Service and Application Engineering*, volume 7365 of *Lecture Notes in Computer Science*, pages 102–119. Springer.

[Papazoglou et al., 2007] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45.

[Peltz, 2003] Peltz, C. (2003). Web services orchestration and choreography. *Computer*, 36(10):46–52.

[Petcu, 2011] Petcu, D. (2011). Portability and interoperability between clouds: Challenges and case study. In Abramowicz, W., Llorente, I., Surridge, M., Zisman, A., and Vayssière, J., editors, *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 62–74. Springer Berlin Heidelberg.

[Petcu et al., 2011] Petcu, D., Crăciun, C., Neagul, M., Panica, S., Martino, B., Venticinque, S., Rak, M., and Aversa, R. (2011). Architecturing a sky computing platform. In Cezon, M. and Wolfsthal, Y., editors, *Towards a Service-Based Internet. ServiceWave 2010 Workshops*, volume 6569 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg.

[Peter Mell, 2011] Peter Mell, T. G. (2011). The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

[Petri, 1962] Petri, C. A. (1962). Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn.

[Qin and Fahringer, 2012] Qin, J. and Fahringer, T. (2012). Prerequisites. In *Scientific Workflows*, pages 15–28. Springer Berlin Heidelberg.

[Rajkumar Buyya, 2009] Rajkumar Buyya, J. Y. (2009). Gridbus workflow enactment engine. In et al. (eds.), L. W., editor, *Grid Computing: Infrastructure, Service, and Applications*, pages 119–146. CRC Press, Boca Raton.

[Reese, 2009] Reese, C. (2009). *Prozess-Infrastruktur für Agentenanwendungen*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg. Pdf: `http://www.sub.uni-hamburg.de/opus/volltexte/2010/4497/`.

[Reese, 2010] Reese, C. (2010). *Prozess-Infrastruktur für Agentenanwendungen*, volume 3 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin. Dissertation. Pdf: http://www.sub.uni-hamburg.de/opus/volltexte/2010/4497/.

[Reichert et al., 2009] Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., and Göser, K. (2009). Architectural principles and components of adaptive process management technology. In Heinzl, A., Dadam, P., Kirn, S., and Lockemann, P. C., editors, *PRIMIUM - Process Innovation for Enterprise Software, 15.04.2009 in Mannheim, Germany*, volume 151 of *LNI*, pages 81–97. GI.

[Rochwerger et al., 2011] Rochwerger, B., Breitgand, D., Epstein, A., Hadas, D., Loy, I., Nagin, K., Tordsson, J., Ragusa, C., Villari, M., Clayman, S., Levy, E., Maraschini, A., Massonet, P., Muñoz, H., and Tofetti, G. (2011). Reservoir - when one cloud is not enough. *Computer*, 44(3):44 –51.

[Rochwerger et al., 2009] Rochwerger et al., B. (2009). The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1–4:11.

[Rölke, 1999] Rölke, H. (1999). Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen. Diploma thesis, University of Hamburg, Department of Computer Science.

[Rölke, 2004] Rölke, H. (2004). *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin.

[Salimifard and Wright, 2001] Salimifard, K. and Wright, M. (2001). Petri net-based modelling of workflow systems: An overview. *European Journal of Operational Research*, 134(3):664–676.

[Scheer, 2002] Scheer, A. (2002). *ARIS - vom Geschäftsprozess zum Anwendungssystem*. Springer, Berlin [u.a.], 4., durchges. aufl. edition.

[Schumacher, 2003] Schumacher, J. (2003). Eine Plugin-Architektur für Renew – Konzepte, Methoden, Umsetzung. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg.

[SIIF, 2011] SIIF (2011). P2302 - standard for intercloud interoperability and federation (siif). URL: http://standards.ieee.org/develop/project/2302.html.

[Sim, 2012] Sim, K. M. (2012). Agent-based cloud computing. *IEEE Transactions on Services Computing*, 5(4):564–577.

[Simon, 2014] Simon, M. (2014). Concept and implementation of distributed simulations in Renew. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Singh, 1998] Singh, M. (1998). Agent communication languages: rethinking the principles. *Computer*, 31(12):40 –47.

[Sonntag et al., 2010a] Sonntag, M., Karastoyanova, D., and Deelman, E. (2010a). Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows. In *Sixth International Conference on e-Science, e-Science 2010, 7-10 December 2010, Brisbane, QLD, Australia*, pages 206–213. IEEE Computer Society.

[Sonntag et al., 2010b] Sonntag, M., Karastoyanova, D., and Leymann, F. (2010b). The missing features of workflow systems for scientific computations. In Engels, G., Luckey, M., Pretschner, A., and Reussner, R., editors, *Software Engineering 2010 - Workshopband (inkl. Doktorandensymposium), Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26.02.2010, Paderborn*, volume 160 of *LNI*, pages 209–216. GI.

[Stevens et al., 2003] Stevens, R. D., Robinson, A. J., and Goble, C. A. (2003). mygrid: personalised bioinformatics on the information grid. In *ISMB (Supplement of Bioinformatics)*, pages 302–304.

[Strauch et al., 2013] Strauch, S., Andrikopoulos, V., Bachmann, T., and Leymann, F. (2013). Migrating application data to the cloud using cloud data patterns. In *Proceedings of the 3 rd International Conference on Cloud Computing and Service Science, CLOSER 2013, 8-10 May 2013, Aachen, Germany*, pages 36–46. SciTePress.

[Sycara, 1998] Sycara, K. P. (1998). Multiagent systems. *The AI Magazine*, 19(2):79–92.

[Talia, 2012] Talia, D. (2012). Clouds meet agents: Toward intelligent cloud services. *Internet Computing, IEEE*, 16(2):78–81.

[Taylor et al., 2003] Taylor, I., Shields, M., Wang, I., and Rana, O. (2003). Triana applications within grid computing and peer to peer environments. *Journal of Grid Computing*, 1(2):199–217.

[Tolosana-Calasanz et al., 2012] Tolosana-Calasanz, R., BañAres, J. Á., Pham, C., and Rana, O. F. (2012). Enforcing qos in scientific workflow systems enacted over cloud infrastructures. *Journal of Computer and System Sciences*, 78(5):1300–1315.

[Toosi et al., 2014] Toosi, A. N., Calheiros, R. N., and Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Comput. Surv.*, 47(1):7.

[Tsuda et al., 2012] Tsuda, H., Matsuo, A., Abiru, K., and Hasebe, T. (2012). Inter-cloud data security for secure cloud-based business collaborations. *FUJITSU SCIENTIFIC and TECHNICAL JOURNAL (FSTJ)*, 48(2).

[Turner and Tan, 2007] Turner, K. and Tan, K. (2007). Graphical composition of grid services. In Guelfi, N. and Buchs, D., editors, *Rapid Integration of Software Engineering Techniques*, volume 4401 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg.

[Valk, 1998] Valk, R. (1998). Petri nets as token objects - an introduction to elementary object nets. In Desel, J. and Silva, M., editors, *19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal*, number 1420 in Lecture Notes in Computer Science, pages 1–25, Berlin Heidelberg New York. Springer-Verlag.

[Valk, 2004] Valk, R. (2004). Object Petri Nets – Using the Nets-within-Nets Paradigm. In Desel, J., Reisig, W., and Rozenberg, G., editors, *Advances in Petri Nets: Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, Berlin Heidelberg New York.

[van der Aalst, 1997] van der Aalst, W. (1997). Verification of workflow nets. In in Computer Science, L. N., editor, *Application and Theory of Petri Nets 1997*, volume 1248, pages 407–426.

[van der Aalst, 1998] van der Aalst, W. (1998). The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66.

[van der Aalst, 2011a] van der Aalst, W. (2011a). Business process configuration in the cloud: How to support and analyze multi-tenant processes? In Zavattaro, G., Schreier, U., and Pautasso, C., editors, *9th IEEE European Conference on Web Services, ECOWS 2011, Lugano, Switzerland, September 14-16, 2011*, pages 3–10. IEEE.

[van der Aalst, 2011b] van der Aalst, W. (2011b). Intra- and inter-organizational process mining: Discovering processes within and between organizations. In Johannesson, P., Krogstie, J., and Opdahl, A. L., editors, *PoEM*, volume 92 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer.

[van der Aalst and van Hee, 2002] van der Aalst, W. and van Hee, K. (2002). *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press.

[van der Aalst and van Hee, 2004] van der Aalst, W. and van Hee, K. (2004). *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA.

[Vaquero et al., 2008] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55.

[Vidal et al., 2012] Vidal, J., Lama, M., and Bugarín, A. (2012). Toward the use of Petri nets for the formalization of OWL-S choreographies. *Knowledge and Information Systems*, 32(3):629–665.

[W3C, 2012] W3C (2012). Extensible markup language (xml) 1.0 (fifth edition). URL:`http://www.w3.org/TR/REC-xml/`.

[W3C), 2012] W3C) (2012). World wide web consortium (w3c). URL:`http://www.w3.org/`.

[Wagner, 2009a] Wagner, T. (2009a). A centralized Petri net- and agent-based workflow management system. In Duvigneau, M. and Moldt, D., editors, *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, number FBI-HH-B-290/09 in Bericht, pages 29–44, Vogt-Kölln Str. 30, D-22527 Hamburg. University of Hamburg, Department of Informatics.

[Wagner, 2009b] Wagner, T. (2009b). Modeling of a centralized Petri net- and agent-based workflow management system. Bachelor thesis, University of Hamburg, Department of Informatics.

[Wagner, 2010] Wagner, T. (2010). Prototypische Realisierung einer Integration von Agenten und Workflows. In *Informatiktage 2010 - Fachwissenschaftlicher Informatik-Kongress 19. und 20. März 2010, B-IT Bonn-Aachen International Center for Information Technology in Bonn*, volume S-9 of *LNI*, pages 101–104. GI.

[Wagner, 2012] Wagner, T. (2012). Agentworkflows for flexible workflow execution. In [Cabac et al., 2012], pages 199–214.

[Wagner, 2016] Wagner, T. (planned for end of 2016). *Petri Net-based Combination and Integration of Agents and Workflows*. Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Wagner and Moldt, 2011] Wagner, T. and Moldt, D. (2011). Approaching the integration of agents and workflows. In Bergenthum, R. and Desel, J., editors, *Algorithmen und Werkzeuge für Petrinetze. 18. Workshop AWPN 2011, Hagen, September 2011. Tagungsband*, pages 55–61.

[Wagner and Moldt, 2015a] Wagner, T. and Moldt, D. (2015a). Integrating agent actions and workflow operations. In Müller, J. P., Ketter, W., Kaminka, G., Wagner, G., and Bulling, N., editors, *Multiagent System Technologies - 13th German Conference, MATES 2015, Cottbus, Germany, September 28-30, 2015, Revised Selected Papers*, volume 9433 of *Lecture Notes in Computer Science*, pages 61–78. Springer.

[Wagner and Moldt, 2015b] Wagner, T. and Moldt, D. (2015b). Workflow management principles for interactions between Petri net-based agents. In Devillers, R. and Valmari, A., editors, *Application and Theory of Petri Nets and Concurrency - 36th International Conference, Petri Nets 2015, Brussels, Belgium, June 21-26, 2015, Proceedings*, volume 9115 of *Lecture Notes in Computer Science*, pages 329–349. Springer-Verlag.

[Waschke, 2012] Waschke, M. (2012). *Cloud*, pages 43–59. Apress.

[Wassermann et al., 2007] Wassermann, B., Emmerich, W., Butchart, B., Cameron, N., Chen, L., and Patel, J. (2007). Sedna: A Bpel-based environment for visual scientific workflow modelling. In *In Workflows for eScience - Scientific Workflows for Grids*. Springer Verlag.

[Wei and Blake, 2010] Wei, Y. and Blake, M. B. (2010). Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75.

[Weinman, 2011] Weinman, J. (2011). Axiomatic cloud theory. Working Paper. Available at http://www.joeweinman.com/Resources/Joe_Weinman_Axiomatic_Cloud_Theory.pdf.

[Weske and Puhlmann, 2005] Weske, Mathias Vossen, G. and Puhlmann, F. (2005). Workflow and service composition languages. In Bernus, P., Mertins, K., and Schmidt, G., editors, *Handbook on Architectures of Information Systems*, International Handbooks on Information Systems, pages 369–390. Springer Berlin Heidelberg.

[WfMC, 1999] WfMC (1999). *Workflow Management Coalition, Terminology & Glossary (Document No. WFMC-TC-1011)*. Workflow Management Coalition Specification.

[White, 2008] White, S. A. (2008). *BPMN modeling and reference guide: understanding and using BPMN*. Future Strategies Inc.

[Wooldridge, 1997] Wooldridge, M. (1997). Agent-based software engineering. In Proc, I., editor, *IEE Proceedings on Software Engineering*, volume 144, pages 26–37.

[Wooldridge, 2009] Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems (2. ed.)*. Wiley.

[Wu et al., 2013] Wu, Z., Liu, X., Ni, Z., Yuan, D., and Yang, Y. (2013). A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293.

[Yang et al., 2007] Yang, Y., Liu, K., Chen, J., Lignier, J., and Jin, H. (2007). Peer-to-peer based grid workflow runtime environment of swindew-g. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, pages 51–58, Washington, DC, USA. IEEE Computer Society.

[Yu and Buyya, 2005] Yu, J. and Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, 3(3-4):171–200.

[Zaplata, 2012] Zaplata, S. (2012). *Flexibilisierung verteilter Prozessausführung: Zur dynamischen Verteilung, Überwachung und Steuerung individueller Prozessinstanzen auf Anwendungsebene*. PhD thesis, University of Hamburg.

[Zaplata et al., 2009] Zaplata, S., Kottke, K., Meiners, M., and Lamersdorf, W. (2009). Towards runtime migration of WS-BPEL processes. In Dan, A., Gittler, F., and Toumani, F., editors, *ICSOC/ServiceWave Workshops*, volume 6275 of *Lecture Notes in Computer Science*, pages 477–487.

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

[Zhang and Zhang, 2009] Zhang, Z. and Zhang, X. (2009). Realization of open cloud computing federation based on mobile agent. In *IEEE International Conference on*

*Intelligent Computing and Intelligent Systems, 2009. ICIS 2009.*, volume 3, pages 642–646.