Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 1656-0001

Design and implementation of TOSCA Service Templates for provisioning and executing bone simulation in cloud environments

Marzieh Dehghanipour

Course of Study:	INFOTECH
Examiner:	PD Dr. rer. nat. habil. Holger Schwarz
Supervisor:	Peter Reimann, Tim Waizenegger
Commenced:	29.January, 2015
Completed:	31.July, 2015
CR-Classification:	C.2.4, C.5.5, D.2.11, D.3.2, G.2.3, H.3.4, I.6.3,

Abstract

Recent years have shown an increasing trend to move applications and services into cloud infrastructures. Cloud-based applications typically consist of distributed components which are connected and communicate with each other. Automating the deployment and management of these components is one of the major challenges in IT world. The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard provides a meta-model for describing the structure of composite cloud-based applications, which provides automation for deployment and management of these applications. TOSCA-based applications may be executed via the OpenTOSCA (a run-time environment for TOSCA-based applications) environment, which has been developed by the University of Stuttgart.

Simulation applications deal with heterogeneous and huge data sources. Adequate data management and data provisioning for these applications are some of the most significant challenges for simulation applications. SimTech – Information Management, Processes, and Languages (SIMPL) is a framework which provides a generic approach for data management and data provisioning in simulation applications. SIMPL frees users to deal with any low-level details of data sources and corresponding data management operations.

Both the TOSCA standard and the SIMPL framework are based on workflows. The first goal of this master's thesis is to combine the TOSCA standard with the SIMPL framework in order to enable the generic data provisioning and data management approach offered by SIMPL as an integral part of the TOSCA standard.

A further and main part of this work is to design and implement TOSCA Service Templates for provisioning and executing bone simulations in cloud environments. Different variants of a TOSCA Service Template realizing a bone simulation in a cloud-native way have to be developed and implemented. In other words, a Software as a Service (SaaS) solution for PANDAS bone simulation is provided in the scope of this master's thesis with the help of TOSCA and SIMPL technologies.

Contents

List of Figures

Listings

Acronyms and Abbreviations

TOSCA	Topology and Orchestration Specification for Cloud Applications	iii
SIMPL	SimTech – Information Management, Processes, and Languages	iii
CSV	Comma-Separated Values	2
SQL	Structured Query Language	2
XML	Extensible Markup Language	2
BPEL	Business Process Execution Language	2
IA	Implementation Artifact	3
ODE	Orchestration Director Engine	3
API	Application Program Interface	3
SaaS	Software as a Service	iii
VM	Virtual Machine	3
DM	Data Management	2
NIST	National Institute of Standards and Technology	7
PaaS	Platform as a Service	8
IaaS	Infrastructure as a Service	8
SOAP	Simple Object Access Protocol	8
WSDL	Web Services Description Language	8
UDDI	Universal Description, Discovery and Integration	8
HTTP	Hypertext Transfer Protocol	9
IEEE	Institute of Electrical and Electronics Engineers	10
ISO	International Organization for Standardization	10
ESB	Enterprise Service Bus	10
DA	Deployment Artifact	11
BPMN	Business Process Model and Notation	12
CSAR	Cloud Service ARchive	13
UI	User Interface	14
DBMS	Database Management Systems	15
WfMS	Workflow Management Systems	15
SOA	Service-Oriented Architecture	16

Chapter 1

Introduction

This chapter provides an overview for the motivations and goals behind the present research. Furthermore, the problem statement, scope of work as well as the structure of this master's thesis are introduced in the following sections.

1.1 Motivation and problem statement

Recent researches in science and technology have explored a number of reasons why cloud computing is widely used in today's enterprises. Many organizations are continuously moving their legacy applications to the cloud in order to benefit from significant efficiency and cost advantages which are provided by cloud infrastructures. Using cloud infrastructures within organizations, enables business applications to become more mobile and collaborative. Drawbacks of traditional and on-premise solutions, such as costly infrastructures, lack of flexibility in system performance and accessibility, deployment speed, etc., derive IT enterprises to think about cloud-based solutions instead. There are many developed solutions and approaches for moving traditional and on-premise applications into cloud infrastructures. In other words, a large number of solutions have been discovered in the state of the art for mapping traditional applications into cloud infrastructures regarding provisioning and managing software components as well as required data in cloud environments. The main contribution of this work is to move a legacy bone simulation software, so-called PANDAS[1], into cloud infrastructures and turn the PANDAS software into a fully integrated SaaS solution[2].

Rapidly changing business and IT environments make systems and applications more complex and distributed. The past few years have seen increasing technological advances in IT applications and solutions. Composite applications, built by combining pieces of other applications and components, are the solution for today's enterprises. Software deployment processes as well as setting up and managing cloud resources in composite applications are problematic and burdensome. Automating the deployment and management of the various components of cloud-based applications is one of the key aspects of moving applications into the cloud. By automation, the provisioning of new software instances for new customers becomes cheaper and faster in terms of money and time. Cloud computing characteristics, such as elasticity, pay per use, rapid provisioning, etc., are highly dependant on the degree of automation in deployment and management of applications. Finding an efficient solution for automat-

[1]http://www.mechbau.uni-stuttgart.de/pandas/index.html

ing the deployment and management of cloud-based applications, such as OASIS[2] TOSCA standard[3], which is an Extensible Markup Language (XML)-based[4] language, is beneficial. The TOSCA standard addresses three main problems which cannot be solved all together in other approaches as follows: (1) automation in deployment and management of cloud-based applications, (2) portability of applications, and (3) reusability and interoperability of components in composite applications. The TOSCA standard is based on XML language, which is a well-known and easy language. In fact, using TOSCA for cloud-based applications is an efficient and easy approach. For orchestration and automation processes in the scope of this master's thesis, different variants of a TOSCA Service Template realizing a bone simulation via PANDAS in a cloud-native way are provided[26].

Data and information explosion in most of simulation applications is one of the biggest issues that scientists and engineers are now dealing with. Data management and data provisioning in cloud infrastructures is one of the most challenging aspects regarding to data and information explosion in simulation applications. In other words, simulation applications work with heterogeneous and huge data in various formats. These huge data need to be queried, transferred, maintained or stored as the result. Most of approaches in this field require lots of effort in order to find appropriate data sources, and to prepare these data in order that simulation applications can properly use them. Most of scientists do not have the necessary skills to provide and prepare appropriate data sources for the PANDAS bone simulation software. Accordingly, provisioning and managing all required data sources for the PANDAS tool with the SIMPL technology removes the burden from scientists to worry about this deficiency and complexity. The PANDAS bone simulation software deals with heterogeneous data sources such as text files, XML databases, Comma-Separated Values (CSV) files and Structured Query Language (SQL) databases, respectively. The SIMPL framework[16], which is an efficient approach that eliminates lots of effort to locate adequate data sources and find appropriate solutions for data transformations is used in this master's thesis. SIMPL extends workflow languages, such as Business Process Execution Language (BPEL)[5], by a small set of Data Management (DM) activities and DM patterns which provide a generic and uniform approach to access any kind of data sources. Similar to TOSCA, the SIMPL framework is also based on workflow languages. To the extent of our knowledge, there is no approach like SIMPL which can provide this level of abstraction in accessing heterogeneous data sources. In other words, all other approaches require to specify low-level details of data sources for managing and provisioning data, and impose a burden on scientists and engineers[16]. Implementing different variants of workflows with SIMPL for provisioning data as well as returning the calculation results to the users is another important goal in this master's thesis.

Both TOSCA standard and SIMPL framework are based on workflow technology. Integration of these two approaches can benefit users to have an integrated support for automating the application deployment as well as provisioning heterogeneous data in cloud-native applications. In the scope of this research, an approach for integrating the SIMPL framework to the workflow engine of OpenTOSCA (a run-time environment for TOSCA-based applications)[6], in order to make the SIMPL generic access for data provisioning and data management an integral

[2] Advancing Open Standards for the Information Society
[3] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[4] http://www.w3schools.com/xml/
[5] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[6] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

part of the TOSCA standard definition, is developed and implemented. Consequently, data management and data provisioning techniques offered by SIMPL can be a constituent part of the TOSCA specifications. In conclusion, moving traditional and on-premise PANDAS bone simulation software into cloud infrastructures by using all above-mentioned technologies enables users to benefit in different aspects and solves all the obstacles which existed before this work[18].

1.2 Scope of work

In the scope of this research, different variants of a TOSCA Service Template for provisioning and executing PANDAS bone simulation software in cloud environments are designed and implemented. To address this goal, the main objectives of this research are divided into two parts. First, the SIMPL framework is integrated with the TOSCA standard in order to offer a full-fledged and integrated support for deploying and managing cloud-native simulation applications as well as for data provisioning regarding their input and result data. In other words, the prototype for SIMPL is integrated with the OpenTOSCA engine in order to provide the abstract data provisioning and data management offered by SIMPL as an integral part of the TOSCA definitions. For this purpose, a TOSCA-based Implementation Artifact (IA) was implemented, so-called *ODE-Service*, as shown in Figure 1.1. This IA is mainly based on Apache Orchestration Director Engine (ODE) deployment Application Program Interface (API)[7] in order to deploy and undeploy BPEL processes to and from the Apache ODE engine remotely.

As Figure 1.1 illustrates, the second and main contribution of this master's thesis is to elaborate, develop, and implement different variants of a TOSCA Service Template realizing a bone simulation in a cloud-native way. The Service Template should provide corresponding service topologies, as well as the following plans which turn the bone simulation into a fully integrated SaaS solution:

- Provisioning plan which instantiates Virtual Machine (VM)s and prepares the infrastructures for installing the PANDAS software components and related resources.
- PANDAS software provisioning plan which sets up the necessary simulation software components, in particular the PANDAS calculation tool and different kinds as well as configurations of database systems that store the result data of PANDAS.
- Different variants of management plans that provide and prepare heterogeneous input data of the simulation in order that PANDAS can properly ingest these data.
- A management plan that orchestrates the simulation calculation in PANDAS.
- Different variants of management plans that return the result data of PANDAS back to the user with respect to the data formats and granularity this user requires.
- A termination plan which terminates the installed VMs and undeploys all other software components.

Moreover, Chapter 6 of this master's thesis compares the corresponding approaches using the TOSCA standard and the SIMPL framework for moving the traditional and on-premise applications into cloud infrastructures with the other existing approaches, in terms of advantages and limitations. This chapter evaluates the generalization capabilities of the proposed approaches

[7]http://ode.apache.org/management-api.html

to other simulation examples, other simulation software and for developing other variants of a TOSCA Service Template. As shown in Figure 1.1, the above-mentioned discussions and evaluations are in the scope of this master's thesis.

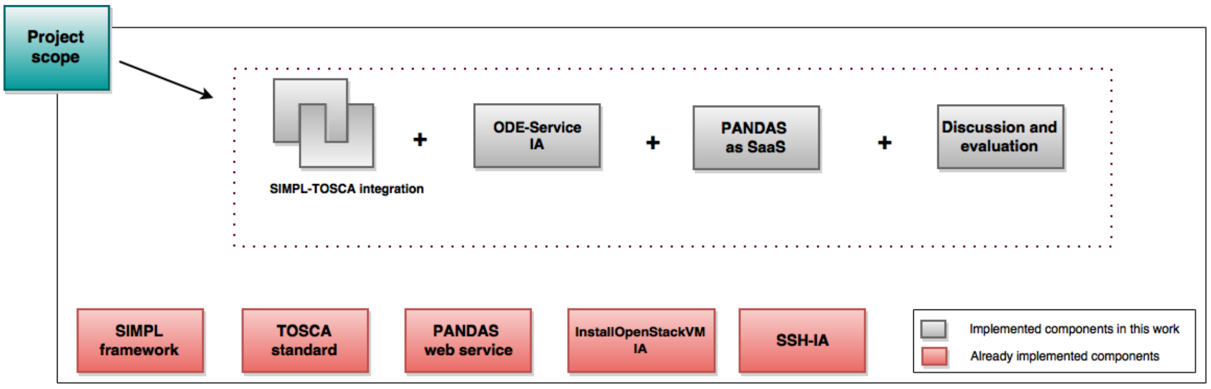


Figure 1.1: Scope of thesis

As Figure 1.1 depicts, the SIMPL framework, TOSCA standard, PANDAS web service, and two IAs are not in the scope of this master's thesis. They are already implemented and only used in this work. The TOSCA standard and the SIMPL framework are discussed in Sections 2.2 and 2.5, respectively. The first IA is *InstallOpenStackVM*, which is a Java-based implementation and contains several operations, such as `InstallVMwithCustomKeypair`, `InstallVMwithGeneratedKey`, `InstallVMwithCustomFlavor`, etc., for installing a new VM on the OpenStack[8] cloud provider. This IA contains the termination function as well, which terminates VMs. The actual implementation of this IA can be found on the GitHub[9]. *SSH-IA* is another already implemented IA, which is used to send some shell commands to a remote server.

1.3 Outline

This section specifies the structure of chapters which is used in this master's thesis. This thesis contains all the following chapters:

Chapter 1 - "Introduction" covers a brief introduction to the topic of this master's thesis. This chapter discusses the motivation scenario, problem statement as well as the scope of this work.

Chapter 2 - "Background" explains all required background information and technologies related to this master's thesis. This chapter includes an introduction to cloud computing, OASIS TOSCA specification, workflow technology, SIMPL framework and all other technologies and tools which are used for realizing the goals and motivations behind this thesis.

Chapter 3 - "Related Work" discusses preliminary and ongoing related work to the scope

[8] https://www.openstack.org/
[9] https://github.com/tosca-types/openstack

of this master's thesis. This chapter evaluates related work in running simulation applications in cloud infrastructures. Furthermore, similar approaches to the TOSCA standard for automating the deployment and management of composite applications are discussed. Various approaches similar to SIMPL, to manage and provision heterogeneous data sources in simulation applications, are discussed as well.

Chapter 4 - "Mapping PANDAS into cloud" discusses the main concepts developed in the course of this master's thesis for mapping PANDAS into cloud environments regarding software provisioning as well as data provisioning. Besides, different variants of a PANDAS Service Template in the TOSCA standard, which includes an application topology and several management plans, are discussed from a conceptual point of view. Following that, the method which is used in this master's thesis for integrating the TOSCA standard with the SIMPL framework is explained.

Chapter 5 - "Implementation" lists all the libraries and techniques which are used for implementing different variants of a PANDAS Service Template. In other words, the proposed approach for moving an on-premise PANDAS bone simulation software into cloud infrastructures is implemented in detail. Moreover, an approach for the first contribution of this master's thesis which is integration of the SIMPL framework with the TOSCA standard is discussed. Besides, this chapter also discusses challenges the author faced during design and implementation phases.

Chapter 6 - "Discussion" restates general goals and motivations of this master's thesis, which is followed by a statement about whether or not, and to what extent, our findings address these goals. Furthermore, the lessons which were learned in this master's thesis with respect to the approaches and contributions are discussed in this chapter. This chapter also evaluates whether and how the approach which is used in this work for the PANDAS bone simulation software can be generalized to other simulation examples, other simulation software and for developing other variants of a TOSCA Service Template. Besides, this chapter discusses briefly about using other cloud providers for the PANDAS Service Template instead of the OpenStack cloud provider, which is used in this master's thesis.

Chapter 7 - "Summary, conclusion and future work" summarizes briefly the main contributions of this master's thesis and brings this work to conclusions. This chapter discusses the possible future use and extensions for the resulting system. Furthermore, some opportunities and future work for extending the application scope of the results of this thesis are realized as well.

Chapter 2

Background

This chapter provides the basic background knowledge necessary for this thesis. In Section 2.1 a brief introduction to cloud computing is provided. The OASIS TOSCA standard is discussed in Section 2.2 as an OASIS standard to define composite[1] cloud-based applications and their management functionalities. Section 2.3 presents the concept of Workflows in web services as well as the most common language, BPEL, for defining business processes and workflows. Section 2.4 argues about simulation applications as well as an example of a simulation workflow for structure changes within a human bone. This section provides an overview about the concepts which are related to data management and data provisioning in simulation applications. Section 2.5 emphasizes on a generic framework for data management and data provisioning in simulation applications, so-called SIMPL. Following that, a brief overview of technologies and tools which are used in this master's thesis is the main goal of Section 2.6.

2.1 Cloud Computing

According to National Institute of Standards and Technology (NIST), 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"*. The cloud computing paradigm consists of five crucial characteristics, as well as four deployment models and three service models, which are described as follows[50].

Characteristics of cloud computing:
The five fundamental characteristics of cloud computing are: (1) On-demand self-service, (2) Broad network access, (3) Resource pooling, (4) Rapid elasticity and (5) Measured service. *On-demand self-service* refers to the provision of cloud resources, such as data storage, whenever they are required by a customer. The customer can access the resources through the Internet. In other words, services and resources are accessible over the network through different client devices such as mobile phones, tablets and laptops (*Broad network access*). *Resource pooling* describes a situation in which multiple clients, customers or tenants can be served by a single provider. The service can be distributed over clients based on their needs. *Rapid elasticity* is defined as the ability to scale provisioning and deprovisioning of the resources. To put it another way, customers can request more or less resources in the cloud whenever it is demanded. Lastly, *Measured Service* refers to measuring and monitoring the

[1] Applications which consist of heterogeneous distributed component

provision of services in cloud infrastructures by cloud providers. This measurement deals with multiple reasons, such as billing or efficient use of services[50].

Service Models of cloud computing:
In agreement with NIST, there exist three cloud service models: (1) Infrastructure as a Service (IaaS), (2) Platform as a Service (PaaS) and (3) SaaS. With these cloud computing service models, customers can access computing resources in a virtualized environment over a public network. In IaaS, the computing resources are virtualized hardware, such as virtual server spaces, networks and storage. PaaS is a cloud computing model which provides a platform for the users to develop, run and manage web applications. In other words, customers can benefit from hardware and software tools as a service which are needed for application development. Therefore, customers do not need to install their own hardware and software to develop a new application. SaaS refers to delivering of applications to the customers as a service over the Internet. SaaS applications are on the service provider's servers. This model frees customers from installing and managing software on their own local machines[50].

Deployment Models of cloud computing:
According to NIST, four primary deployment models for cloud services are: Private cloud, Community cloud, Public cloud and Hybrid cloud. Private cloud is a particular model of cloud computing which computing power within a virtualized environment is only accessible by a single organization. This model provides a secure cloud-based environment for the customers of that organization. Community cloud describes a multi-tenant infrastructure[2] which is shared between multiple but still defined organizations with common computing concerns. Public cloud is a model in which cloud providers provide computing power, such as storage and applications, to general public over the Internet. Last but not least, Hybrid cloud is the combination of two or more of the above-mentioned cloud deployment models. Accordingly, an organization can maximize its efficiency by using public cloud for unimportant operations in the organization and using private and secure cloud for sensitive operations on the other hand[50].

2.1.1 Web Service

The term `Web Service` describes a standardized way of integrating applications by using different technologies such as XML, Simple Object Access Protocol (SOAP)[3], Web Services Description Language (WSDL)[4] and Universal Description, Discovery and Integration (UDDI)[5] over the Internet Protocol (IP). XML is a language that specifies how web documents should be formatted. Designers use XML to validate and interpret data which is transferred between applications and organizations over the Internet[15]. Listing 2.1 shows the overall structure of a XML-based document.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
```

[2]Multi-tenant infrastructure is an infrastructure where multiple customers can be served by a single instance of a software application[2]

[3]http://www.w3schools.com/webservices/ws_soap_intro.asp

[4]http://www.w3schools.com/webservices/ws_wsdl_intro.asp

[5]http://w3schools.sinsixx.com/wsdl/wsdl_uddi.asp.htm

```
</root>
```

Listing 2.1: XML syntax [22]

SOAP is a XML-based messaging protocol which allows applications running on different operating systems, such as Windows or Linux, to communicate with each other by using Hypertext Transfer Protocol (HTTP) and XML language[15]. Listing 2.2 represents an example of SOAP envelop message.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  <!-- Message information goes here -->
  ...
</soap:Envelope>
```

Listing 2.2: SOAP Envelope syntax [22]

WSDL is another XML-based formatting which is used to describe the functionalities and interfaces of web services. A WSDL file contains several tags which explain the operations of a web service, how these operations are to be called, as well as input and output parameters which are expected by each operation[15]. Listing 2.3 depicts the syntax of elements in a WSDL file.

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

Listing 2.3: WSDL file syntax [22]

UDDI is a directory for storing information related to web services. Businesses can register their own web services in the UDDI directory. Discovering other web services around the world is another aspect of using UDDI. To illustrate, if there exists a web service for flight rate checking and reservation in UDDI, travel agencies could search the directory to discover the specific web service and then communicate with it immediately[15].

2.1.2 Cloud-based applications

Cloud-based applications (Cloud App) are applications and programs within cloud environments. These applications can be accessed over the Internet, e.g., through a web browser. Automating the deployment and management of different components of cloud-based applications and services is one of the most significant challenges in cloud computing paradigm. Automated deployment enables the provisioning of new service instances for new customers cheaper and faster[27]. The subsequent sections describe scalability, elasticity, interoperability and portability characteristics in cloud environments. Scalability and elasticity in cloud-based services fulfill the requirements of automated management.

2.1.2.1 Scalability and elasticity in cloud-based services

Elasticity refers to the ability of a system to access and release computing resources whenever it is actually requested by clients based on the workloads. Scalability is the prerequisite for elasticity. Scalability is the ability of a system to continue to function well whenever computing resources are provisioned or deprovisioned according to user needs. In contrast to elasticity, scalability does not consider how well the resources, which are provisioned or deprovisioned, satisfy customer needs. In other words, these two properties enable cloud-based applications to be managed in an automated way[10].

2.1.2.2 Interoperability and portability in cloud computing

According to Institute of Electrical and Electronics Engineers (IEEE)[6] and International Organization for Standardization (ISO)[7], interoperability is *"the ability for two or more systems or applications to exchange information and mutually use the information that has been exchanged"*. In the context of cloud computing, interoperability provides the capability of cloud-based applications to understand their interfaces, authentication and authorization mechanisms, data formats needed to be transferred, etc., for communication and cooperation with each other. Portability in cloud computing presents the facility to move a component of a system to another system without significant effort for transformation of the source format to the one which is compatible to the target system[4].

There are many challenges associated with interoperability and portability in cloud computing. Interfaces and APIs of cloud services are not standardized. In consequence, interoperability between services is troublesome. One practical approach to handle interoperability in cloud computing is providing an isolation layer between customer application interfaces and cloud service interfaces. Accordingly, the service is not called directly by the customer application. Technologies such as Enterprise Service Bus (ESB)[8] is appropriate for this aspect[4]. OASIS defines the TOSCA standard which made significant improvements and enhancements to interoperability and portability of cloud-based applications and services [26]. In the following sections, a brief overview on the main concepts of TOSCA is provided.

[6]https://www.ieee.org/index.html

[7]http://www.iso.org/iso/home.html

[8]http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html

2.2 OASIS TOSCA

TOSCA is an OASIS standard to define composite cloud-based applications and their management functionalities. In other words, by using the TOSCA standard, users are capable to define the components of an application, relationships between these components as well as their related management functionalities in a standardized, portable and well-defined structure. A TOSCA-based application, so-called `Service Template`, consists of two main concepts: (1) an application topology, and (2) management plans. The topology defines different components of an application and the relationships between these components as a graph. Each vertex in the graph is a `Node Template`. Node Templates can be derived from `Node Types` and then can be instantiated as `Node Instances`. Management plans on the other hand, can be used to create a high-level management task of these components. In other words, management plans are used to automate the deployment, configuration, as well as management of various components of applications. They can be executed in an automated manner in order to deploy, configure, manage and operate the application[5].

Figure 2.1 depicts the main two concepts of the TOSCA standard. As shown in the figure, a Topology Template consists of several nodes which are connected with relationships. These nodes and their relationships can have various types. For instance, Node Types can be IA, Deployment Artifact (DA), script, policy, etc. Implementation and deployment artifacts are the actual implementation of nodes in the topology. To illustrate, an operating system type in the topology may have an image hosting the operating system as DA. IAs are small management applications which provide management capabilities of a node via Representational State Transfer (REST), WSDL or script interfaces. In the example of an operating system, an IA could be a REST-based web service, which has a function to connect via Secure Shell (SSH) to a virtual server with the defined operating system, in order to run some configuration commands on the server. Implementation and deployment artifacts are deployed onto the corresponding nodes of the topology in a proper time. Similarly, relationships between these nodes are from different types, such as `hosted on`[9], `depends on`[10] or `communicates with`[11][5].

[9] An operating system can be hosted on a server
[10] One component depends on another component
[11] An application can communicate with its related database

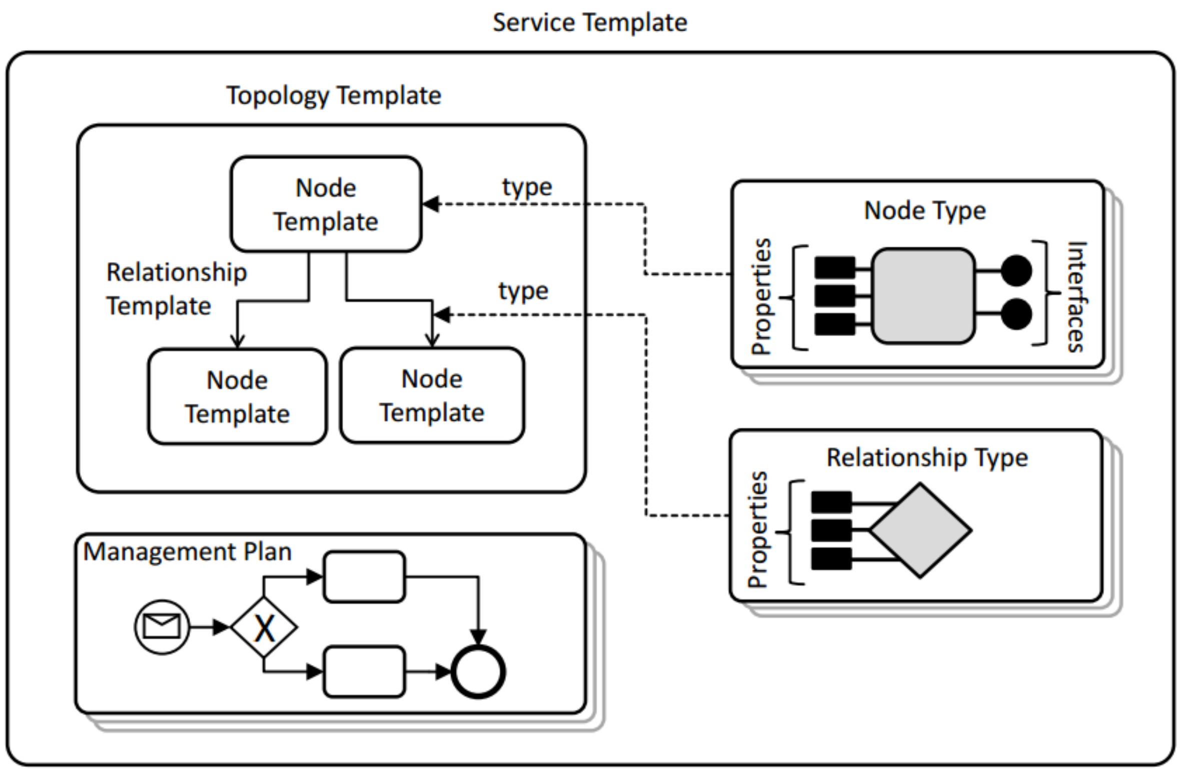


Figure 2.1: TOSCA Service Template (adapted from OASIS (2012))[5]

Management plans on the other hand, are sequences of activities which can be started and executed fully automatedly by receiving an external message. The TOSCA standard uses workflow languages, such as Business Process Model and Notation (BPMN)[12] or BPEL, to define management plans. Figure 2.2 represents a simple management plan which installs a MySQL[13] database on a Linux VM inside the OpenStack cloud provider.

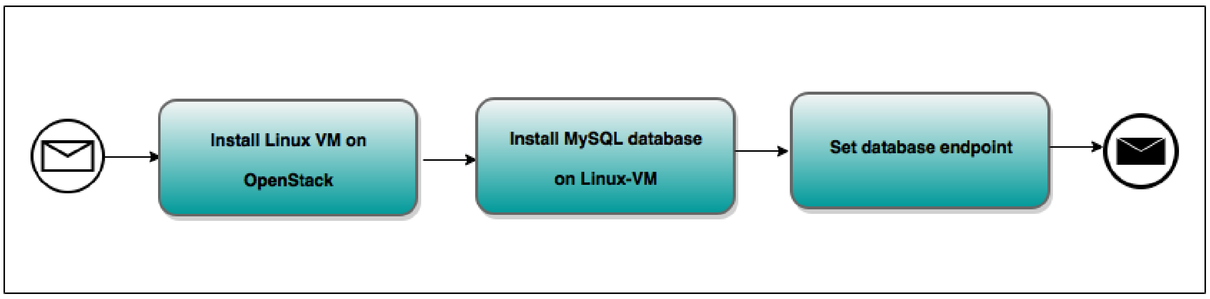

Install Linux VM on OpenStack
Install MySQL database on Linux-VM
Set database endpoint

Figure 2.2: A simple TOSCA plan

This plan is started by receiving a message from outside. The first circle represents a start event which instantiates the plan immediately after receiving a message. The execution continues to the next activity and installs a Linux VM on the OpenStack cloud provider. The third activity

[12]http://www.bpmn.org/
[13]https://www.mysql.com/

installs a MySQL database on the instantiated VM. The next activity sets an endpoint[14] for the database. Endpoint is the point of entry into a database Server. Users and other applications can connect to a database server via these endpoints. The execution continues until it reaches the last circle, which is an end event. After the execution of this end event, the plan is terminated by sending some notifications to other or external partners in the process. This simple workflow uses BPMN modelling notation as a modeling language.

2.2.1 Challenges addressed by OASIS TOSCA

Three primary challenges can be addressed by using TOSCA-based applications: (1) Automated management (2) Portability of applications and (3) Interoperability and reusability of application components[5]. In the following sections, each of these three characteristics are discussed in detail.

Automated management: defines the capability of management plans, which can be executed fully automated, to improve the self-service[15] management and elasticity of cloud-based applications whenever provisioning or deprovisioning is happened. The portability of the management plans between various environments is another aspect. Workflow languages, such as BPMN or BPEL, are used to implement management plans (see Section 2.3.1)[5].

Portability of applications: the term `vendor lock-in` refers to the fact that moving an application from one cloud provider to another cloud provider is too expensive and impossible for customers. In order to reduce the `vendor lock-in` problem, portability of applications in cloud infrastructures is essential. Each component in TOSCA-based applications defines its functionality as well as management in a portable way. Consequently, having portable applications with TOSCA is another considerable benefit of using this standard[5].

Interoperability and reusability of application components:
TOSCA achieves reusability and interoperability of applications by defining application components in a self-contained and reusable manner. These components can be defined, packaged and combined in order to provide composite applications. The following section provides a short overview about packaging the TOSCA-based applications[5].

2.2.2 TOSCA packaging

In TOSCA, topologies as well as management plans together with all other resources that are needed for deploying or managing the relevant cloud-based application, such as XML schemes, scripts, etc., are packaged into TOSCA archive, so-called Cloud Service ARchive (CSAR) (OASIS, 2013, Sect. 16)[16]. All components of a composite application are packaged into one single archive and can be considered as the only installable file which should be deployed and executed on a TOSCA run-time environment, such as OpenTOSCA. OpenTOSCA, which is an open source ecosystem for OASIS TOSCA, is discussed as follows.

[14]https://www.simple-talk.com/sql/database-administration/sql-server-endpoints-soup-to-nuts/
[15]It means that a customer can instantiate, manage and terminate his application instances himself
[16]http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

2.2.3 OpenTOSCA

OpenTOSCA[17] is an open source environment for TOSCA-based applications which has been developed at the University of Stuttgart[18]. OpenTOSCA is an environment for running and installing the TOSCA packaging format, i.e. CSAR. The OpenTOSCA environment consists of three main elements as follows[47]:

1. OpenTOSCA container: an open source TOSCA run-time environment
2. Winery[19]: a graphical web-based environment for modeling the topologies of TOSCA-based applications
3. Vinothek[6]: a portal for end users to deploy and instantiate CSARs easily

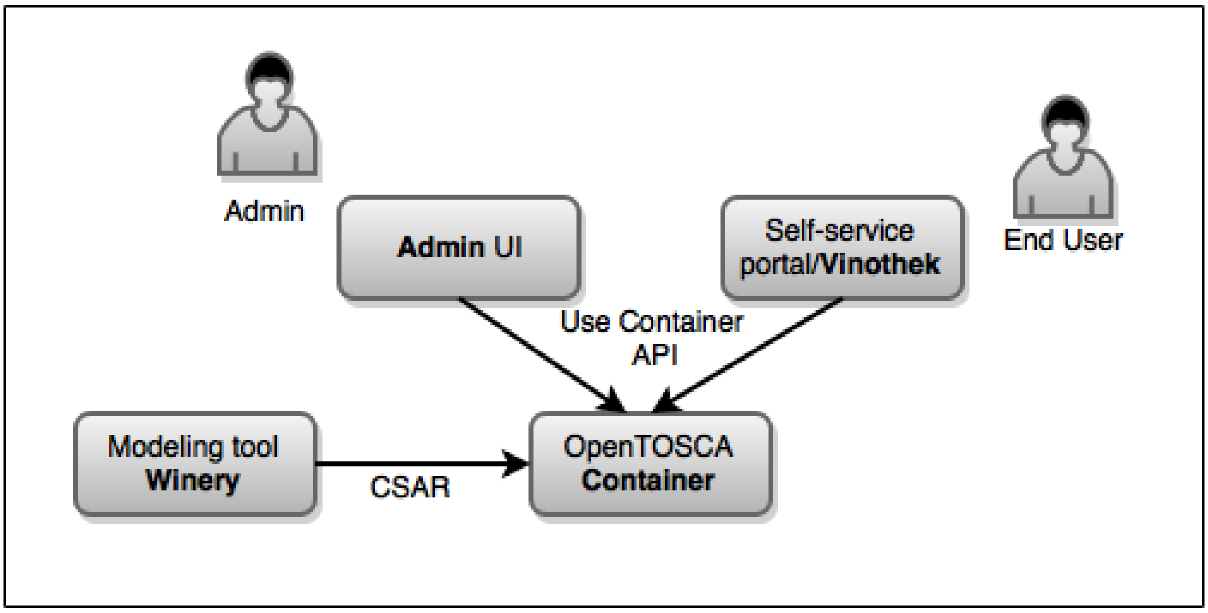


Figure 2.3: Ecosystem structures and relations[48]

Figure 2.3 illustrates the relationship between these three elements inside the OpenTOSCA environment. Developers can use the Winery tool to model TOSCA-based applications, and then export the whole application as a CSAR package. A CSAR file which contains all components of an application can be deployed and executed in OpenTOSCA container. End users and the administrator can access the OpenTOSCA container via the Vinothek and Admin User Interface (UI), respectively[48].

As mentioned already, the TOSCA Service Templates, which consist of application topologies and management plans, are packaged in a CSAR file format, and then deployed on the OpenTOSCA container for execution. Management plans are Workflow-based language (e.g. BPEL) and require a workflow engine, such as Apache ODE or WSO2 Business Process Server[20], for running and execution. Application topoloies on the other hand, consist of various components connecting to each other as well as the actual implementations of these components. Figure 2.4 depicts the process of deploying a CSAR file on the OpenTOSCA environment. As shown

[17]http://www.iaas.uni-stuttgart.de/OpenTOSCA/
[18]http://www.uni-stuttgart.de/home/
[19]https://projects.eclipse.org/projects/soa.winery
[20]http://wso2.com/products/business-process-server/

in the figure, a CSAR file consists of management plans as well as various Web application ARchive (WAR) files. The WAR files are the actual implementations of the components inside the topology. The user can develop a CSAR file with a modeling tool, such as Winery, and then deploy the CSAR on the OpenTOSCA container. The container of OpenTOSCA is responsible to deploy different resources to the appropriate locations. To clarify, it deploys various management plans on a plan engine for running and execution. The container deploys the WAR files on an application server (like Apache Tomcat[21]) for execution as well[47].

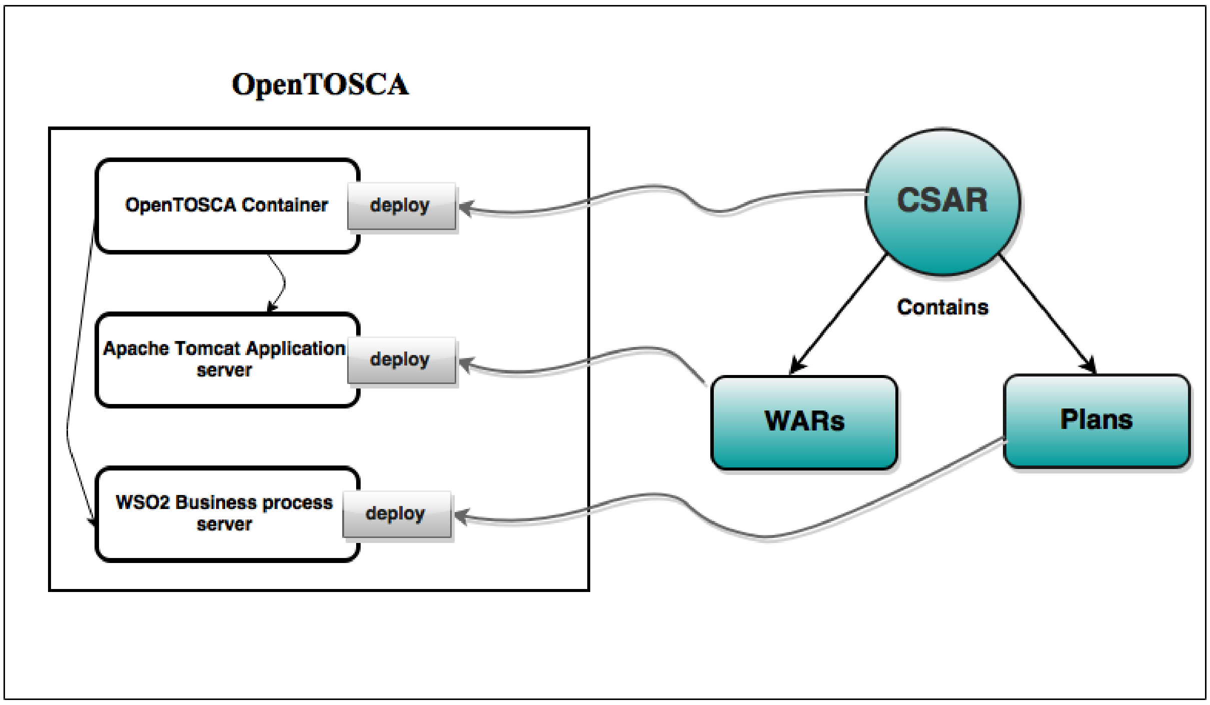


Figure 2.4: Deployment of a CSAR file

2.3 Workflows

Workflows are used in many fields of science, computing and simulations. Workflows consist of a sequence of activities which are connected to each other. Like Database Management Systems (DBMS), Workflow Management Systems (WfMS) are used for managing and executing workflows. WfMS consist of several workflow engines which can be used for process deployment and execution. The following section illustrates some basic concepts in WfMS. In general, WfMS consist of two primary environments: (1) Build-time environment and (2) Run-time environment[14]. Figure 2.5 illustrates the major building blocks of WfMS.

[21]http://tomcat.apache.org/

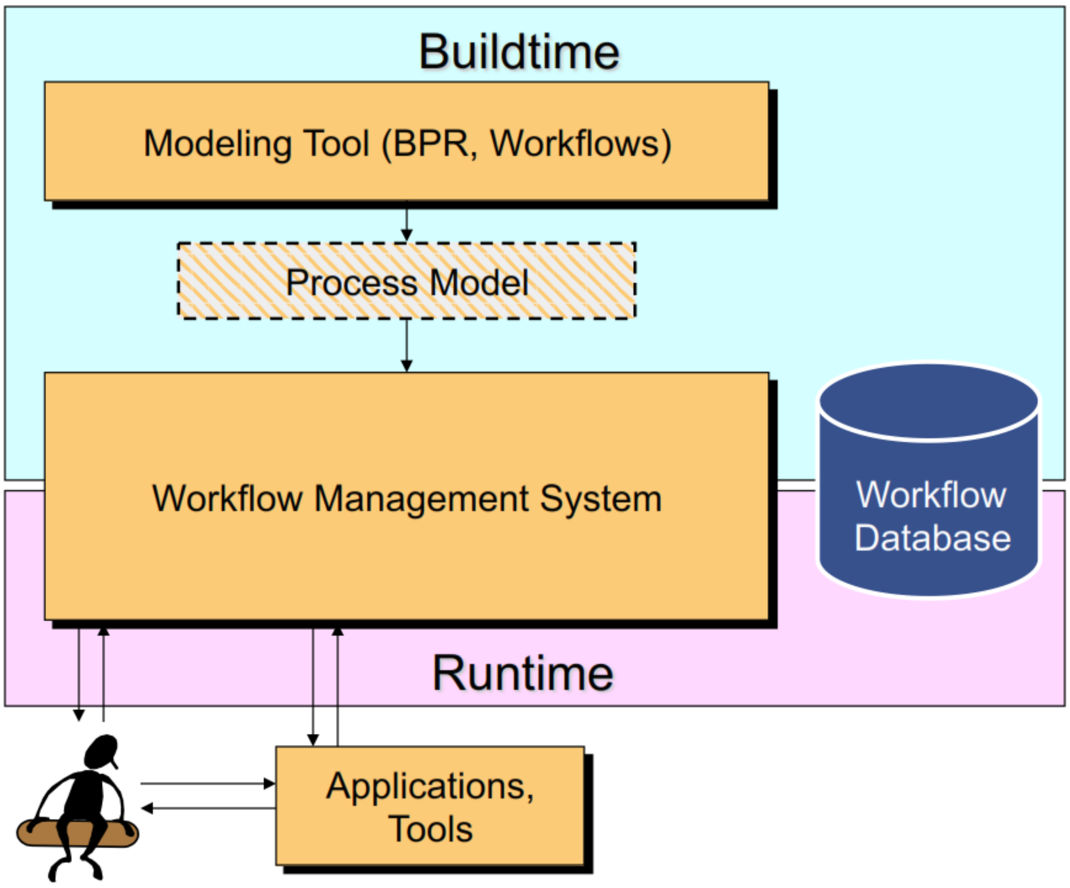


Figure 2.5: Major building blocks of WfMS[14]

WfMS consist of the build-time environment for building or designing the workflows by using some modelling tools. The run-time environment on the other hand, is used for executing the workflows. The workflow database, which is shown in Figure 2.5, is a repository which is shared between the build-time and the run-time environments. This database is used to store and manage all the workflows during the design and implementation phases. After modeling the workflows inside the build-time environment, they can be imported, i.e. deployed, to the run-time environment for execution. After deployment, execution is started. The user can communicate with the WfMS directly or indirectly with the help of some graphical-based applications and tools[14].

The OASIS TOSCA standard uses workflow languages, such as BPMN or BPEL, to define management plans of composite applications. Workflows have some properties which provide portability in cloud-based applications as well as fully automated execution of management plans. TOSCA uses BPMN and BPEL as workflow languages. After modeling the workflows, they need to be deployed and executed on a workflow engine such as Apache ODE engine (see Section 2.6.3)[14]. The subsequent section provides a concise review about the BPEL language in the Service-Oriented Architecture (SOA)[22] paradigm.

[22]http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html

2.3.1 Web Services Business Process Execution Language (WS-BPEL)

Different web services can be integrated or composed with each other in order to develop a composite web service. Integrating different web services needs `orchestration`[14]. It is an important concept in SOA. The term `orchestration` for web services refers to coordinating and integrating several web services and the order in which their service operations are invoked as a single web service. One of the most common standard for defining the `orchestration` of web services is BPEL[1].

BPEL is the standard which was born with the combination of a XML-based language for defining business (XLANG)[23] and Web Services Flow Language (WSFL)[24] technology. This XML-based language is an OASIS standard. WS-BPEL (or BPEL) is a language for orchestrating web services. BPEL standard uses the WSDL definitions of web services in order to coordinate and define relationships between various web services[46]. The subsequent section discusses a brief overview of the BPEL process layout.

2.3.2 WS-BPEL Layout

Previously mentioned, BPEL is a XML-based language which consists of several sub elements. As Listing 2.4 illustrates, a BPEL definition file starts with a `<process>` tag which is the main activity and encapsulates all other elements and activities[1].

```
<?xml version="1.0"?>
<process name="NCName" ...>
  ...
  <partnerLinks>?        ... </partnerLinks>
  <messageExchanges>?   ... </messageExchanges>
  <variables>?          ... </variables>
  <correlationSets>?    ... </correlationSets>

  <faultHandlers>?      ... </faultHandlers>
  <eventHandlers>?      ... </eventHandlers>

   <!-- main activity -->
</process>
```

Listing 2.4: BPEL process syntax[22]

All service-based partners, which are going to communicate with the present BPEL process, can be defined inside the BPEL process with the `<partnerLinks>` tag. This node refers to the WSDL description of the web services. Global and local variables are declared inside the `<variables>` node. Multiple instances of a BPEL process can be instantiated and executed simultaneously. The `<correlationSets>` define a set of properties which are unique in each instance of a BPEL process. Theses properties can be used to select the correct instance during communication. In other words, by having the `<correlationSets>` element inside a BPEL process, instances do not need to have artificial Identity (ID)[1].

[23]https://msdn.microsoft.com/en-us/library/aa577463.aspx
[24]http://www.service-architecture.com/articles/web-services/web_services_flow_language_wsfl.html

Like all programming languages, BPEL language has capabilities to handle faults, exceptions and events during the execution of a process. BPEL achieves this goal by using `<faultHandlers>` and `<eventHandlers>` nodes. In the main activity part, BPEL provides several basic as well as structured activities, such as `<invoke>`, `<receive>`, `<reply>`, `<assign>`, `<flow>`, `<if>`, `<scope>`, `<sequence>`, `<wait>`, etc. The users use the `<invoke>` activity to specify the operations they want to invoke for a specific service. With the `<receive>` element, users can define the partner link from which to receive information as well as the specific port type and operation related to the partner link which they should invoke. The `<reply>` element enables users to respond a message after receiving it with the `<receive>` activity. If there exist two or more requests to the same operation which are sent from one process to another process, `<messageExchanges>` node can be used to make distinction between multiple receives/replies activities. The `<assign>` element is used in order to manipulate data, such as copying the value of one variable to another variable. With a `<flow>` activity, users can define one or more activities to be performed concurrently. The `<if>` element is used to specify conditional behavior for specific activities. The `<scope>` activity enables users to define a set of child activities with their local variables, fault and event handlers and correlation sets. The activities inside the `<sequence>` element are performed in a sequential order. The BPEL process can wait for a certain period of time or until a time limit has been reached with the help of a `<wait>` activity[1].

2.4 Simulation Applications

Simulation play a significant role in many aspects of science. By running simulation, scientists can create models in order to virtually evaluate and test important aspects of the real world. Therefore, developing simulation applications is one of the biggest challenging topics in IT world over the last decades. Simulation applications consist of complicated procedures for their calculations. These applications provide less expensive repeatable and easy ways to represent and test the complex behaviour of a system in real world. Most of the simulation applications are based on workflows. As a consequence, workflow technology plays an important role in developing simulation applications[28]. An example of a simulation workflow is provided as follows.

2.4.1 Simulation workflow for structure changes within a human bone

Recent researches have shown that the structure of a human bone is changing over the time. The load caused on human bones according to daily activities of the relevant person as well as mechanical activities during sports are the main reasons of structure changing within a human bone. In other words, mechanical and biochemical reasons can be considered as the main logic in human bone changing processes. Mechanical aspects, such as sport loads and daily activities can affect dynamics in bone cells as well as tissue properties. Besides mechanical reasons, biochemical changes in human body have influences on structure changing within a human bone, as they may contribute to the mass exchange between different solids and fluids within the bone tissue[29].

The simulation of structure changes within a human bone plays an investigating role of the healing process after bone fractures. The simulation can be divided into two parts: (1) the bio-mechanical simulation which simulates mass movements between porous media and the liquids contained therein at the level of the bone tissue and (2) the systems-biological simulation that determines tissue formation processes at the cellular level. The bio-mechanical simulation is sub-divided into a mechanical and a chemical part. For the bio-mechanical simulation, the PANDAS framework is used and the systems-biological simulation is performed by the computing environment GNU Octave[25]. GNU Octave is an open source version of MATLAB[26]. Running bone simulation with MATLAB or other simulation software could be challenging and complicated. Therefore, the PANDAS bone simulation software is only used in the scope of this master's thesis for simulating the structure changes within a human bone. On the other hand, as PANDAS bone simulation software is not available in the cloud so far, the main goal of this thesis which is mapping PANDAS into cloud is obvious.

The PANDAS bone simulation software is a framework which calculates simulations on the tissue level within a human bone. PANDAS is based on the Finite Element Method (FEM)[27]. First, it calculates changes in the structure of a human bone for one FEM element. Then, it combines all FEM elements. Previously mentioned, PANDAS is used for the bio-mechanical simulation which consists of mechanical and a chemical parts. Therefore, PANDAS simulates two different instances for mechanical and chemical bone structures in the scope of this master's thesis. Figure 2.6 shows the high-level workflow of a PANDAS bone simulation software. This workflow starts with the `Data provisioning` step. This step provides huge and heterogeneous data sources which are required for PANDAS simulation. Then, the workflow continues to the second high-level step, which is PANDAS calculation. In other words, the PANDAS instances (two instances in this work) are simulated in this step. Evaluation of the simulation results can be performed more accurately by visualizing the PANDAS outputs. As a consequence, the results may be transformed to an appropriate format for a visualization software. Therefore, the next step of this workflow is transforming the results of calculation. Following that, the results are visualized in order to facilitate the interpretation of the simulation outcome. After visualization, the workflow and the actual calculation is completed[17].

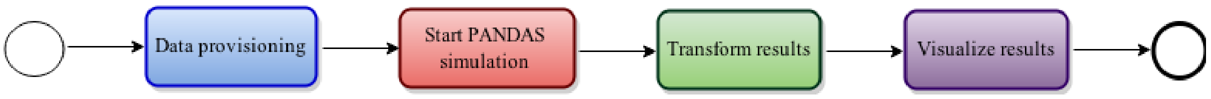


Figure 2.6: PANDAS simulation workflow, cf.[17]

Figure 2.7 depicts the main activities, input and output data required for the PANDAS bone simulation software. The workflow consists of three phases: (1) preprocessing phase, (2) solving phase and (3) post-processing phase. The workflow is started by receiving a message and continues until it reaches the end event. The *preprocessing* phase starts by preparing basic data sources from various databases or file systems about the bone which needs to be simulated. As input parameters, PANDAS needs various and heterogeneous data sources. It requires a text-based geometrical bone shape, XML-based material parameters, CSV-based boundary conditions (e.g. time-dependant pressures on the upper joint of the bone which originate from outside) and FEM parameters (e.g. some parameters for the basic functions such as interpolation functions) which come from SQL databases. Interpolation functions are also known as shape functions or blending functions[30]. As Figure 2.7 depicts, all these various

[25]http://www.gnu.org/software/octave/
[26]http://de.mathworks.com/products/matlab/
[27]http://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/Home.html

data sources should be transformed into text-based data files in order to be handled by PANDAS instances[17].

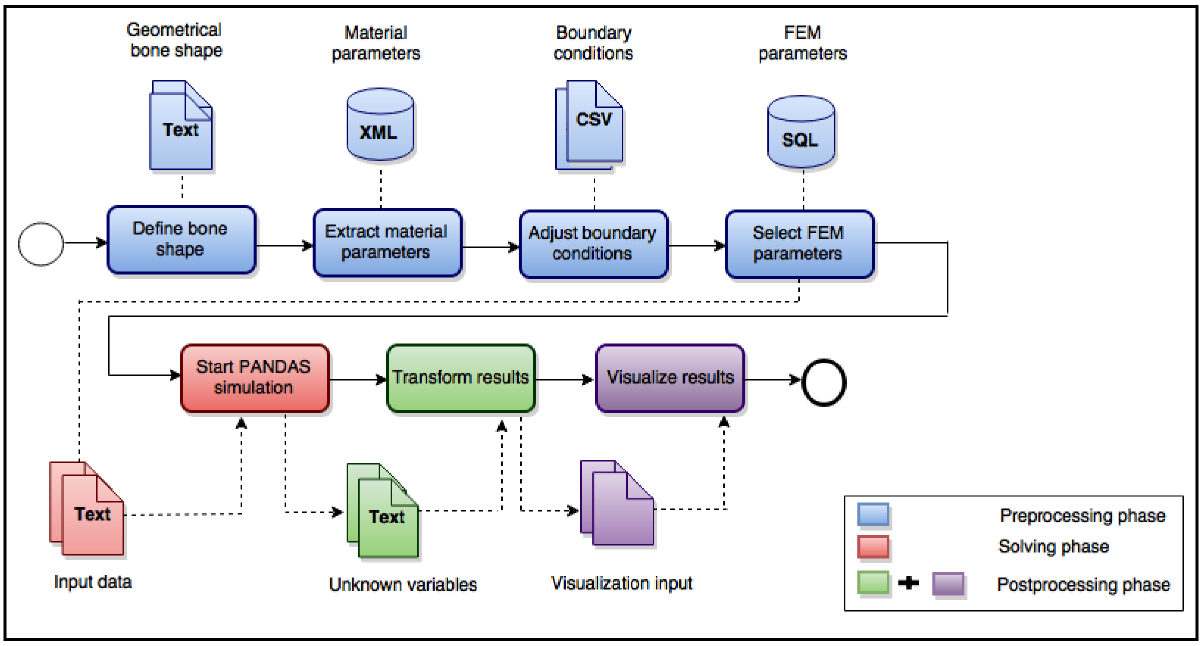


Figure 2.7: Workflow for bone remodeling simulation[17]

The *solving* phase starts the actual simulation of PANDAS. The workflow uses the input data sources, which were prepared in the *preprocessing* phase, in order to create the intermediate as well as the final results of the simulation. These input data sources are used to create and solve some matrix equations. To put it another way, a FEM grid which is necessary to set up the matrix equations is created for each time step t_i. Millions of mesh points are existed in a FEM grid. After the time step t_n, the *solving* phase is completed. Following that, the intermediate and the final results are stored in some CSV files. These CSV files are then transformed into another file format in the *post-processing* phase, which is suitable for the visualization tools. To facilitate the interpretation of the simulation outcome, the results of the simulation can be visualized with some visualization tools. Therefore, the text-based unknown variables are transformed into the formats which are appropriate for the visualization tool. All these procedures for data provisioning and data transformations provide complexities for simulation workflows[16].

In conclusion, as Figure 2.7 illustrated, the simulation workflow for structure changes within a human bone consists of huge and heterogeneous data sources, such as databases, CSV files, text documents, etc., for running the actual simulation. Most of the data management and data provisioning activities, which are done manually, cause a high error rate. As the result, a generic way to access different data sources decreases this error rate, and removes the burden from the scientists and engineers to define the low-level details of data sources[16]. The subsequent sections provides an overview about data management and data provisioning processes in simulation applications as well as a generic framework, so-called SIMPL, to access different kinds of data sources.

2.4.2 Data management and data provisioning in simulation applications

As it can be seen in the example workflow depicted in Figure 2.7, simulation applications deal with heterogeneous, distributed and huge amount of data sources as input parameters for their calculation, as well as huge output data sources which they produce as the results. Data management and data provisioning in simulation applications is one of the most challenging topics these days. In general, the primary data management activities are Extract, Transform and Load (ETL). ETL refers to extracting data from homogeneous or heterogeneous data sources, transforming the data into an appropriate format or structure for querying and analyzing processes and finally loading the data into final target systems[8].

Workflow technology can be considered as one of the primary technologies which are used for managing heterogeneous data formats in simulation applications. Most of the WfMS suffer from inadequate approaches for managing and provisioning huge and heterogeneous data sources. Scientists have to spend lots of time and effort in order to provide appropriate input data formats needed for simulation applications. By having a generic platform for data provisioning and deprovisioning of simulation applications, scientists and engineers do not need to define concrete details of data formats and operations of data provisioning for simulation applications. The following section discusses a generic framework, so-called SIMPL, for data management and data provisioning in simulation fields[16].

2.5 SIMPL framework

The subsequent sections are based on [16] and [17]. The SIMPL framework was a collaboration between the Institute of Parallel and Distributed Systems (IPVS)[28] and the Institute of Architecture of Application Systems (IAAS)[29] at the University of Stuttgart. SIMPL is the framework which compensates the shortage of abstract and generic data management mechanisms in workflow technologies. It extends the BPEL workflow language, by introducing some additional DM activities and DM patterns. In this work as mentioned already, the SIMPL framework is used for data provisioning and data management of PANDAS instances. Figure 2.8 shows the SIMPL framework as an extension of a scientific WfMS. The *SIMPL core* embedded in the *Service Bus*, provides a unified interface to access huge, heterogeneous and distributed data sources. The *Workflow Execution Environment* is extended by the SIMPL framework for additional DM activities. For using these DM activities during the development phase, the *Workflow Design Tool* is extended as well.

[28]http://www.ipvs.uni-stuttgart.de/index1.html

[29]http://www.iaas.uni-stuttgart.de/

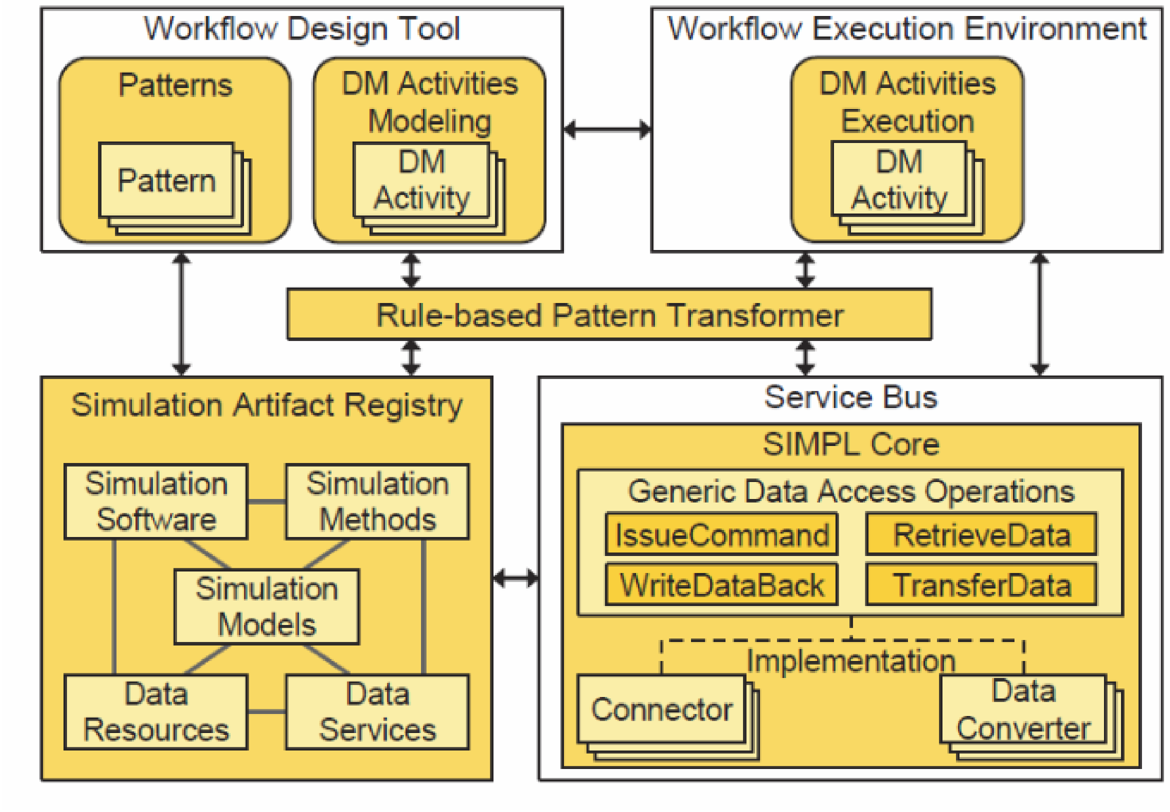


Figure 2.8: Structure of the SIMPL framework embedded in a SWfMS[17]

The current prototype of SIMPL is based on BPEL as the workflow language, Eclipse BPEL Designer[30] version 0.8.0 (see Section 2.6.1) as the workflow designer or modeling tool and Apache ODE version 1.3.5 which is discussed in Section 2.6.3 as the workflow engine. Figure 2.9 depicts the Graphical User Interface (GUI) of the extended Eclipse BPEL Designer. Besides DM activities, the SIMPL framework offers some DM patterns which ease the design of data provisioning tasks required in simulation workflows. Workflows can be implemented with a pattern-based approach. Appropriate DM activities and DM patterns can be selected from the plug-ins as shown in Figure 2.9. Each pattern abstracts away several low-level workflow steps which hide all the complexities from the scientists. The abstract patterns which are combined in the workflow can be transformed, so-called *Rule-based Pattern Transformation*, to an executable workflow during both workflow design and execution. As shown in Figure 2.8, SIMPL uses a *Rule-based Pattern Transformer* component for this transformation. This component uses the *Simulation Artifact Registry* component for providing appropriate information, which are required for this transformation. Inside the GUI of the extended Eclipse BPEL Designer as shown in Figure 2.9, the abstract patterns are transformed to an executable workflow via the pattern transformation buttons[17].

[30]https://eclipse.org/bpel/

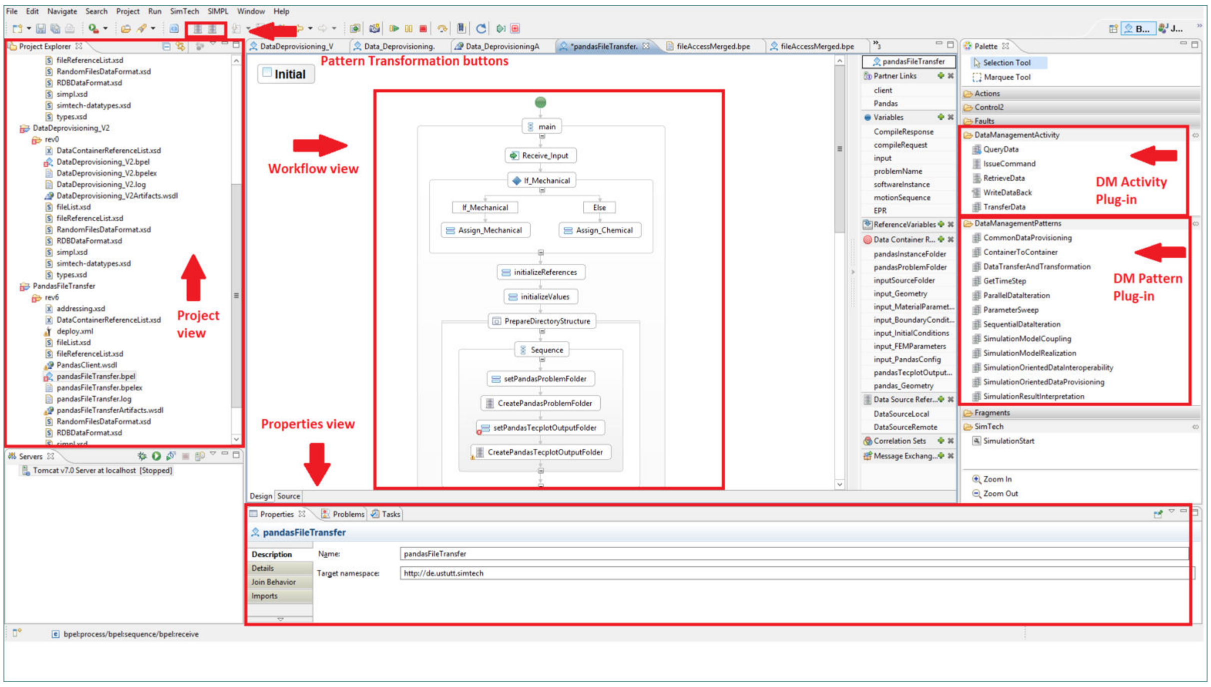


Figure 2.9: User Interface of extended Eclipse BPEL Designer - SIMPL framework

In this master's thesis, only DM activities are used in *DataProvisioning* and *DataDeprovisioning* workflows, which are discussed in Sections 4.2.3 and 4.2.5. In other words, DM patterns are not used in this master's thesis. For this purpose, the discussion in the following section is based on the detailed information related to SIMPL-based DM activities.

2.5.1 SIMPL Data Management Activities

There are four different types of DM activities: (1) *TransferData* activity, (2) *IssueCommand* activity, (3) *RetrieveData* activity and (4) *WriteDataBack* activity. The *SIMPL core*, which is shown in Figure 2.8, is called by each of these DM activities. In other words, these activities send the appropriate DM operations to the *SIMPL core* in order to provide a generic access for heterogeneous and huge data sources[16]. These DM activities are discussed deliberately as follows.

TransferData activity can be used for transferring data from one location to another location. This activity consists of two input parameters for defining a source and a target. To illustrate, in different variants of the *DataProvisioning* plans of PANDAS bone simulation software, this activity transfers some distributed and heterogeneous data from a Windows server instance to a Linux-VM. This activity sends the DM functions to the *SIMPL core*, and waits until receiving a notification for the successful execution of the functions. In case of failure, the fault handling process is executed[16].

IssueCommand activity is another DM activity which may be used to manipulate data. This activity contains a data source reference as well as a data management command as input parameters, e.g. a SQL statement or a shell command to access files. By execution, the command is executed on the appropriate data source. The workflow engine then sends a notification of success. Similarly in case of failure, the engine can enable the fault handling process for recovery[16]. For example, in different variants of the *DataProvisioning* workflows of PANDAS bone simulation software, this activity is used several times for creating some sub-folders in the root folder of the relevant PANDAS software instance.

RetrieveData activity has two input parameters: (1) a data management command for retrieving the appropriate data sources and (2) a data set variable. This activity provides data related to the data management command, and stores the result into the data set variable within the workflow context. For instance, the data management command can be a `SQL SELECT` statement which retrieves appropriate data from a database. After the successful execution of this activity, the workflow engine is notified and the execution is continued to the next activities. In case of failure, the engine can enable the fault handling process similar to other above-mentioned data management activities[16].

WriteDataBack activity stores data from data variable within the workflow context to an external target data container. The workflow engine then notifies the success of the process and the execution is continued to the next step. In case of failure, the fault handling is enabled to recover the failure[16].

In the SIMPL framework, `data source` and `data container` terms are used frequently. The term `data source` is used for a system which can store and manage data. The commands that are embedded in DM activities are issued to the `data sources` and then executed by them. `Data source reference` variable is a variable which refers to a `data source` from within a workflow. A `data source` can manage different `data containers`. Each `data container` is used for a collection of data such as a table in a database system or a file in a file system. `Data container reference` variable is a variable which refers to a `data container`. There are different options in the SIMPL framework for choosing the types of `Data source reference` variables. There are different concrete data sources, such as `Windows Local`, `Unix Local`, `PostgreSQL`, etc., which `Data source reference` variables can refer to. Based on theses `Data source reference` variables which are used in the workflow, a SSH[31] connection for example, only when the `data sources` are remote file systems, is established to connect different file systems. The current implementation of the SIMPL web service contains some operations which enable users to add, delete or update different types of `Data source reference` variables, `Data container reference` variables, etc.[16].

2.5.2 Components of SIMPL run-time environment

The current implementation of the SIMPL framework consists of various components as follows:

- JDK[32], but an original Oracle JDK Version 1.6 (not openJDK for example)
- Apache ActiveMQ version equal or greater than 5.3.2 (standalone application)
- Apache Tomcat version 7.0
- Apache Axis2 version equal or greater than 1.5 (deployed inside Tomcat)
- MySQL Community Server[33] (standalone)

[31]http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man1/slogin.1?query=ssh&sec=1
[32]http://www.oracle.com/technetwork/java/javase/downloads/index.html
[33]https://dev.mysql.com/downloads/mysql/

- PostgreSQL[34] version 9.2 (standalone)

.
Furthermore, the following WARs or libraries are required to be deployed on Apache Tomcat:

- Orchestration Director Engine-Pluggable Framework (ODE-PGF): connected to ActiveMQ and to the MySQL database for communicating and storing auditing information
- Fragmento: connected to a PostgreSQL database for storing the workflow fragments
- The ZIP archive containing the plug-ins of the SIMPL framework

Figure 2.10 depicts the existing components in the SIMPL framework and how these components interact with each other. As shown in the figure, the Apache Tomcat application server is considered as a run-time environment of the workflow middle-ware components. This includes (1) an advanced workflow engine, so-called *ODE-PGF*, (2) a repository for processing the workflow fragments, so-called *Fragmento*, (3) an Apache Axis2 engine and (4) some other ZIP archives containing the plug-ins of the SIMPL framework. These ZIP archives only need to be unzipped into the root folder of the Apache Tomcat server. It was designed to directly work with the ODE-PGF engine (tight integration and no communication channel is required). Furthermore, Tomcat is connected to another PostgreSQL database in order to store the meta-data of SIMPL (simulation artifact registry).

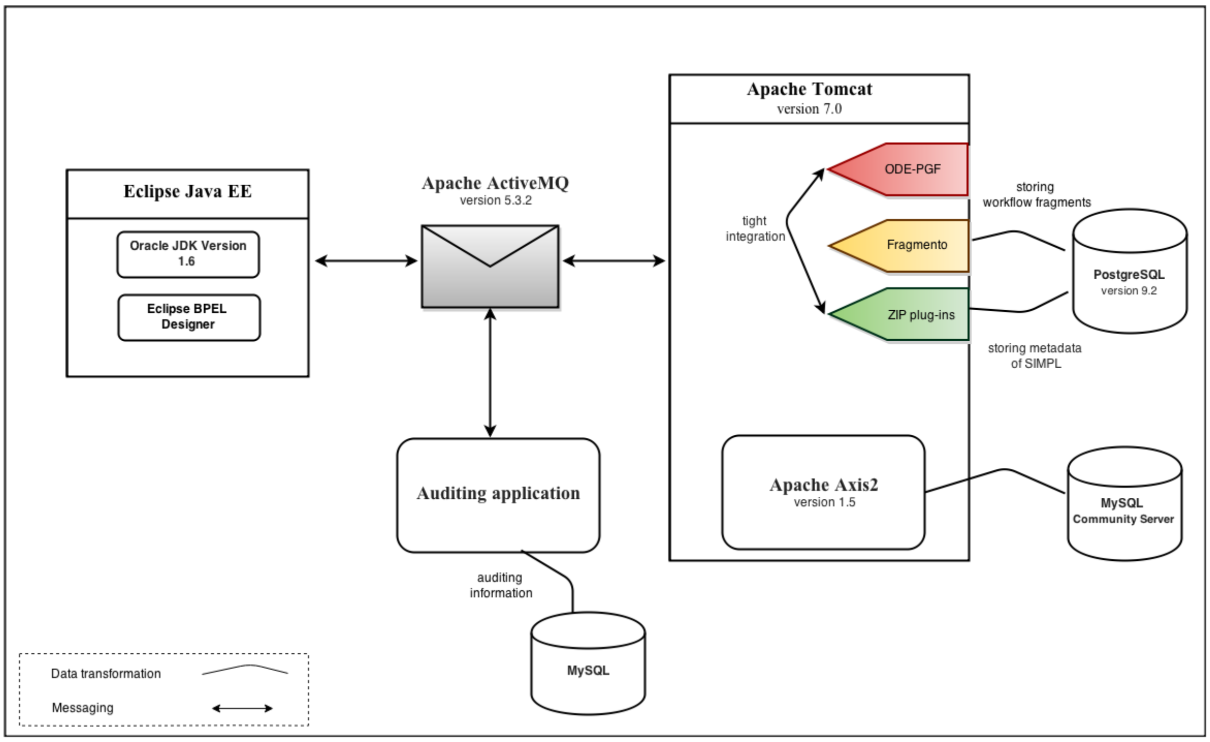


Figure 2.10: Components of SIMPL prototype

The *Auditing application* is a Java stand-alone application which is implemented with a connection to a MySQL database for logging and storing the events of the workflow engine. It monitors all process instances which are created and started.

[34]http://www.postgresql.org/

The *Apache ActiveMQ*[35] is the most common messaging server. The SIMPL framework is designed in a way that components are loosely coupled. In consequence, the components can be connected to each other by sending and receiving messages via a messaging server like *Apache ActiveMQ*. Actually, SIMPL does not use the *Apache ActiveMQ* so far. *Apache ActiveMQ* is only used for other components in the overall architecture of the SIMPL framework that are not directly relevant to SIMPL.

Apache ODE-PGF[31], was implemented as part of a thesis and extends Apache ODE engine v1.3.5. This extension makes the events of the ODE engine visible to the users. For example, there are events that occur during deployment or execution of process instances and activities. It uses *Apache ActiveMQ* in order to publish the events during deployment and execution of process instances.

The *Fragmento* (Fragment-Oriented) repository was developed at IAAS institute at the University of Stuttgart. Business process management is an integral part in process-based applications. Each business process contains of several reusable process fragments. Process fragments can be extracted from a business process and then stored in a *Fragmento* repository. The appropriate fragments can be retrieved and reused in another business process[32]. Figure 2.11 depicts the concepts of extracting and retrieving fragments of a business process. In SIMPL, it is connected to the PostgreSQL database of the SIMPL framework for storing the workflow fragments. These fragments can be used later in other workflows inside the SIMPL framework.

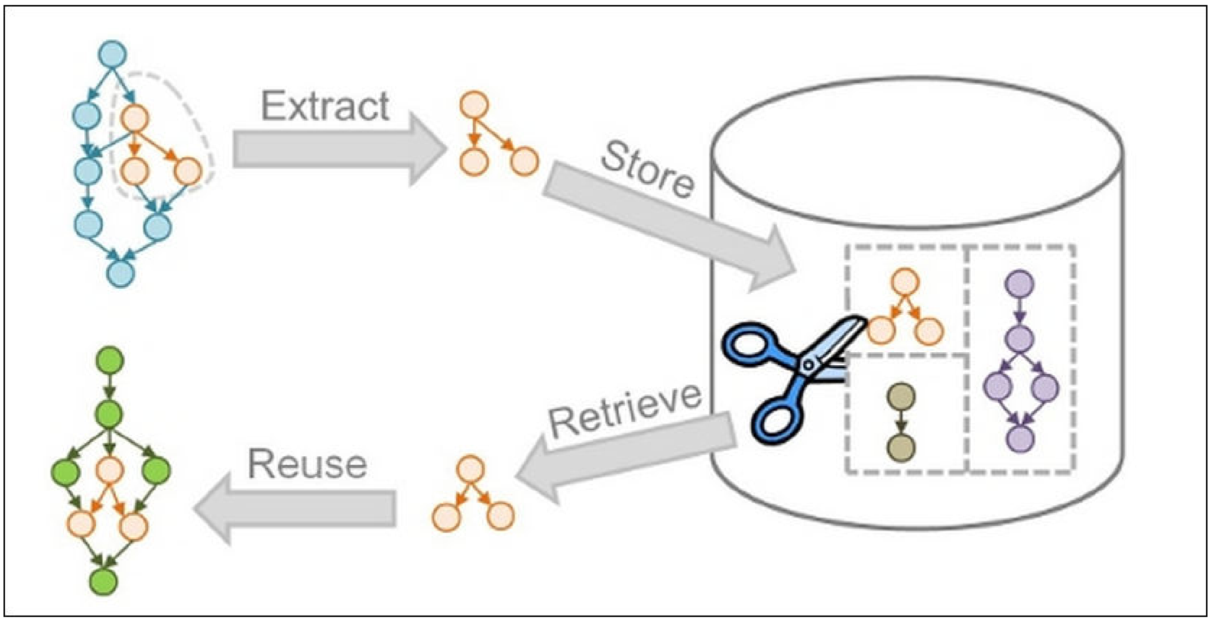


Figure 2.11: Concept of Fargmento[32]

[35]http://activemq.apache.org/

2.6 Technologies

This section discusses all necessary technologies and tools which realize the concepts in all chapters of this master's thesis. The work which is designed and implemented in this thesis is mainly based on these technologies and tools.

2.6.1 Eclipse BPEL Designer

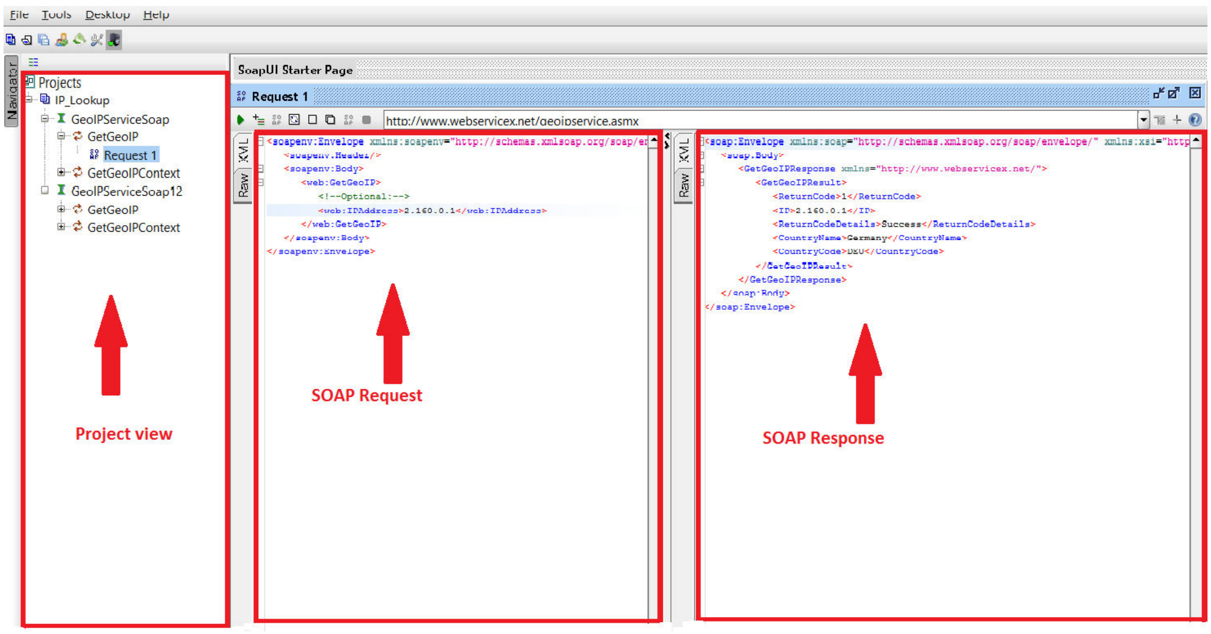
Eclipse BPEL Designer[33] was created by IBM and Oracle in May 2005. It is an extension to Eclipse[36] in order to support defining, editing, deploying and testing BPEL processes. The author categorizes the primary features of Eclipse BPEL Designer as follows:

- Designer: a Graphical Editing Framework (GEF)-based[37]editor which provides a graphical means to design the BPEL processes
- Model: an Eclipse Modeling Framework (EMF)[38] model that represents the designed BPEL process in an internal data format
- Validation: a validator which operates on the EMF model and produces errors and warnings in case of faulty specifications
- Run-time framework: an extensible framework which allows to deploy and execute the BPEL processes on a BPEL engine from the tools
- Debug: a framework which allows the user to step through the execution of a process, including support for breakpoints

2.6.2 SoapUI

SoapUI[34] is an open source platform for testing web services. It should be noted that working with SoapUI is simple because of its user friendly GUI. By creating a project in SoapUI and importing the WSDL files of the web services, SOAP request and response messages to different operations of a web service can be easily examined. SoapUI provides complete test coverage for SOAP-based web services[49] as well as REST-based web services[35]. Figure 2.12 shows the GUI of the SoapUI tool.

[36]https://eclipse.org/
[37]https://eclipse.org/gef/
[38]http://www.eclipse.org/modeling/emf/



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/en
   <soapenv:Header/>
   <soapenv:Body>
      <web:GetGeoIP>
         <!--Optional:-->
         <web:IPAddress>2.160.0.1</web:IPAddress>
      </web:GetGeoIP>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http
   <soap:Body>
      <GetGeoIPResponse xmlns="http://www.webservicex.net/">
         <GetGeoIPResult>
            <ReturnCode>1</ReturnCode>
            <IP>2.160.0.1</IP>
            <ReturnCodeDetails>Success</ReturnCodeDetails>
            <CountryName>Germany</CountryName>
            <CountryCode>DEU</CountryCode>
         </GetGeoIPResult>
      </GetGeoIPResponse>
   </soap:Body>
</soap:Envelope>
```

Figure 2.12: User Interface of SoapUI

In Figure 2.12 the functionality of a web service, so-called `GeoIPService`[39], is tested with SoapUI. This web service enables users to easily look up countries by IP address. In the SOAP request pallet, the IP address of `2.160.0.1` is sent to the web service operation. The SOAP response window shows the result of calling this function. As shown in the figure, this IP address belongs to `Germany`.

2.6.3 Apache Orchestration Director Engine (ODE)

Apache ODE[40] server executes and orchestrates business processes or workflows which are implemented with WS-BPEL standards (see Section 2.3.1). This engine talks to the web services inside the BPEL process, sends and receives messages and handles errors and exceptions. At the time of writing this thesis, the latest version of Apache ODE server is version 1.3.6.

2.6.4 Apache Tomcat

Apache Tomcat[41] is an open source server for running Java Servlets[42] and JavaServer pages[43]. It is released under the Apache License version 2[44]. The current version of Apache Tomcat is Tomcat 8.0.24. Apache Tomcat can be used as a container for deploying Java-based web services. Java-based web services are packaged in a WAR file format and then can be deployed on the Apache Tomcat server. In other words, a WAR is a web archive that can be deployed to any Java Enterprise Edition (EE) application server.

[39]http://www.webservicex.net/ws/WSDetails.aspx?WSID=64
[40]http://ode.apache.org/index.html
[41]http://tomcat.apache.org/
[42]http://www.oracle.com/technetwork/java/index-jsp-135475.html
[43]http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[44]http://www.apache.org/licenses/

2.6.5 Apache Axis

Apache Axis[45], and the the second generation of that, so-called Apache Axis2, are two well-known containers for web services. These two containers enable users to create, deploy, test and run web services. Apache Axis2 was implemented in both Java and C languages. Web services are packaged in Axis ARchive (AAR) file format and then can be deployed on the Apache Axis server. In other words, an AAR is a specific Axis2 artifact that can be deployed in an application server where there exists Axis2 standard web application, which is deployed already. This archive file contains all the implementations of a web service and corresponding configuration files. A good developer decides to build loosely coupled systems using services as a smallest piece of functionality. Therefore, it simplifies the development, provides the capability to develop web services in parallel, simplifies the testing, etc. In this case, it is more efficient to use a lightweight approach as Axis Archives instead of developing all these services as separate applications (separate WARs)[36].

2.6.6 Tools/Technologies chain diagram

Figure 2.13 illustrates how tools and technologies mentioned in Section 2.6 are related and work with each other. BPEL plans which orchestrate web services can be designed and implemented in the Eclipse BPEL Designer environment. Apache ODE engine which is deployed inside Tomcat is used for deploying BPEL workflows. After deployment of these workflows, these plans are considered as web services and can be started and tested with the SoapUI tool. Previously mentioned, Apache Axis2 is another engine for deploying web services. This engine can be installed and deployed inside the Apache Tomcat server. The functionality of each web service, which is deployed on the Apache Axis2 engine, can be tested with SoapUI. In other words, the SOAP messages and related input and output parameters, which are sent to and received from web services can be tested easily by the SoapUI tool.

[45]http://axis.apache.org/

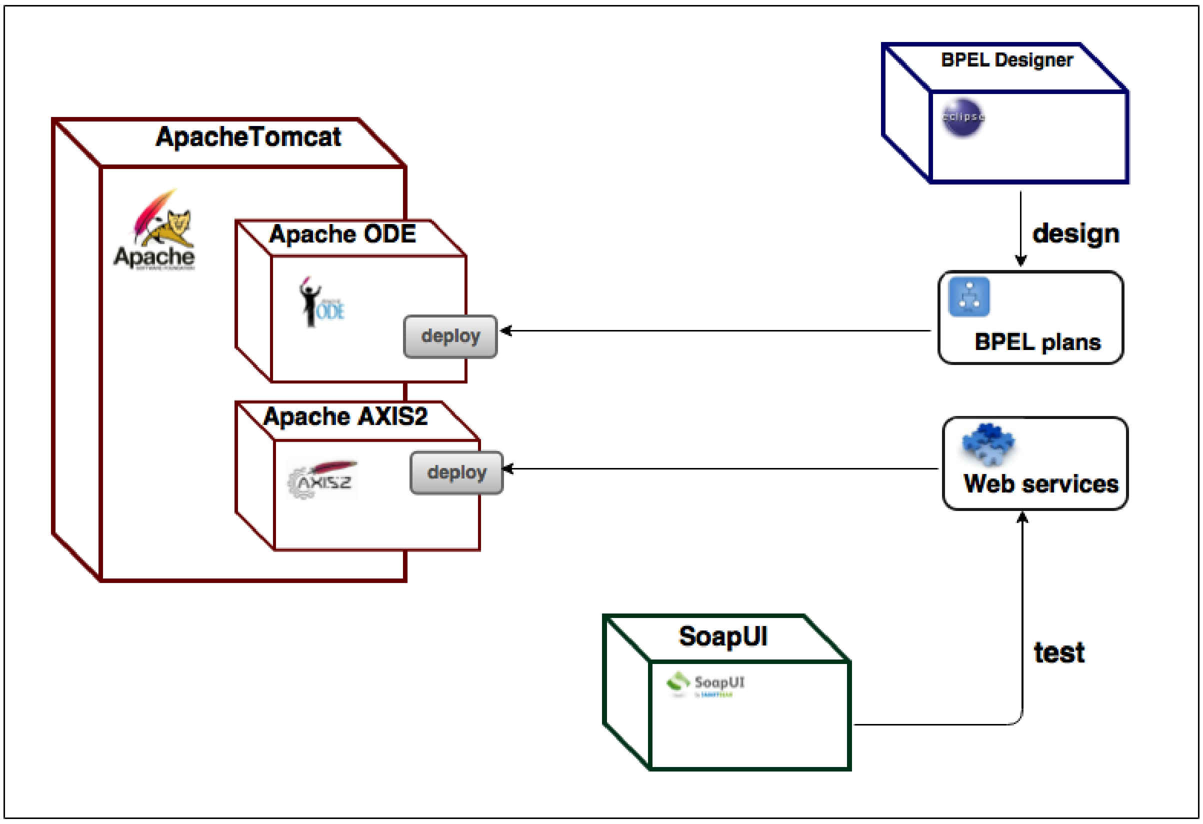


Figure 2.13: Relation between tools and technologies

Chapter 3

Related Work

This chapter discusses related work in the area of moving legacy applications into cloud environments. The main goal of this master's thesis is bringing a PANDAS bone simulation software to cloud infrastructures and deploy it as a SaaS solution. Running legacy simulation applications in cloud infrastructures can be evaluated from various aspects: (1) running simulation calculations in the cloud, (2) provisioning simulation software as well as other components and resources and (3) data provisioning for cloud-based simulation applications. In other words, these three aspects are specifically important with respect to the goals of this master's thesis which were discussed in Chapter 1. By discussing similar work related to these three aspects, the concepts of design and implementing the PANDAS bone simulation software as a SaaS solution in the cloud can be elicited. Section 3.1 discusses two simulation software, which are running in the cloud and may thus be used, in order that simulation calculations may benefit from the characteristics of cloud infrastructures. Then, an overview of various approaches for automating the deployment and management of simulation software in cloud infrastructures is given in Section 3.2. Section 3.3 specifies some related approaches for data provisioning and data management in cloud environment.

3.1 Running simulation calculations in the cloud

Running simulations in cloud infrastructures is one of the most important areas in cloud computing. There is a noticeable trend to move simulation calculations into cloud infrastructures. Simulation applications hosted on a cloud environment have many advantages compared to traditional and on-premise solutions as listed below [37]:

- Faster: In general, running simulation calculations needs massive processing power. By using powerful virtual infrastructures in cloud environments, cloud-based simulation software can run their actual calculations faster.
- Better design: Simulation is an iterative procedure and may have parallel and distributed processing. In other words, parallel simulation refers to the execution of a discrete simulation software across multiple processors. Cloud infrastructures enable scientists to have parallel and distributed processing for designing their applications and services more efficiently. Scientists may design simulation software better by running multiple simulations at the same time. Therefore, cloud infrastructures eliminate the need to purchase, operate and maintain computing resources at the local site for parallel and distributed simulations.

- Cheaper: Simulation software are overpriced. Moving simulation software to cloud infrastructures frees scientists to buy their own software and licenses. They can access different software in cloud environments, and then pay only for the amounts which they used (pay per use feature).
- Better collaboration: Some simulation software collaborate with each other in order to generate better calculations and results. Moving applications to cloud infrastructures provides possibilities for applications to collaborate easily over the Internet.
- Ubiquitous simulations: Simulation software can be accessed from all locations through the Internet.

With respect to the above-mentioned criteria in cloud environments, it is important to discuss these reasons for moving bone simulation software into the cloud. Most of the bone simulation software require lots of computational resources and dependant on a High-Performance Computing (HPC)[1] system. Accordingly, they need massive processing power in general. By using powerful virtual infrastructures in cloud environments, bone simulation software can execute their simulations faster and with better performance as well. It is not required to set up costly infrastructures and buy simulation software and licenses. Users of bone simulation software have to pay only for the amount which they used [Faster and Cheaper criteria].

Most of the simulation software need parallel and distributed processing. By running simulations in parallel and maybe on distributed nodes, the overall simulation time can be decreased and the performance and productivity can be increased. Cloud infrastructures enable scientists to have parallel and distributed processing for designing their applications and services. For example, a situation where the PANDAS simulation software is executed for multiple *motion sequences* in parallel. *Motion sequences* in the PANDAS bone simulation software refer to the boundary conditions, e.g. sleeping or standing. These boundary conditions define the state of a bone, which needs to be simulated. In the scope of this master's thesis, the PANDAS simulation software is executed for only one *motion sequence* at each execution. But, in the situation where the PANDAS simulation software is executed for multiple *motion sequences* in parallel, cloud infrastructures are beneficial for running multiple simulations at the same time [Better design criterion].

The PANDAS bone simulation software can be coupled with other simulation software, such as GNU Octave. The PANDAS bone simulation software is responsible for simulation on the tissue level. PANDAS calculates the structure changes within a human bone for one FEM element and then combines all the FEM elements within a loop activity. The simulation for the structure changes within a human bone can also be done on the cell level. For this purpose, the GNU Octave can be used. For visualizing the results of the PANDAS simulation, it can collaborate with some visualization tools at the end of simulation[18]. Consequently, the PANDAS bone simulation software may have collaboration with other software in order to generate better calculations and results. On cloud environments, cloud-based applications can communicate and collaborate easily with each other over the Internet. Therefore, moving the bone simulation software, especially PANDAS bone simulation software, into cloud environments enables the related cloud-based software to collaborate easily and efficiently [Better collaboration criterion]. Furthermore, these simulation software can be accessed from all locations through the Internet [Ubiquitous simulations criterion].

[1]http://searchenterpriselinux.techtarget.com/definition/high-performance-computing

Above-mentioned challenges are true for other simulation software as well. These criteria derive scientists and engineers to move their traditional and on-premise simulation calculations into cloud infrastructures in order to benefit from significant advantages of cloud environments. Many simulation software, such as MATLAB, were moved into the cloud for better performance. For instance, cloud-based MATLAB speeds up processing by accessing powerful computing resources on Amazon Elastic Compute Cloud (EC2)[2] or other cloud providers.

COMSOL Multiphysics[3] is another simulation software which has been moved into cloud environments. COMSOL is the Platform for Physics-based modeling and simulation. In other words, this software is a general purpose platform which is based on advanced numerical methods for simulating problems that may combine multiple physics. Similar to MATLAB, COMSOL requires HPC power for running its simulations. Therefore, scientists have moved this platform into cloud infrastructures. By running COMSOL on the Amazon EC2, users can benefit from virtual computing resources at very low and economical prices[38].

In general, it is possible to run the bone simulation with another cloud-based software, such as MATLAB or COMSOL. But the implementation of the numerical calculation of the bone simulation is very sophisticated, and to some part even domain- or problem-specific. Consequently, other software, such as MATLAB or COMSOL, do not provide this implementation so far. Accordingly, it is required to re-implement all or at least much of the numerical calculation within other software. To clarify, some complex MATLAB scripts is necessary, in the case of using MATLAB for bone simulation. Furthermore, this would have similar impacts on the data provisioning, since other software might also need different formats for the input data sources. As a result, moving the numerical calculation and all other related parts of the simulation process (especially data provisioning) to another software, such as MATLAB, is a more complex task than just bringing the PANDAS bone simulation software into the cloud. For this purpose, in order to provide bone simulation in cloud environments, the PANDAS bone simulation software is moved into cloud in the scope of this master's thesis.

3.2 Software provisioning in the cloud

In contrast to MATLAB and COMSOL applications, which are generic simulation platforms used by millions of engineers and scientists worldwide and which are thus already available in the cloud, the PANDAS bone simulation software is a specific purpose tool which has a limited number of users. A large number of automated and efficient approaches for software provisioning and management in cloud infrastructures were invented for standard and general purpose software like MATLAB. PANDAS can be used in some fields of engineering, e.g. in soil and rock mechanics or foam and tissue engineering. In other words, the usage of the PANDAS bone simulation software is restricted to a few number of fields. Therefore, this simulation tool has limited number of users in contrast to the general purpose software like Matlab[39]. To the extent of our knowledge, there does not exist a solution so far which automatically deploy and manage the proprietary legacy software PANDAS. Consequently, in the scope of this master's thesis, an automated approach for software provisioning as well as data provisioning for the PANDAS bone simulation platform is provided by using the OASIS

[2]http://aws.amazon.com/de/ec2/

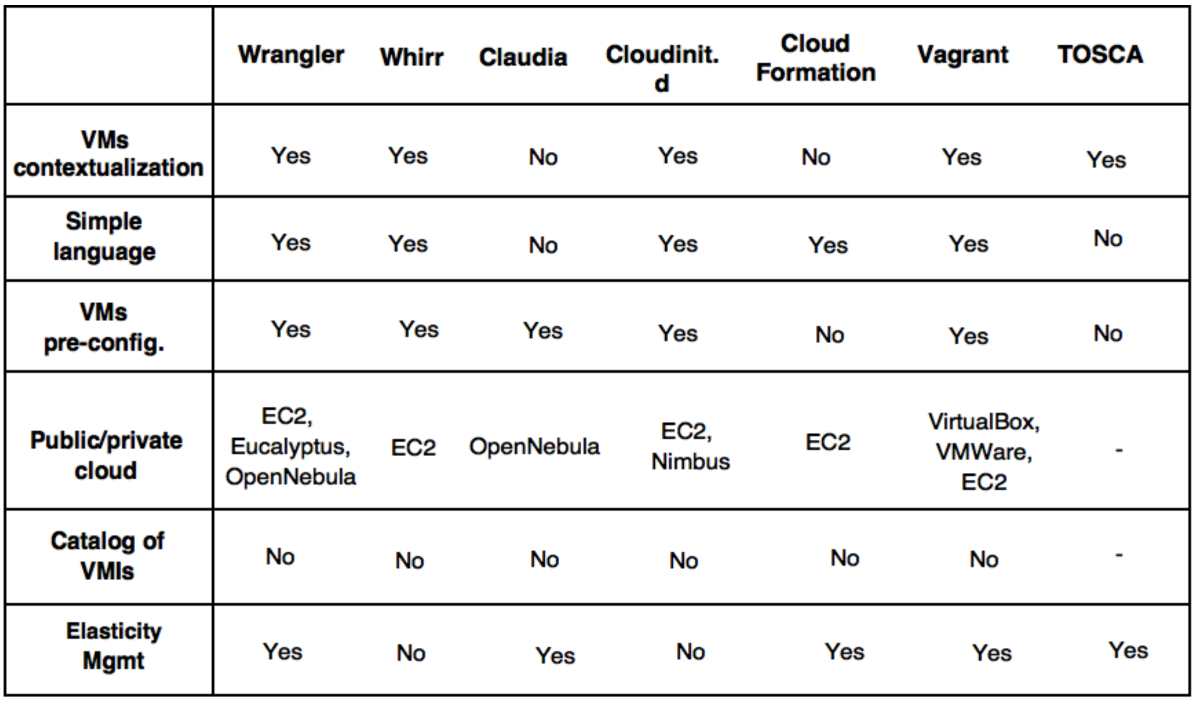
[3]http://www.comsol.com/comsol-multiphysics

TOSCA standard and the SIMPL framework. MATLAB and COMSOL software do not use the OASIS TOSCA standard in order to automate the deployment and management of their components in cloud environments. But, with the high productivity of the TOSCA standard, which is discussed in the following, the PANDAS bone simulation software uses the TOSCA standard for its software provisioning in the cloud.

As Figure 3.1 illustrates, many approaches have been developed and implemented to simplify the automatic deployment and configuration of virtual infrastructures. All the developed approaches can be evaluated from six different points of view, which are required for the deployment and management of virtual infrastructures in cloud environments. These criteria are as follows[7]:

1. The capability of customizing virtual infrastructures at the run-time by installing and configuring all the required software [VMs contextualization].
2. Having a simple and easy to understand language, which can be used to define all the hardware and software requirements specification [Simple language].
3. Restrictions such as pre-configuration of VMs [VMs pre-config.]
4. The capability of provisioning virtual infrastructures on public cloud providers (like Amazon Web Services[4]) as well as on the on-premise resource provisioning systems, such as OpenNebula[5] [Public/private cloud]
5. Existing of the Virtual Machine Image (VMI) in the application architecture to select the appropriate image for each VM [Catalog of VMIs].
6. Elasticity capability, both horizontal (adding/removing nodes) and vertical elasticity, which is growing or shrinking the capacity of the existing nodes [Elasticity Mgmt]

[4]http://aws.amazon.com/de/
[5]http://opennebula.org/



	Wrangler	Whirr	Claudia	Cloudinit.d	Cloud Formation	Vagrant	TOSCA
VMs contextualization	Yes	Yes	No	Yes	No	Yes	Yes
Simple language	Yes	Yes	No	Yes	Yes	Yes	No
VMs pre-config.	Yes	Yes	Yes	Yes	No	Yes	No
Public/private cloud	EC2, Eucalyptus, OpenNebula	EC2	OpenNebula	EC2, Nimbus	EC2	VirtualBox, VMWare, EC2	-
Catalog of VMIs	No	No	No	No	No	No	-
Elasticity Mgmt	Yes	No	Yes	No	Yes	Yes	Yes

Figure 3.1: Virtual infrastructure deployment tools comparison, cf.[7]

In [12], a method to automate the installation of VMs, application components as well as configuring and managing these components is provided. The system is called Wrangler and consists of three components: (1) *clients*, (2) *coordinator* and (3) *agents*. Figure 3.2 clarifies all these three components and their relationships.

- Clients are hosted on user's local machines and are responsible for sending requests to the coordinator in order to start, manage, query and terminate deployment processes.
- Coordinator is the central manager which accepts requests from clients, provisions VMs on appropriate cloud providers and controls application deployment on the installed VMs as well. As Figure 3.2 illustrates, all information required for these steps are stored in a database.
- Agents run on each VM node, which are provided by the coordinator. Agents are responsible for gathering information about the nodes, such as IP addresses, deploying appropriate software and applications on the nodes as well as sending reports to the coordinator about the state and health of each node.

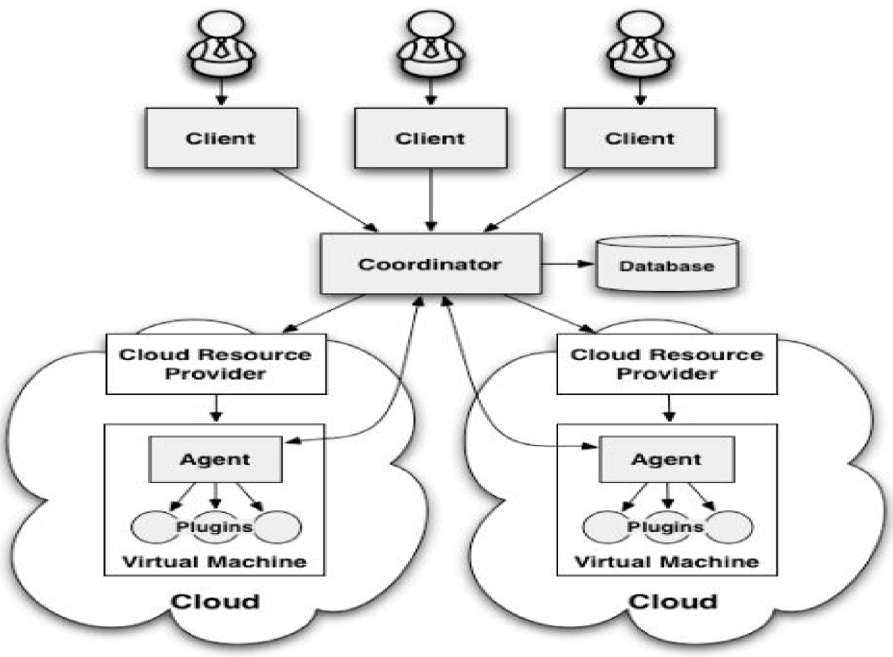


Figure 3.2: Wrangler architecture[12]

All required deployment configurations are provided by users in a simple XML file. This XML document consists of information, such as number of VMs, required information about cloud providers (e.g. Amazon cloud provider), some characteristics of VMs (e.g. image type), etc. As shown in Figure 3.2, each node has one or more plug-ins which provide the functionalities needed to be implemented by VMs. These plug-ins can be configured and customized by users with some related parameters. In other words, plug-ins are user-defined scripts which define the behaviour of a node. These scripts can be invoked and executed by the agents in order to configure and manage a node. There are different types of plug-ins, such as *application* plug-ins, *configuration* plug-ins, *data* plug-ins, etc., inside the Wrangler system. *Application* plug-ins are responsible for installing the software used by the application. *Configuration* plug-ins apply application-specific settings, and *data* plug-ins download and install application data. Listing 3.1 shows an example of a simple XML file which defines some configurations for application deployment in the Wrangler system. This file describes four nodes (one server and three clients), a cloud provider and other characteristics of the system[12].

```
<deployment>

<!-- Server node -->
<node name=server>
<provider name=amazon>
<image>ami-912837</image>
<instance-type>c1.xlarge</instance-type>
      ......
</provider>
<plugin script=nfs_server.sh>
<param name="EXPORT">/mnt</param>
</plugin>
```

```
</node>


<!-- Client nodes -->
<node name=client count=3 group=clients>
<provider name=amazon>
<image>ami-901873</image>
<instance-type>m1.small</instance-type>
   ......
</provider>
<plugin script=nfs_client.sh>
<param name="SERVER">
<ref node="server" attribute="local-ipv4">
</param>
<param name=PATH>/mnt</param>
<param name=MOUNT>/nfs/data</param>
</plugin>
<depends node=server/>
</node>

</deployment>
```

Listing 3.1: XML configuration file for deployment [12]

As mentioned already in Section 2.2, users can deploy and manage various components of composite applications in an automated way by using the OASIS TOSCA standard. TOSCA components and topologies as well as all related management plans are packaged in a CSAR file and executed on the OpenTOSCA environment. This package contains all the deployment and management steps, such as installing VMs on a cloud platform, deploying components of the application on the installed VMs, managing these components by running the TOSCA management plans, etc.

In contrast to the OASIS TOSCA standard, which does not need any pre-configurations for VMs, the Wrangler system requires some prior steps to configure VMs. For instance, this concerns some configurations like installing Wrangler agents on VMs and connecting them to the coordinator. In addition, Wrangler is only supported by EC2, Eucalyptus[6] and OpenNebula cloud providers. The TOSCA standard on the other hand, does not have any limitations in choosing the appropriate cloud provider. For TOSCA-based applications, any type of cloud providers (public, private, hybrid, community) can be used. Figure 3.1 depicts this criterion for other approaches as well. In opposition to the TOSCA standard, the selection of the most appropriate VMIs, which are based on user requirements, are necessary in the Wrangler approach. Bone simulations can be applied to different levels, such as tissue level, cell level, etc., within a human bone. Therefore, the situation for running a bone simulation in the cloud is dynamic and it is more efficient to use the TOSCA standard, which does not need any VMs pre-configurations. TOSCA also enables users to configure VMs at run-time with the help of some simple executable files[12].

Vagrant[40] is another approach for software provisioning in cloud infrastructures. It can be

[6]https://www.eucalyptus.com/

used for installing VMs on VirtualBox[7], VMware Fusion[8] and EC2. This approach benefits from a multi-machine[9] environment to manage a set of VMs. By using some provisioning tools, such as shell scripts, Chef[10] or Puppet[11], Vagrant enables users to deploy their software automatically on VMs. In contrast to the TOSCA standard, Vagrant provides high-level languages for contextualization of virtual infrastructures at run-time.

Like Wrangler, Vagrant approach requires pre-configurations of VMs. Vagrant is appropriate for simple and small virtual infrastructures. Therefore, Vagrant is not an efficient approach for moving the PANDAS bone simulation software, which is a complex and distributed software that can be coupled with other software like MATLAB, into cloud environments.

In conclusion, as Figure 3.1 illustrated, the TOSCA standard provides the contextualization of VMs at run-time. To clarify, the installed VMs in the TOSCA topology can be customized during the application execution. In other words, it is not necessary to apply all the configuration settings for VMs during the deployment phase. TOSCA uses some executable files to customize VMs at run-time instead of some high level languages, such as Puppet, Chef, Ansible[12], etc. Unlike other approaches as shown in Figure 3.1, TOSCA does not require any pre-configurations for VMs. Bone simulations, in particular the PANDAS bone simulation software, are highly dependant on run-time environments. It is not possible to define all configurations and settings for these simulations before running the actual simulations. As a matter of fact, it is important to use an approach like TOSCA, which enables users to have a dynamic situation for managing as well as configuring virtual infrastructures. Unlike Whirr[13] and CloudInit.d[14] approaches, the TOSCA standard has elasticity capability. For bone simulations in general, which have dynamic environments, the elasticity capability enables bone simulation applications to benefit from virtual resources more efficiently whenever they are demanded. Therefore, in the scope of this master's thesis, the OASIS TOSCA standard is used to move the PANDAS bone simulation software into cloud infrastructures. In general, OASIS TOSCA has some features which make this standard distinct from other existing approaches. Some of these attributes are discussed as follows[26]:

- TOSCA provides the orchestration of web services via existing workflow languages, like BPMN or BPEL. Therefore, TOSCA benefits from the properties of workflow languages, such as portability as well as automated execution. The PANDAS bone simulation software requires portability to move easily from one cloud provider to other cloud providers. As the bone simulation software are complex and distributed, by using workflow technologies the deployment and management of the software components can be automated.
- TOSCA standardizes the application typologies as well as management plans in a self-governing way. In other words, each node inside the TOSCA topology provides some functionalities to manage itself.

[7]https://www.virtualbox.org/

[8]http://www.vmware.com/

[9]Vagrant is able to define and control multiple guest machines per Vagrant file.

[10]https://www.chef.io/solutions/configuration-management/

[11]https://puppetlabs.com/

[12]http://www.ansible.com/home

[13]https://whirr.apache.org/

[14]http://www.nimbusproject.org/doc/cloudinitd/latest/

- TOSCA provides a syntax to define application components, their relationships, types and other properties as a graph. As the bone simulation software are complex and distributed, this feature helps developers to understand the components inside the TOSCA topology better. This feature also enables developers to provide better management operations for their components.
- TOSCA makes it possible to define various constraints on the nodes and components in the topology with some TOSCA policies. The PANDAS bone simulation software requires various non functional requirements, such as data security, data quality, etc. With the help of the TOSCA policies, these non functional requirements can be defined in a standardized way.
- TOSCA provides the ability to use virtual images as an IA inside the topology. Therefore, there is no need to have the catalog of VMI in some other approaches. As IAs are part of the TOSCA topology, they can be customized with some parameters and attributes at any time.
- TOSCA enables users to define non functional behaviours of web services, such as monitoring behaviour, by means of policies.
- TOSCA provides the dynamic and automated usage of application topology. Accordingly, it leads to propagate critical information at an appropriate time during the application life cycle. This feature is appropriate for dynamic situations, like running the PANDAS bone simulation software in the cloud.

3.3 Data provisioning in the cloud

Most of the simulation applications deal with huge, distributed and heterogeneous data sources. Therefore, having an efficient approach for provisioning and managing data sources may lead to high performance in cloud-based applications. The standard tools for defining and executing of ETL processes, e.g. IBM-InfoSphere Information Server[15], Pentaho[16] and Talend[17] put burden on scientists and engineers to define the low-level implementation details of data sources. To clarify, for retrieving the appropriate data from the results of PANDAS calculation, scientists have to define some complex SQL commands or scripts for filtering, aggregating and transforming the data. Therefore, they should be expert in the fields of their simulation applications in order to deal with these low-level details. As a matter of fact, these ETL tools do not provide the adequate abstraction supports for data provisioning and data management of simulation software in cloud environments. It is efficient to have more abstract approaches for data management and data provisioning of the PANDAS bone simulation software in the cloud. Accordingly, in the scope of this master's thesis, the SIMPL framework which was described in section 2.5 is used for data provisioning and deprovisioning of a PANDAS bone simulation software.

The Scientific Data Management (SDM) center[18] offers an end-to-end data management approach for providing data as well as analyzing and visualizing data generated by simulations and experiments. This approach consists of powerful means for accessing data. To put it another way, the SDM approach offers various algorithms, database indexes, etc., for providing

[15]http://www.ibm.com/software/data/integration/info_server/
[16]http://www.pentaho.de/
[17]http://de.talend.com/
[18]https://sdm.lbl.gov/sdmcenter/

and analyzing huge and heterogeneous data sources. SDM focuses on analyzing data that may be generated by simulation software as their output, but not on providing input data for simulations in a generic way. The latter issue is exactly where the SIMPL framework comes into play[19].

Open Grid Services Architecture – Data Access and Integration (OGSA-DAI)[19] framework provides an innovative approach for accessing and managing data resources via web services on the web or within grids or clouds. It encapsulates huge and heterogeneous data sources and enables users to query, update, transform and combine data. This approach frees users to deal with low level details of the location, structure or format of data sources which are transferred or queried. In contrast to the SIMPL framework, the abstraction provided by the OGSA-DAI framework is one of the limitation of using this approach. In other words, the abstraction level depends on abstractions which is supported by each web service[41]. On the other hand, the SIMPL framework provides a unified access to heterogeneous data sources. There exists abstract supports in the SIMPL framework with the help of DM activities and DM patterns. In other words, SIMPL provides a generic and unified abstraction mechanism[16].

There are various scientific WfMS which support running of workflow-based applications in the cloud. To clarify, the scientific workflow management system Pegasus[20], automatically handle all the heterogeneous input and output data sources which are required for running workflow-based applications. In other words, it locates data sources and computational resources, which are required for the workflow execution, automatically. Pegasus provides capabilities for users to form and manage their abstract workflows without worrying about the details of the execution environment. Abstract workflows are workflows which contain just information about the overall tasks, which should be done in the workflows, as well as how these tasks are interconnected. To put it another way, abstract workflows do not provide any information about how input data sources are actually delivered. Furthermore, they do not provide any information regarding the implementations of these tasks. Pegasus workflow management system transforms abstract workflows into executable workflows. During this transformation, Pegasus adds some data management operations for distributed and heterogeneous data sources to the workflows. These data management operations have to be implemented by the modeler himself. In contrast to Pegasus, SIMPL provides an abstraction level via DM pattern, which are automatically converted to some executable workflows and frees modelers to implement some additional data management operations.

Data management and data provisioning for heterogeneous data sources in cloud environments is a hot research topic these days. For managing heterogeneous and distributed data in cloud environments, traditional relational databases cannot provide a high level of scalability, availability and performance which is required for cloud-based applications. NoSQL database systems[21] overcome some restrictions and limitations of traditional relational databases. These databases are pattern free and accept key-value pair for their storage mode. A typical NoSQL database can be extended transparently, deployed on cheap hardware and are appropriate for distributed storage[20]. However, this approach does not deal with data management abstractions as a generic and consolidated solution, which can be provided in the SIMPL framework.

[19]http://www.ogsadai.org.uk/index.php
[20]http://pegasus.isi.edu/
[21]http://nosql-database.org/

In [11], a workflow system, which provides a multi-level abstraction for decomposing tasks within workflows from required resources and services, is introduced. General Workflow Execution Service (GWES) enables users to manage execution and composition of processes within the workflows with a high level of abstraction in distributed environments, such as SOA, Cluster, Grid, or cloud environments. GWES maps abstract workflows, which have been constructed by users, to some concrete and executable workflows. This transformation provides adequate and available data resources for each task within workflows. When a user sends a request for some data resources, the request is first mapped to an abstraction layer. Then, the abstraction layer locates the appropriate candidates for the requested resource. One of the candidates is selected and the task is mapped to the selected resource. Unlike SIMPL which provides an abstract mechanism via DM activities and DM patterns, GWES supports data management abstractions by providing an abstraction layer between user's requests and actual resources.

Chapter 4

Mapping PANDAS into cloud

This chapter provides an overview about requirements and necessary components for moving the PANDAS bone simulation software into cloud infrastructures. As already mentioned in the previous chapters, the main goal of this master's thesis is to design and implement different variants of a TOSCA Service Template for provisioning and executing the PANDAS bone simulation software in cloud environments. Section 4.1 discusses the concept of application topology for the PANDAS bone simulation software in the OASIS TOSCA standard. The management plans, which are included in a TOSCA-based Service Template for the PANDAS bone simulation software, are designed and developed in Section 4.2. Following that, in Section 4.3 an approach for integrating the SIMPL framework with the OASIS TOSCA standard is discussed.

4.1 PANDAS application topology

As already mentioned in the previous chapters, the main contribution of this master's thesis is bringing the traditional and on-premise PANDAS bone simulation software into the cloud and turning the bone simulation into a fully integrated SaaS solution. The manual installation and configuration of the PANDAS bone simulation software in cloud infrastructures is burdensome and time-consuming. To automate software provisioning in the cloud for PANDAS, the OASIS TOSCA standard, which was described in Section 2.2, is used in this work. TOSCA automates the deployment and management of PANDAS components in arbitrary cloud infrastructures with reduced efforts and costs. A TOSCA-based Service Template for the PANDAS bone simulation software consists of an application topology as well as management plans. For developing PANDAS as a SaaS solution, three VMs are provided. As Figure 4.1 depicts, a Linux-VM is deployed on a cloud infrastructure such as OpenStack cloud provider which runs the actual calculation of PANDAS. This VM is provisioned dynamically during the execution of the PANDAS Service Template. The PANDAS software package and a related PostgreSQL database for storing the results of PANDAS are installed on this VM with the help of some management workflows. The second VM, which is a Windows-VM using the operating system Microsoft Windows Server 2012 R2 Standard 64-bit, is deployed on the OpenStack cloud provider. This VM consists of the SIMPL framework with the ODE-PGF engine. Furthermore, this VM is also used for storing the results of PANDAS simulation. In other words, different variants of the *DataDeprovisioning* workflow, which is discussed in Section 4.2.5, return the result data of a PANDAS simulation back to the user and store them in some files and directories on the Windows-VM. This VM is part of the static infrastructures, which have been deployed already. As the figure depicts, The OpenTOSCA environment is installed on the third Linux-VM in the topology. This VM is used to execute the PANDAS Service Template, which is exported from the Winery tool as a CSAR. Like the second above-mentioned VM, this VM is part of the static infrastructures.

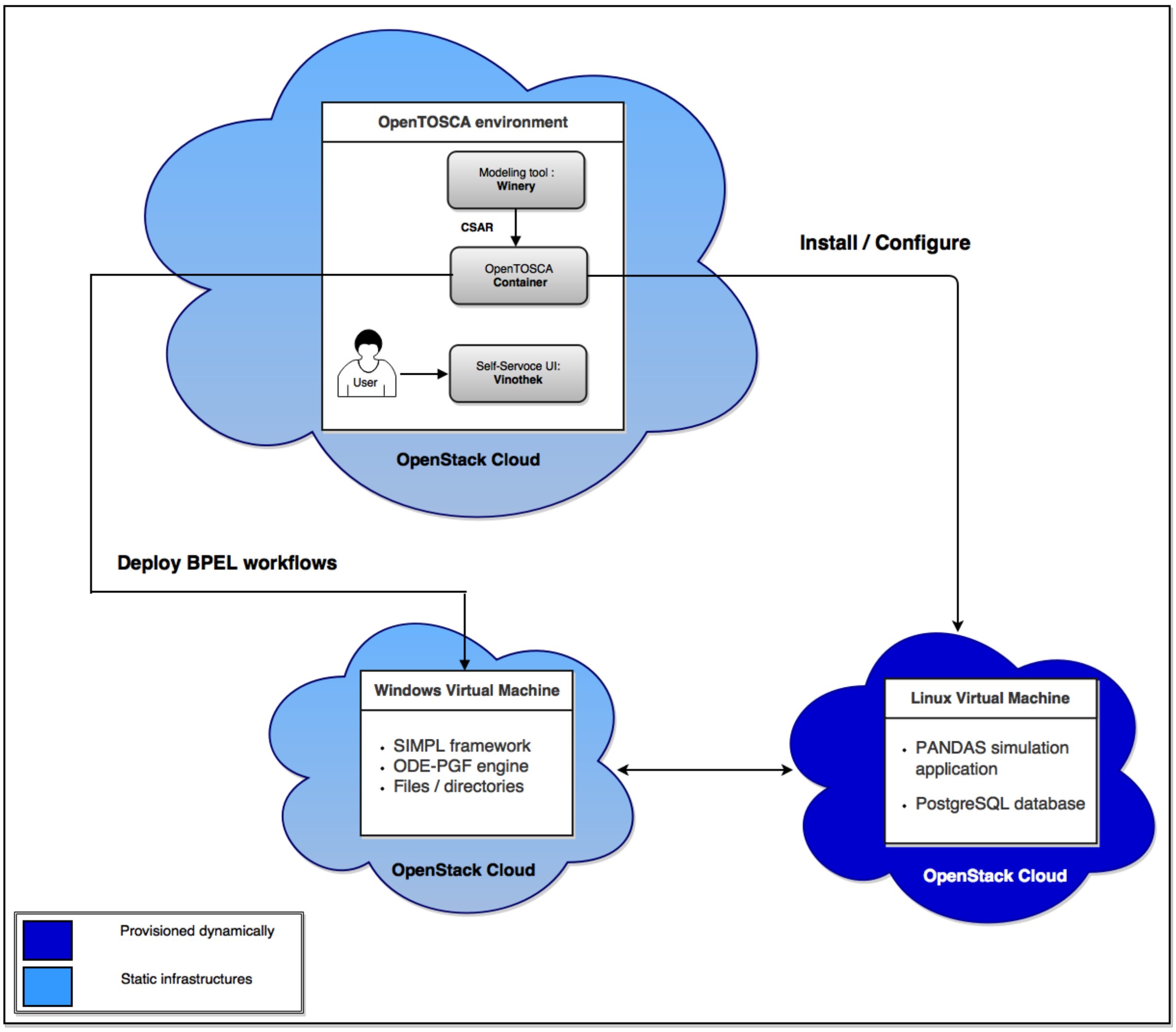


Figure 4.1: Cloud-based topology for PANDAS Service Template

The components of the first VM, which is provisioned dynamically by executing the PANDAS Service Template inside the OpenTOSCA environment, can be modelled as Figure 4.2. In other words, the developed CSAR file for the PANDAS bone simulation software is executed inside the OpenTOSCA environment. Based on the service topology and the management plans inside the CSAR, a new VM is installed on the OpenStack cloud provider. Then, a Linux operating system will be hosted on the created VM. Following that as the figure shows, the source code of the PANDAS bone simulation software and the related PostgreSQL database is installed on the operating system. The current implementation of the PANDAS web service installs the actual source code and the related PostgreSQL database on the same VM. As discussed in [18], another TOSCA-based service topology for the PANDAS bone simulation software with two VMs, one for the PANDAS software and one for the related database, can be considered as well. In the scope of this master's thesis, only one VM is provisioned for both the PANDAS software and the related PostgreSQL database.

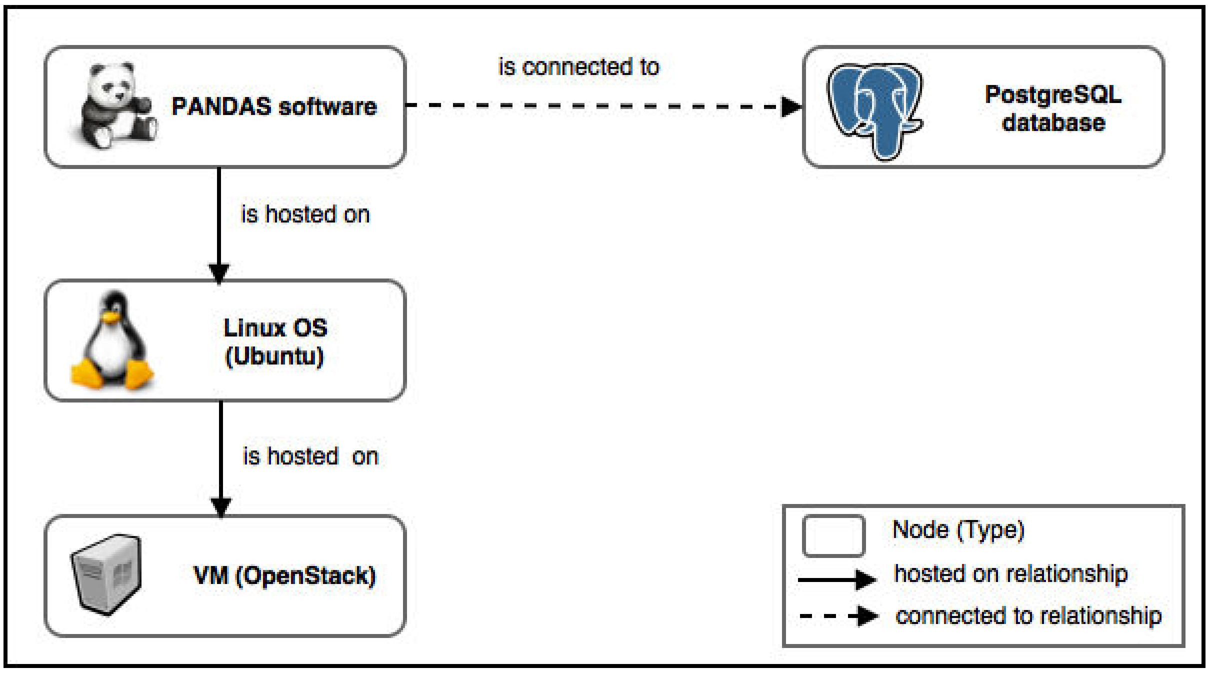


Figure 4.2: TOSCA-based service topology for PANDAS

Regarding to the TOSCA-based service topology of the PANDAS bone simulation software, several IAs are used in order to develop PANDAS as a SaaS solution in cloud environments. The first IA, so-called *InstallOpenStackVM*, is a Java-based implementation which has been developed already and is only used in this work. This IA contains several functions such as *InstallVMwithCustomKeypair*, *InstallVMwithGeneratedKeypair*, *InstallVMwithCustomFlavor*, etc., for installing a new VM on the OpenStack cloud provider. This IA contains a termination function as well, which terminates VMs inside the topology. The actual implementation of this web service can be found on GitHub[1].

SSH-IA is another IA inside the PANDAS service topology. This IA enables users to customize configurations of the installed VMs by making a SSH connection to the servers and running some shell commands on the servers. To clarify, there are two issues in the current implementation of the PANDAS Web Service which can be solved with this IA:

1. After each calculation of PANDAS instances, Apache Tomcat should be restarted. Otherwise, the calculation might not be executed properly. For this purpose, a shell command which restarts Tomcat is sent via the *SSH-IA* to the server.
2. After each calculation of PANDAS instances, the PostgreSQL database should be restarted. The *SSH-IA* is used to send the related shell command, which is `sudo service postgresql restart`, to the remote server.

Another IA, so-called *ODE-Service* IA, is designed and implemented in this master's thesis. Some PANDAS management plans, which are discussed completely in the subsequent sections, need to be deployed on a workflow engine, like Apache ODE-PGF, for execution. The *ODE-Service* IA is a SOAP-based web service which consists of *Deploy* and *Undeploy* functions for deploying and undeploying some BPEL workflows on and from an Apache ODE-PGF

[1]https://github.com/tosca-types/openstack

engine, respectively.

4.2 PANDAS management plans

OASIS TOSCA management plans are XML-based workflows which provide automation for deployment, configuration as well as management of components inside the application topology. The TOSCA standard uses existing workflow languages, such as BPMN or BPEL. In the scope of this master's thesis, BPEL is used as the workflow language for modeling and executing TOSCA-based management plans[5]. Figure 4.3 illustrates high-level activities inside the PANDAS management plans.

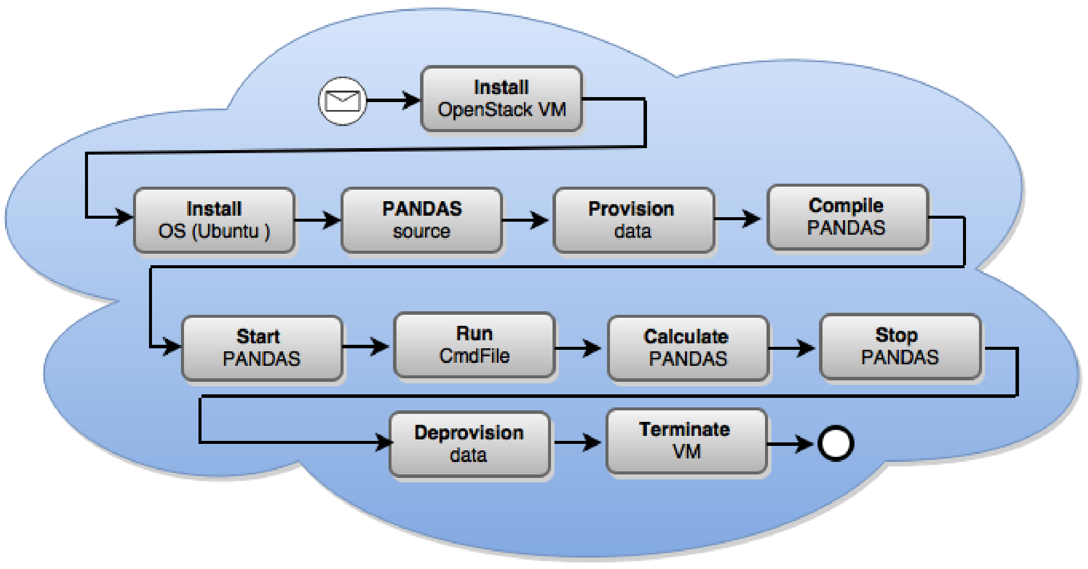


Figure 4.3: Activities inside the PANDAS management plans

As shown in Figure 4.3, a VM with a Linux operating system is installed on the OpenStack with the help of *InstallOpenStackVM* IA. Then the new installed VM can be configured via the *SSH-IA*. As the figure shows, the third activity in the workflow copies the PANDAS source code on the created Linux-VM [`PANDAS source`]. Previously mentioned, the PANDAS bone simulation software requires huge and heterogeneous input data sources for its calculation. Therefore, the SIMPL framework is used in this step to provision huge and distributed data sources [`Provision data`]. At the end of data provisioning phase, the PANDAS source code and the provisioned data are compiled together [`Compile PANDAS`]. Then, PANDAS instances are started [`Start PANDAS`]. In `Run CmdFile` step, some commands are sent to PANDAS instances, so they can execute their cmd-files and read some input files properly. After that, the actual calculations of instances are started [`Calculate PANDAS`]. Following that, PANDAS instances are stopped simultaneously [`Stop PANDAS`], and the workflow continues to the next step which is data deprovisioning [`Deprovision data`]. In this phase, the results of PANDAS simulation are transferred from the Linux-VM to the Windows-VM via the SIMPL framework. The activities inside data provisioning, calculation and data deprovisioning phases need to be executed on the Apache ODE-PGF engine. For this purpose, the *ODE-Service* IA is used to deploy these workflows on that engine. Then, the whole workflow is terminated in termination phase. In this phase [Terminate VM], the installed VM, which was created in the first step, is terminated and all the workflows on the ODE-PGF engine are undeployed.

Figure 4.3 depicted high-level activities inside the management plans of the PANDAS Service Template. In the scope of this master's thesis, six different types of workflows as management plans have been designed and implemented in order to fulfill all the above-mentioned activities. These workflows are discussed in detail as follows:

1. Provisioning plan which instantiates VMs and prepares some configurations of these VMs for installing the PANDAS software components via the next plan.
2. PANDAS software provisioning plan which sets up the necessary simulation software components, in particular the PANDAS calculation tool and different kinds as well as configurations of database systems that store the result data of PANDAS.
3. Different variants of management plans that provide and prepare heterogeneous input data of the simulation in order that PANDAS can properly ingest these data.
4. A management plan that orchestrates the simulation calculation in the PANDAS bone simulation software.
5. Different variants of management plans that return the result data of PANDAS back to the user.
6. Termination plan which terminates the installed VMs in the first plan, and undeploys all other software components.

As it is discussed in the subsequent sections, some of these workflows have to be designed and implemented in multiple variants in the scope of this master's thesis. In other words, different variants of plan number two, three and five are designed and implemented in this work. In the following, the middle-level design of the PANDAS management workflows are discussed in detail.

4.2.1 Plan number one

The first workflow, so-called *InfrastructureProvisioning* workflow, is shown in Figure 4.4. This plan is started by receiving a message and then continues to the first activity which is *InstallOpenStackVM* for installing a new VM on the OpenStack cloud provider. The appropriate operation of the *InstallOpenStackVM* IA is called in this step. Type of the image for the operating system, which should be installed on the created VM, can be passed to the workflow via an input parameter. If this value is not received from input parameters, the default image type is used. All operations of the *InstallOpenStackVM* IA require at least two input parameters, so-called `Credentials` and `EndpointsAPI`. These values should be passed to the workflow via input parameters. The format of these parameters are described on GitHub[2]. In other words, `Credentials` consist of the tenant ID, user name as well as password of the registered user on the OpenStack. The `EndpointsAPI` on the other hand, defines the location of some remote methods of the OpenStack API, which are accessible to external clients. It is an entry point to the OpenStack system for external clients. This entry point provides a REST-based API to call different operations inside the OpenStack. The next activity in the workflow in-

[2]https://github.com/tosca-types/openstack

vokes the *SSH-IA* to run some shell commands on the installed VM remotely. These shell commands configure the VM. To clarify, as discussed in Section 4.1, this IA is used to restart a Tomcat server and a PostgreSQL database remotely. This workflow includes the deployment of other BPEL plans on the Apache ODE-PGF engine of the SIMPL framework inside the Windows-VM on the OpenStack cloud provider. For this purpose, the *ODE-Service* IA is used to deploy the plans on the ODE-PGF engine. As shown in Figure 4.4, the last step of this workflow invokes the *SIMPLResourceManagement* web service to register the IP address of the new installed Linux-VM to the SIMPL framework. A new entry for the IP address of the created Linux-VM should be inserted into the data source table of the SIMPL database. In other words, by adding the new IP address to the PostgreSQL database of the SIMPL framework, the data sources and the results of PANDAS calculation are transferred correctly to the right VM in *DataProvisioning* and *DataDeprovisioning* workflows, respectively (see Section 4.2.3 and 4.2.5). For this purpose, some other input parameters are passed to the workflow. The `DataSourceID` variable defines a unique ID for the entry, which needs to be registered to the PostgreSQL database of the SIMPL framework. The `DataSourceName` variable provides a name for this entry, and the `DataSourceAddress` variable is the IP address of the created VM which needs to be registered. As results of the workflow execution, the links of deployed BPEL workflows on the ODE-PGF engine, which are used later for starting these workflows, can be retrieved.

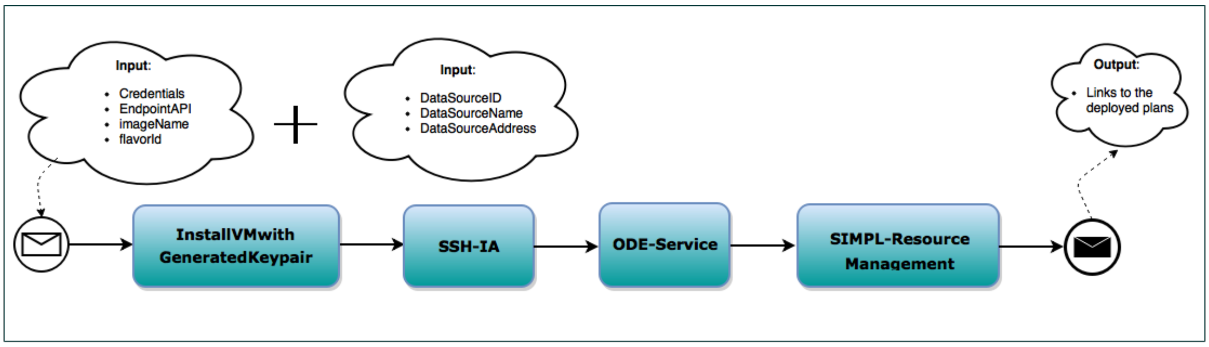


Figure 4.4: InfrastructureProvisioning plan

4.2.2 Plan number two

The second workflow, so-called *PandasSoftwareProvisioning* plan, is a simple plan which is used for preparing a PANDAS instance. In other words, each execution of this workflow prepares one PANDAS instance. As mentioned already, two PANDAS instances are required in the scope of this master's thesis, one for the mechanical and one for the chemical calculation. Accordingly, this workflow is called two times in the PANDAS Service Template. Different variants of this workflow are designed and implemented in the scope of this master's thesis. In the subsequent sections, different variants of this workflow are discussed in detail. All the variants use the below-mentioned two main activities for preparing each PANDAS instance. The following operations are part of the PANDAS web service and can be called in this workflow:

- *Platform_provisioning*: this operation mainly creates a unique ID and a basic directory structure on the Linux-VM for the PANDAS software instance.

- *Pandas_source*: this operation unpacks the archive of the PANDAS source code to the basic directory structure, which was created in the previous operation.

4.2.2.1 Plan number two - First version

After installing a Linux-VM on the OpenStack cloud provider in the *InfrastructureProvisioning* workflow, the floating-IP address of the installed VM is required to install the PANDAS bone simulation source code and related PostgreSQL database on that VM. As Figure 4.5 illustrates, this IP address is passed to the workflow via an input parameter. A unique ID and a path to a basic directory structure for the PANDAS software instance will be created as the result of executing this workflow.

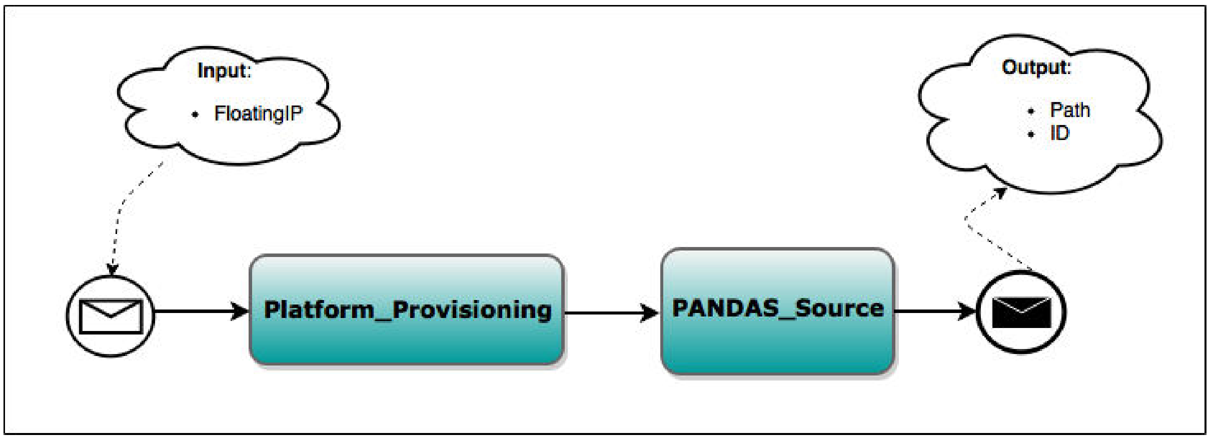


Figure 4.5: PandasSoftwareProvisioning plan - First version

4.2.2.2 Plan number two - Second version

Previously mentioned, the PANDAS bone simulation software uses a PostgreSQL database as the storage system. In *PandasSoftwareProvisioning* plan, the PANDAS source code as well as its related PostgreSQL database with different configurations have to be installed on the VM. Conceptually, different architectures for the PANDAS Service Template which can affect this workflow can be considered as follows:

- If there are different configurations for the database system: in the current implementation of the PANDAS web service, a PostgreSQL database is used as the storage system. In the case that we have different configurations for the PostgreSQL database, different changes are applied to the database system without any need to change the current implementation of the PANDAS web service.
- If the PostgreSQL database is not deployed on the same Linux-VM of PANDAS instance: two VMs should be installed on the cloud provider, one for the PANDAS instance and one for the database system. In this situation, a different and distributed architecture should be considered for the PANDAS bone simulation tool. The implementation of the PANDAS web service and of the PANDAS source code require modifications. In the application topology of the PANDAS Service Template, a separate VM is required for installing the PostgreSQL database of PANDAS.

- If the PANDAS bone simulation software uses different kinds of database systems for storing the results of simulation: for instance, instead of a PostgreSQL database, a different database, e.g. a Microsoft SQL server[3], may be used. In this case, some changes should be applied to the PANDAS source code and PANDAS web service in order to work with this database properly, as the current implementation of PANDAS and of its web service is tightly coupled with the PostgreSQL database system.
- If PANDAS instances are deployed on different VMs: if two PANDAS instances in the scope of this thesis, one for the mechanical and one for the chemical calculation, require two different VMs. In this case, a different and distributed architecture should be considered for the PANDAS bone simulation tool. The implementation of the PANDAS web service and of the PANDAS source code require modifications. Some mechanisms have to be considered to connect the VMs with each other for calculating the final results of simulation.
- The above-mentioned variants might also be combined with each other. For instance, a different database system (not a PostgreSQL database) with a different configuration that is even deployed on a different VM (not the same VM as PANDAS). In this situation, implementations of the PANDAS source code and of the PANDAS web service are more complicated.

There exist various configuration parameters in the PostgreSQL database system which affect the behaviour of the system. All these parameters can take a value of one of these five types: Boolean, integer, floating point, string or enum[4]. There are some various means for changing these parameters in the database. The two most common ways are: (1) editing the related file, so-called *postgresql.conf*, which is inside the data directory of the PostgreSQL database and (2) sending some queries to the database in order to change these parameters. Listing 4.1 indicates an example of how this *postgresql.conf* file might look like inside the data directory of the PostgreSQL database. Each line belongs to one parameter's configuration. Hash marks are used to define a comment and all the values of the parameters must be single-quoted. Editing the *postgresql.conf* file is a traditional approach for changing configurations of the PostgreSQL database. With respect to the TOSCA management plans, which are used in this master's thesis, the second approach for customizing the database is more efficient. In other words, it is more generic to send some queries within the TOSCA management workflows to change configurations than the manual configurations of the *postgresql.conf* file. In the following, some configuration parameters which can affect the behaviour of a PostgreSQL database are discussed. In general, there are some settings and parameters for *Connections and Authentication*, *Resource Consumption*, *Error Reporting and Logging*, *Replication*, *Error Handling*, etc., which can be modified[23].

```
// This is a comment
log_connections = 'yes'
log_destination = 'syslog'
search_path = '$user, public'
shared_buffers = '128MB'
```

Listing 4.1: postgresql.conf file [23]

[3]http://www.microsoft.com/en-us/server-cloud/products/sql-server/
[4]https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

Table 4.1 lists some common parameters in a PostgreSQL server which can be used to change configurations and settings. It is important to note that, a PostgreSQL database contains a large number of parameters which can affect the system's behaviour. Based on user's requirements in the case of PANDAS bone simulation software, these parameters can be modified for the PANDAS PostgreSQL database. For instance, `max_connections` parameter can be used to change the maximum number of users that can access the PostgreSQL server concurrently. To provide a better security for the data inside the PANDAS PostgreSQL database, various authentication parameters can be configured. Replication parameters can be customized as well for replicating the results of PANDAS simulation in multiple places. Table 4.1 indicates only a subset of these parameters which can be used by users to customize the database system[23].

listen_addresses(string)	The address on which the server should listen
port(integer)	The TCP port which server should listen on
max_connections(integer)	The maximum number of concurrent connections
authentication_timeout(integer)	Maximum time for client authentication
ssl(boolean)	To enables SSL connections
shared_buffers(integer)	The amount of memory for shared memory buffers
wal_level(enum)	How much information is written to the WAL
archive_mode(boolean)	If on, segments are sent to archive storage
hot_standby(boolean)	Can or cannot run queries during recovery
enable_mergejoin(boolean)	Enables or disables the usage of merge-join
enable_nestloop(boolean)	Enables or disables the usage of nested-loop join
geqo(boolean)	Enables or disables genetic query optimization
log_destination(string)	Desired log destination
log_filename(string)	The file names of the created log files
log_connections(boolean)	If on, each connection to the server is logged
log_error_verbosity(enum)	The amount of detail written in the server log
log_statement(enum)	Controls which SQL statements are logged
search_path (string)	The order in which schemes are searched
DateStyle(string)	Sets the display format for date and time values
TimeZone(string)	Sets the time zone
deadlock_timeout(integer)	Time to wait on a lock before checking for deadlock
exit_on_error(boolean)	If true, any error will terminate the current session
block_size(integer)	Specifies the size of a disk block
server_version(string)	Reports the version number of the server
trace_notify(boolean)	Generates a great amount of debugging output

Table 4.1: PostgreSQL configuration parameters[23]

Previously mentioned, different variants of *PandasSoftwareProvisioning* workflow are designed and implemented in this thesis in order to install the PANDAS source code as well as different configurations for its PostgreSQL database system. The first version of this workflow was discussed in the previous section in Figure 4.5. In the current implementation of the PANDAS web service, there exists an operation which is used for sending a query to the PostgreSQL database. By sending different queries, in order to change some configuration parameters (as Table 4.1 shows) of the PostgreSQL database, different variants of *PandasSoftwareProvisioning* plan can be designed and implemented. Accordingly, as Figure 4.6 illustrates, in the second version of *PandasSoftwareProvisioning* workflow, the query is received as as input parameter and then applied to the database. As the figure shows, the `Send_Update_Query` operation of the PANDAS web service is used to send some queries to the database. The next activities are the same as in the first version. It is not important to call this operation at the beginning of this workflow. In other words, the queries can be sent to the PostgreSQL database at the end of the workflow after invoking the `Platform_Provisioning` and the `PANDAS_Source` operations.

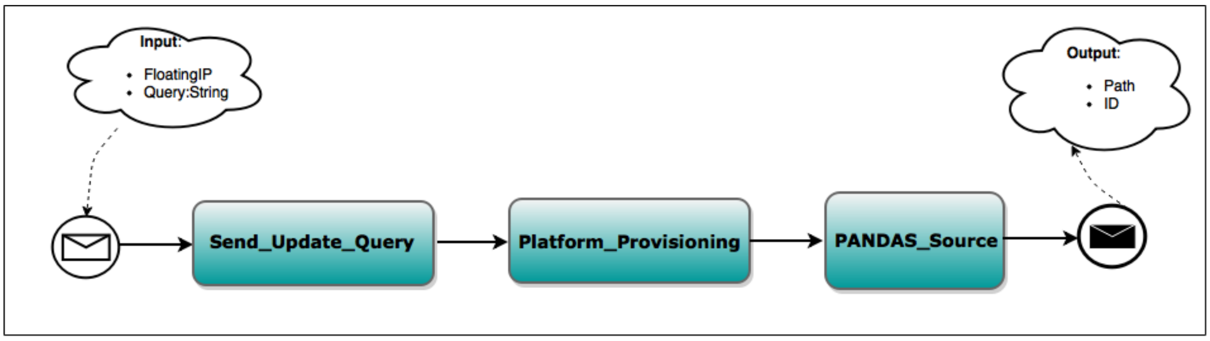


Figure 4.6: PandasSoftwareProvisioning plan - Second version

4.2.2.3 Plan number two - Third version

By using the *SSH-IA* which is used as well in the first plan of the PANDAS Service Template, so-called *InfrastructureProvisioning* workflow (see Section 4.2.1), different SQL commands can be sent to the PostgreSQL server in order to change settings and configuration parameters. Hence, another variant of *PandasSoftwareProvisioning* plan can be designed and implemented by using this IA. As Figure 4.7 illustrates, the `Run Script` operation of this IA is called in order to customize some settings of the PostgreSQL database. Therefore, the appropriate query is received from the input parameters of the workflow and then used as an input for the `Run Script` operation of *SSH-IA*. Besides the query, the `Run Script` operation requires other parameters, such as hostname, SSKkey and SSKUser. These parameters are received from the input parameters of the workflow as well. The rest of the workflow is the same as in the previous versions. Like previous version, it is not important to invoke this operation at the beginning of this workflow. In other words, the queries can be sent to the PostgreSQL database at the end of the workflow after invoking the `Platform_Provisioning` and the `PANDAS_Source` operations.

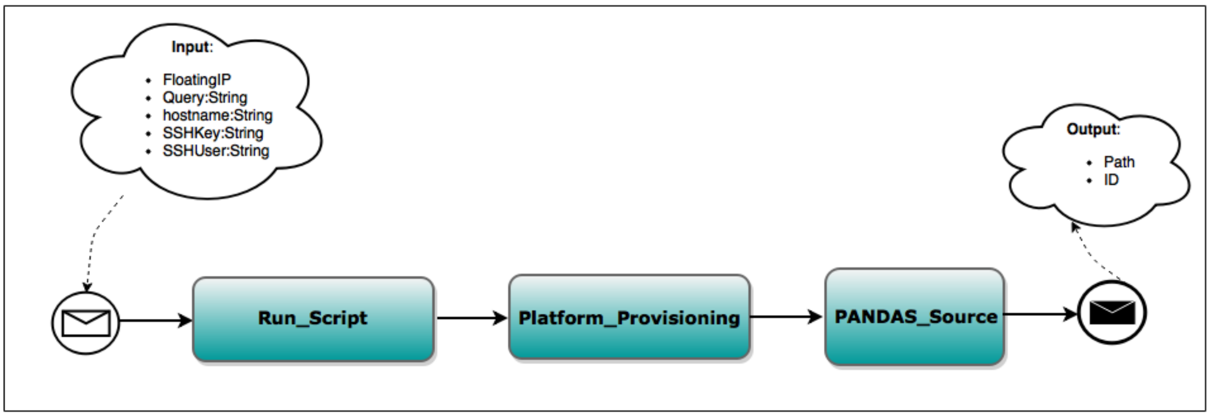


Figure 4.7: PandasSoftwareProvisioning plan - Third version

The second and the third versions of *PandasSoftwareProvisioning* workflow have the same functionality, i.e. sending queries to the PostgreSQL database of PANDAS for modifying configuration parameters. But on the other hand, the third version is more generic with respect to other simulation software than PANDAS. To put it another way, the `Send_Update_Query` operation of the PANDAS bone simulation web service, which is used in the second version, is only specific to the PANDAS bone simulation software. In the case that other simulation software, e.g. MATLAB, is used for simulation, this operation and the related workflow cannot be used. As the *SSH-IA* is used in the third version of *PandasSoftwareProvisioning* workflow, the third version of this workflow is appropriate in case of using other simulation software than PANDAS. The `Run Script` operation of this IA can be used to send appropriate queries to the related database server in order to change some configuration parameters.

4.2.2.4 Plan number two - More versions

More versions can be considered for *PandasSoftwareProvisioning* workflow with respect to the different options, which were discussed in the beginning of Section 4.2.2.2. In the following, different cases are discussed as examples. In the case that the PostgreSQL database is not deployed on the same Linux-VM of PANDAS instance, it is important to connect the Linux-VM of PANDAS to the VM of the PostgreSQL database in this workflow. As the current implementation of PANDAS and of its web service is tightly coupled with the PostgreSQL database system, there is no need to connect them explicitly in the above-mentioned versions of *PandasSoftwareProvisioning* workflow.

The PANDAS bone simulation software can use different kinds of database systems, e.g. a SQL server instead of a PostgreSQL database, for storing the results of simulation. In this case, if the database system is tightly coupled with the PANDAS web service, there is no need to connect them explicitly in *PandasSoftwareProvisioning* workflow. Otherwise, a connection between the Linux-VM of PANDAS instances and the VM of the related databases has to be established in this workflow.

Unlike the current implementation of PANDAS, different instances of the PANDAS bone simulation can be deployed on different VMs. In this case, a connection between these instances has to be established in *PandasSoftwareProvisioning* workflow. In other words, different instances of the PANDAS bone simulation are configured simultaneously and may work with each other in other workflows of the PANDAS Service Template (see Section 4.2.4). Therefore, it is required to connect multiple instances of the PANDAS bone simulation with each other.

4.2.3 Plan number three

The third plan, so-called *DataProvisioning* plan, provides and prepares heterogeneous input data sources of the simulation in order that PANDAS instances can properly ingest these data. In the scope of this master's thesis, two different variants of *DataProvisioning* workflow are designed and implemented. Previously mentioned, two PANDAS instances, one for the mechanical and one for the chemical calculation, are used in this work. Because the input data sources are different for the mechanical and chemical instances, this workflow is called separately for each of the two PANDAS software instances. Based on the value of one of the input parameters which are sent to the workflow, so-called `SoftwareInstance` variable, the path to the appropriate location of data sources for each instance is assigned in the workflow in order to provision the correct input data sources for PANDAS instances. Two different versions of this workflow are discussed in the following.

4.2.3.1 Plan number three - First version

Figure 4.8 depicts the initial version of *DataProvisioning* plan which provisions some heterogeneous files for a PANDAS software instance. Aforementioned in Section 2.5, the SIMPL framework extends workflow languages, such as BPEL, by introducing some additional DM activities and DM patterns. These activities and patterns provide a generic access to many different kinds of data sources. Figure 4.8 illustrates a high-level design of these DM activities in *DataProvisioning* workflow. Basically, this workflow is not only a data provisioning workflow. Instead, it provides and configures a concrete simulation problem or simulation example to be calculated by PANDAS instances (e.g. a certain bone to be simulated). As the figure shows, this workflow consists of three phases as follows:

- The first phase creates some sub-folders in the root folder of the relevant PANDAS software instance. The first *IssueCommand* activity creates a sub-folder for the simulation example in the root folder of the relevant PANDAS software instance. This sub-folder can be used later to store all the input data sources, which are provisioned in this workflow. The second *IssueCommand* activity creates another sub-folder, so-called *PandasTecplotOutputFolder* for instance, to store outputs of PANDAS simulation.
- The second phase provides necessary data sources which describe the simulation example. This phase consists of four *TransferData* activities and one *IssueCommand* activity which transfer heterogeneous data sources to the relevant PANDAS software instance. The first *TransferData* activity in this phase transfers a ZIP file that comprises all files describing the geometrical bone shape from the Windows-VM to the Linux-VM. Then, with an *IssueCommand* activity this ZIP file is unpacked to an appropriate directory. Following that, all the material parameters are transferred to the Linux-VM via a *TransferData* activity. Similarly, the boundary conditions and the FEM parameters are transferred with two *TransferData* activities in this phase.
- The third phase configures the PANDAS instances (e.g. some numerical configurations) in order that they can properly calculate the simulation outcomes. The numerical data sources are transferred with a *TransferData* activity. The last step in this workflow

invokes the PANDAS web service to compile the source code of PANDAS and the transferred simulation example all together.

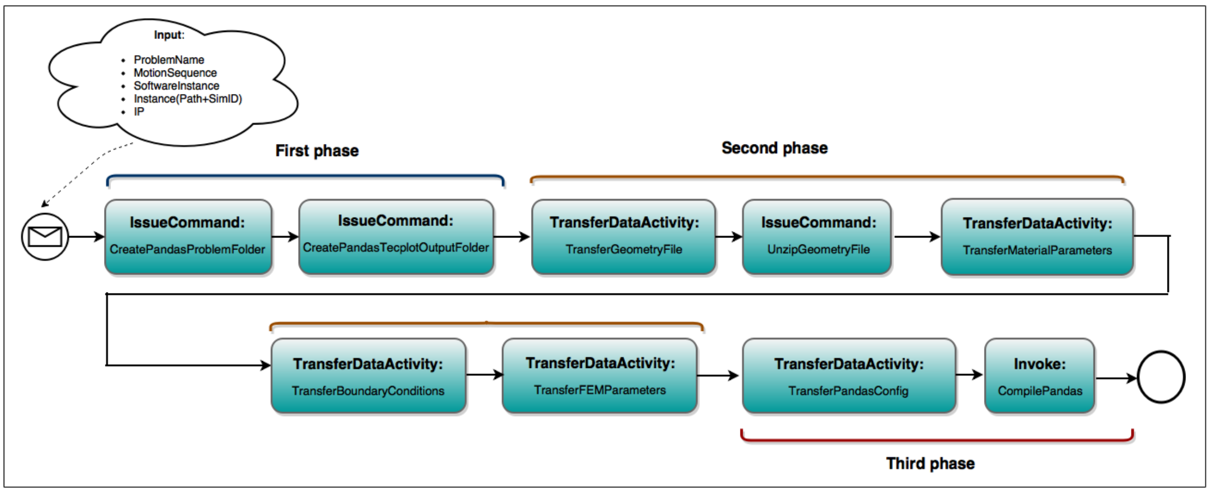

Input:
ProblemName
MotionSequence
SoftwareInstance
Instance(Path+SimID)
IP
First phase
Second phase
Third phase
IssueCommand: CreatePandasProblemFolder
IssueCommand: CreatePandasTecplotOutputFolder
TransferDataActivity: TransferGeometryFile
IssueCommand: UnzipGeometryFile
TransferDataActivity: TransferMaterialParameters
TransferDataActivity: TransferBoundaryConditions
TransferDataActivity: TransferFEMParameters
TransferDataActivity: TransferPandasConfig
Invoke: CompilePandas

Figure 4.8: DataProvisioning plan - First version

As Figure 4.8 depicts, some parameters are sent to the workflow as input parameters. The purpose of these input parameters are as follows:

- ProblemName (String): The name of the problem (or example) to be simulated, e.g. 3d_bone. This is used to generate a correct folder in the directory structure of the PANDAS software instance, where later input files can be copied to.
- MotionSequence (String): The name of the motion sequence that determines the boundary conditions, e.g. sleeping or standing, and thus the correct input files representing these boundary conditions. These motion sequences are used to define the state of a bone which is simulated.
- SoftwareInstance (String): Either mechanical or chemical, depending on whether the workflow shall provide data for the first (mechanical part) or second (chemical part) PANDAS software instance. This is currently used to identify the correct input files in the source folder.
- Instance: A complex type which contains: (1) Path, which is the root folder of the relevant PANDAS simulation software instance. This path is created by *PandasSoftwareProvisioning* workflow and (2) ID, which is a unique ID for each PANDAS instance. This ID is created by *PandasSoftwareProvisioning* workflow.
- IP (String): The IP address of the created VM in the first workflow. This IP address is used in order to compile the source code of PANDAS and the transferred simulation example all together on the right VM.

The idea in this master's thesis is separating *PandasSoftwareProvisioning* plan from *DataProvisioning* plan. As a result, the same *PandasSoftwareProvisioning* workflow can be used for several kinds of simulation problems. Another important issue to mention is including the *CompilePandas* operation as part of *DataProvisioning* plan. Conceptually, this operation should be part of *PandasSoftwareProvisioning* plan, because it compiles the PANDAS source code and thus the Pandas software. But, the current implementation of the PANDAS web service compiles the PANDAS source code with some of the input data sources together. Therefore, the required input data should be provisioned first before the *CompilePandas* operation can be invoked. It would be another option to invoke this operation in *Simulation/Application* plan. *Simulation/Application* plan is discussed in Section 4.2.4. But, by including this operation at the end of *DataProvisioning* workflow, PANDAS simulation can be executed several times without any need to compile the PANDAS source and the simulation problem again and again for each calculation. Subsequently, the overall performance of the actual simulation will be optimized.

4.2.3.2 Plan number three - Second version

In this master's thesis, two different variants of *DataProvisioning* workflow are provided. Figure 4.8 illustrated the concept for the first version of this workflow. To develop the second version of this workflow, different approaches for provisioning heterogeneous and huge data sources of the PANDAS bone simulation software can be considered. As Figure 2.7 in Chapter 2 indicates, PANDAS requires heterogeneous data sources as input parameters. For example, PANDAS instances need some CSV-based *boundary conditions* files as input parameters. These CSV files were transferred to the Linux-VM in the first version of *DataProvisioning* workflow. Some different kinds of CSV files, that have a different format, can be used as well for PANDAS instances. These CSV files require a coordinate transformation in order to be appropriate for PANDAS instances. A Java API for XML Web Service (JAX-WS)[5] is implemented in this master's thesis in order to transform the coordination of these *boundary conditions* files. In other words, a Java-based program for this transformation has been developed before and a web service around this program is implemented only in this work. Accordingly, as Figure 4.9 illustrates, another version of *DataProvisioning* workflow can be developed based on this web service.

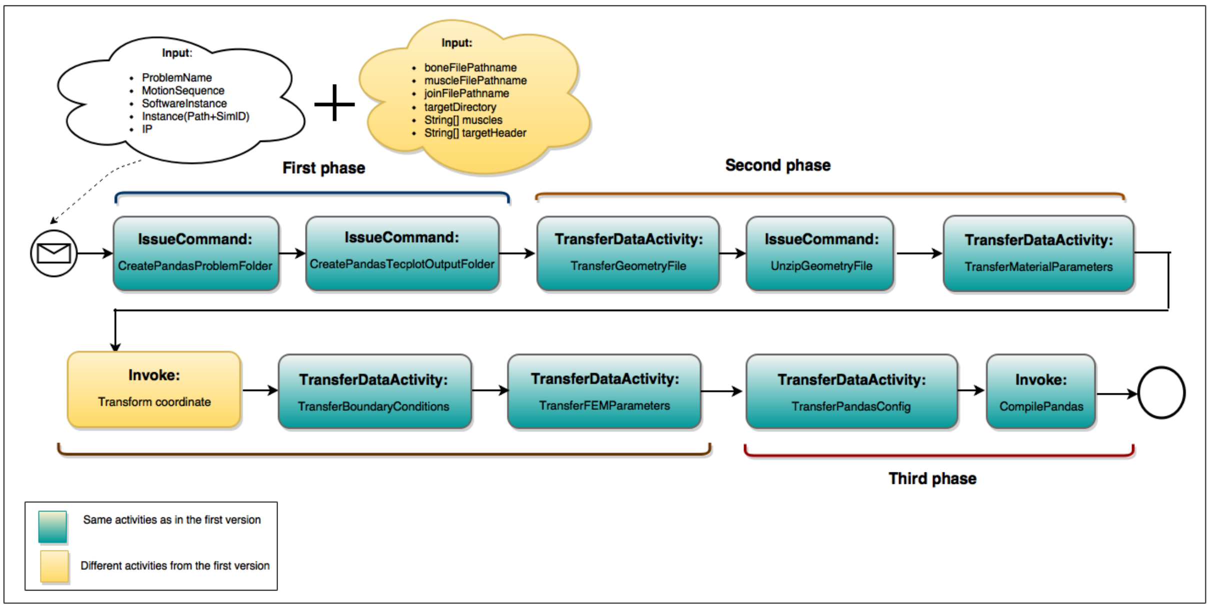

Input:
ProblemName
MotionSequence
SoftwareInstance
Instance(Path+SimID)
IP
+
Input:
boneFilePathname
muscleFilePathname
joinFilePathname
targetDirectory
String[] muscles
String[] targetHeader
First phase
Second phase
IssueCommand: CreatePandasProblemFolder
IssueCommand: CreatePandasTecplotOutputFolder
TransferDataActivity: TransferGeometryFile
IssueCommand: UnzipGeometryFile
TransferDataActivity: TransferMaterialParameters
Invoke: Transform coordinate
TransferDataActivity: TransferBoundaryConditions
TransferDataActivity: TransferFEMParameters
TransferDataActivity: TransferPandasConfig
Invoke: CompilePandas
Third phase
Same activities as in the first version
Different activities from the first version

Figure 4.9: DataProvisioning plan - Second version

[5] http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html

As Figure 4.9 shows, in the second phase of this workflow, the *CoordinateTransformation* web service is invoked in order to transform the coordinate of the above-mentioned CSV files. In other words, before transferring the boundary conditions to the Linux-VM as an input data source for PANDAS instances, the `Transform coordinate` operation of this web service is invoked in order to transform the coordination of these CSV files. As the figure depicts, some additional input parameters are added to the first version of *DataProvisioning* workflow. These additional input parameters are required for invoking the `Transform coordinate` operation. In general, this operation requires three CSV files, two string arrays and a path, which refers to a directory, for storing the results of the operation. These parameters are discussed in detail in the following:

- BoneFilePathName (String): A string which defines the location of a CSV file which describes the relevant information about a bone.
- MuscleFilePathName (String): A string which defines the location of a CSV file which describes the muscle forces working on the bone.
- JointFilePathName (String): A string which defines the location of a CSV file which describes the joint contact forces of adjacent bones.
- TargetDirectory (String): The directory which is used to store results of the *Coordinate-Transformation* web service.
- MuscleArray (String Array): A string array which consists of several names for various muscles where PANDAS needs the corresponding muscle forces as input. The `Transform coordinate` operation of the *CoordinateTransformation* web service considers all the strings inside this array and generate some other CSV files based on these strings. In other words, it is not possible to filter only a sub-set of the muscles in the MuscleArray.
- TargetHeaderArray (String Array): A string array which determines the header values of the CSV files, which are generated in this operation.

As the result, twelve muscle files (based on the input parameters which are used in this work) and one joint file are generated after executing the `Transform coordinate` operation of this web service. Afterwards, these files are transferred to the Linux-VM as an input data source for a PANDAS instance via one *TransferData* activity of the SIMPL prototype.

4.2.4 Plan number four

The fourth plan, so-called *Simulation/Application* workflow, realizes the actual simulation calculation of PANDAS instances. In this master's thesis, this workflow is called after *DataProvisioning* workflow. The workflow runs simulation for both PANDAS instances (mechanical and chemical) simultaneously. Accordingly, this workflow only needs to be executed once for realizing the PANDAS bone simulation software as a SaaS solution in cloud environments. Only one version of this workflow is designed and implemented in this master's thesis, since the actual simulation calculation is not the main focus of this thesis. As Figure 4.10 illustrates, the main activities inside this workflow are as follows. Similar to the operations which were used in *PandasSoftwareProvisioning* plan, the following operations are part of the PANDAS web service:

- *Start_Pandas_Mechanical* and *Start_Pandas_Chemical*: start the respective PANDAS software instances (both instances simultaneously in this work).
- *Execute_Command*: sends some commands to both PANDAS instances (at once), so that they execute their cmd-files and read some input files. The current version of the PANDAS web service sends some commands to both instances at once. But conceptually, it would be more efficient to execute this activity separately and in parallel for each instance of PANDAS.
- *Calculate_Pandas*: invokes the actual calculation (both instances at once). There is one problem with the current version of the PANDAS web service implementation: the calculation actually lasts something about half an hour, but the web service operations access PANDAS in an asynchronous manner and then the invocation is finished after one minute. As a workaround, a BPEL wait activity which waits for 45 minutes is added to the workflow in order that everything works properly.
- *Stop_Pandas*: stops PANDAS instances (both instances at once). Similar to *Execute_Command*, it is more efficient to Stop PANDAS instances separately in two *Stop_Pandas* activities in parallel. But, the current version of the PANDAS web service stops both instances at once via one invoke activity.

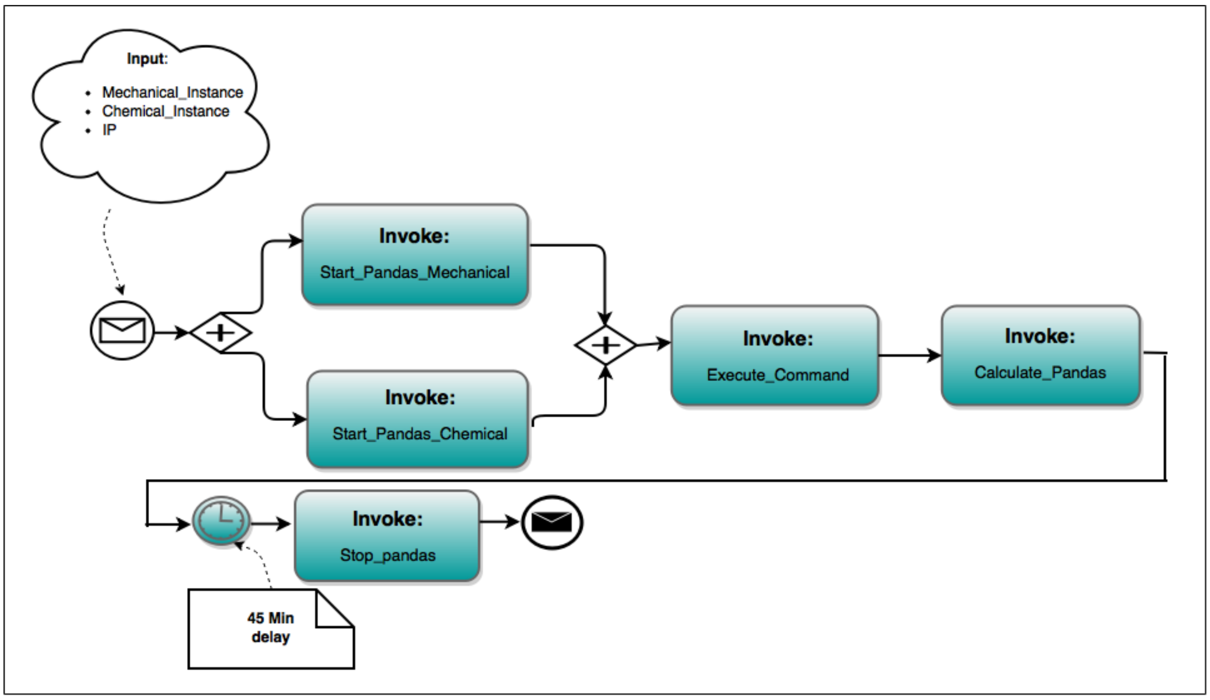


Figure 4.10: Simulation/Application plan

As Figure 4.10 shows, three input parameters are passed to the workflow. The IP address of the created VM in the first workflow of the PANDAS Service Template is sent to this workflow. This IP address is used to execute all the above-mentioned operations on the new created VM in the topology. Two complex type parameters, one for the mechanical and one for the chemical instances, are sent to the workflow as well. These parameters consist of the information related to PANDAS instances. A path and an ID, which are the output of *PandasSoftwareProvisioning* workflow, are passed to the workflow via these complex type parameters in order to run simulation for the correct PANDAS instance. Besides, the `ProblemName` variable, which was also an input parameter in *DataProvisioning* workflow and discussed in Section 4.2.3.1, is part of these complex type parameters and sent to *Simulation/Application* workflow as well. In other words, each of these complex type parameters consists of a path, an ID and a ProblemName variable.

4.2.5 Plan number five

The fifth plan, so-called *DataDeprovisioning* workflow, returns the result data of a PANDAS simulation back to the user. In the scope of this master's thesis, different variants of *DataDeprovisioning* plan are designed and implemented.

4.2.5.1 Plan number five - First version

Figure 4.11 shows the initial version of *DataDeprovisioning* workflow. This is a simple workflow which consists of only one *TransferData* activity that transfers the outcome of a PANDAS calculation from *PandasTecplotOutputFolder*, i.e. all text-based files of PANDAS result are stored in this folder, to an appropriate folder of the Windows-VM on the OpenStack cloud provider. This *PandasTecplotOutputFolder* was discussed in Section 4.2.3.2. A path to an appropriate folder on the Windows-VM is passed to the workflow as an input parameter. Similarly, the actual location of this *PandasTecplotOutputFolder* for each PANDAS instance is passed to the workflow as an input parameter.

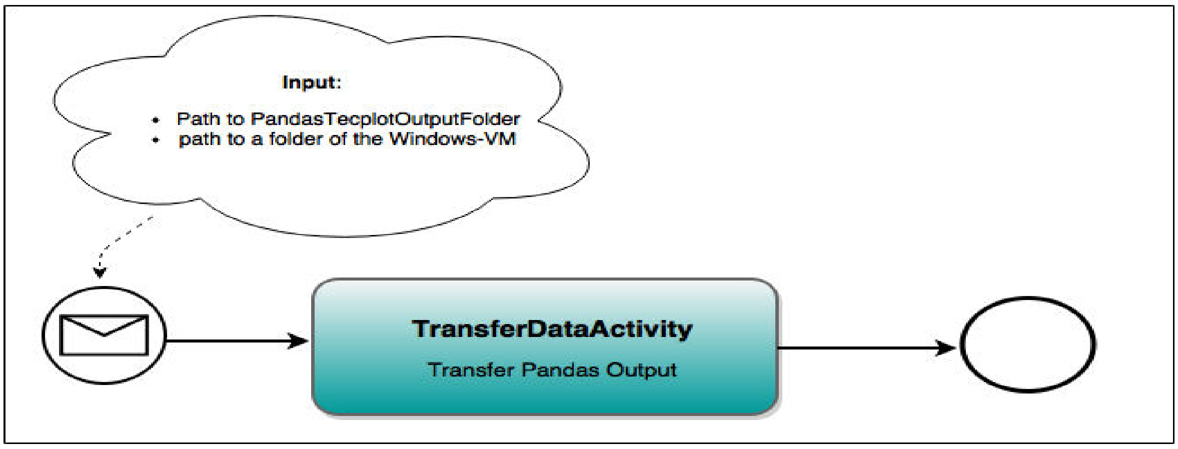

Input:
Path to PandasTecplotOutputFolder
path to a folder of the Windows-VM
TransferDataActivity
Transfer Pandas Output

Figure 4.11: DataDeprovisioning plan - First version

4.2.5.2 Plan number five - Second version

Instead of using *PandasTecplotOutputFolder* and provisioning the results of the simulation to the user from this folder, as Figure 4.12 shows, in the second version of *DataDeprovisioning* workflow, the results can be queried directly from the PostgreSQL database. First, it is required to add a new data source for the PANDAS PostgreSQL database inside the database of the SIMPL framework. To put it another way, after creating a new VM in the first workflow, the new IP address of the created VM has to be registered inside the SIMPL database. This IP address is used later to connect to the right PostgreSQL server of PANDAS, and then extract the appropriate data based on the received query. For this purpose, The `AddDataSource` operation of the SIMPL web service is invoked to register a new entry for the PANDAS PostgreSQL database inside the SIMPL database. This IP address is sent to the workflow as an input parameter. Besides, a simulation ID of a PANDAS instance is sent to the the workflow as well. This ID is used in the query to export the right data for the right PANDAS instance from the PostgreSQL database. With the help of an *IssueCommand* activity inside the SIMPL framework, the query is executed on the server and the results are exported into one or more files. Following that, these files have to be transferred to the Windows-VM by using a *TransferData* activity of SIMPL. Different variants of this workflow can be provided by changing the query which is executed on the PostgreSQL database.

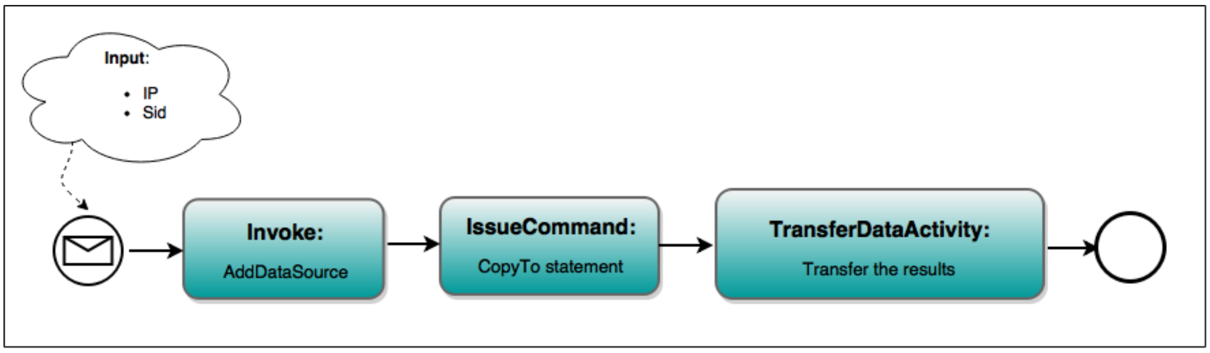


Figure 4.12: DataDeprovisioning plan - Second version

As shown in Listing 4.2, a `COPY TO` statement copies the contents of a table or the results of a query to a file. `COPY TO` statement in PostgreSQL databases reads data from a database and writes it back to an appropriate file. It is also possible to define some options for a `COPY TO` statement in order to customize this statement. For example, the format of the file which is used for storing the results of the query can be from different types such as text, CSV or binary[24].

```
COPY { table_name [ ( column [, ...] ) ] | ( query ) }
TO { filename }
[ [ WITH ] ( option [, ...] ) ]
```

Listing 4.2: PostgreSQL COPY TO statement [24]

Different variants of a workflow with this `COPY TO` statement can be designed and implemented. For instance, one variant can copy the results of PANDAS into one or more text files. Other variant could copy them into one or more CSV files instead. CSV files do not have any limitations in the file length. It is possible to store 100,000 or billions of rows of data in a CSV file[42]. By changing the query inside the `COPY TO` statement, various versions can be considered as well. To clarify, by filtering the results of the simulation based on the simulation ID which was created in *PandasSoftwareProvisioning* plan for each PANDAS instance, the appropriate calculation results of each instance can be derived from the database for users. It is also possible to filter the results for different tables or even for some columns based on user's requirements. The second version of *DataDeprovisioning* workflow can be modified as well to extract appropriate data for multiple PANDAS instances. In other words, different simulation IDs of PANDAS instances can be transferred to the workflow via input parameters. Then, for each PANDAS instance one *IssueCommand* activity is used in order to execute the query on the PostgreSQL server and then copy data into one or more files . Following that, these files for each PANDAS instance are transferred separately to the Windows-VM with the help of a *TransferData* activity inside the SIMPL framework.

4.2.5.3 Plan number five - Third version

Instead of querying tables from the PostgreSQL database of the PANDAS bone simulation software, the whole database and all related data sources can be backed up to a different machine as a data deprovisioning process. One approach to back up PostgreSQL databases is using a SQL dump statement. With a SQL dump command, a text file with some appropriate SQL commands is generated on the database system. Then, by executing this file as a query on the database server, the same database can be recreated in the new location. As shown in Listing 4.3, a pg_dump statement writes a database to an standard output. The text files which are created by a pg_dump statement can be restored by a psql command. In other words, the result of a pg_dump command can be transferred to the *dbname* database with a psql command. This *dbname* database in the third command as Listing 4.3 shows should be created before executing a psql command. The *dbname* database can be created with the help of some commands, such as createdb, as the second command in the listing depicts[25]. Subsequently, another version for *DataDeprovisioning* workflow can be designed and implemented by using the above-mentioned SQL statements.

```
pg_dump dbname > outfile
createdb -T template0 dbname
Psql dbname < outfile
```

Listing 4.3: PostgreSQL Backup statement [25]

Figure 4.13 depicts the middle-level design of the third version of *DataDeprovisioning* workflow. The pg_dump statement requires a user name and password for connecting to the database server. It is also necessary to define the name of the database which needs to be backed up. All these values are transferred to the workflow as input parameters. Then, an *IssueCommand* activity inside the SIMPL framework can be used to execute the pg_dump statement on the database server. This command produces a .sql file as the result. Following that, this file is transferred to the Windows-VM via a *TransferData* activity. With an *IssuCommand* activity, a database can be created on the Windows-VM before executing a psql command. Then, a psql command can be executed on the PostgreSQL database inside the Windows-VM. As the result, the database is backed up from the Linux-VM to the Windows-VM.

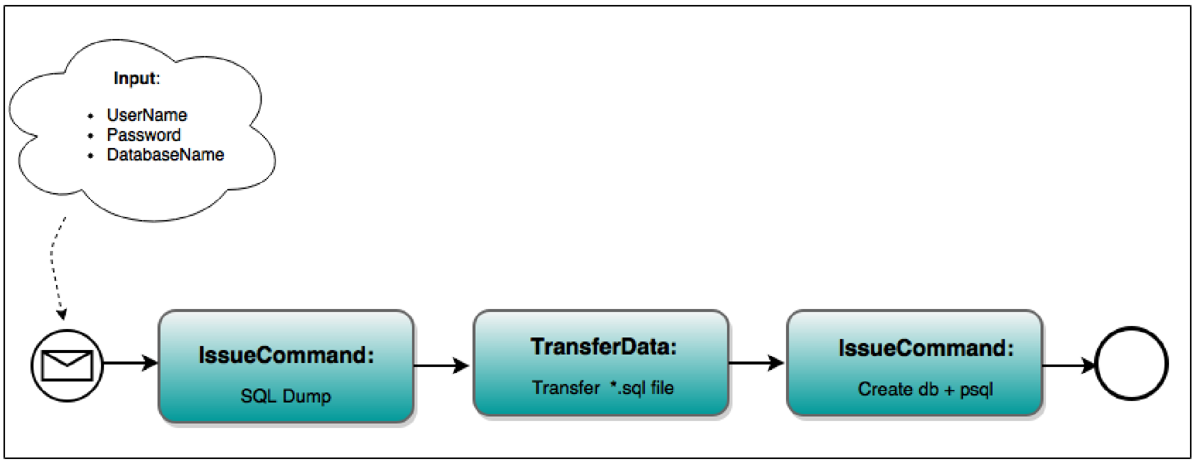


Figure 4.13: DataDeprovisioning plan - Third version

4.2.5.4 Plan number five - Fourth version

Most of simulation applications, the PANDAS bone simulation software for instance, generate heterogeneous and huge amount of data as simulation results. Therefore, tables inside their databases could grow dramatically in size. The PostgreSQL database allows user to create tables larger than the maximum size allowed on the system. By using a `SQL dump` command, dumping those huge tables to one or more files can be problematic. Using compression in a `SQL dump` command can be considered as a solution in order to deal with this problem. The appropriate compression program, such as `gzip`[6], can be used to compress tables and then store them to an appropriate file[25].

```
pg_dump dbname | gzip > filename.gz
createdb -T template0 dbname
gunzip -c filename.gz | psql dbname
```

Listing 4.4: PostgreSQL Backup statement with compression[25]

Listing 4.4 indicates the commands for dumping the compressed database to a file and then restoring it using a `gunzip` command. Another version for *DataDeprovisioning* workflow can be implemented by using these commands. Figure 4.14 shows the main activities for the forth version of *DataDeprovisioning* workflow. In this version, all tables inside the database are compressed first and then backed up to another machine.

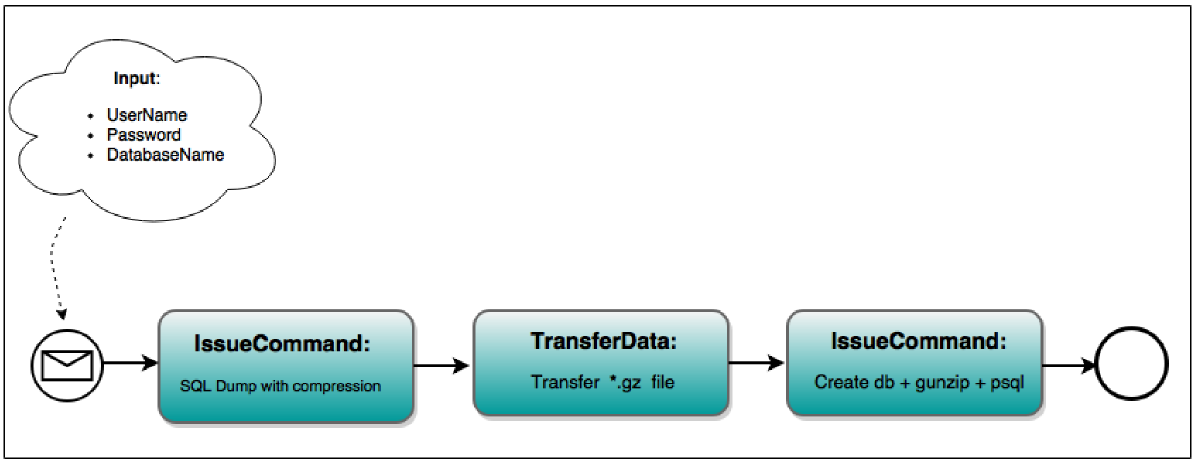


Figure 4.14: DataDeprovisioning plan - Forth version

4.2.5.5 Plan number five - Fifth version

Instead of compressing tables inside the database, there is an option in PostgreSQL databases which allows users to split the output into different pieces. In other words, users can control large tables and split them into different pieces with an appropriate size which is acceptable on the target system. For instance, Listing 4.5 shows the usage of the `split` option in order to make chunks of one megabyte [`pg_dump dbname | split -b 1m - filename`]. Then, the generated files can be restored with the help of a `cat` statement. This statement concatenates the splitted files in order to regenerate the database again and back it up to the Windows-VM[25].

```
pg_dump dbname | split -b 1m - filename
createdb -T template0 dbname
```

[6]http://www.gzip.org/

```
cat filename* | psql dbname
```

Listing 4.5: PostgreSQL Backup statement with split option [25]

Accordingly, another version for *DataDeprovisioning* workflow can be designed by splitting tables into different chunks and restoring them again on the Windows-VM. Figure 4.15 illustrates the middle-level design of the fifth version.

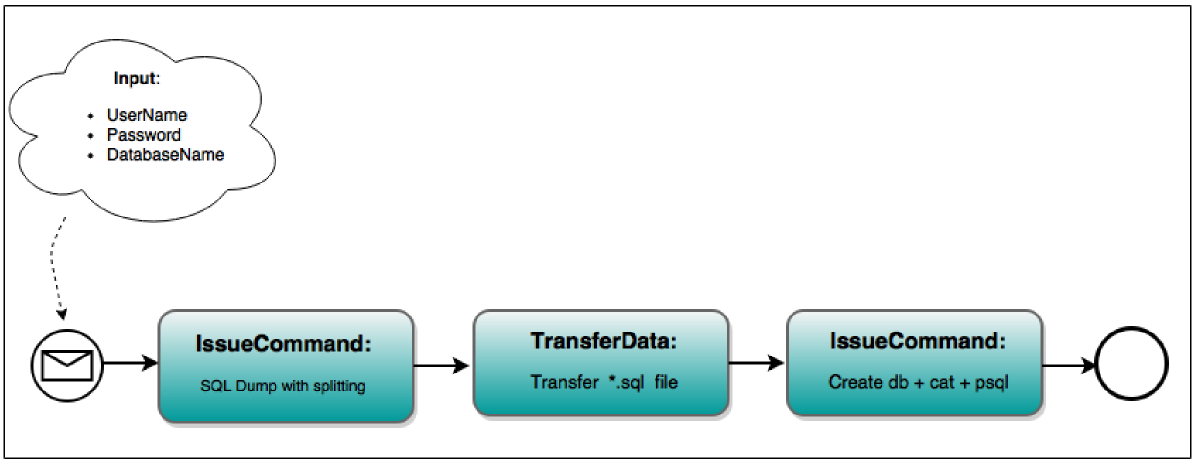


Figure 4.15: DataDeprovisioning plan - Fifth version

4.2.6 Plan number six

The last plan for realizing the PANDAS bone simulation software as a SaaS solution in cloud environments is *Termination* plan. As Figure 4.16 illustrates, the first activity of this workflow invokes the *TerminateVMbyServerid* operation of the *InstallOpenStackVM* IA inside the service topology of the PANDAS Service Template. This is a loop activity which can be executed several times. To put it another way, several VMs might be installed in the first workflow by users, depending on which variants of the plans are used. For this purpose, this operation reads the TOSCA service topology and then determines the number of installed VMs which need to be terminated. The *TerminateVMbyServerid* operation requires `credentials`, `endpointsAPI` and `serverId` as input variables. In Section 4.2.1, the `credentials` and `endpointsAPI` variables were discussed. The `ServerId` variable defines a unique ID for each server on the OpenStack cloud provider. Therefore, these values are transferred to the workflow via input parameters. Following that, in the second activity of this workflow, the *Undeploy* operation of the *ODE-Service* IA is called in order to undeploy all BPEL plans from the ODE-PGF engine inside the Windows-VM. For this purpose, the names of the workflows which need to be removed from the ODE-PGF engine are passed to the workflow as input parameters. In this master's thesis, three workflows have to be undeployed from the ODE-PGF engine. As the result, the names of these three workflows are passed to the workflow, as Figure 4.16 illustrates. Finally, in the last step of this workflow, the new SIMPL data source which was created and added to the SIMPL resource management in the first workflow is unregistered from SIMPL again. The *DeleteDataSource* operation of the SIMPL web service is called in this step to remove the entry from the database. This operation needs an ID, which determines the entry that should be removed from the database. Therefore, the unique ID of that entry is transferred to the workflow as an input parameter, as Figure 4.16 depicts. In this master's thesis, only one variant of this workflow is designed and implemented.

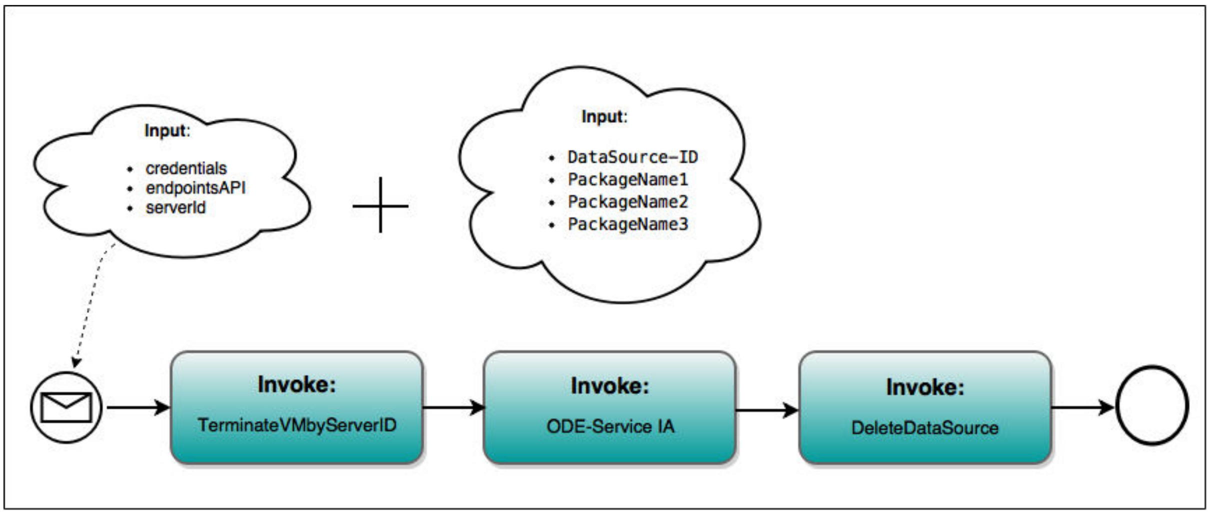


Figure 4.16: Termination plan

4.3 Integration of SIMPL with TOSCA

Previously mentioned, both TOSCA and SIMPL rely on workflows to orchestrate processes for on-demand service deployment and management, as well as for data provisioning and simulation calculation, respectively. One goal in this master's thesis is to combine both approaches in order to offer a full-fledged and integrated support for cloud-native simulation software that spans all these relevant goals[18]. Consequently, one contribution of this master's thesis is to integrate the prototype of SIMPL with the OpenTOSCA engine in order to make the data provisioning technology offered by SIMPL an integral part of a TOSCA description. For data provisioning and deprovisioning of simulation applications in cloud environments, the DM activities and DM patterns of SIMPL can be integrated to TOSCA plans. Different approaches can be considered for integrating SIMPL with TOSCA as follows.

4.3.1 Extending the workflow engine inside the OpenTOSCA environment

One approach for integrating the TOSCA standard with the SIMPL framework could be extending the workflow engine inside the OpenTOSCA environment to accept the DM activities and DM patterns of the SIMPL framework. To put it another way, the workflow engine inside the OpenTOSCA environment, which is a WSO2 Business Process Server, needs to be extended by the plugable framework of the ODE-PGF engine (PGF means plugable framework). In this approach, one could simply copy the ZIP-archive of the SIMPL ODE extensions, maybe adjust a configuration file to register this extension bundle, and then everything should work properly. Only one workflow engine, i.e. the WSO2 Business Process Server inside the OpenTOSCA environment, is used in this approach for executing all workflows inside the TOSCA Service Template. The other components of the SIMPL framework can be connected to OpenTOSCA components in a loosely coupled way[18].

4.3.2 Extending the OpenTOSCA environment via plug-ins

Another approach for integrating the SIMPL framework with the TOSCA standard is developing some plug-ins[7] for the OpenTOSCA environment. Based on these plug-ins, the SIMPL-based workflows are recognized automatically from other types of workflows inside the TOSCA Service Template, and then deployed on the right workflow engine. In this thesis, the ODE-PGF engine on the SIMPL framework should be selected for these workflows. Consequently, this approach relies on two different workflow engines (ODE-PGF and WSO2 Business Process Server).

4.3.3 Integrating via some features inside the TOSCA management plans

Similar to the previous method, this approach uses the plug-in architecture as well. The plug-in functionality is already provided in the OpenTOSCA environment. In this case, scientists are able to define in the TOSCA plans on which engine they should be deployed. Therefore, there is only need to develop the corresponding plug-in and then deploy it inside the OpenTOSCA environment. With the help of this plug-in, the OpenTOSCA environment automatically deploy the corresponding workflows on the defined workflow engine. This might be a combination of the two above-mentioned approaches. In other words, this approach relies on two different workflow engines (ODE-PGF and WSO2 Business Process Server) like in the second approach, and similar to the first and second approaches the workflows are deployed automatically to one of these engines.

4.3.4 Integrating via the ODE-Service IA

In this master's thesis, a different approach for integrating the SIMPL framework with the TOSCA standard is described as follows: both workflow engines in the SIMPL framework and the OpenTOSCA environment work together in a loosely coupled way. In other words, we still have two different workflow engines: (1) the WSO2 Business Process Server inside the OpenTOSCA environment and (2) the separate ODE-PGF engine. Based on the activities inside the workflows, the appropriate engine for running and executing the BPEL plans is selected and then the workflows are deployed on that engine. For this purpose, the *ODE-Service* IA is an integral part of the TOSCA service topology inside the PANDAS Service Template. This is a Java-based IA which contains two main functions for deploying to and undeploying BPEL plans from a workflow engine and was designed and implemented in this thesis. As mentioned already, the Winery tool is used in order to create a CSAR file which contains the PANDAS Service Template. The Service Template contains an application topology as well as management plans. The *ODE-Service* IA is inside the service topology and can be used for deploying and undeploying SIMPL workflows. As Figure 4.17 shows, if the workflow contains some SIMPL DM activities or DM patterns, the *ODE-Service* IA is called in order to deploy the workflows on the ODE-PGF engine within the SIMPL framework. This engine is extended already in order to understand and execute the SIMPL activities. These workflows can be undeployed from the engine with this IA as well. On the other hand, if the workflows do not contain any SIMPL activities, the plans are deployed on the workflow engine inside the OpenTOSCA environment. In this case, the deployment and undeployment of the plans are happened automatically and there is no need to call the `Deploy` and `Undeploy` operations of the *ODE-Service* IA. It is important to note that the process of deploying and undeploying

[7]https://github.com/decebals/pf4j

workflows via the *ODE-Service* IA cannot be done automatically like in the above-mentioned approaches. In other words, two TOSCA plans do the deployment and undeployment tasks. In this work, the first workflow inside the PANDAS Service Template defines the workflows which need to be deployed on the ODE-PGF engine. To put it another way, the `Deploy` function of the *ODE-Service* IA is called inside the first plan for each workflow which needs to be deployed on the ODE-PGF engine. The `Undeploy` operation of this IA is called in the last plan of the PANDAS Service Template for each workflow which needs to be undeployed from the ODE-PGF engine. Therefore, with the help of two TOSCA plans [the first and the last plans in this work], the deployment and undeployment processes can be done in this approach.

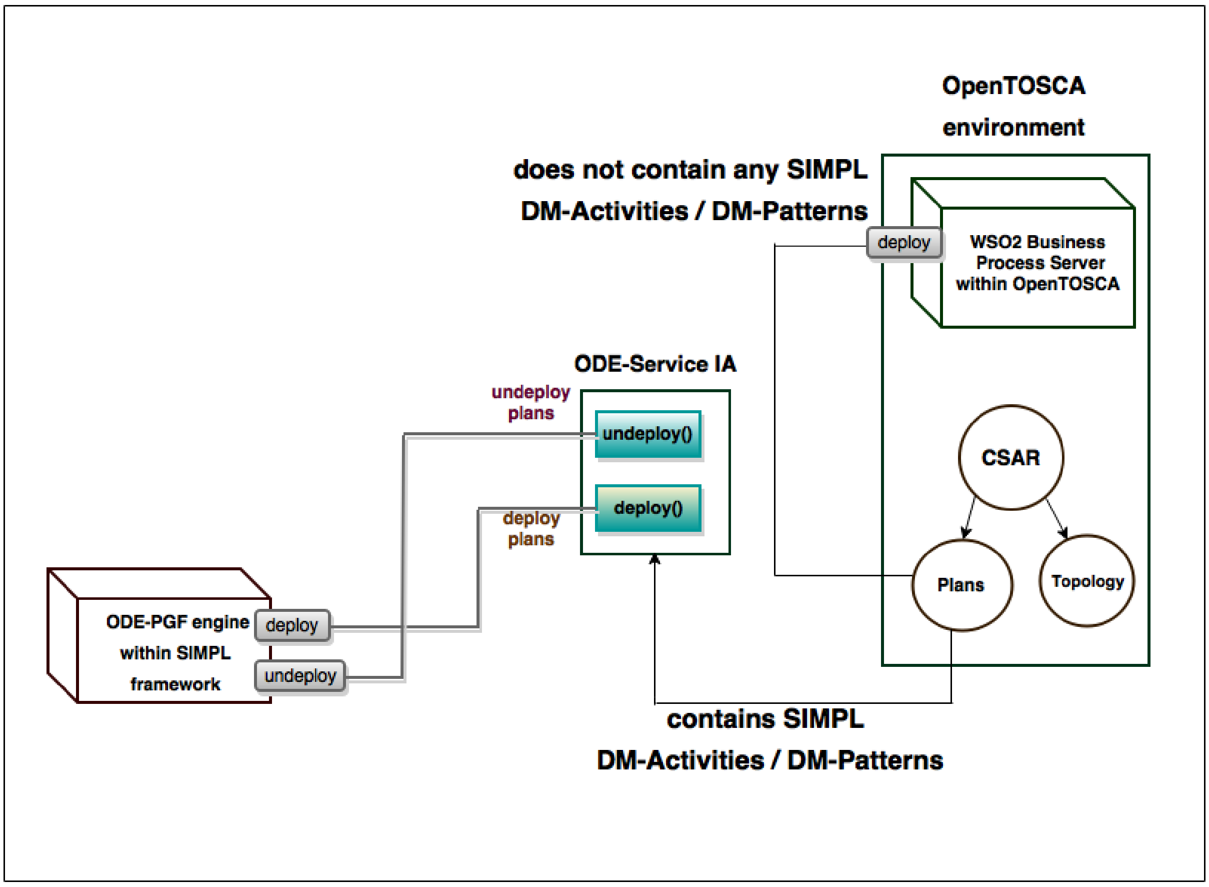

OpenTOSCA environment
does not contain any SIMPL DM-Activities / DM-Patterns
deploy
WSO2 Business Process Server within OpenTOSCA
ODE-Service IA
undeploy plans
undeploy()
deploy()
deploy plans
CSAR
Plans
Topology
ODE-PGF engine within SIMPL framework
deploy
undeploy
contains SIMPL DM-Activities / DM-Patterns

Figure 4.17: Integration of SIMPL with TOSCA

4.3.5 Evaluation of the proposed approaches for TOSCA and SIMPL integration

This section evaluates the above-mentioned approaches for integrating the TOSCA standard and the SIMPL framework from a conceptual point of view. The first three approaches are mainly based on the plug-in architecture. But, the last approach which was designed and implemented in this master's thesis is different from the first three approaches. For this purpose, this section compares the implemented approach in this master's thesis [the last approach] with the plug-in architecture [the first three approaches] in general.

Using the *ODE-Service* IA for integrating SIMPL with the TOSCA standard is an easy and straightforward approach to understand. This IA consists of only two simple operations, so-called `Deploy` and `Undeploy`, which can be used in the first and the last workflows of the PANDAS Service Template, respectively. Developers only need to define a node template inside the application topology of TOSCA for each SIMPL-based workflow. SIMPL-based workflows are workflows which consist of one or more DM activities or DM pattern. Following that, the `Deploy` operation of the *ODE-Service* IA is called for each workflow separately in the first plan to deploy the workflow. Similarly, in the *Termination* plan, the `Undeploy` operation of this IA is called for each workflow which needs to be removed from the ODE-PGF engine.

Besides, the TOSCA-based service topology of simulation software, inside the TOSCA Service Template, is self–explanatory. To put it another way, by looking at the service topology, it is simple enough to understand which workflows are SIMPL-based and need an extended version of Apache ODE engine for execution. Each SIMPL-based workflow requires a node template in the topology, which represents the actual workflow implementations and may consist of some properties. In other words, the actual implementation of these workflows are added to theses node types as DAs. For instance, three different workflows (*DataProvisioning*, *Simulation/Application* and *DataDeprovisioning*) inside the PANDAS Service Template require an ODE-PGF engine for their execution. Moreover, the *Simulation/Application* workflow in the PANDAS Service Template is basically some kind of `application logic` workflow, but not a TOSCA-based workflow from a conceptual point of view. So, it might also be more intuitive to define this workflow as a DA for a node template than as a TOSCA-based plan. Therefore, by using the *ODE-Service* approach for integrating TOSCA with SIMPL, engineers are capable to separate the real TOSCA-based workflows from other types of workflows [so-called `application logic` workflows].

Furthermore, it is always a question for engineers which workflow engine to use for which plan, either the workflow engine inside the OpenTOSCA environment or the ODE-PGF engine. This issue can be seen from the perspective of which workflow design tool is used and which person has designed the corresponding workflows. Conceivably, the actual TOSCA-based plans are designed by different persons than the SIMPL-based ones. Then, these different persons maybe also need different workflow design tools depending on their knowledge and on the way how they design workflows (maybe they even use different workflow languages or workflow technologies). As a consequence, it might also be beneficial to see workflows that are designed with an external design tool as DAs, and not as TOSCA-based workflows. Based on the above-mentioned desirable features, which can be provided by the *ODE-Service* approach, the author prefers this method of integration to the other approaches.

At first, the plug-in architecture seems to be an efficient way to integrate TOSCA with SIMPL. In other words, because this architecture constitutes a generic solution where every workflow in a TOSCA Service Template, from a design perspective, is treated in the same way and there is no need to define any additional activities in some of the plans to deploy and undeploy these workflows. But on the other hand, this generic solution might not be the right one as discussed above. Furthermore, there are some issues and challenges for a pluggable system as follows, which turn away most of the developers[43]:

- Maintainability: maintainability of plug-ins can be difficult most of the time. Managing versions and backwards compatibility with existing plug-ins can be challenging in a plug-in framework.

- Complexity: a plug-in can work fine when it is tested alone. But, when different plug-ins are integrated with each other, the interactions between plug-ins can cause many problems.
- Testing: testing a plug-in can have many difficulties if the plug-in system does not provide some form of `plug-in runner` for testing. Therefore, developers only have to test plug-ins in the real world and this decreases the speed of development.

One limitation of the *ODE-Service* approach for integrating TOSCA with SIMPL is the lack of automation. In other words, the deployment and undeployment processes of the workflows inside the PANDAS Service Template are not automated without any human interventions. The developers need to determine in advance the workflows which require an extended ODE engine for their executions. Then, the `Deploy` operation of the *ODE-Service* IA is called for these workflows inside the first plan. In a similar way, the `Undeploy` operation of this IA is invoked in the last plan of the PANDAS Service Template in order to remove the workflows from the ODE-PGF engine. As it is discussed in Section 7.2, an optimized approach for this IA which deploys and undeploys workflows automatically can be designed and implemented in future.

4.3.6 Adopting the PANDAS management plans to the different approaches of TOSCA and SIMPL integration

The third approach, which were discussed in Section 4.3.3, requires possible changes that need to be done to the management plans inside the PANDAS Service Template. Inside all the management plans, the developer has to determine the appropriate workflow engine. Therefore, some changes need to be applied to the current versions of all management plans inside the PANDAS Service Template. In the first and second approaches, which were discussed in Sections 4.3.1 and 4.3.2 respectively, there is no need to change the management plans significantly. Because the workflow engine inside the OpenTOSCA environment [First approach] and the OpenTOSCA environment itself [Second approach] are extended via plug-ins in order to understand different types of workflows [SIMPL-based, TOSCA-based].

Chapter 5

Implementation

In this chapter, the implementation of the prototype for realizing the PANDAS bone simulation software in cloud infrastructures is described, with respect to the requirements and concepts specified in the previous chapters. In Section 5.1, the detailed approach for the first contribution of this master's thesis which is the integration of the SIMPL prototype with the TOSCA standard is discussed. Section 5.2 describes the design, development and implementation of different variants of a TOSCA Service Template realizing the PANDAS bone simulation software in a cloud-native way, which is the second and basically the main goal of this master's thesis. Furthermore, in each section the challenges the author faced during design and implementation are discussed as well.

5.1 Integration of SIMPL with TOSCA

Aforementioned, both the OASIS TOSCA standard and the SIMPL prototype rely on workflows to orchestrate processes for on-demand service deployment and management, as well as for data provisioning and simulation calculation, respectively. The first goal of this master's thesis is to combine both approaches in order to offer a full-fledged and integrated support for cloud-native simulation applications that spans all these relevant processes[18]. In the following, the implementation of the presented approach for this goal in this master's thesis is described in detail.

This approach is mainly based on the Apache ODE deployment API. Apache ODE engine has some communication layers, such as Axis2, and integration layers which are used by ODE BPEL Engine run-time in order to interact with the outside world. Already mentioned in section 2.6.5, Apache Axis2 is a container for web services. This container enables users to create, deploy, test and run web services. Apache ODE deployment web service[1], which is used for deploying WS-BPEL processes, is an Axis2 web service which is deployed on the ODE engine during the initiation of ODE run-time[44].

The *ODE-Service* IA which was developed in this master's thesis is mainly based on the ODE deployment web service. This IA reads the BPEL process artifact (zip package containing .bpel, .wsdl, deploy.xml, etc., files) and then encodes the package with Base64 encoding schemes[2]. Then, the package is sent to the ODE deployment web service in order to deploy and undeploy

[1]https://github.com/apache/ode/tree/master/axis2/src/main/java/org/apache/ode/axis2/service

[2]https://www.base64encode.org/

BPEL processes from Apache ODE engine as a payload inside a SOAP message. Winery, a graphical modelling TOSCA tool, is used in this work to develop and design the *ODE-Service* IA. The subsequent section gives an overview about using the Winery tool in correspondence with developing the *ODE-Service* IA.

5.1.1 ODE-Service IA

The following subsections provide some implementation details about the *ODE-Service* IA. In Subsection 5.1.1.1, the interface design of the *ODE-Service* IA in the Winery tool is discussed. Following that, the actual implementation of operations, which are inside the *ODE-Service* IA, with the help of Eclipse is explained in Subsection 5.1.1.2. Then, Subsection 5.1.1.3 tests the *ODE-Service* IA with the SoapUI tool. Finally, the challenges the author faced during design, implementation and deployment of this IA are discussed in Subsection 5.1.1.4.

5.1.1.1 ODE-Service interface design in Winery

Winery is a graphical environment which can be used to model the TOSCA Service Templates. A TOSCA Service Template consists of an application topology as well as some management plans. With Winery, all elements inside a TOSCA specification can be modeled and designed. In other words, all components are stored within a repository in a CSAR format and can be imported to and exported from the Winery web-based environment. Figure 5.1 depicts the GUI of the Winery tool. With Winery, users can create TOSCA-based node types, relationship types, policy types, etc., which can be used inside the application topology of a TOSCA Service Template. The `Add new` button in the figure is used to generate different element types. Following that, the created elements can be exported as a CSAR format from the Winery tool. As shown in Figure 5.1, three buttons are used for exporting, editing and deleting the elements. In other words, an already created CSAR file can be modified as well by importing it again to the Winery tool[45]. In this master's thesis, different node types with their corresponding relationships are used in the service topology of the PANDAS bone simulation software. The service topology of PANDAS is described in Section 5.2.1.

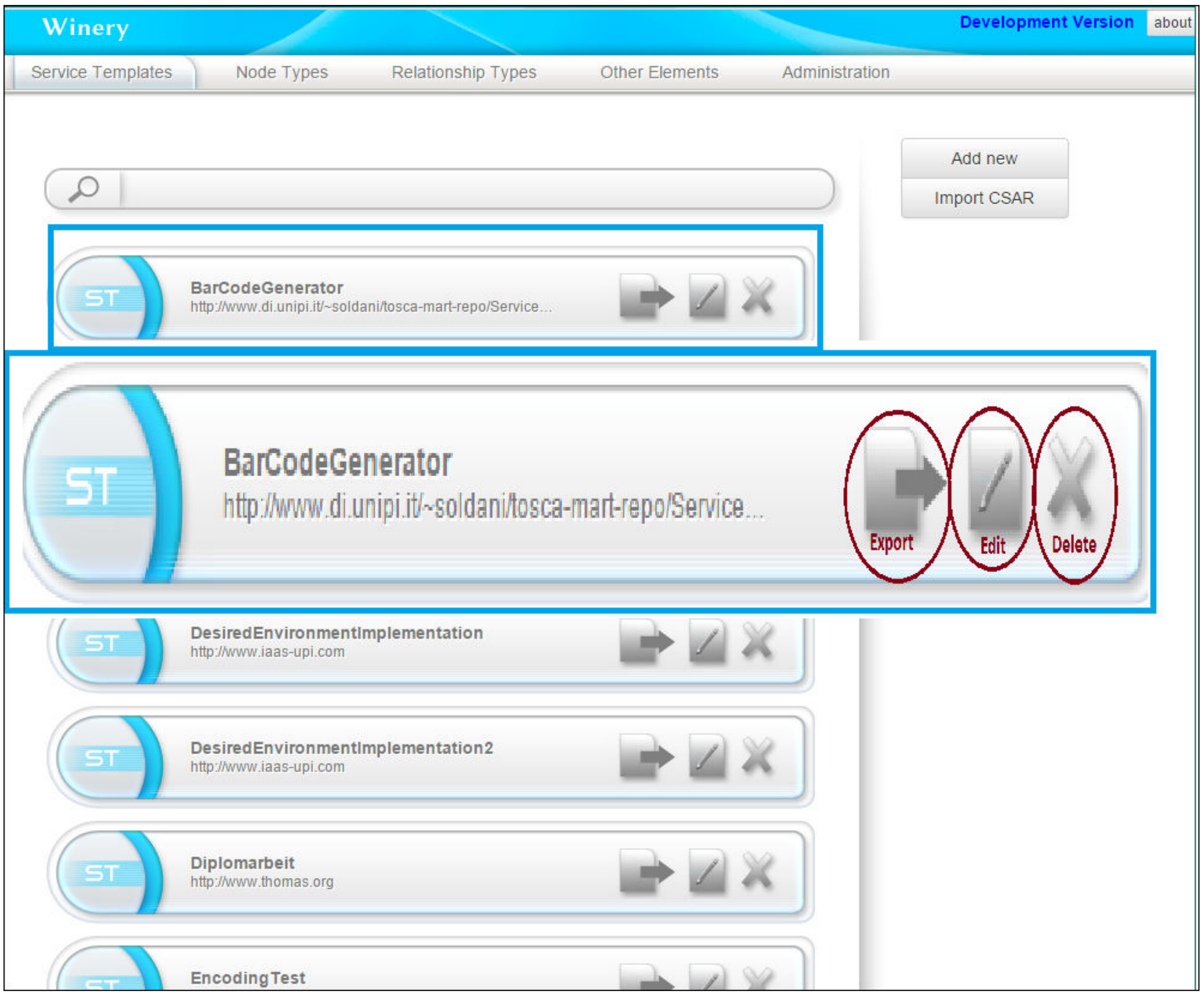


Figure 5.1: Winery Graphical User Interface[45]

Aforementioned, the *ODE-Service* IA is one of the main IAs, which are used in the PANDAS bone simulation service topology. As shown in Figure 5.2, this IA consist of two main operations, so-called `Deploy` and `Undeploy`, for deploying and undeploying BPEL processes to and from Apache ODE engine remotely. The *Deploy* operation of the *ODE-Service* IA requires a `PackageName`, a `Path` and an `OdeEngineIpAddress` as input variables. The `PackageName` defines the name of the BPEL package, which needs to be deployed on the ODE-PGF engine. `Path` variable is the location which this package is stored. The `OdeEngineIpAddress` variable defines the IP address of the workflow engine, which is used for deploying and undeploying processes. Similarly, BPEL workflows can be undeployed from the engine by using the *Undeploy* operation of this *ODE-Service* IA. This operation requires a `PackageName` and an `OdeEngineIpAddress` as input parameters. A specific package, which is defined by the `PackageName` variable, can be removed from the ODE engine with the IP address which can be determined by the `OdeEngineIpAddress` variable. The interface of this IA is generated as a WAR file by clicking on the `generate Implementation Artifact` button as the figure illustrates. Then, the Winery tool creates a Maven[3] project for this IA automatically. The created Maven project can be imported into Eclipse for implementing the actual codes of these operations. To put it another way, with the help of the Winery tool, the web service

[3]https://maven.apache.org/

interface for each IA can be generated, and then the developer can implement the actual web service from the generated interface using a development environment of his or her choice. In the following, the process of developing the *ODE-Service* IA in Eclipse is described.

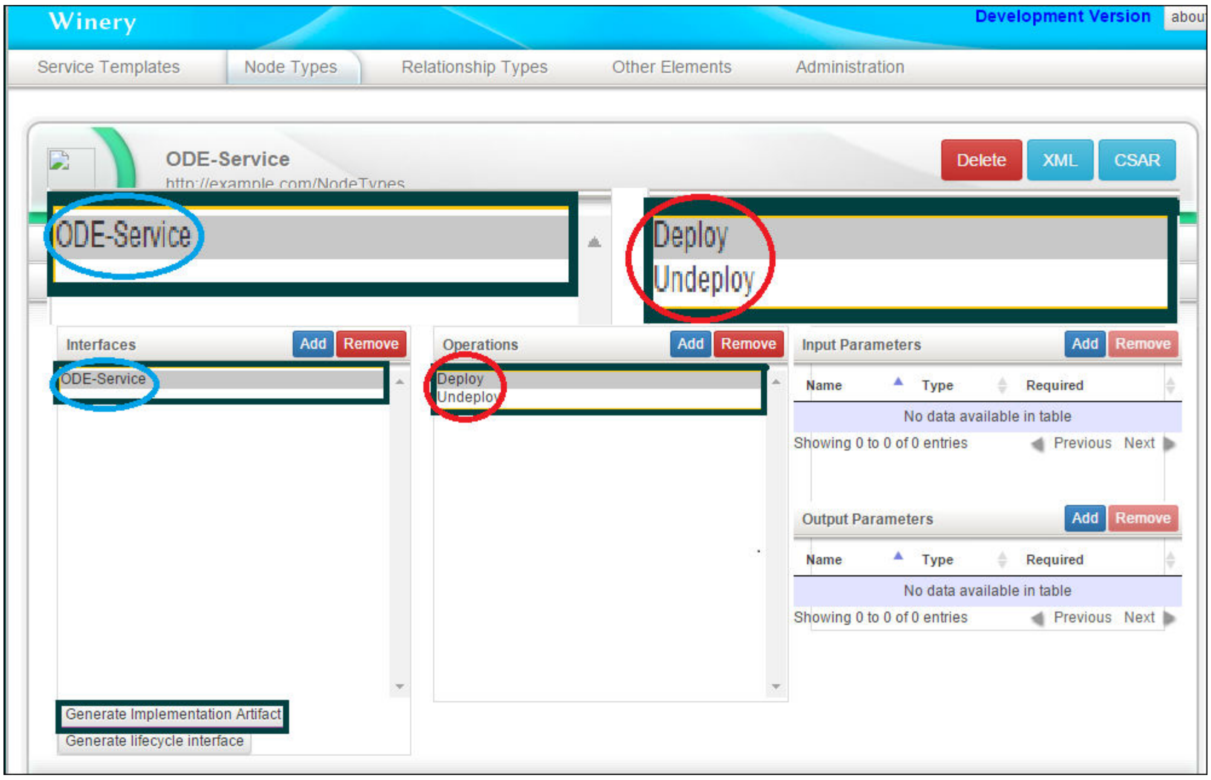


Figure 5.2: ODE-Service IA in Winery[45]

5.1.1.2 ODE-Service IA in Eclipse

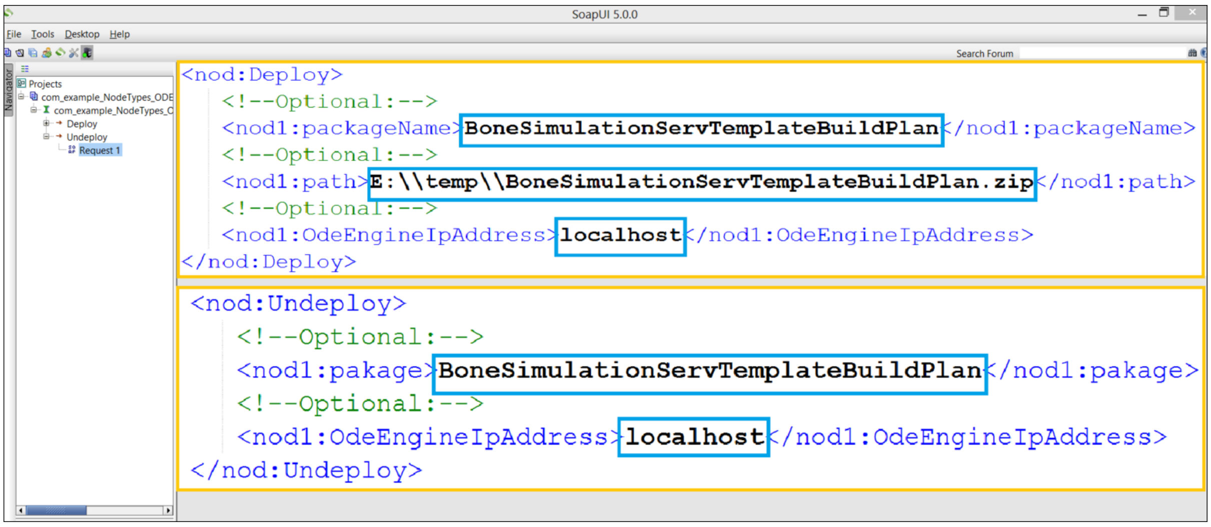
After exporting the *ODE-Service* IA from the Winery tool, it can be imported into Eclipse as a Maven project in order to design and implement the operations. Previously mentioned, this IA is based on the Apache ODE deployment API which is an Axis2 web service. Thence, `org.apache.axis2` library[4] should be included in the project as the main dependency. Other libraries, such as `org.apache.ode`[5], `org.apache.axiom`[6], etc., are used to design and implement this *ODE-Service* IA. Then, after implementing the *ODE-Service* IA, it should be an integral part of the PANDAS bone simulation service topology. This IA is called inside the first workflow of the PANDAS Service Template (see Section 4.2.1), in order to deploy BPEL workflows on the ODE-PGF engine inside the Windows-VM. Similarly, the *Undeploy* function of this IA can be used in the last workflow of the PANDAS Service Template as well (see Section 4.2.6) to undeploy all workflows from the ODE-PGF engine.

5.1.1.3 ODE-Service IA in SoapUI

The *ODE-Service* IA is a WAR file which can be deployed on an application server, such as Apache Tomcat, and then tested with the SoapUI tool. Figure 5.3 shows the GUI of the

[4]http://axis.apache.org/axis2/java/core/api/
[5]http://grepcode.com/project/repo1.maven.org/maven2/org.apache.ode/ode-tools/
[6]http://grepcode.com/search/?start=0&query=org.apache.axiom&entity=type

SoapUI tool for testing the *Deploy* and *Undeploy* functions of this IA. In other words, with the help of the SoapUI tool the input parameters of the operations as well as the SOAP messages which are required to invoke the web service operations can be determined. These input parameters can be used to invoke the operations inside the workflows properly. In other words, these parameters are used to invoke the *Deploy* and *Undeploy* operations inside the first and the last workflows of the PANDAS Service Template.



```
<nod:Deploy>
   <!--Optional:-->
   <nod1:packageName>BoneSimulationServTemplateBuildPlan</nod1:packageName>
   <!--Optional:-->
   <nod1:path>E:\\temp\\BoneSimulationServTemplateBuildPlan.zip</nod1:path>
   <!--Optional:-->
   <nod1:OdeEngineIpAddress>localhost</nod1:OdeEngineIpAddress>
</nod:Deploy>
```

```
<nod:Undeploy>
   <!--Optional:-->
   <nod1:pakage>BoneSimulationServTemplateBuildPlan</nod1:pakage>
   <!--Optional:-->
   <nod1:OdeEngineIpAddress>localhost</nod1:OdeEngineIpAddress>
</nod:Undeploy>
```

Figure 5.3: Testing ODE-Service IA with SoapUI

5.1.1.4 Challenges during implementation and deployment phases

The aforementioned section provided the main details of the implemented prototype for the first goal of this master's thesis, which is integrating TOSCA with the SIMPL framework. Previously mentioned, the *ODE-Service* IA is one of the main IAs inside the TOSCA service topology. As this IA is based on many libraries such as Axis2, Apache ODE, etc., there were many challenges during the implementation phase to deal with all these dependencies and to avoid conflicts between them as well. These issues can be solved by using the Maven tool to ease the management of all these dependencies. Furthermore, the version of each dependency should be consistent with the versions of other dependencies. The author faced other challenges as well for deploying this IA on an application sever, such as Tomcat, during the deployment phase. In other words, Apache ODE and Axis2 engines are installed inside the Apache Tomcat application server. Accordingly, the ODE and Axis2 libraries inside this IA had so many conflicts, according to different versions, with these libraries inside Tomcat. To solve this issue, some libraries inside the *ODE-Service* IA, which have conflict with the ones inside Tomcat and Axis2 have to be removed from the library folder of this IA. For instance, `servlet-api.jar`[7] and `XmlSchema.jar`[8] libraries have to be removed from the library folder of this IA after building the WAR file of this web service.

[7]https://tomcat.apache.org/tomcat-5.5-doc/servletapi/
[8]https://ws.apache.org/commons/XmlSchema/

5.2 Implementation of PANDAS Service Template

This section describes the actual implementations of the prototype which moves the PANDAS bone simulation software into cloud infrastructures. As already mentioned in the previous chapters, the second goal of this master's thesis is to elaborate, develop and implement different variants of a TOSCA Service Template realizing a bone simulation in a cloud-native way. The subsequent sections provide implementation details of the prototype which is designed and implemented in this work in order to turn the PANDAS bone simulation into a SaaS solution. A Service Template consists of an application topology as well as some management plans. As a consequence, the application topology and the management plans of the PANDAS Service Template are discussed in Sections 5.2.1 and 5.2.2 deliberately.

5.2.1 PANDAS service topology

Previously mentioned, the Winery, which is a graphical modelling TOSCA tool, is used in this work in order to design the service topology for the PANDAS bone simulation software. The Winery tool can not be used for developing the management plans inside the Service Template. It can be used only for design and implementation of service topology. Accordingly, these workflows are designed and implemented with other tools, such as Eclipse BPEL Designer, and then imported to the Winery in order to have the complete Service Template. Following that, the TOSCA service topology with all the corresponding workflows are exported from the Winery tool as a CSAR. The OpenTOSCA environment can be used to execute this CSAR file, which realizes the whole Service Template of the PANDAS bone simulation software. Figure 5.4 depicts the service topology with all components and relationships between these components, which was designed and developed for the PANDAS bone simulation in this work.

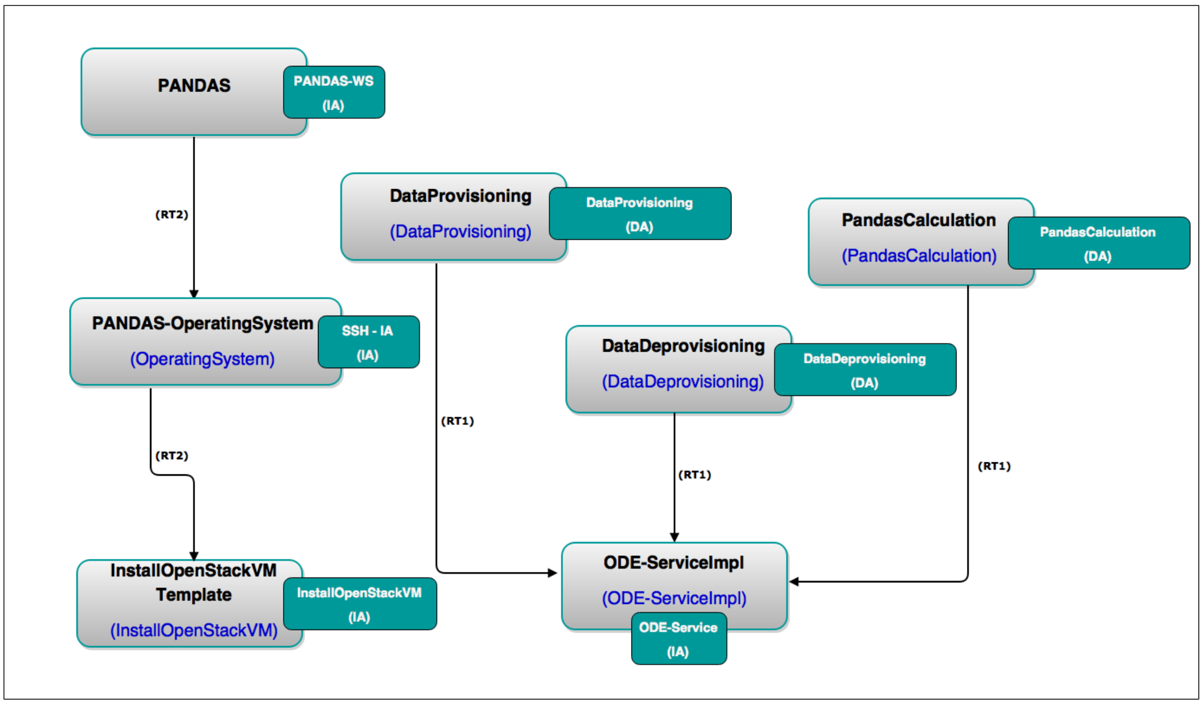


Figure 5.4: PANDAS service topology in Winery

As Figure 5.4 illustrates, four IAs are used in the PANDAS service topology. The first one is the *ODE-Service* IA which is designed and implemented in this master's thesis in order to deploy and undeploy BPEL workflows to an ODE engine remotely. The other three IAs, so-called *SSH-IA*, PANDAS-WS and *InstallOpenStackVM*, were already implemented and only used in this thesis. The *SSH-IA* is used in *InfrastructureProvisioning* plan and in the third variant of *PandasSoftwareProvisioning* plan to run some shell commands on a remote server. The workflows with this IA are discussed in detail in the following sections. The third IA, so-called *InstallOpenStackVM*, installs a new VM with a defined operating system on the OpenStack cloud provider. The fourth IA is the PANDAS-WS IA which is a web service that provides all operations which are related to the PANDAS tool and are used in the workflows.

Besides the above-mentioned IAs, there exist other nodes in the service topology of the PANDAS bone simulation software as Figure 5.4 depicts. The *PANDAS-OperatingSystem* node inside the topology realizes a Linux operating system which is required to be installed on top of the VM which was created with the *InstallOpenStackVM* IA. Previously mentioned, a service topology shows relationships between components inside the topology as well. In the PANDAS service topology, there is a relationship between the *PANDAS-OperatingSystem* node and the *InstallOpenStackVM* node. The *InstallOpenStackVM* node realizes the created VM inside the topology. This node has an attached *InstallOpenStackVM* IA. The relationship between the *PANDAS-OperatingSystem* node and the *InstallOpenStackVM* node realizes that an operating system is installed on top of a VM. This relationship is of type `Installed-On`, as the operating system requires to be installed on top of the created VM after invoking the appropriate operation of the *InstallOpenStackVM* IA. Another node in the topology is the *PANDAS* node which realizes the PANDAS tool and has an attached IA, which is the PANDAS web service. This node shows that the PANDAS tool is installed on top of the *PANDAS-OperatingSystem* node. Therefore, a relationship of type `Installed-On` connects this node to the *PANDAS-OperatingSystem* node.

Three different workflows inside the PANDAS Service Topology (so-called *DataProvisioning*, *Simulation/Application* and *DataDeprovisioning* workflows) need to be deployed on the ODE-PGF engine inside the Windows-VM for execution. Aforementioned, the *ODE-Service* IA is used to deploy these workflows to the Windows-VM remotely from the OpenTOSCA environment. In the PANDAS service topology, it is required to include three node types for these three different workflows. In other words, the actual implementation of these three workflows are added to theses node types as DAs. These three nodes inside the topology are connected to the *ODE-Service* IA with three `Connected-To` relationships. To put it another way, these nodes have to be connected to this IA in order to be able to use the *Deploy* and *Undeploy* operations of this IA. In the following sections, the implementation of six different management plans, which are existed inside the Service Template of PANDAS, are discussed deliberately.

5.2.2 PANDAS management plans

For realizing the PANDAS bone simulation software as a SaaS solution in cloud infrastructures, six various types of plans are required, as they were discussed in Section 4.2. The overall and complex TOSCA management plan is divided into six different workflows as this kind of modularization is useful for the application. Previously mentioned, BPEL is used as a workflow language to design and implement all management plans in this master's thesis.

Some of these workflows are deployed on the OpenTOSCA environment and executed on the workflow engine inside OpenTOSCA. On the other hand, some other plans are deployed on the SIMPL prototype and then executed on the ODE-PGF engine inside the SIMPL framework. For implementing most of the management plans in this thesis, the Eclipse BPEL Designer as the standard workflow design tool and the extended Eclipse BPEL Designer inside the SIMPL prototype are used. For implementing the *InfrastructureProvisioning* and *Termination* workflows, Notepad++[9] was used. The following subsections define the implementation of these management plans inside the Service Template of PANDAS in detail.

5.2.2.1 Plan number one: InfrastructureProvisioning plan

InfrastructureProvisioning plan is the first plan in the PANDAS Service Template, which is invoked in order to prepare the main infrastructures for the PANDAS bone simulation software in cloud environments. This workflow is executed inside the OpenTOSCA container, and provides an essential framework for other workflows which are inside the PANDAS Service Template. In this workflow the IAs inside the PANDAS service topology are called via a service invoker, so-called *OpenTOSCA Service Invoker*[10], which was originally developed by Michael Zimmermann as a bachelor thesis[21] at IAAS of the University of Stuttgart. Accordingly, in implementation phase of this workflow, the BPEL operations are called via the *OpenTOSCA Service Invoker* in an asynchronous way. This workflow consists of four primary tasks, which were already discussed in Section 4.2.1, as follows: (1) creating a VM for the PANDAS bone simulation software and its corresponding PostgreSQL database, (2) executing the required shell commands on the created VM in order to configure the VM, (3) deploying some BPEL workflows, which are inside the PANDAS Service Template, to the ODE-PGF engine inside the Windows-VM and (4) registering the new IP address of the created VM on the resource management database of the SIMPL framework.

For creating a new VM on the OpenStack cloud provider, the *InstallOpenStackVM* IA, which is inside the PANDAS service topology, is used. This is an already implemented IA which creates a new VM on the OpenStack cloud provider. All operations of this IA are called via the *OpenTOSCA Service Invoker*. The actual implementation of this implementation artifact can be found on the GitHub[11]. In this work, the *InstallVMwithGeneratedKeypair* operation of this IA was used for creating a new VM on the OpenStack cloud provider. The *SSH-IA* is another IA which is used in this work to run some shell commands on a server remotely. In the current implementation of the PANDAS web service for example, we need to restart the PostgreSQL database and the Apache Tomcat server after each simulation run. Otherwise, the calculation might not be executed properly. Accordingly, the *runScript* operation of this *SSH-IA* is used in this work to send some shell commands for these two issues. Like other invocations, this operation is called via the *OpenTOSCA Service Invoker*. Listing 5.1 illustrates the shell commands which are sent to the Linux-VM with the help of this *SSH-IA*.

```
# Tomcat shutting down
./apache-tomcat-6.0.43/bin/shutdown.sh;

# Delay for 5s
```

[9]https://notepad-plus-plus.org/
[10]http://install.opentosca.org/documentation/Documents/InstallationGuide.pdf
[11]https://github.com/tosca-types/openstack

```
sleep 5s;

# Solve a problem related to shared kernel memory. The sysctl command
   is used to configure a shared memory. The shmmax command is used
  to define the maximum size (in bytes) for a shared memory segment.
sudo sysctl -w kernel.shmmax=1073741824;

# Starting Tomcat and delay for 20s
nohup ./apache-tomcat-6.0.43/bin/startup.sh; sleep 20s;

# Restart PostgreSQL
sudo service postgresql restart;
```

Listing 5.1: Restarting Tomcat and PostgreSQL database

Aforementioned, for deploying the BPEL workflows on the ODE-PGF engine inside the SIMPL prototype, the *ODE-Service* IA was designed and developed in this master's thesis (see Section 5.1.1). As Figure 5.4 shows, one node template is used for each workflow which needs to be deployed on the Windows-VM. The actual implementation of these workflows (see Sections 5.2.2.3, 5.2.2.4 and 5.2.2.5) are added to these nodes as DAs. In implementation phase of this workflow, there were many problems for integrating this step with the previous steps. In order to address the goal, a separate workflow was designed and implemented in order to call the `Deploy` operation of the *ODE-Service* IA for these three workflows. Therefore, the *InfrastructureProvisioning* workflow was divided into two workflows. The last step in *InfrastructureProvisioning* workflow is registering the new IP address of the created VM on the resource management database of the SIMPL framework. The SIMPL resource management web service is used in this work for registering a new entry in the database. This web service contains a function, so-called *AddDataSource*, to add a new data source inside the PostgreSQL database of the SIMPL framework. By adding a new data source for the created VM, the provisioning and deprovisioning processes of the required data sources for the PANDAS bone simulation become feasible. Similar to the previous step, adding the *AddDataSource* operation of the SIMPL web service to the *InfrastructureProvisioning* workflow was challenging and problematic in the implementation phase. Therefore, this operation was added to the second workflow, which was implemented for calling the `Deploy` operation of the *ODE-Service* IA. In conclusion, the *InfrastructureProvisioning* workflow was divided into two workflows. The first workflow creates a new VM on the OpenStack cloud provider and configures the VM by running some shell commands. The second workflow invokes the `Deploy` operation of the *ODE-Service* IA for the three above-mentioned workflows, which need to be deployed on the ODE-PGF engine, and registers the IP address of the new VM to the PostgreSQL database of the SIMPL framework as well.

5.2.2.2 Plan number two: PandasSoftwareProvisioning plan

The second plan, so-called *PandasSoftwareProvisioning* plan, is a simple plan which is used for preparing a PANDAS instance, one for the mechanical and one for the chemical calculation. For realizing the PANDAS bone simulation software as a SaaS solution in cloud infrastructures, this plan is called two times in order to create two PANDAS instances for the mechanical and chemical calculations, respectively. In the scope of this master's thesis, three different variants for this workflow were designed and implemented. The first version has exactly the same activities which were discussed in Section 4.2.2.1. In this workflow, the IP address of the created VM in the first plan is used in order to copy the PANDAS source code on the right VM. For this purpose, this IP address is received from the input parameter of the workflow and then assigned to the PANDAS partnerLink[12]. The PANDAS partnerLink is a partnerLink which was defined inside the Eclipse BPEL Designer and realizes the PANDAS web service. Then, the *Platform_ Provisioning* and *Pandas_ Source* operations of the PANDAS web service are executed on this new created VM. Figure 5.5 depicts the actual implementation of this workflow with the help of Eclipse BPEL Designer.

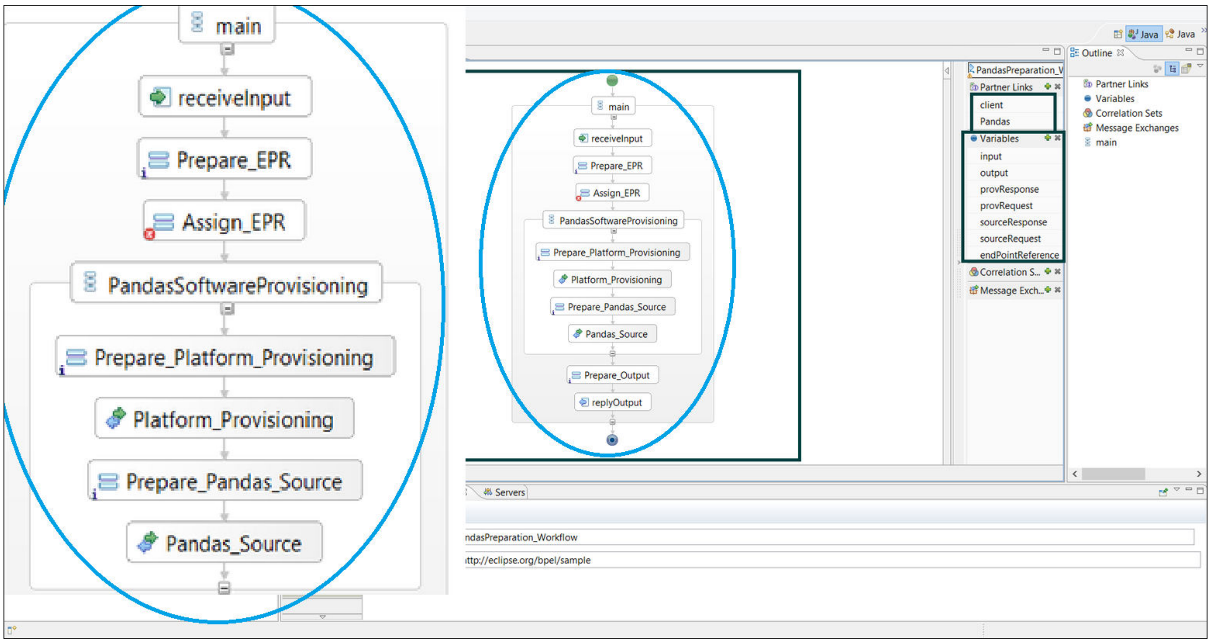


Figure 5.5: Implementation of PandasSoftwareProvisioning plan - First version

As mentioned in Section 4.2.2.2, there are many configuration parameters in the PostgreSQL database which affect the behaviour of the database server. By modifying these parameters, different variants for the *PandasSoftwareProvisioning* plan can be designed and implemented. For instance, the PANDAS web service has an operation, so-called *Send_ Update_ Query*, which can be used to set different configuration parameters for the PostgreSQL database of the PANDAS bone simulation tool. Figure 5.6 shows the implementation of the second variant of the *PandasSoftwareProvisioning* plan. The query which is required to change these configuration parameters can be received as a string from the input parameter of the workflow. Then, the query is executed to the database server by calling the *Send_ Update_ Query* operation of the PANDAS web service. Some of these queries, which were used in this work for testing the second variant of the *PandasSoftwareProvisioning* plan, are discussed as follows.

[12]https://www.packtpub.com/sites/default/files/downloads/7948_AppendixA.pdf

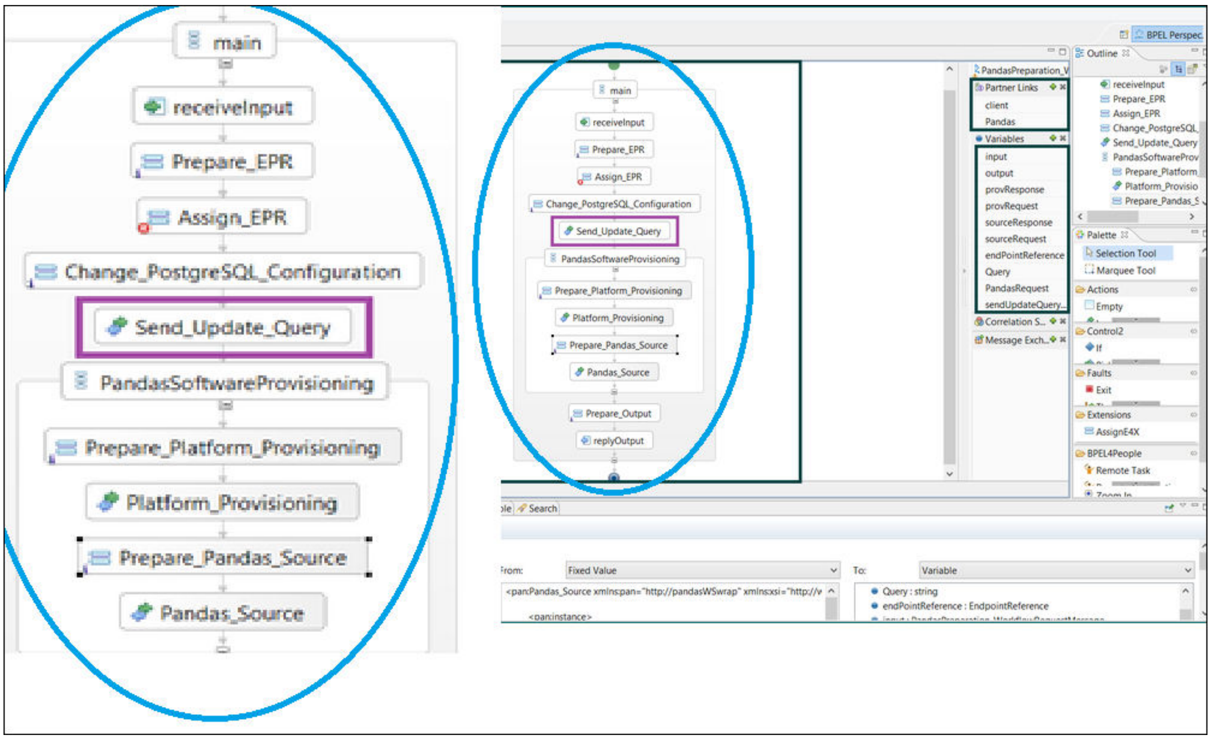


Figure 5.6: Implementation of PandasSoftwareProvisioning plan - Second version

The PostgreSQL database provides three SQL commands as the following, which can establish different configuration parameters[23]:

- `ALTER SYSTEM`: this command can be used to change global settings of the whole PostgreSQL server. This command is equivalent to edit the `postgresql.conf` file, which was discussed in Section 4.2.2.2, and are applied across the entire database cluster.
- `ALTER DATABASE`: this command allows global settings to be overwritten for a single database within the cluster.
- `ALTER ROLE`: this command allows both global settings as well as the database settings to be customized with user-specific values.

Listing 5.2 illustrate how these commands can be used in order to change different configurations of the PostgreSQL database. The `ALTER SYSTEM` and the `ALTER DATABASE` commands are used in this work to test functionalities of PostgreSQL configuration parameters of the PANDAS database. The first code example shows the usage of the `ALTER SYSTEM` command. By using the `DEFAULT` value, the customized setting for that specific parameter will be removed from the configuration files. Customized configurations for parameters can additionally be removed by using the `RESET` keyword for each parameter separately, or by using the `RESET ALL` keyword for all configuration parameters[23].

```
ALTER SYSTEM SET configuration_parameter{TO|=}{value|'value'|DEFAULT}
ALTER SYSTEM RESET configuration_parameter
ALTER SYSTEM RESET ALL
```

Listing 5.2: ALTER SYSTEM command in PostgreSQL database [23]

The `ALTER DATABASE` command changes configuration parameters for a specific database. As shown in Listing 5.3, different names, owners, table spaces, and some configuration parameters can be customized for each table in the PostgreSQL database system. Therefore, this offers fine-grained, but also more complicated possibilities to change configuration parameters[23].

```
ALTER DATABASE name [ [ WITH ] option [ ... ] ]
where option can be:
    CONNECTION LIMIT \emph{connlimit}
    NAME \emph{newname}
    NEW_OWNER \emph{newowner}
    NEW_TABLESPACE \emph{newtablespace}
    SET configuration_parameter { TO | = } { value | DEFAULT }
    RESET configuration_parameter
    RESET ALL
```

Listing 5.3: ALTER DATABASE command in PostgreSQL database [23]

Most of configuration parameters can be customized locally for a session with the `SET`[13] command as well. This command has no effect on other sessions. A simple example could be a query which can change the time zone of the database server. This query can be received as a string from the input parameter of the *PandasSoftwareProvisioning* plan and then applied to the database system. Listing 5.4 shows this query. In this example, the time zone for the database is first set to Berkeley, California and then changed to the Italy time zone. For testing all these SQL commands, the second variant of the *PandasSoftwareProvisioning* workflow was used.

```
--set the time zone to Berkeley, California
SET TIME ZONE 'PST8PDT';

--set the time zone to Italy
SET TIME ZONE 'Europe/Rome';
```

Listing 5.4: Changing the time zone of the PostgreSQL database [23]

In the third version of the *PandasSoftwareProvisioning* plan as Figure 5.7 depicts, the *SSH-IA* is used to make a connection to the PostgreSQL database of the PANDAS bone simulation software, and then execute some queries on the server remotely. These queries can be used to change configuration parameters of the database. Accordingly, the *runScript* operation of this IA is invoked in this workflow to change database settings. The rest of the plan remains the same as in the previous versions. All activities which were used during the implementation phase of the third version of the *PandasSoftwareProvisioning* plan are exactly the same as in Section 4.2.2.3.

[13]http://www.postgresql.org/docs/current/interactive/sql-set.html

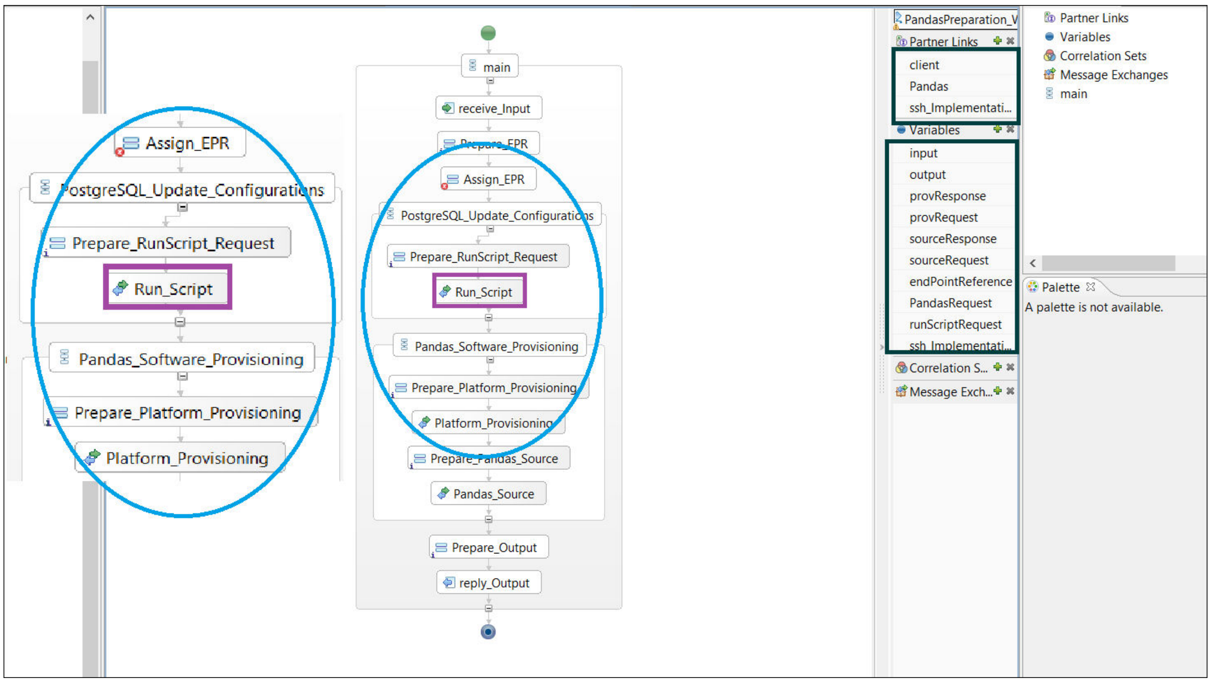
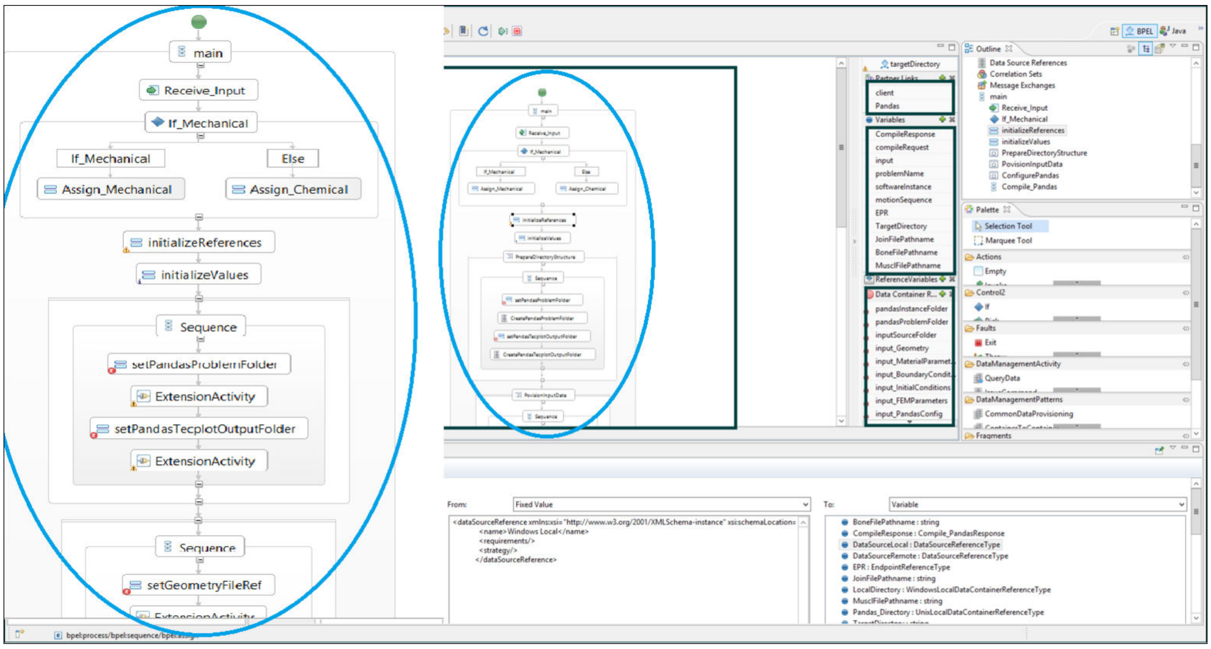


Figure 5.7: Implementation of PandasSoftwareProvisioning plan - Third version

5.2.2.3 Plan number three: DataProvisioning plan

As mentioned in Section 4.2.3, the third workflow inside the PANDAS Service Template, so-called *DataProvisioning* plan, provides and prepares heterogeneous input data of the simulation in order that PANDAS instances can properly ingest these data. As the input data sources are different for the mechanical and chemical instances, this workflow is executed separately for each of the two PANDAS software instances. In other words, in the PANDAS Service template which realizes PANDAS as a SaaS solution in cloud infrastructures, this workflow is called two times. The SIMPL framework was used to design and implement different versions of this workflow. In the scope of this master's thesis, two different variants for the *DataProvisioning* workflow were designed and implemented. These two variants can be used to provide and prepare heterogeneous input data of both mechanical and chemical instances of the PANDAS bone simulation software. As discussed before in Section 4.2.3.1, based on one of the input parameters of this workflow, so-called `SoftwareInstance` parameter, the correct input files in the source folder are selected for the mechanical or chemical instances.

In the first variant of this workflow, the main activities inside the workflow are exactly the same as discussed in Section 4.2.3.1. Figure 5.8 shows the actual implementation of the first version of this plan inside Eclipse BPEL Designer. This workflow consist of a large number of activities, but the figure below illustrates only some first activities of the workflow for the sake of simplicity. As Figure 5.8 depicts, this workflow needs the PANDAS web service as a partnerLink. This partnerLink is required in order to invoke the *CompilePandas* operation of this web service at the end of the workflow. The new IP address of the created VM is assigned to the PANDAS partnerLink before invoking the *CompilePandas* operation. As a result, the source code of the PANDAS tool and all the provisioned data are compiled together. As the figure illustrates, some SIMPL variables are used in this workflow to define different *DataContainer* and *DataSource* variables. All activities which were used in the implementation phase of this workflow are exactly the same as in Section 4.2.3.1.



From: Fixed Value

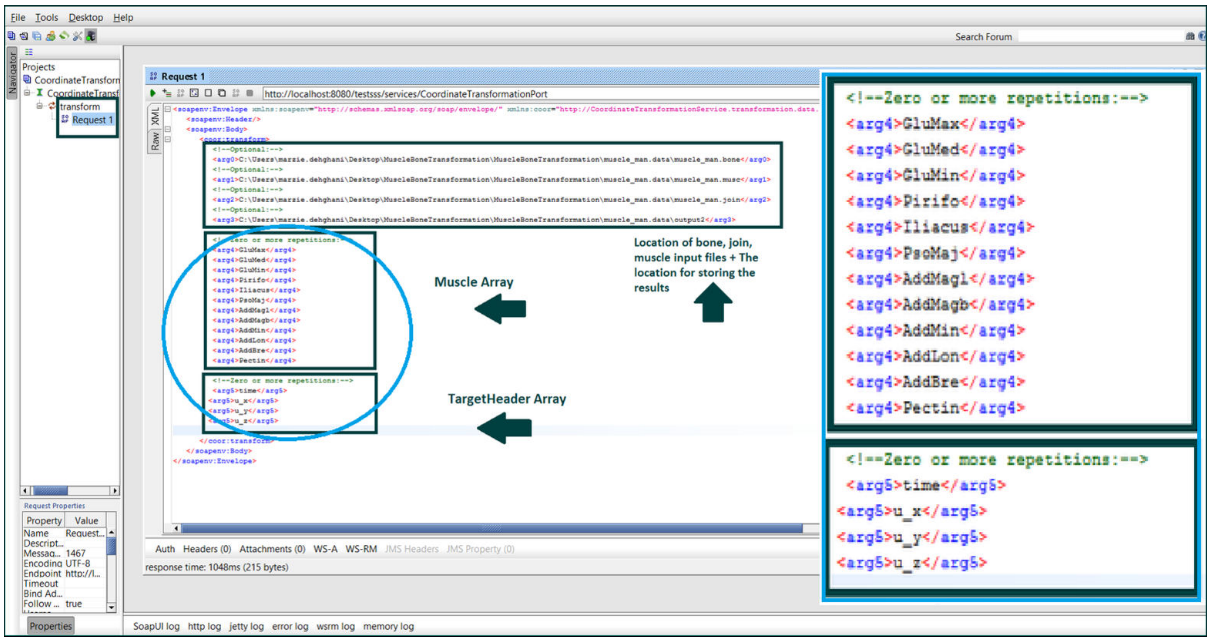
```
<dataSourceReference xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
        <name>Windows Local</name>
        <requirements/>
        <strategy/>
      </dataSourceReference>
```

To: Variable

- BoneFilePathname : string
- CompileResponse : Compile_PandasResponse
- DataSourceLocal : DataSourceReferenceType
- DataSourceRemote : DataSourceReferenceType
- EPR : EndpointReferenceType
- JoinFilePathname : string
- LocalDirectory : WindowsLocalDataContainerReferenceType
- MusclFilePathname : string
- Pandas_Directory : UnixLocalDataContainerReferenceType

Figure 5.8: Implementation of DataProvisioning plan - First version

Previously mentioned, the PANDAS bone simulation software requires heterogeneous data sources as input. As shown in Figure 2.7, PANDAS requires some CSV files for its *boundary conditions* input parameter. Different kinds of CSV-based files, which have a different format, can be used as well for PANDAS instances. These CSV files require a coordinate transformation in order to be appropriate for PANDAS instances as an input parameter. Therefore, the second version of the *DataProvisioning* workflow uses an additional web service in order to transform the coordination of these CSV files. For this purpose, a JAX-WS was designed and implemented in this master's thesis for transforming the coordination of the *boundary conditions* CSV files. As Figure 5.9 depicts, the *transform* operation of this web service requires six different input variables. In other words, it needs references to three CSV files (CSV files for bone, joint and muscle) as input parameters. Based on these CSV files and two string arrays for the muscles and target header, it creates some other CSV files as output. The target directory for storing the generated CSV files should be passed as an input to this operation. For example as the figure shows, the muscle array consists of twelve strings which define the names of different muscles inside a human body. Based on these twelve strings inside the array, twelve CSV files for the muscles and one CSV file for the joint are created as output of calling the *transform* operation.


Request 1
http://localhost:8080/testsss/services/CoordinateTransformationPort
Location of bone, join, muscle input files + The location for storing the results
Muscle Array
TargetHeader Array
response time: 1048ms (215 bytes)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:coor="http://CoordinateTransformationService.transformation.data.
   <soapenv:Header/>
   <soapenv:Body>
      <coor:transform>
         <!--Optional:-->
         <arg0>C:\Users\marzie.dehghani\Desktop\MuscleBoneTransformation\MuscleBoneTransformation\muscle_man.data\muscle_man.bone</arg0>
         <!--Optional:-->
         <arg1>C:\Users\marzie.dehghani\Desktop\MuscleBoneTransformation\MuscleBoneTransformation\muscle_man.data\muscle_man.musc</arg1>
         <!--Optional:-->
         <arg2>C:\Users\marzie.dehghani\Desktop\MuscleBoneTransformation\MuscleBoneTransformation\muscle_man.data\muscle_man.join</arg2>
         <!--Optional:-->
         <arg3>C:\Users\marzie.dehghani\Desktop\MuscleBoneTransformation\MuscleBoneTransformation\muscle_man.data\output2</arg3>
         <!--Zero or more repetitions:-->
         <arg4>GluMax</arg4>
         <arg4>GluMed</arg4>
         <arg4>GluMin</arg4>
         <arg4>Pirifo</arg4>
         <arg4>Iliacus</arg4>
         <arg4>PsoMaj</arg4>
         <arg4>AddMagl</arg4>
         <arg4>AddMagb</arg4>
         <arg4>AddMin</arg4>
         <arg4>AddLon</arg4>
         <arg4>AddBre</arg4>
         <arg4>Pectin</arg4>
         <!--Zero or more repetitions:-->
         <arg5>time</arg5>
        <arg5>u_x</arg5>
        <arg5>u_y</arg5>
        <arg5>u_z</arg5>
      </coor:transform>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.9: Testing CoordinateTransformer web service with SoapUI

Following that, the second version of the *DataProvisioning* plan can be designed and implemented. As figure 5.10 illustrates, this version uses the *transform* operation of the above-mentioned web service in order to transform the coordination of the *boundary conditions* CSV files. Then, all other required data sources and theses CSV files can be transferred to the Linux-VM. As Figure 4.9 in the previous chapter shows, all these required files and variables for calling the *transform* operation can be received in the workflow as input parameters. As a consequence, this operation is invoked inside the workflow instead of reading the already created *boundary conditions* CSV files from the local directory. The rest of the workflow for provisioning and transferring other data sources to the Linux-VM remain the same as in the previous version.

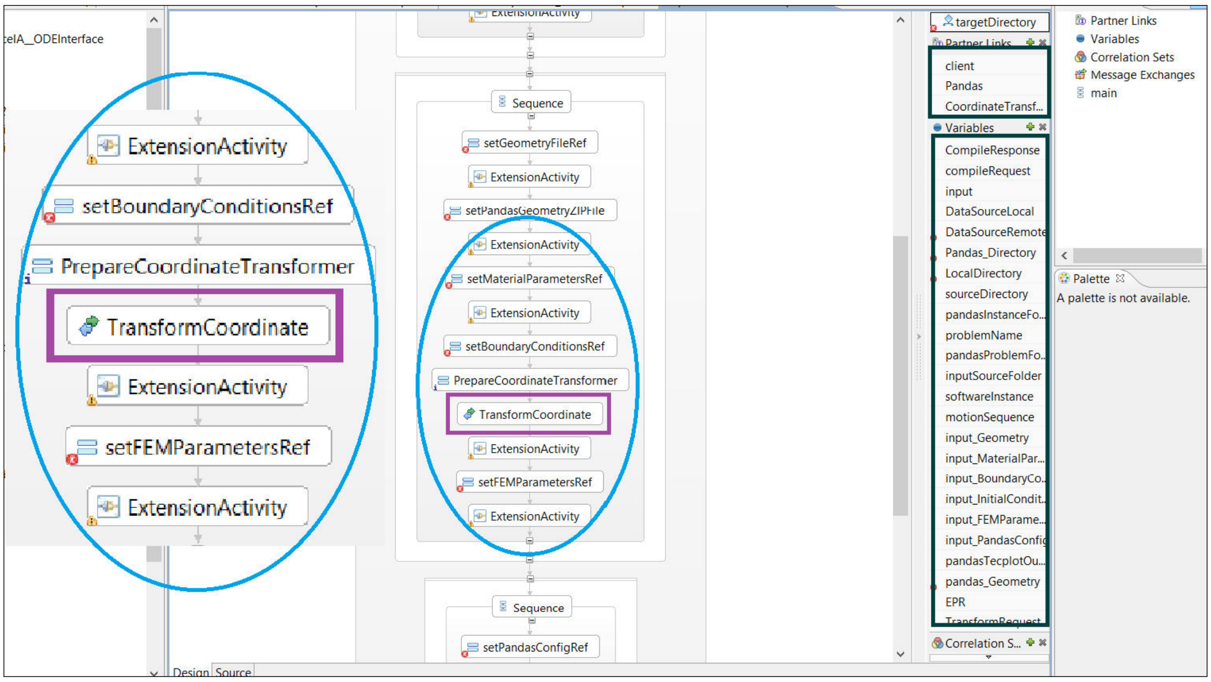


Figure 5.10: Implementation of DataProvisioning plan - Second version

5.2.2.4 Plan number four: Simulation/Application plan

The fourth plan, so-called *Simulation/Application* plan, realizes the actual simulation of the PANDAS bone simulation tool. The main activities of this workflow are the same as mentioned in Section 4.2.4. This workflow calculates the simulation for both mechanical and chemical PANDAS instances at the same time. Accordingly, this workflow is executed only once inside the PANDAS Service Template. As Figure 5.11 illustrates, this workflow contains the PANDAS web service as a partnerLink in order to invoke different operations of this web service. The IP address of the created VM in the first plan is received as an input parameter of the workflow and then assigned to the PANDAS partnerLink in order to execute all the activities on the new created VM. There are two important issues to be considered inside this workflow:

1. The calculation of the instances lasts something about half an hour. But the web service operation accesses PANDAS in an asynchronous manner and is thus finished after one minute. As a workaround, a BPEL wait activity which waits for 45 minutes was added after calling the *Calculate_ Pandas* operation in order that everything works properly.
2. It is more efficient to implement the PANDAS web service in a way which calls *Execute_ Command* and *Stop_ Pandas* operations separately for each PANDAS instance. But, the current version of PANDAS calls the *Execute_ Command* as well as the *Stop_ Pandas* operations only once for both instances.

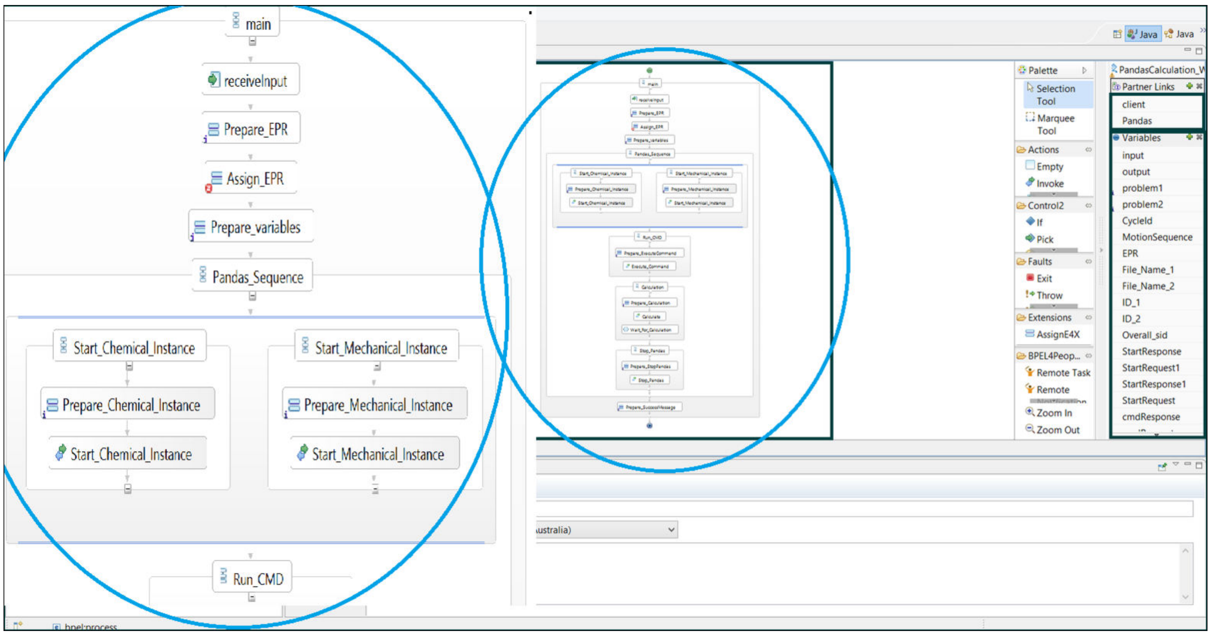


Figure 5.11: Implementation of Simulation/Application plan

5.2.2.5 Plan number five: DataDeprovisioning plan

The fifth plan, so-called *DataDeprovisioning* plan, returns the result data of PANDAS instances back to users. In other words, in the *DataDeprovisioning* workflow the calculation results are transferred from the Linux-VM, which was installed in the first plan, to the Windows-VM in order to save the calculations in an appropriate folder. In the scope of this master's thesis, five different approaches for transferring the results of the calculation to the Windows VM were discussed conceptually in the previous chapter. In the implementation phase of this workflow, five various versions were implemented and tested based on the approaches which were discussed in the previous chapter. It is important to note that in Section 4.2.5, five different approaches for transferring data from the Linux-VM to the Windows-VM were discussed conceptually. In the implementation phase, five different variants were designed and implemented which are not totally the same as the variants in the previous chapter. All these variants are discussed in detail as follows. For implementing all these different variants, the SIMPL framework with extended Eclipse BPEL Designer was used.

In the first variant of this workflow, one SIMPL *TransferData* activity is used to transfer the outcome of the PANDAS calculation from the *PandasTecplotOutputFolder* to an appropriate folder in the Windows-VM on the OpenStack cloud provider. All activities in this version are exactly the same as discussed in Section 4.2.5.1. The *PandasTecplotOutputFolder* folder is a folder which was created in the *DataProvisioning* plan for storing the results of the calculation. As Figure 5.12 shows, this simple workflow contains two *DataSource Reference* variables, so-called *DataSourcelocal* and *DataSourceRemote*. The *DataSourcelocal* variable is of type `Unix Local`, and the *DataSourceRemote* variable is of type `Windows Local` as we need to transfer data from a Linux-VM to a Windows-VM.

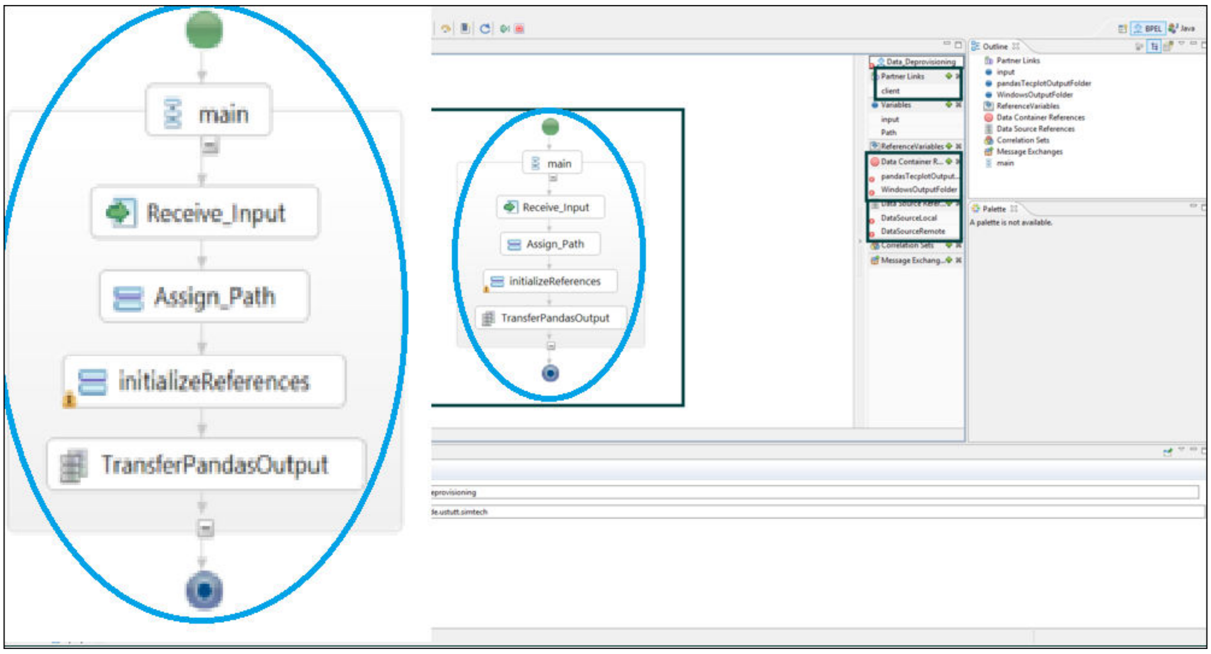
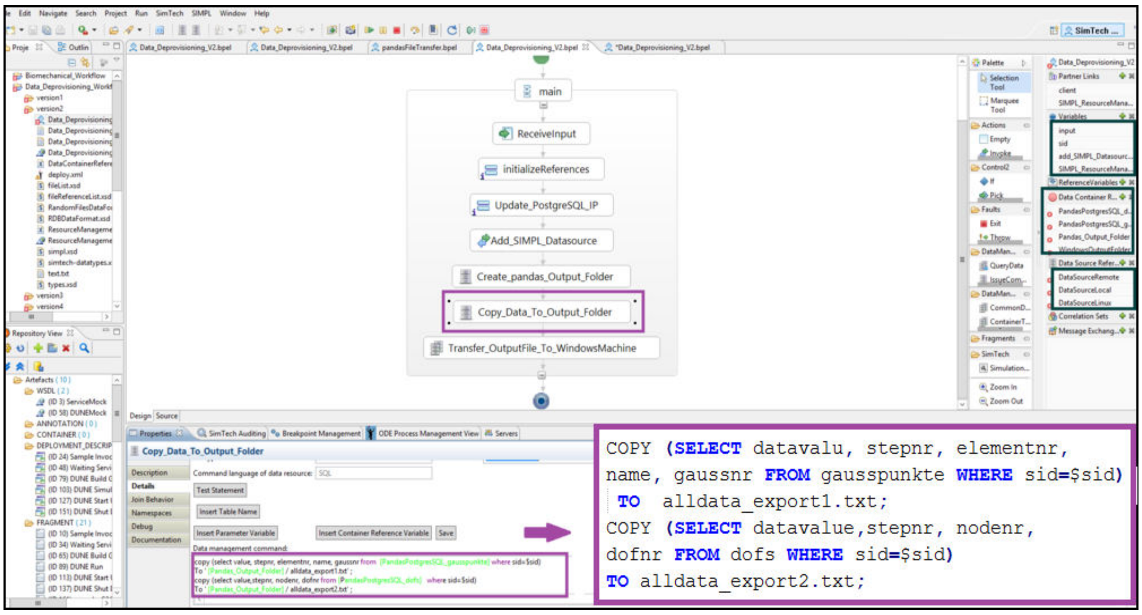


Figure 5.12: Implementation of DataDeprovisioning plan - First version

In the second version of the *DataDeprovisioning* workflow, instead of transferring PANDAS results from the *PandasTecplotOutputFolder* folder to the Windows-VM, a query is sent to the PostgreSQL database of PANDAS in order to extract the data. By executing this query, the appropriate results are copied to one or more specific files. Then, the next activity in this workflow transfers these files to the Windows-VM by using a *TransferData* activity of the SIMPL prototype. It is important to note that the second version of the *DataDeprovisioning* workflow, as shown in Figure 5.13, was implemented exactly the same as discussed in Section 4.2.5.2. Different versions of this workflow can also be provided by changing the query which is sent to the PostgreSQL database. In the second version, the query in Listing 5.5 copies the results of the calculation for a specific PANDAS instance to a specific file. To put it another way, the simulation ID of a PANDAS instance can be received from an input parameter of the workflow, and then the appropriate data based on the query is retrieved from two specific tables, so-called *gausspunkte* and *dofs*, in the PostgreSQL database. The results of the query is copied to some text files, and then these text files are transferred to the Windows-VM. By executing this workflow, the calculation results for one PANDAS instance can be retrieved from the PostgreSQL database.

```
COPY (SELECT datavalu, stepnr, elementnr, name, gaussnr FROM
    gausspunkte WHERE sid=$sid) TO  alldata_export1.txt;
COPY (SELECT datavalue,stepnr, nodenr, dofnr FROM dofs WHERE sid=$sid)
     TO alldata_export2.txt;
```

Listing 5.5: Query for the second version of DataDeprovisioning plan



```
COPY (SELECT datavalu, stepnr, elementnr,
name, gaussnr FROM gausspunkte WHERE sid=$sid)
 TO  alldata_export1.txt;
COPY (SELECT datavalue,stepnr, nodenr,
dofnr FROM dofs WHERE sid=$sid)
TO alldata_export2.txt;
```

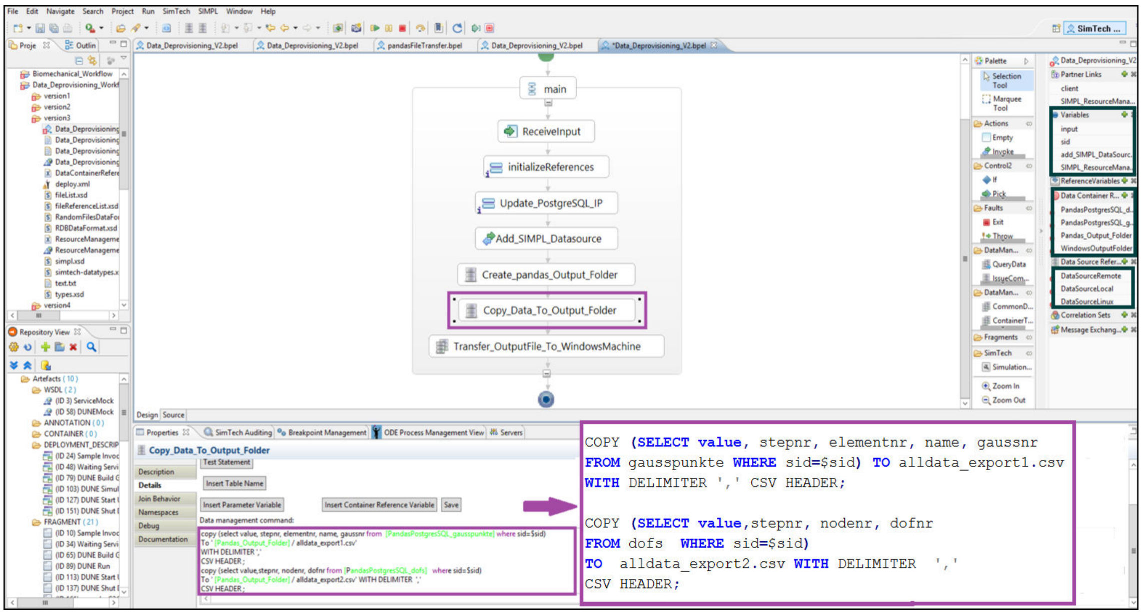
Figure 5.13: Implementation of DataDeprovisioning plan - Second version

The third version of the *DataDeprovisioning* workflow in the implementation phase is not the same as the third version which were discussed in Section 4.2.5.3. In other words, this variant as well as the following variants of the *DataDeprovisioning* workflow in this chapter can be considered simply as other variants of the second version, which was discussed in Section 4.2.5.2. As Listing 5.6 depicts, instead of using text files for copying the results of the query, some CSV files are used. To put it another way as Figure 5.14 illustrates, the simulation ID of a PANDAS instance is received from an input parameter of the workflow. Then, the query is executed and the results are copied to the CSV files. As the listing below illustrates, some different options for the CSV files can be determined. To clarify, with the DELIMITER keyword the delimiter for the CSV files can be customized. In this query, the delimiter is defined as a comma.

```
COPY (SELECT value, stepnr, elementnr, name, gaussnr FROM gausspunkte
    WHERE sid=$sid) TO alldata_export1.csv  WITH DELIMITER ',' CSV
    HEADER;

COPY (SELECT value,stepnr, nodenr, dofnr FROM dofs  WHERE sid=$sid) TO
     alldata_export2.csv WITH DELIMITER  ','
CSV HEADER;
```

Listing 5.6: Query for the third version of DataDeprovisioning plan


main
ReceiveInput
initializeReferences
Update_PostgreSQL_IP
Add_SIMPL_Datasource
Create_pandas_Output_Folder
Copy_Data_To_Output_Folder
Transfer_OutputFile_To_WindowsMachine
Copy_Data_To_Output_Folder
Test Statement
Insert Table Name
Insert Parameter Variable
Insert Container Reference Variable
Save
Data management command:

```
copy (select value, stepnr, elementnr, name, gaussnr from [PandasPostgresSQL_gausspunkte] where sid=$sid)
To ' [Pandas_Output_Folder] / alldata_export1.csv'
WITH DELIMITER ','
CSV HEADER;
copy (select value,stepnr, nodenr, dofnr from [PandasPostgresSQL_dofs]  where sid=$sid)
To ' [Pandas_Output_Folder] / alldata_export2.csv' WITH DELIMITER ','
CSV HEADER;
```

```
COPY (SELECT value, stepnr, elementnr, name, gaussnr
FROM gausspunkte WHERE sid=$sid) TO alldata_export1.csv
WITH DELIMITER ',' CSV HEADER;

COPY (SELECT value,stepnr, nodenr, dofnr
FROM dofs  WHERE sid=$sid)
TO  alldata_export2.csv WITH DELIMITER  ','
CSV HEADER;
```

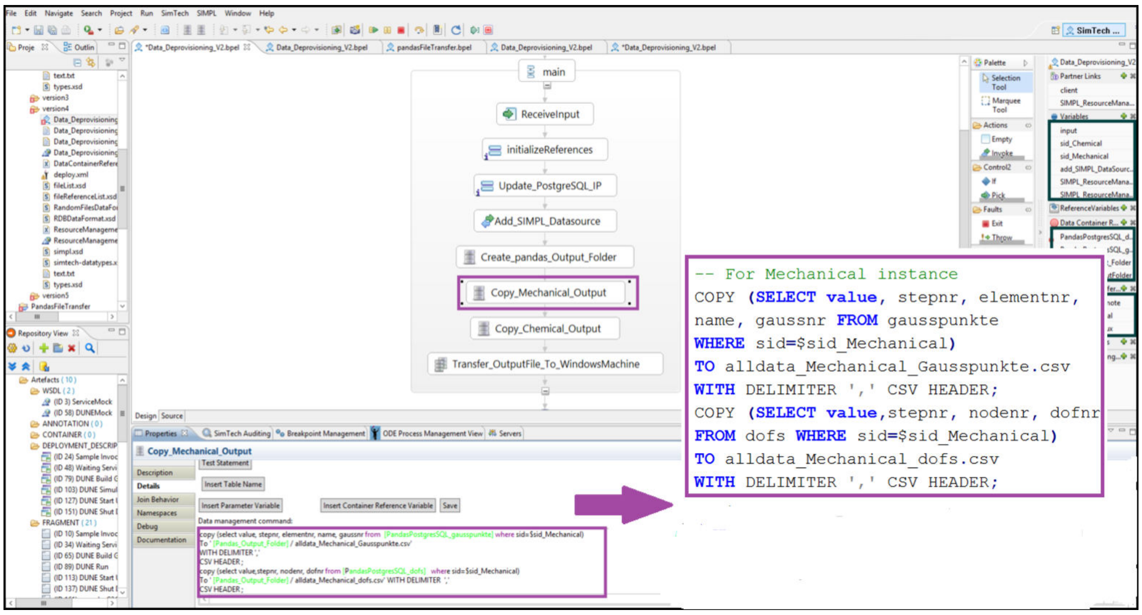
Figure 5.14: Implementation of DataDeprovisioning plan - Third version

In the fourth version of the *DataDeprovisioning* workflow as Figure 5.15 depicts, the results of the calculation can be retrieved from the database for both PANDAS instances, which are mechanical and chemical instances. The simulation IDs for both instances are received from the input parameters of the workflow, and based on the query the results are copied to some CSV files. Following that, these CSV files are transferred to the Windows machine. Listing 5.7 illustrates the query which was used for this version.

```
-- For Mechanical instance
COPY (SELECT value, stepnr, elementnr, name, gaussnr FROM gausspunkte
    WHERE sid=$sid_Mechanical) TO alldata_Mechanical_Gausspunkte.csv
    WITH DELIMITER ',' CSV HEADER;
COPY (SELECT value,stepnr, nodenr, dofnr FROM dofs WHERE sid=
    $sid_Mechanical) TO alldata_Mechanical_dofs.csv
WITH DELIMITER ',' CSV HEADER;


--For Chemical instance
COPY (SELECT value, stepnr, elementnr, name, gaussnr FROM  gausspunkte
     WHERE sid=$sid_Chemical) TO alldata_Chemical_Gausspunkte.csv
WITH DELIMITER ',' CSV HEADER;
COPY (SELECT value,stepnr, nodenr, dofnr FROM dofs WHERE sid=
    $sid_Chemical) TO alldata_Chemical_dofs.csv
WITH DELIMITER',' CSV HEADER;
```

Listing 5.7: Query for the fourth version of DataDeprovisioning plan



```
-- For Mechanical instance
COPY (SELECT value, stepnr, elementnr,
name, gaussnr FROM gausspunkte
WHERE sid=$sid_Mechanical)
TO alldata_Mechanical_Gausspunkte.csv
WITH DELIMITER ',' CSV HEADER;
COPY (SELECT value,stepnr, nodenr, dofnr
FROM dofs WHERE sid=$sid_Mechanical)
TO alldata_Mechanical_dofs.csv
WITH DELIMITER ',' CSV HEADER;
```

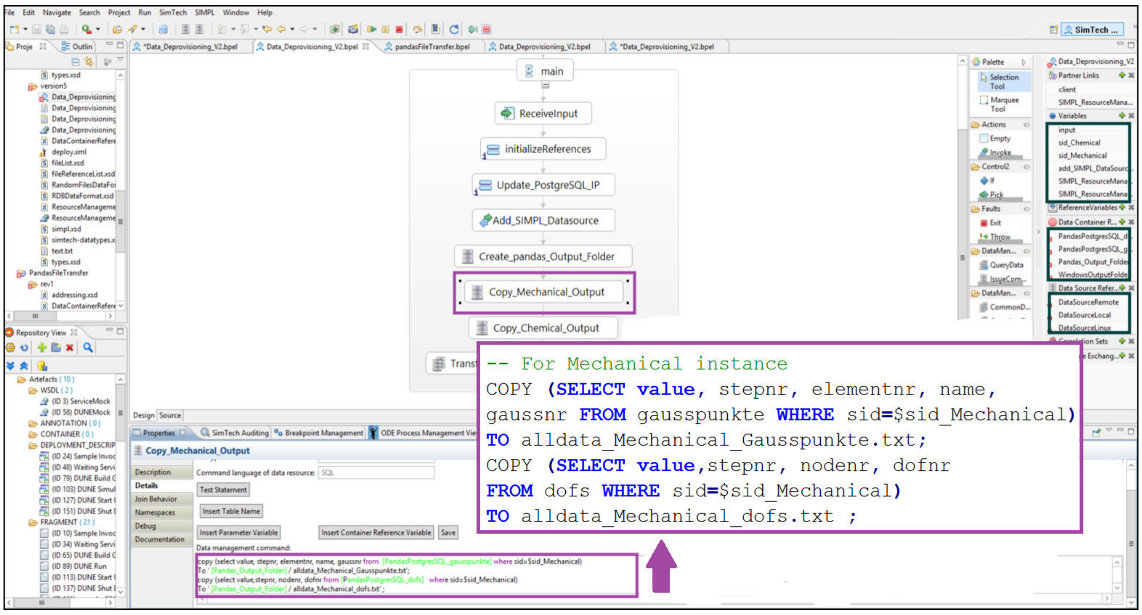
Figure 5.15: Implementation of DataDeprovisioning plan - Fourth version

In the fifth version of the *DataDeprovisioning* workflow as Figure 5.16 depicts, instead of using CSV files, some text files are used for storing the results. In other words, the calculation results for both PANDAS instances are retrieved from the PostgreSQL database, and then stored in some text files. Following that, similar to the previous versions, these files are transferred to the Windows-VM. Listing 5.8 illustrates the query for this version.

```
-- For Mechanical instance
COPY (SELECT value, stepnr, elementnr, name, gaussnr FROM gausspunkte
    WHERE sid=$sid_Mechanical) TO alldata_Mechanical_Gausspunkte.txt;
COPY (SELECT value,stepnr, nodenr, dofnr FROM dofs WHERE sid=
    $sid_Mechanical) TO alldata_Mechanical_dofs.txt ;

--For Chemical instance
COPY (SELECT value, stepnr, elementnr, name, gaussnr FROM gausspunkte
    WHERE sid=$sid_Chemical) TO alldata_Chemical_Gausspunkte.txt; COPY
    (SELECT value,stepnr, nodenr, dofnr FROM dofs WHERE sid=
    $sid_Chemical) TO alldata_Chemical_dofs.txt;
```

Listing 5.8: Query for the fourth version of DataDeprovisioning plan



```
-- For Mechanical instance
COPY (SELECT value, stepnr, elementnr, name,
gaussnr FROM gausspunkte WHERE sid=$sid_Mechanical)
TO alldata_Mechanical_Gausspunkte.txt;
COPY (SELECT value,stepnr, nodenr, dofnr
FROM dofs WHERE sid=$sid_Mechanical)
TO alldata_Mechanical_dofs.txt ;
```

Figure 5.16: Implementation of DataDeprovisioning plan - Fifth version

5.2.2.6 Plan number six: Termination plan

In the last plan, so-called *Termination* plan, all components inside the PANDAS service topology should be removed from cloud infrastructures, and terminated as well. This workflow is executed inside the OpenTOSCA container and similar to the *InfrastructureProvisioning* workflow, the IAs are called via the *OpenTOSCA Service Invoker* in an asynchronous way. This workflow consists of three primary tasks: (1) terminating the created VMs inside the PANDAS service topology, (2) undeploying the BPEL workflows from the ODE-PGF engine of the Windows machine and (3) removing the created data source from the PostgreSQL database of the SIMPL framework.

The termination of the VMs (only one VM in this work) can be done via the *TerminateVMbyServerid* operation of the *InstallOpenStackVM* IA in the PANDAS service topology. This part of the workflow was already implemented before and only used in this master's thesis. The actual implementation of this part can be found on GitHub[14]. In the next step, the *Undeploy* function of the *ODE-Service* IA which was developed in this work is used to undeploy some workflows from the ODE-PGF engine of the SIMPL framework. Integrating this step into the previous step was challenging and problematic in the implementation phase. In order to address the goal, a separate workflow which calls the *Undeploy* operation of the *ODE-Service* IA for three workflows [*DataProvisioning*, *Simulation/Application* and *DataDeprovisioning*] was designed and implemented. The third step in the *Termination* workflow is removing the created data source from the PostgreSQL database of the SIMPL framework. Like the previous step, integrating this step to the main *Termination* workflow had many problems. Therefore, this step was added to the second workflow, which was created for the second step [undeploying plans from the ODE-PGF engine]. In other words, the *DeleteDataSource* operation of the SIMPL resource management web service is invoked in the second workflow to delete the previously created entry from the PostgreSQL database. In conclusion, similar to the first

[14]https://github.com/CloudCycle2/Enterprise_CSAR

workflow, which was discussed in Section 5.2.2.1, the *Termination* workflow was divided into two workflows. The first workflow deletes the created VM from the PANDAS service topology, and the second workflow undeploys the plans and then removes the created data source from the PostgreSQL database of the SIMPL framework.

5.2.2.7 Challenges during implementation and deployment phases

Previously mentioned, the TOSCA Service Template consists of two main concepts : (1) application topology and (2) management plans. The Winery tool is a graphical tool which can be used to design and develop the topology of a TOSCA Service Template. But, for implementing the management plans inside the Service Template, there is not so far any designer like Eclipse BPEL Designer to design and implement the TOSCA-based management plans. Accordingly, all the steps for implementing these plans need to be done manually. In other words, for implementing the first and the last workflows which need to be executed inside the OpenTOSCA environment and the operations are invoked via the *OpenTOSCA Service Invoker*, the author faced lots of challenges at the implementation phase. In other words, using only Notepad++ for writing these workflows are problematic. As it will be discussed later in Chapter 7 Section 7.2.3, one of the future work of this master's thesis can be conducted by the extension of the Winery tool with a graphical editor for developing the TOSCA-based management plans.

5.2.3 The overall PANDAS Service Template

As mentioned already, a Service Template consists of an application topology and some management plans. Similarly, the PANDAS Service Template consists of an application topology, which was discussed in Section 5.2.1, and several management plans, which were discussed in Section 5.2.2. Previously mentioned, the overall and complex TOSCA management plan for the PANDAS bone simulation software is divided into six different workflows as this kind of modularization is useful for the application. Therefore, the PANDAS Service Template has an application topology and six different types of management plans. To simulate theses management plans as an automated overall plan, a SoapUI test case[15] can be used. This test case can be part of a test unit and consists of several test steps. This kind of testing, so-called functional testing, tests the functional behaviour of the management plans which are executed one after each other. For validating the responses of this test case, several assertions[16] can be added to the test case. There are different types of assertions in SoapUI, such as SOAP Fault, XPath[17] Match, XQuery[18] Match, Not SOAP Fault, etc. If all assertions are successful, the test case responses are valid. In the PANDAS Service Template, the test case consists of an appropriate order for all management plans. In other words, the test case starts by executing the first workflow, which is *InfrastructureProvisioning* plan. Then, the execution continues to the second plan, which is *PandasSoftwareProvisioning* plan. Following that, the third workflow, which is *DataProvisioning* plan is executed. The next workflow in this test case is *Simulation/Application* workflow. Then, the *DataDeprovisioning* plan is executed. Lastly, the test case can be completed by executing the *Termination* plan. It is important to note that, all the input and output parameters, which are required for each workflow, have to be

[15]http://www.soapui.org/getting-started/functional-testing.html
[16]http://www.soapui.org/functional-testing/validating-messages/getting-started-with-assertions.html
[17]http://www.w3schools.com/xpath/
[18]http://www.w3schools.com/xquery/

defined in the test case. As a result, the overall functionality of the PANDAS Service Template can be tested in this way. As it will be discussed in Section 7.2.4, running choreographies[3] for the PANDAS Service Template is a better approach and can be conducted as a future work.

Chapter 6

Discussion

This chapter discusses the significant arguments which were set up in Chapter `Introduction`. Section 6.1 restates the general goals and motivations of this master's thesis which is followed by a statement about whether or not, and how much, our findings address these goals. In Section 6.2, the lessons which were learned in this master's thesis with respect to the approaches and contributions were discussed. Section 6.3 discusses whether and how the approach which is used in this work for the PANDAS bone simulation software can be generalized to other simulation examples, other simulation software and for developing other variants of a TOSCA Service Template. Furthermore, this chapter discusses briefly about using other cloud providers for the PANDAS Service Template instead of OpenStack, which is used in this master's thesis.

6.1 Evaluation of the findings regarding goals and motivations

The following section evaluates the approaches presented in this master's thesis with regard to the goals and motivations specified in the `Introduction` chapter. The general goals of this thesis are: (1) *moving traditional and on-premise simulation applications, namely PANDAS, into cloud infrastructures*, (2) *automating the deployment and management of applications in cloud environments*, (3) *dealing with huge and heterogeneous data sources in simulation applications* and (4) *integration of the OASIS TOSCA standard with the SIMPL framework.*

6.1.1 Mapping traditional applications into cloud infrastructures

Moving traditional and on-premise applications into cloud infrastructures is turning to one of the main challenges in IT organizations over the last decades. Virtual resources instead of physical resources and pay per use capability for resources in the cloud are the two most important and biggest advantages of cloud-based applications. Having virtual resources for storage, processing, etc., removes the cost which is required for providing technology infrastructures in traditional solutions. Most of complex applications, especially simulation applications, depend on a large number of software and servers. Moving applications into cloud infrastructures eliminates the cost for data storage, software updates, management, etc., by providing virtual resources instead of physical resources for users. Many solutions have been discovered in the last few years for mapping traditional applications into cloud environments. The main goal of this master's thesis is turning the PANDAS bone simulation software into cloud infrastructures in order to benefit from the advantages of cloud-based applications. For this purpose, the TOSCA standard and the SIMPL framework are used in this thesis. As the result, different variants of a TOSCA Service Templates which realize the PANDAS bone simulation in cloud environments as a SaaS solution are designed and implemented.

In this master's thesis, the PANDAS bone simulation software has been moved to cloud infrastructures with the help of TOSCA and SIMPL. A new VM is created on the OpenStack cloud provider. Then, the PANDAS source code and its related PostgreSQL database can be installed on the created VM. Following that, the required data sources are provisioned for the PANDAS bone simulation software and the actual simulation for two PANDAS instances (mechanical and chemical) are executed on that machine. Then, the results of the calculations are deprovisioned to another machine for later usage. Finally, the created VM is removed from the OpenStack cloud provider. As the result, the goal of mapping the traditional PANDAS bone simulation into cloud infrastructures has been reached.

6.1.2 Automating the deployment and management of cloud-based applications

Composite cloud-based applications consist of various and distributed components. The manual deployment and management of these components is an expensive and tedious process. By automating the deployment and management of these components, the provisioning of software instances for new customers gets cheaper and faster in cloud infrastructures. Most of cloud computing advantages, such as elasticity, pay per use, etc., are significantly dependant on the degree of automation in deployment and management of cloud-based applications. Various approaches have been discovered in the state of the art for automating the deployment and management of composite applications in cloud infrastructures.

In this master's thesis, the OASIS TOSCA standard is used for automating the deployment and management of the PANDAS bone simulation software. Previously discussed in Section 2.2, a TOSCA Service Template consists of an application topology and management plans. TOSCA management plans fulfill the automation goal. The management plans are described via workflow-based languages, such as BPEL or BPMN, which describe all the detailed steps for executing and managing application components. These workflows can be started and executed automatically by receiving an external message. As the result, all the required steps for deploying and managing different components of a composite cloud-based application are fully automated without any human interventions. As the result of this master thesis, different variants of a PANDAS Service Template, which consists of one service topology and six different types of management plans, were designed and implemented. As mentioned above, with the help of these management plans, the deployment and management of the PANDAS Service Template is automated in the cloud. In other words, these management plans can be executed automatically one after each other. As discussed in Section 5.2.3, in the current implementation of the PANDAS Service Template, a SoapUI test case which defines an appropriate order for the management plans and then executes them one after each other is used to automate all the steps of moving PANDAS into cloud infrastructures. As it is discussed in Section 7.2.4, running choreographies[3] for the PANDAS Service Template is a better approach and can be conducted as a future work. With choreographies, running all the management plans inside the Service Template are automated and each workflow calls its subsequent workflow in an appropriate order. Therefore, there is no need to define a SoapUI test case for automating the overall workflow of the PANDAS Service Template.

6.1.3 Data management and data provisioning of simulation applications in the cloud

A large number of applications, particularly simulation applications, deal with huge, heterogeneous and distributed data sources these days regarding to the world of information explosion. Finding an appropriate format for data sources and dealing with all the required data transformations lead to a large number of complexities for data provisioning and data management in the cloud for simulation applications. Having a generic access to many kinds of data sources, frees scientists and engineers to deal with the low level details of data sources.

The required heterogeneous data sources for a PANDAS bone simulation software are defined in Figure 2.7. Consequently, a generic approach for data management and data provisioning of the PANDAS bone simulation in cloud infrastructures is demanded. The SIMPL framework is used in this master's thesis in order to address this goal. The SIMPL framework fulfills this goal by extending the BPEL workflow language with some DM activities and DM patterns. Data provisioning and data deprovisioning workflows for the PANDAS bone simulation software were designed and implemented with the SIMPL framework in this master's thesis. As a result, users of PANDAS do not need to deal with the low level information of each data source. Data provisioning and data deprovisioning of the PANDAS bone simulation in the cloud benefits from an efficient approach to access data by using the SIMPL prototype. The two implemented versions of the *DataProvisioning* plan, which were discussed in Section 5.2.2.4, do not provision all different types of data sources for the PANDAS bone simulation software, as Figure 2.7 shows. In other words, all different types of data sources in Figure 2.7, which are text-based, XML-based, CSV-based and SQL-based, are not provisioned completely in these workflows. These workflows provision some kinds of text-based and XML-based data sources for the PANDAS bone simulation software. As it is discussed in Section 7.2.2, other variants for *DataProvisioning* workflow, which provision all types of data sources which are required by PANDAS, can be designed and implemented as a future work.

6.1.4 Integration of TOSCA with SIMPL

The OASIS TOSCA standard and the SIMPL framework are both based on workflow languages, such as BPEL. By integrating these two standards, cloud-based applications can benefit from an integrated support. To put it another way, a generic approach for data provisioning and data management of cloud-native applications with the SIMPL framework becomes an integral part of the TOSCA standard definition. This goal is fulfilled by the *ODE-Service* IA which was designed and implemented in this work. The *ODE-Service* IA is an integral part of the TOSCA service topology in order to combine the TOSCA management plans with the SIMPL workflows. The management plans of the TOSCA Service Template, realizing the PANDAS bone simulation software in the cloud, are deployed on the appropriate workflow engine for running and execution with the *ODE-Service* IA. Some workflows inside the PANDAS Service Template require an extended workflow engine for execution. Therefore, this *ODE-Service* IA uses the Apache ODE deployment API in order to deploy and undeploy the plans to and from an extended workflow engine remotely. As the result, the TOSCA Service Template can consist of TOSCA-based as well as SIMPL-based (workflows which consist of some DM activities or DM patterns) workflows. Based on the activities inside the workflows, the appropriate engine is selected. This approach provides a generic solution of using SIMPL advantages inside the OpenTOSCA run-time environment.

The goal of integrating the TOSCA standard and the SIMPL framework has been reached via this *ODE-Service* IA. But, this approach does not deploy and undeploy workflows automatically. In other words, as discussed in Section 4.3.5, one problem in this approach is the lack of automation. The deployment and undeployment processes of the workflows inside the PANDAS Service Template are not automated without any human interventions. Developers need to determine the workflows which require an extended ODE engine for their executions in advance. Then, the `Deploy` operation of this *ODE-Service* IA is called for these workflows inside the first plan. In a similar way, the `Undeploy` operation of this IA is invoked in the last plan of the PANDAS Service Template for the workflows which need to be removed from the ODE-PGF engine. As it is discussed in Section 7.2.1, an optimized approach for this IA which deploys and undeploys workflows automatically can be designed and implemented in future.

6.2 Lessons learned with respect to the approaches and contributions

In this master's thesis, the traditional and on-premise PANDAS bone simulation software is moved into cloud infrastructures. The OASIS TOSCA standard is used in this work in order to automate the deployment and management of the PANDAS bone simulation software in the cloud. As PANDAS requires distributed and heterogeneous data sources, the SIMPL framework is used for data provisioning and data management in this work. In this thesis, an efficient approach for integrating SIMPL with TOSCA was developed and implemented. Moreover, different variants of a TOSCA Service Template realizing the PANDAS bone simulation in the cloud were elaborated, developed and implemented. This Service Template provides the corresponding service topologies, as well as management plans which turn PANDAS into a fully integrated SaaS solution. This section discusses the author's lessons learned in this thesis with respect to the approaches, contributions and implementations. Some of these lessons are listed as follows:

- Mapping the PANDAS bone simulation software into cloud infrastructures in order to benefit from the advantages of cloud-based applications is a beneficial decision.
- Using the OASIS TOSCA standard for automating the deployment and management of the PANDAS bone simulation software is an efficient approach. Therefore, the assumptions about TOSCA, which the author made in Section 3.2 are correct. In other words, the TOSCA standard is an easy and straightforward approach. Working with the Winery tool for developing the service topology as well as using the BPEL workflow language for implementing the management plans provide simplicity for developers. TOSCA-based applications are portable from one cloud provider to another cloud provider. Therefore, developers are not restricted in choosing a cloud provider.
- On the other hand, the Winery tool does not support graphical editors, such as Eclipse BPEL Designer, for developing the management plans inside TOSCA Service Templates. Consequently, it is problematic to develop the management plans without help of editors. As it is discussed in Section 7.2.3, extending the Winery tool to have an editor for developing TOSCA management plans can be conducted as a future work.
- Using the SIMPL framework as a generic approach for data management and data pro-

visioning of the PANDAS bone simulation software in cloud infrastructures is a good decision. As a consequence, the assumptions in Section 3.3 about SIMPL with respect to other approaches for data provisioning and data management are correct. In reality, the SIMPL framework frees developers to deal with the low-level details of data sources. Therefore, the author believes that using the SIMPL framework for extending the presented approach in this thesis, as it is discussed in Section 6.3, is a good choice. In other words, SIMPL is a good choice in future for developing other variants of *DataProvisioning* and *DataDeprovisioning* workflows inside the PANDAS Service Template. Therefore, as it is discussed in Section 7.2.2, future work for developing other variants of a TOSCA Service Template for the PANDAS bone simulation software can be conducted with the help of SIMPL as a data provisioning and data management tool.

- It is a little bit challenging in the beginning of working with the SIMPL framework. At first, developers may think that SIMPL provides many complexities instead of simplifying the data management and data provisioning processes. But, after working a little bit with this framework, developers can make sure that SIMPL is a good choice for data management and data provisioning processes.
- The *ODE-Service* IA is evaluated as a good approach for integrating TOSCA and SIMPL from the author's point of view. As discussed in Section 4.3.5, this approach has some advantages and limitations. The author believes that optimizing this approach in future provides the generic SIMPL-based approach for data provisioning and data management of cloud-native applications as an integral part of the TOSCA standard definition. Therefore, as it is discussed in Section 7.2.1, optimizing this approach can be conducted as a future work.

6.3 Generalization of the presented approach

This section investigates generalization capabilities of the approach proposed in this master's thesis from four different aspects: (1) generalization capabilities for other simulation examples, (2) generalization capabilities for other simulation software instead of PANDAS, (3) generalization capabilities for developing other variants of a TOSCA Service Templates and (4) generalization capabilities of the proposed approach with respect to other cloud providers instead of OpenStack. As mentioned already, the overall TOSCA management plan is divided into six different workflows which realize the PANDAS bone simulation software in cloud infrastructures. Accordingly, the subsequent sections evaluate generalization capabilities for each workflow from the above-mentioned points of view. In Subsection 6.3.1, generalization capabilities of the presented approach for other simulation examples are discussed. Subsection 6.3.2 investigates generalization capabilities of the proposed approach for other simulation software instead of PANDAS. The discussion in Subsection 6.3.3 is centered around the development of other variants of a TOSCA Service Template for PANDAS bone simulation software. Finally, Subsection 6.3.4 discusses briefly about generalizing the approach proposed in this master's thesis with respect to other cloud providers instead of OpenStack.

6.3.1 Generalization of the presented approach for other simulation examples

This section assesses the generalization capability of each workflow inside the PANDAS Service Template for other simulation examples, which need to be calculated by PANDAS (e.g., a different bone to be simulated or different boundary conditions). To put it another way, the question is what happens when the PANDAS bone simulation uses different simulation problems for its calculation from the one which was used in this master's thesis. Basically, the PANDAS software can calculate other simulations than the bone simulation as well. For example, this can also concern other tissues such as cancer. PANDAS was originally developed to simulate some problems from civil engineering or structural mechanics. PANDAS is based on the, so-called, Theory of Porous Media[9] which may be used for both civil engineering and tissue simulations. From a conceptual point of view, in case of other simulation examples as mentioned above, it is required to provide different input data (as these input data describe the problem to be simulated). Furthermore, it is necessary sometimes to change the service or plan that is used for the calculation (in case the numerical scheme needs to be adjusted). Besides, some more sophisticated simulation examples might also require different topologies for the PANDAS environment (for example even more PANDAS instances that are involved in one distributed calculation). It is important to note that all the following discussions neglect these kinds of simulation examples. Some workflows are generic enough and do not need any changes in order to be adopted to support other simulation examples, and some are not on the other hand. The subsequent sections evaluate generalization capabilities of six different workflows, which were discussed in Section 4.2, in detail.

6.3.1.1 InfrastructureProvisioning plan

This workflow installs the required VMs (only one VM in this work) on the OpenStack cloud provider, in order to realize the PANDAS bone simulation software in cloud infrastructures as a SaaS solution. The next step in this workflow, deploys some other workflows on the ODE-PGF engine of the Windows-VM. Then, the new IP address of the created VM is added to the resource management database of the SIMPL framework. This workflow is executed on the OpenTOSCA environment. If PANDAS requires another simulation example for its calculation, this workflow can be adopted easily with the new simulation example without any modifications. In other words, the main steps in this workflow and the artifacts the workflow provisions with these steps are necessary for all other simulation examples, which are simulated by PANDAS. As the result, this workflow can work perfectly with other simulation examples. Previously mentioned, developing TOSCA-based workflows are challenging as there does not exist any graphical tool like Eclipse BPEL Designer to develop and implement these workflows. Accordingly, all the steps inside the TOSCA-based workflows need to be implemented manually without the help of graphical editors, and the debugging process is really problematic. Therefore, basically it is good that this workflow is generic enough for other simulation examples. Because it was implemented in this master's thesis and no one else needs to bother with all these issues mentioned above.

6.3.1.2 PandasSoftwareProvisioning plan

PandasSoftwareProvisioning workflow prepares a PANDAS instance. In other words, the workflow creates a unique ID and a basic directory structure for the PANDAS software instances, and then unpacks the archive of the PANDAS source code to the basic directory structure. To the greatest extent, this workflow is independent of simulation examples. It only deals with the number of instances which are required for a simulation. To clarify, in this master's thesis, two PANDAS instances are required, one for the the mechanical calculation and one for the chemical. For this purpose, this workflow is executed two times inside the PANDAS Service Template to realize two PANDAS instances in cloud environments. In other simulation examples, this workflow can be executed less or more times based on the defined simulation problem. Basically, the main steps within this workflow do not need any changes to be adopted to other simulation examples, and can work perfectly with other simulation examples as well.

6.3.1.3 DataProvisioning plan

This workflow provides and prepares heterogeneous input data of the simulation in order that PANDAS can properly ingest these data. As mentioned already, this workflow consists of three phases. In the first phase, some sub-folders are created in the root folder of the relevant PANDAS software instance. As all other simulation examples require some folders and directories for storing all input data sources, this phase can be adopted easily by any simulation examples without any changes. The second phase, provides the necessary data which describe the simulation example. This phase is highly dependant on a simulation example, since a simulation example or a simulation problem is mostly described by these data which need to be provisioned. In case these input data are different because of a different simulation problem, this phase of the workflow requires the most changes in order to be accepted by a specific simulation example. As shown in Figure 2.7, PANDAS requires heterogeneous data sources. These data sources can be different in various simulation examples. For instance, if PANDAS simulates other simulation examples than bone simulations, the text-based files in Figure 2.7 which define bone shapes are not required anymore. The other example could be the second variant of *DataProvisioning* workflow, which were discussed in Section 4.2.3.2. This variant defines another simulation example which requires different CSV files from the first variant. Therefore, these CSV files are different and this workflow needs to be adopted to provision the new CSV files. These are only some simple examples, but there might be even more sophisticated examples which require different data sources. The third phase of this workflow configures PANDAS (e.g. some numerical configurations) in order that it can properly calculate the simulation outcome. Following that, the actual compilation of the PANDAS source code with some of the input data, which is transferred in the second phase of this workflow, is the final step of the third phase in this plan. Similar to the second phase, the first part of this phase, which configures some numerical configurations, needs to be adopted for the new simulation example. In other words, different simulation examples, which simulate structure changes within a human bone, may require different configuration parameters for their calculation. Subsequently, the files or directories which determine these configuration parameters may differ from one simulation example to another simulation example. But on the other hand, the second part of the third phase, which compiles the PANDAS source code with some of the input data, does not need to be adopted to other simulation examples.

Aforementioned, in this master's thesis, the SIMPL framework was used for developing all variants of *DataProvisioning* workflow. The SIMPL framework defines some DM activities, which were discussed in Section 2.5.1, and some DM patterns. In other words, these activities and the pattern-based approach in SIMPL ease workflow development even more, in particular for scientists. The GUI of the SIMPL framework, with extended Eclipse BPEL Designer, is straightforward and easy to use. The users may have some challenges at the beginning of using this framework. But, everything becomes straightforward after working a little bit. Consequently, developing or adopting different variant of *DataProvisioning* workflow with the new simulation examples would not require lots of effort.

6.3.1.4 Simulation/Application plan

This workflow realizes the actual simulation calculation for PANDAS bone simulation instances. First, PANDAS instances (in this work, only two instances) are started at the same time. After starting the PANDAS instances, this workflow invokes the *Execute_Command* operation of the PANDAS web service in order to send some commands to both instances at once. Conceivably, these commands may not be the same for different simulation examples and need to be adopted for the new simulation example. Following that, the actual calculation for instances is started. It is not possible to make sure that this step, which performs the calculation, is generic enough. In other words, there might be different kinds of calculation schemes that might need different web service operations implementing these schemes. The last step of the *Simulation/Application* workflow stops the instances simultaneously. This step is generic enough, and do not need any changes for the new simulation examples in case these examples again rely on exactly two PANDAS software instances. On the assumption that the new simulation example is based on more or less instances for its calculation, the PANDAS web service implementation as well as the operations of this workflow need to be adopted with the new one. To put it another way, the current implementation of the PANDAS web service works only for two instances. If the new simulation example requires more instances for its calculation, the implementation of the PANDAS web service needs to be adopted with the new requirements.

6.3.1.5 DataDeprovisioning plan

This workflow returns the result data of one or more PANDAS instances back to users. In this master's thesis, different variants of the *DataDeprovisioning* workflow were designed and implemented with the help of the SIMPL framework. In the first variant of this workflow for instance, results are transferred from one folder on the Linux-VM to an appropriate folder on the Windows-VM. If other simulation examples require all the generated data to be transferred to the Windows-VM, this variant of the workflow works fine with other simulation examples as well. The other variants, query data from the PostgreSQL database and then transfer them to the Windows-VM. Based on the requirements which exist for new simulation examples, these workflows may need some changes. To clarify, the query which is used for fetching data from the PostgreSQL database of PANDAS, may vary for the new simulation examples. In other words, the data structure of the corresponding database tables can be different in other simulation examples. For example, the tables or their related columns may differ in other simulation examples. It might be required to join different tables first, and then retrieve the appropriate data. As a result, the queries which were used in the implementation phase of different variants of this workflow (see Section 5.2.2.5) may need to be adopted to the new simulation examples. Similar to the *DataProvisioning* plan, working with SIMPL is convenient and straightforward. Besides, DM activities and DM patterns inside the SIMPL framework ease the workflow development even more. Therefore, adopting the *DataDeprovisioning* workflow with the new simulation examples would not be challenging, and most of the implementations of the required workflow can be derived from the old versions.

6.3.1.6 Termination plan

The last plan terminates the VMs, which are created in the first workflow. In this master's thesis, only one VM is terminated in this plan. This workflow however contains a BPEL loop activity which terminates all the VMs existing inside the TOSCA service topology one after another. If the new simulation example instantiated more than one VM, this workflow would be generic enough to be adopted with the new simulation example without any modifications. In other words, the BPEL loop activity in this workflow reads the appropriate data from the TOSCA service topology in order to determine how many VMs were instantiated in the first plan. For this purpose, the number of VMs, which need to be terminated, is not important and this workflow can work perfectly for any number of VMs in the application topology. Moreover, terminating one VM is a generic operation that works for all kinds of VMs in the same way. In other words, this termination step works perfectly with Linux-VMs or Window-VMs for example.

6.3.1.7 Generalization degree as a graph

Figure 6.1 depicts the degree of generalization capabilities of the proposed approach in this master's thesis for other simulation examples with respect to each management plan inside the PANDAS Service Template. As mentioned in Section 6.3.1, the PANDAS software can calculate other simulations as well than the bone simulations. But, the discussion in Section 6.3.1 eliminated all these sophisticated simulation examples. Previously mentioned, If PANDAS requires another simulation example for its calculation, the *InfrastructureProvisioning* workflow can be adopted easily with the new simulation example without any modifications. Similarly, the *Termination* plan is generic enough to work with other simulation examples. In general, as mentioned already, TOSCA-based workflows are more generic than other types of workflows. Therefore, as shown in Figure 6.1, these two plans have the highest number, which shows the highest degree of the generalization, than other workflows. On the other hand, *DataProvisioning* and *DataDeprovisioning* workflows are specific to simulation examples as discussed in Sections 6.3.1.3 and 6.3.1.5, respectively. As a result, they have the lowest number in Figure 6.1. The main steps within the *PandasSoftwareProvisioning* workflow do not need any changes to be adopted to other simulation examples, and can work perfectly with other simulation examples. This workflow can be executed less or more times based on the defined simulation problem. Therefore, this workflow is more generic than *DataProvisioning* and *DataDeprovisioning* workflows. Finally, as discussed in Section 6.3.1.4, the *Simulation/Application* workflow needs some changes to be adopted to other simulation examples in some cases. Accordingly, as the figure shows, this workflow is less generic than *InfrastructureProvisioning*, *PandasSoftwareProvisioning* and *Termination* workflows. But on the other hand, it is more generic than *DataProvisioning* and *DataDeprovisioning* workflows. It is important to mention that these numbers are based on the author's assumption. Therefore, it is possible that one considers a different assumption with respect to a different generalization degree for each workflow.

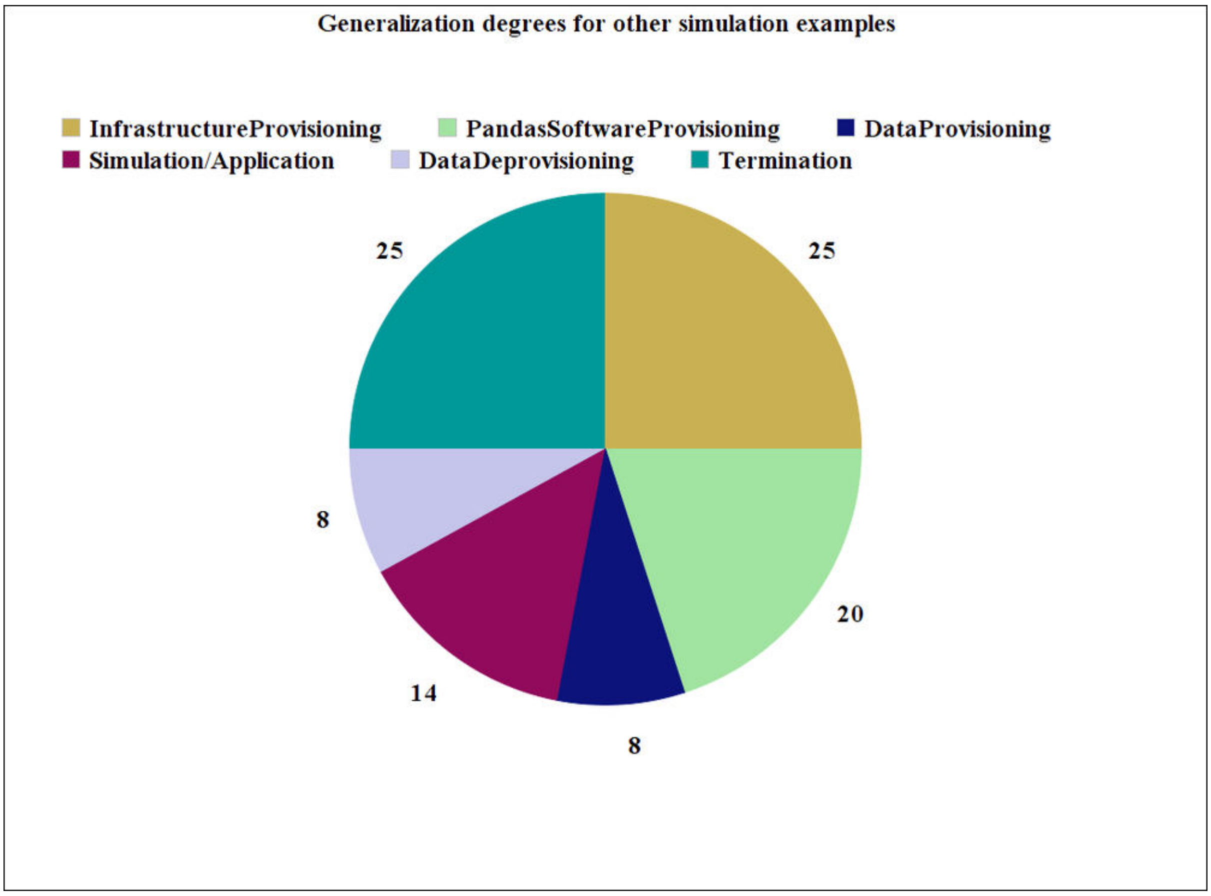


Figure 6.1: Generalization degree for other simulation examples

6.3.2 Generalization of the presented approach for other simulation software

This section investigates generalization capabilities of the proposed approach for other simulation software instead of PANDAS. In this master's thesis, the TOSCA standard and the SIMPL framework were used to move the PANDAS bone simulation software into cloud infrastructures. This section discusses generalization capabilities of using these two approaches for other simulation software. Moreover, it evaluates to what extent the TOSCA management plans for the PANDAS bone simulation software are generic enough to incorporate other simulation software, such as GNU Octave. To realize this assumption, the installation process, required data sources and other features of the GNU Octave software are studied, as a case study, to get an overview about general and common processes of working with different simulation software in cloud environments. Like in the previous sections, this evaluation considers all workflows inside the TOSCA Service Template separately.

The TOSCA standard improves the portability and automated deployment and management of cloud-based applications. Most of simulation software are composite applications, and consist of distributed and heterogeneous components in their architectures. Accordingly, automating the deployment, termination and management of these components in the topology improves the overall performance of these software. The OASIS TOSCA standard is a generic approach which can be used to move many simulation software into cloud infrastructures. The TOSCA standard defines an application topology and then manage the components inside this topology by means of management plans. These two concepts of TOSCA are generic enough to be used for most of applications. Here, we assume that the application topology inside the TOSCA Service Template remains the same in other simulation software. In other words, we assume that the only thing which is required to be changed is the software itself, but the general number of software instances and the general structure of the topology remain the same. Therefore, generalization capabilities of the TOSCA management plans for other simulation software are discussed as follows.

6.3.2.1 InfrastructureProvisioning plan

In the first workflow of almost all simulation software, the main infrastructures which are required for the software should be provided in cloud environments. Provisioning a set of VMs and customizing some configurations on the VMs are some common steps in the first workflow for many simulation software. Generating a new VM, which is the first activity in this workflow, is compatible to other simulation software if only one VM is required to be provisioned. In case other simulation software need additional ones, this step of the workflow need to be adjusted accordingly. In the second activity of this workflow, some shell commands are sent to the new created VM in order to configure it properly. This might be quite different and more complex for other simulation software. Accordingly, this step has to be adopted to the specific requirements in other simulation software. The *InfrastructureProvisioning* workflow may also embed some other steps, which are unique for each simulation software. For instance, the PANDAS bone simulation software works with the SIMPL framework in this master's thesis for data provisioning and data deprovisioning steps. Consequently, the last two step in this workflow are registering the new IP address of the created VM on the PostgreSQL database of SIMPL and deploying some workflows to an ODE-PGF engine. These steps might be quite different and more complex for other simulation software.

6.3.2.2 PandasSoftwareProvisioning plan

All simulation software require software instances on cloud infrastructures to run their simulation calculations. To clarify, in this work, the PANDAS bone simulation software needs two software instances for running their mechanical as well as chemical calculations on the cloud. As a consequence, the *PandasSoftwareProvisioning* workflow is called two times inside the PANDAS Service Template. Other simulation software may require different number of software instances for their calculations. Accordingly, the number of execution of this workflow has to be adopted in different simulation software. It is compatible for most of simulation software to prepare a platform (e.g. some basic directory structures for each instance) in the first step of this workflow. Then, the source code of the simulation software is unpacked on the corresponding directory. In other words, preparing a platform for each software instance and then copying the software source code on the appropriate directory of the instance are generic steps in most of simulation software. In this master's thesis, the current implementation of the PANDAS web service includes a connection between PANDAS and the PostgreSQL database. Therefore, there is no need in this workflow to set up the connection between PANDAS and PostgreSQL database. However, other simulation software might not be tightly coupled with their databases. In this case, it is also required to make a connection between a simulation software and its related database system in this workflow.

6.3.2.3 DataProvisioning plan

Different simulation software require various data sources for their calculations. As simulation software are based on a large number of criteria in order to simulate the real phenomenons, their required data sources are distributed and heterogeneous. Accordingly, the *DataProvisioning* workflow varies from one simulation software to other simulation software in general. As mentioned already in Section 6.3.1.3, this workflow consists of three phases. In the first phase, some sub-folders are created in the root folder of the relevant PANDAS software instance. As all other simulation software require some folders and directories for storing all input data sources, this phase can be adopted easily to any simulation software. The second phase provides the necessary data which describe the simulation example which is simulated by the software. This phase is highly dependant on the simulation software and their requirements, since a simulation example or a simulation problem is mostly described by these data to be provisioned. In this case, these input data are different because of a different simulation problem in a different simulation software. Therefore, this phase of the workflow requires the most changes in order to be accepted by a specific simulation software. As shown in Figure 2.7, PANDAS requires heterogeneous data sources. These data sources can be different in various simulation software. For instance, other simulation software might not use any CSV-based data sources or SQL-based files. The GNU Octave simulation software for instance, may accept only text-based files as input data sources. These are only some simple examples, but there might be even more sophisticated examples which require different data sources. The third phase of this workflow configures PANDAS (e.g. some numerical configurations) in order that it can properly calculate the simulation outcome. Following that, the actual compilation of the PANDAS source code with some of the input data, which is transferred in the second phase of this workflow, is the final step of the third phase in this plan. These two steps may not be part of the new simulation software. In other words, the new simulation software might not need any numerical configurations or use another mechanism for compiling the source code. Previously mentioned, the SIMPL framework is used for data provisioning and data management in this master's thesis. This framework has a user friendly GUI and it would be convenient for developers to modify the workflows in order to be adoptable to other simulation software. Furthermore, the SIMPL framework has extensibility features, which enable a developer to define new types of data sources in the SIMPL framework. As a consequence, if the simulation software require some new types for their data sources, it is possible to customize the SIMPL framework in order to understand the new data sources. Furthermore, the SIMPL framework defines some DM activities, which were discussed in Section 2.5.1, and some DM patterns. To put it another way, these activities and the pattern-based approach in SIMPL ease workflow development even more, in particular for scientists. To conclude, with the help of SIMPL, developers can use the existing variants of *DataProvisioning* workflow in order to derive a new workflow for the new simulation software.

6.3.2.4 Simulation/Application plan

This workflow realizes the actual simulation calculation for PANDAS bone simulation instances. The discussion here can be centered around three main steps in this workflow: (1) starting software instances, (2) performing the actual calculation and (3) stopping software instances. On the one hand, other simulation software will rely on other web services. Accordingly, this workflow needs some changes to work with the new web services. On the other hand, different web services may require a different kind of workflow or orchestration of the web service operations. In other words, the general structure of starting a software instance, performing the calculation, and stopping the instance is a good choice for legacy applications. But, it is good only for legacy applications. In other words, a real workflow might need more sophisticated workflow steps within these three phases of the workflow.

6.3.2.5 DataDeprovisioning plan

This workflow returns the result data of one or more PANDAS instances back to users. In this master's thesis, different variants of the *DataDeprovisioning* workflow were designed and implemented with the help of the SIMPL framework. In the first variant of this workflow for instance, results are transferred from one folder on the Linux-VM to an appropriate folder on the Windows-VM. If other simulation software require all the generated data to be transferred to the Windows-VM, this variant of the workflow works fine with other simulation software as well. The other variants, query data from the PostgreSQL database, and then transfer them to the Windows-VM. Based on the requirements which exist for the new simulation software, this workflow may need some changes. To clarify, the query which is used for fetching data from the PostgreSQL database of PANDAS, may vary for the new simulation software. In general, the new simulation software may not use the PostgreSQL database as a database system. Other database systems, such as SQL server, may be used in other simulation software. In other words, the data structure of the corresponding database tables can be different in other simulation software as well. For example, the overall schema, tables or their related columns maybe different in other simulation software. It might be required to join different tables first, and then retrieve the appropriate data. As a result, the queries which were used in the implementation phase of different variants of this workflow (see Section 5.2.2.5) may need to be adopted to the new simulation software. Similar to the *DataProvisioning* plan, working with SIMPL is convenient and straightforward. Besides, the DM activities and the DM pattern ease workflow development even more. Therefore, adopting the *DataDeprovisioning* workflow with the new simulation software would not be challenging.

6.3.2.6 Termination plan

Most of simulation software require some kinds of termination or garbage collection plans to terminate or remove cloud infrastructures at the end of simulation. In the case that other simulation software require to terminate all the VMs, which were created in the beginning of the deployment phase, this workflow can be adopted to the new simulation software as well. The *Termination* plan for most of simulation software terminates the VMs and undeploys the other components from the topology of the simulation software. But there might also be cases, where one does not want to undeploy the whole VMs from the topology. Instead one might only want to delete directories on the machine, that hold some of the software instances, but not the actual VM. Because, these VMs might be used later. This is may be a very rare case, but this *Termination* plan cannot be used in this case. For each VM in the topology, some directory structures need to be removed from the VMs. In BPEL workflow language, a loop activity can be used to loop over all the VMs in the topology and then delete directories on each machine. Consequently, other simulation software maybe require other workflow steps. Some other tasks which are specific only to some simulation software can be done in this workflow. To clarify, in this master's thesis, the entry which was created inside the PostgreSQL database of the SIMPL framework in the first plan, is removed in the last step of this workflow. In other simulation software different tasks maybe required in respect to the topology and requirements. In conclusion, based on the different cases, which were discussed above, other simulation software might require different steps in their *Termination* workflows.

6.3.2.7 Generalization degree as a graph

As discussed in Section 6.3.2, generalization capabilities of the proposed approach in this master's thesis for other simulation software instead of PANDAS depend on different cases that can be considered in each management plan. In other words, it is a little bit difficult to show generalization capabilities of all these cases as a graph. Therefore, as shown in Figure 6.2, the *PandasSoftwareProvisioning* and *Termination* workflows have the highest degree with respect to the several reasons which were mentioned in Sections 6.3.2.2 and 6.3.2.6, respectively. As mentioned in Sections 6.3.2.3 and 6.3.2.5, the *DataProvisioning* and *DataDeprovisioning* workflows have the second highest degree as shown in Figure 6.2. These workflows are highly dependant on the simulation software, but working with the SIMPL framework is convenient and straightforward. Therefore, it eases the development process in general. Lastly, as discussed in Sections 6.3.2.1 and 6.3.2.4, the *InfrastructureProvisioning* and *Simulation/Application* workflows are the least generic workflows with respect to other simulation software. It is important to note that these numbers and scales are based on the author's assumption. Therefore, it is possible that one considers a different assumption with respect to a different generalization degree for each workflow.

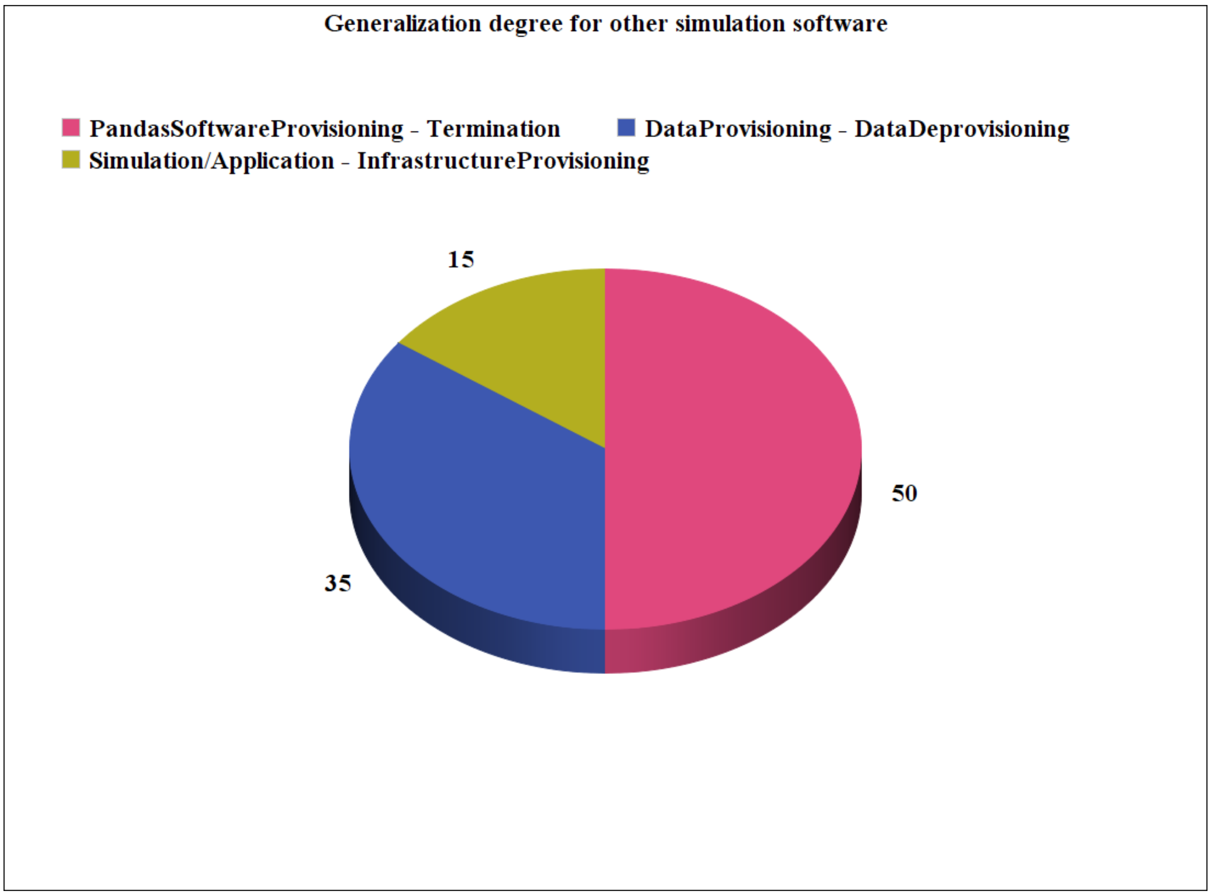


Figure 6.2: Generalization degree for other simulation software

6.3.2.8 The SIMPL framework in other simulation software

Most of simulation applications are based on huge and heterogeneous data sources for their simulation processes. Finding a standard format for data sources and provisioning all distributed data which are required for simulations are one of the most difficult steps for scientists and engineers who work with simulation applications. The SIMPL framework removes this burden from scientists, and provides a generic access to many kinds of data sources. SIMPL is generic enough to deal with many kinds of data sources which are required for many simulation software. Aforementioned, DM activities, DM patterns and the specifications of the SIMPL core operations are independent of data sources. Therefore, whenever it is required to extend SIMPL with new kinds of data sources, which are required for a new simulation software, they do not need to be modified or adopted to the new data sources. This feature provides extensibility for the SIMPL framework. Accordingly, using the SIMPL framework for data management and data provisioning in simulation applications reduces the complexity and improves the overall performance in cloud infrastructures. Nevertheless, one might still need to change some workflow steps in the workflow that provision or deprovision data with the help of the SIMPL framework.

6.3.3 Generalization of the presented approach for developing other variants of a TOSCA Service Template

This section discusses generalization capabilities of the presented approach for developing other variants of a TOSCA Service Template. Previously mentioned, a TOSCA Service Template consists of an application topology and management plans. The different variants of a TOSCA Service Template can be developed by using different versions for management plans or changing completely the structure of the application topology. In Subsection 6.3.3.1 the discussion is centered around using different versions of management plans inside the Service Template in order to develop different variants of a TOSCA Service Template. Subsection 6.3.3.2 evaluates generalization capabilities of the presented approach for developing different variants of a TOSCA Service Template via changing the application topology instead of management plans.

6.3.3.1 Management plans

In this master's thesis, some workflows were designed and implemented in more than one version. Different variants for the *PandasSoftwareProvisioning*, *DataProvisioning* and *DataDeprovisioning* workflows were implemented. Accordingly, different variants of a TOSCA Service Template can be designed and developed with support of these different variants. This section figures out how uncomplicated and straightforward it is to derive a new variant for one specific workflow from its other implemented variants. The evaluation in the following sections is centered around three different kinds of workflows, so-called *PandasSoftwareProvisioning*, *DataProvisioning* and *DataDeprovisioning*. Because, different variants were designed and implemented in this work only for these workflows, a different TOSCA Service Template for the PANDAS bone simulation software can be derived by using these different variants.

PandasSoftwareProvisioning plan:
In this master's thesis, three different variants for *PandasSoftwareProvisioning* plan were designed and implemented. The second and the third versions have some additional operations

compared to the first version. These additional operations send some commands to the PostgreSQL database server in order to change the database configuration parameters. Except those activities and operations, the rest of the workflow is the same as in the first version, and can typically be derived from the already existing implementations. With the help of Eclipse BPEL Designer, implementing different variants of this plan is not troublesome. Therefore, other variants of a TOSCA Service Template can be derived with the help of different variants of the *PandasSoftwareProvisioning* workflow. As mentioned already is Section 4.2.2.2, different architectures can be considered for this workflow. Scientists might need to develop other variants for this workflow than the ones which were implemented in this work. In other words, the new variants can be developed with respect to the different conditions and architectures which were listed in Section 4.2.2.2. With respect to these different architectures, the application topology inside the TOSCA Service Template might be different from the topology which was developed in this work. Therefore, in all above-mentioned cases, different variants for a TOSCA Service Template can be derived. Section 6.3.3.2 discusses more about developing other variants of a TOSCA Service Template via changing the application topology. In conclusion, the workflow steps might be different in different variants. If the required changes can be applied to the PANDAS web service and its source code with respect to each option in Section 4.2.2.2, developing other variants of this workflow might not be problematic with the help of Eclipse BPEL Designer.

DataProvisioning plan:
As mentioned already in Section 4.2.3, two different variants for the *DataProvisioning* plan were designed and implemented in the scope of this master's thesis. The SIMPL framework was used to design and implement these different variants. SIMPL has a user friendly GUI and eases difficulties of developing different variants. Moreover, DM activities and DM patterns in the SIMPL framework ease the whole development process. In the second variant of this workflow, a web service was developed in this work to change the coordination of some CSV-based data sources, which are required by the PANDAS bone simulation software. For implementing the second version, much of the already existing code in the first version were reused. In other words, only a new JAX-WS was designed and then used in the second variant. The second variant (see Section 4.2.3.2) only has an additional `Invoke` activity for calling this web service in the middle of the workflow. The results of this web service is stored in a directory on the Window-VM. The path to this directory is only assigned to the variable which is responsible for holding the right location of these input sources on the Windows-VM. Furthermore, some additional input parameters have to be passed to the workflow in the second variant. These input parameters are used for invoking this web service. The above-mentioned changes are only required for developing the second version of this workflow. If engineers are able to develop web services in general, developing the second variant for this workflow might not be difficult with the help of the SIMPL framework. Other variants could also be considered for this workflow than the variants which were developed in this work. For example, some pattern-based approaches in the SIMPL framework can be used for provisioning all the required data for the PANDAS bone simulation software. If scientists can work with the SIMPL framework in general, developing other variants might not be problematic.

DataDeprovisioning plan:
In this master's thesis, five different variants for the *DataDeprovisioning* workflow were designed and implemented (see Section 5.2.2.5). Similar to the *DataProvisioning* workflow, the SIMPL framework was used to design all variants of this workflow. As discussed before, the

first variant transfers results of the calculation from the created Linux-VM to the Windows-VM. The other variants on the other hand, query data directly from the PostgreSQL database of PANDAS and then transfer them to the Window-VM. The first variant of this workflow, except the usage of the *TransferData* activity of the SIMPL framework, is completely different from other variants. But, with the help of the existing implementations of the second variant, the other variants can be easily derived from the second variant. For instance, some data formats for storing the results of the queries are distinct in different versions. As an alternative, some versions have been implemented to query the result of one PANDAS instance, while some other versions can be used to query data of both PANDAS instances (mechanical and chemical in this work) simultaneously. In general, much of the already existing workflow steps in the second version were reused to implement the third, fourth and the fifth versions in this master's thesis. Furthermore, other variants could also be considered for this workflow than the variants which were developed in this work. For example, the approaches which were discussed in Section 4.2.5 can be used for developing other variants. But, from a conceptual point of view, it is required to retrieve results of the simulation from the related databases somehow in all variants. Therefore, if scientists are able to work with database systems, developing other variants would not be complicated.

6.3.3.2 Application topology

The discussion in the previous sections were centered around the development of different variants for a TOSCA Service Template by changing the management plans inside the Service Template. This section on the other hand, discusses the development of different variants for a TOSCA Service Template by having a completely different service topology. Different topologies for the PANDAS Service Template can be considered with respect to the different architectures, which were discussed in Section 4.2.2.2. For instance, if the related database is not deployed on the same Linux-VM of PANDAS instance. In other words, two VMs should be installed on the cloud, one for the PANDAS instance and one for the database system. In this situation, a different and distributed architecture should be considered for the PANDAS bone simulation tool than the one we used in this work. The implementation of the PANDAS web service and of the PANDAS source code require modifications. In the application topology of the PANDAS Service Template, a separate VM is required for installing the PostgreSQL database of PANDAS. A different topology for the PANDAS Service Template can be derived as well if the PANDAS instances are deployed on different VMs. Furthermore, a combination of the different options in Section 4.2.2.2 might change the topology of the PANDAS Service Template. In most of the above-mentioned cases, the PANDAS web service, PANDAS source code, PANDAS topology and related management plans require very sophisticated changes which are out of the scope of this master's thesis.

6.3.4 Using other cloud providers instead of OpenStack

The OpenStack cloud provider was used in this master's thesis for moving the PANDAS bone simulation software into cloud infrastructures. Other cloud providers can be considered as well. The OASIS TOSCA standard, which provides portability for cloud-based applications, was used in this work. Therefore, moving the PANDAS Service Template from the OpenStack to another cloud provider might not be very complicated. Using other cloud providers instead of the OpenStack for the PANDAS Service Template can be discussed from two different perspectives: (1) different cloud providers with interfaces similar to the OpenStack interface and (2) different cloud providers with interfaces different to the OpenStack interface. In the first case, as interfaces are similar, there is no need to change anything. In other words, if the API of the new cloud provider is similar to the OpenStack API, creating VMs and configuring them might not be different. Therefore, the topology and the management plans inside the PANDAS Service Template do not require any modifications. In the second case, the interface, so-called theAPI, input and output parameters are different for the new cloud provider. Consequently, in the first workflow of the PANDAS Service Template, which is *InfrastructureProvisioning* workflow, the first two steps which create a new VM and then configure the VM have to be adopted to the new interfaces and operations of the new cloud provider. To clarify, different input and output parameters can be used for calling the operation of creating a new VM. In conclusion, as the TOSCA standard was used in this master's thesis for moving the PANDAS Bone simulation software into cloud environments, developers do not need to worry about choosing a cloud provider.

Chapter 7

Summary, conclusion and future work

This chapter summarizes this master's thesis, brings this thesis to conclusions and outlines directions for future research. This chapter is divided into two sections as follows. In Section 7.1, a brief summary about the main contributions of this work as well as the implemented approaches is provided. This section brings this master's thesis to conclusions as well. Section 7.2, realizes some opportunities and future work for extending the application scope of the results of this thesis.

7.1 Summary and conclusion

Recent years have shown a steadily increasing trend to move applications and services into cloud infrastructures. Cloud-based applications benefit from virtual infrastructures in the cloud, such as processing, memory, storage, networking, etc. Automating the deployment and management of composite cloud-based applications is one of the main topics in relevant research. Automation makes the provisioning of new service instances for new customers cheaper and faster. The OASIS TOSCA standard is a formal language to describe the service topology and orchestration processes by means of management plans. The University of Stuttgart has developed an open source container for TOSCA, so-called OpenTOSCA [47].

Appropriate data management and data provisioning in simulation applications is one of the most challenging topics over the last decades. Lots of effort have to be done to find an adequate format for data sources in simulation applications, and to specify and implement required data transformations as well. SIMPL is an extensible framework which provides a generic access to many kinds of data sources in simulation workflows. With the SIMPL framework, the difficulties to deal with low-level details of data sources are removed from scientists and engineers[16].

In this master's thesis, different variants of a TOSCA Service Template for provisioning and executing PANDAS bone simulations in cloud infrastructures were designed and implemented. In other words, the main contribution of this work is to elaborate, develop and implement different variants of a TOSCA Service Template realizing the PANDAS bone simulation software in a cloud-native way. This Service Template provides an application topology as well as the corresponding management plans, which orchestrate the components inside the TOSCA service topology. To put it another way, a SaaS solution for the PANDAS bone simulation software has been provided.

The second contribution of this thesis is integrating the prototype of the SIMPL framework with the TOSCA standard. As both TOSCA and SIMPL are based on workflow languages, such as BPEL, the integration of these two approaches make the data provisioning technology, offered by SIMPL, as an integral part of the TOSCA descriptions. For this purpose, the *ODE-Service* IA was designed and implemented in this work in order to combine these two technologies. By using this IA inside the PANDAS service topology, it is possible for both TOSCA and SIMPL to work perfectly with each other. The outlines of the chapters in this work are summarized as follows.

Chapter one is introductory, and defined the motivations behind this master's thesis, the problem statement as well as the scope of work.

Chapter two discussed the basic knowledge regarding tools and technologies, which are necessary for this master's thesis. This chapter was subdivided into six parts. Part one provided a brief introduction to cloud computing. The OASIS TOSCA standard was discussed in part two. In part three, the concept of workflows and web services was discussed. The fourth part argued about simulation applications in general as well as a simulation workflow for structure changes within a human bone. A generic framework for data management and data provisioning in simulation applications, so-called SIMPL, was discussed in part five. Finally, Part six addressed a brief overview about technologies and tools which were used in this master's thesis.

Chapter three provided an extensive study of related work in the area of moving legacy and on-premise applications into cloud infrastructures. In this chapter, the related work were investigated from three different points of view: (1) running simulation software in the cloud, (2) provisioning components of simulation software and related resources in cloud environments and (3) provisioning and managing data for cloud-based simulation software. By discussing some related work to PANDAS as a simulation software, TOSCA as a software provisioning tool in the cloud and SIMPL as a data provisioning tool in this chapter, some results can be derived as follows:

- Bringing the PANDAS bone simulation software into cloud for simulating structure changes within a human bone is a better and easier approach than using other simulation software in the cloud, such as cloud-based MATLAB, for bone simulations.
- The OASIS TOSCA standard is a more efficient approach for moving the PANDAS bone simulation software into cloud than other approaches, such as Vagrant, Wrangler, etc.
- SIMPL is a generic and consolidated approach for data management and data provisioning of a PANDAS bone simulation software. It would be more efficient to use the SIMPL framework than other approaches, such as standard tools for the ETL processes, the method from the SDM center, the OGSA-DAI framework, etc.

Chapter four concentrated conceptually on the proposed approaches for the main two contributions in this master's thesis. First, an approach to move the PANDAS bone simulation software into cloud infrastructures with the help of TOSCA and SIMPL technologies was illustrated. Then, an efficient method for integrating the SIMPL prototype with the TOSCA standard was realized. The conceptual results of this master's thesis can be summarized and concluded as follows:

- In Section 4.1, the TOSCA-based application topology for the PANDAS bone simulation software was designed. The three VMs which were used for mapping the PANDAS bone simulation software into cloud were discussed. Furthermore, the different IAs, which were used inside the PANDAS service topology, were discussed in detail.

- Section 4.2 argued six different types of management plans, which are used for moving PANDAS into the cloud. The middle-level design of all management plans as well as their different variants were discussed. From the conceptual point of view, different variants of a Service Template, which consist of an application topology and management plans, for the PANDAS bone simulation software were designed in this chapter.
- An approach for integrating the SIMPL framework with the OASIS TOSCA standard was designed in Section 4.3. Conceptually, different approaches for this integration were discussed in this section and compared as well with the one (integrating via the *ODE-Service* IA) which were implemented in Chapter Implementation. As mentioned in Section 4.3.5, the implemented approach in this thesis is concluded as a more efficient approach than plug-in architectures from a conceptual point of view.

In Chapter *Implementation*, the implementations of the discussed approach for integrating SIMPL and TOSCA were argued in detail. An IA inside the PANDAS service topology was designed and implemented in order to integrate these two technologies. Furthermore, the actual implementations of the PANDAS bone simulation software, as a SaaS solution, in cloud environments were discussed. In other words, different variants of a TOSCA Service Templates for the PANDAS bone simulation software, which consists of a corresponding services topology and management plans, were implemented. Finally, all challenges and problems the author faced during the deployment and implementation phases were discussed.

In Chapter six, the results and findings of this master's thesis were evaluated from various aspects. This chapter was subdivided into three sections. Section 6.1 evaluated whether or not, and to what extent, the findings of this work addressed the goals and motivations of this thesis, which were mentioned in Chapter one. In Section 6.2, the lessons which were learned in this master's thesis with respect to the approaches and contributions were argued. Section 6.3 discussed generalization capabilities of the presented approaches to other simulation examples, other simulation software and for developing other variants of a TOSCA Service Template. Furthermore, this chapter discussed briefly about using other cloud providers for the PANDAS Service Template instead of the OpenStack, which was used in this master's thesis. The results of this chapter can be summarized and concluded as follows:

- Using cloud infrastructure within organizations turns applications to become more mobile and collaborative. Cloud computing provides a large number of various advantages for simulation applications. In brief, moving composite simulation software, such as the PANDAS bone simulation software, into cloud infrastructures provides a large number of desirable features. In this master's thesis, the OASIS TOSCA standard and the SIMPL framework were used to move the PANDAS bone simulation software into cloud environments and to realize this simulation software as a SaaS solution on cloud infrastructures. According to the goals of this master's thesis, which were discussed in Section 6.1, the goal of mapping the traditional PANDAS bone simulation into cloud infrastructures has been reached. With the help of TOSCA management plans, the second goal of this thesis, which is automating the deployment and management of the PANDAS bone simulation software, has been reached as well. Furthermore, users of PANDAS do not need to deal with low level information of each data source by using the SIMPL framework. In other words, the third goal, which is data management and data provisioning of PANDAS in the cloud, has been reached. Lastly, The goal of integrating the TOSCA standard and the SIMPL framework has been reached via the *ODE-Service* IA. The implemented approaches in this thesis have some limitations, which were discussed in Section 6.2, and can be used to derive some recommendations for future work which is discussed in the following in Section 7.2. A brief summary of the lessons which were learned in this thesis are listed as follows.

- Using the OASIS TOSCA standard for automating the deployment and management of the PANDAS bone simulation software is an efficient approach. Therefore, the assumptions about TOSCA, which the author made in Section 3.2 are correct.
- On the other hand, the Winery tool does not support graphical editors, such as Eclipse BPEL Designer, for developing the management plans inside TOSCA Service Templates. Consequently, it is problematic to develop the management plans without help of editors. As it is discussed in Section 7.2.3, extending the Winery tool to have an editor for developing TOSCA management plans can be conducted as a future work.
- Using the SIMPL framework as a generic approach for data management and data provisioning of the PANDAS bone simulation software in cloud infrastructures is a good decision. As a consequence, the assumptions in Section 3.3 about SIMPL with respect to other approaches for data provisioning and data management are correct. In reality, the SIMPL framework frees developers to deal with the low-level details of data sources. Therefore, the author believes that using the SIMPL framework for extending the presented approach in this thesis, as it is discussed in Section 6.3, is a good choice.
- It is a little bit challenging in the beginning of working with the SIMPL framework. At first, developers may think that SIMPL provides many complexities instead of simplifying the data management and data provisioning processes. But, after working a little bit with this framework, developers can make sure that SIMPL is a good choice for data management and data provisioning processes.
- The *ODE-Service* IA is evaluated as a good approach for integrating TOSCA and SIMPL from the author's point of view. As discussed in Section 4.3.5, this approach has some advantages and limitations. The author believes that optimizing this approach in future provides the generic SIMPL-based approach for data provisioning and data management of cloud-native applications as an integral part of the TOSCA standard definition. Therefore, as it is discussed in Section 7.2.1, optimizing this approach can be conducted as a future work.
- The most important conclusions which can be derived from discussions in Section 6.3 are: (1) the presented approach in this master's thesis can be generalized to other simulation examples, which need to be calculated by PANDAS without many difficulties, (2) the generalization capability of the proposed approach to other simulation software, different to PANDAS, is negotiable and can be so sophisticated in some cases (3) the different variants of the workflows inside the PANDAS Service Template can be derived without many difficulties from the existing versions.

7.2 Recommendations for future work

While this master's thesis has demonstrated the potential of using the OASIS TOSCA standard and the SIMPL framework to move a simulation software, so-called PANDAS, into cloud infrastructures, many opportunities for extending the scope of this thesis are remained. The following subsections present some of these directions and opportunities.

7.2.1 TOSCA and SIMPL integration

A future work has to be conducted to optimize the implemented approach in this work for TOSCA and SIMPL integration. This future work is considered with respect to the *ODE-Service* IA, which was designed and implemented in this master's thesis. The current implementation of this IA deploys and undeploys workflows perfectly. But, developers need to determine the workflows which require an extended ODE engine for their executions in advance. Then, the `Deploy` operation of the *ODE-Service* IA is called for these workflows inside the first plan. In a similar way, the `Undeploy` operation of this IA is invoked in the last plan of the PANDAS Service Template for the workflows which need to be removed from the ODE-PGF engine. Accordingly, an optimized approach for this IA which deploys and undeploys workflows automatically can be designed and implemented as a future work.

Another future approach could be extending the workflow engine inside the OpenTOSCA environment by introducing some plug-ins, as discussed in Section 4.3.1. The workflow engine inside the OpenTOSCA environment can be extended to accept the DM activities and DM patterns of the SIMPL framework. To put it another way, the workflow engine inside the OpenTOSCA environment, which is a WSO2 Business Process Server, needs to be extended by the plugable framework of the ODE-PGF engine. Only one workflow engine, i.e. the WSO2 Business Process Server inside the OpenTOSCA environment, is used in this approach for executing all workflows inside the TOSCA Service Template. The other components of the SIMPL framework can be connected to OpenTOSCA components in a loosely coupled way.

Furthermore, a future work for integrating TOSCA and SIMPL could be extending the OpenTOSCA environment with some plug-ins to distinguish SIMPL-based workflows from TOSCA-based workflows. After deploying a CSAR file, which consists of an application topology and management plans, on the OpenTOSCA environment, the deployment of workflows has to be done automatically. Based on the plug-ins, the SIMPL-based workflows are distinguished in an automated-manner from the TOSCA-based workflows, and then the appropriate engine for executing these workflows is selected. Obviously, this approach relies on two different workflow engines (ODE-PGF and WSO2 Business Process Server). To put it another way, there is no need for developers to explicitly define workflow steps for deploying the SIMPL-based workflows in the first workflow. In the *ODE-Service* approach, the first workflow of the PANDAS Service Template determines the SIMPL-based workflows which need to be deployed on the ODE-PGF engine.

Similar to the previous methods, another future work can be conducted by using plug-in architectures as well. The plug-in functionality is already provided in the OpenTOSCA environment. In this case, scientists are able to define in the TOSCA plans on which engine they should be deployed. Therefore, there is only need to develop the corresponding plug-ins and then deploy them inside the OpenTOSCA environment. With the help of these plug-ins, the OpenTOSCA environment automatically deploy the corresponding workflows on the defined workflow engine. This approach relies on two different workflow engines (ODE-PGF and WSO2 Business Process Server). As discussed in Section 4.3.5, the efficiency of the implemented approach in this thesis to integrating TOSCA and SIMPL is arguable as compared to plug-in architectures. In other words, this approach can be considered as a more efficient approach than the plug-in based approaches which were mentioned above. Therefore, this theory should also be investigated in detail in future work.

7.2.2 Generalizing the implemented PANDAS Service Template

The generalization capabilities of the approach proposed in this master's thesis were discussed in Section 6.3 from four different aspects: (1) generalization capability for other simulation examples, (2) generalization capability for other simulation software instead of PANDAS, (3) generalization capability for developing other variants of a TOSCA Service Templates and (4) generalization capability of the approach with respect to other cloud providers instead of the OpenStack. This section evaluated the generalization capabilities of each workflow inside the PANDAS Service Template. Therefore, some future work can be conducted by extending the approach proposed in this master's thesis to other simulation examples, other simulation software, developing other variants of a TOSCA Service Template and using other cloud providers than the OpenStack.

7.2.3 Extending the Winery tool to have an editor for developing TOSCA management plans

As mentioned already, Winery is a graphical environment tool which can be used to model the TOSCA Service Templates. A TOSCA Service Template consists of an application topology as well as management plans. The current version of the Winery tool provides a user friendly GUI for developing the application topology. The management plans on the other hand, cannot be designed and implemented inside the Winery tool. Different management plans are implemented manually and then imported into the Winery tool in order to be included in the TOSCA Service Templates. Writing all steps of the management plans manually without the help of any graphical editors, such as Eclipse BPEL Designer, is cumbersome and error-prone. Therefore, one of the future work of this master's thesis has to be conducted by the extension of the Winery tool with a graphical editor for developing the TOSCA management plans. Furthermore, the Winery tool is based on Eclipse as well. Maybe it would be possible to integrate Winery with the Eclipse BPEL Designer (maybe even with the one that is extended by the SIMPL framework). The possibilities for this integration can be investigated in future work.

7.2.4 Running choreographies for the PANDAS bone simulation software

Most of simulation applications consist of data-intensive computing. The size of data and number of services involved in simulation applications are increased significantly. Therefore, centralized orchestration techniques[14] are not so efficient. In the orchestration approach, all data are passed into a centralized engine, which can be a bottleneck to the execution of a workflow. Therefore, if huge amounts of data can be passed directly to where they are required, it prevents from overloading the workflow engine and increases the overall performance. In general, the interaction between multiple workflows can be modeled by using choreographies[3]. In other words, the participating web services are collaborating and communicating with each other in a peer-to-peer fashion in choreographies approaches. Instead of using orchestration languages, such as BPEL, different languages such as Web Services Choreography Description Language (WS-CDL)[1] are used in choreographies. Choreographies can also be modeled with BPEL4Chor[13]. Then, the model can be transformed into executable BPEL processes. A future work can be conducted by running choreographies for the PANDAS bone simulation software in order to connect the required web services. To put it another way, the PANDAS bone simulation software is based on intensive and heterogeneous data sources. Different workflows inside the PANDAS Service Template have to collaborate and communicate with each other. In the current implementation of the PANDAS Service Template, different workflows inside the Service Template have to be called via a SoapUI test case. Developers need to define an order for running the management plans in order to automate the deployment and management of the PANDAS bone simulation software in the cloud. Each workflow can be started automatically by receiving a notification from the previous executed workflow by using choreographies.

7.2.5 Running a coupled simulation for PANDAS

As mentioned already, the PANDAS bone simulation software is responsible for simulation on the tissue level. PANDAS calculates structure changes within a human bone for one FEM element and then combines all the FEM elements within a loop activity. The simulation for structure changes within a human bone can also be done on the cell level. For this purpose, Matlab[2]-Bone workflows have to communicate and collaborate with the workflows for the PANDAS bone simulation software, which have been developed and implemented in this master's thesis. In other words, for each FEM element, which is calculated by PANDAS, 8 cells can be calculated with the help of Matlab simulator. It is also required to have some `Data-Manager` workflows, which are responsible to transform the data that is used by PANDAS at the tissue level to the appropriate formats for Matlab simulation and vice versa. The collaboration between simulations and workflows can be done via the choreographies concept. The execution of multiple simulations can be linked with each other via choreographies. As a consequence, a future work for this master's thesis can be conducted by coupling the simulation at the tissue level, i.e. PANDAS simulation, with the simulation at the cell level, i.e. Matlab simulation by using choreographies[29].

It is also possible to couple the PANDAS bone simulation software with the GNU Octave software, which simulates structure changes within a human bone on the cell level[18]. These two simulations can be linked with each other with the help of choreographies. Therefore, the contributions of this master's thesis are the steps towards moving these kinds of coupled simulation into the cloud via choreographies.

Furthermore, the PANDAS bone simulation software can collaborate with some visualization tools in order to visualize the results of the calculations. A future work can be conducted as well to couple the PANDAS bone simulation software with these visualization tools in cloud.

[1]http://www.w3.org/TR/ws-cdl-10/
[2]http://de.mathworks.com/products/matlab/

References

Literature

[1] Arun Poduval et al. *BPEL Cookbook: Best Practices for SOA-based integration and composite applications development*. Packt Publishing, July 2006 (cit. on pp. 17, 18).

[2] Arshdeep Bahga and Vijay Madisetti. *Cloud Computing: A Hands-On Approach*. CreateSpace Independent Publishing Platform, December 9, 2013 (cit. on pp. 1, 8).

[3] *The Benefits of Service Choreography for Data-Intensive Computing*. New York, USA: Proceedings of the 7th international workshop on Challenges of large applications in distributed environments, 2009 (cit. on pp. 94, 96, 118).

[4] Claude Baudoin and Eliezer Dekel et al. *Interoperability and Portability for Cloud Computing: A Guide*. Tech. rep. United States: Cloud Standards Customer Council, Nov. 2014. URL: http://www.cloud-council.org/CSCC-Cloud-Interoperability-and-Portability.pdf (cit. on p. 10).

[5] Tobias Binz et al. In: *Advanced Web Services*. New York: Springer, Jan. 2014. Chap. TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549 (cit. on pp. 11–13, 46).

[6] Uwe Breitenbücher et al. *Vinothek – A Self-Service Portal for TOSCA*. Tech. rep. Germany: Institute of Architecture of Application Systems, University of Stuttgart, Germany, Institute for Parallel, and Distributed Systems, University of Stuttgart, Germany. URL: http://ceur-ws.org/Vol-1140/paper11.pdf (cit. on p. 14).

[7] Miguel Caballer and Blanquer et al. “Dynamic Management of Virtual Infrastructures”. In: *Journal of Grid Computing* 13.1 (2015), pp. 53–70. URL: http://dx.doi.org/10.1007/s10723-014-9296-5 (cit. on pp. 34, 35).

[8] C.Müller. “Development of an Integrated Database Architecture for a Runtime Environment for Simulation Workflows”. Diploma Thesis. Stuttgart, Germany: University of Stuttgart, 2010 (cit. on p. 21).

[9] W. Ehlers. *IUTAM Symposium on Theoretical and Numerical Methods in Continuum Mechanics of Porous Materials: Proceedings of the IUTAM Symposium held at the University of Stuttgart, Germany, September 5–10, 1999*. Solid Mechanics and Its Applications. Springer Netherlands, 2006. URL: https://books.google.de/books?id=biXrBwAAQBAJ (cit. on p. 100).

[10] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. “Elasticity in Cloud Computing: What It Is, and What It Is Not”. In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 23–27. URL: https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst (cit. on p. 10).

[11] Andreas Hoheisel. “User Tools and Languages for Graph-based Grid Workflows”. In: *Concurrency Computation Practice and Experience* 18.10 (25 August 2006), pp. 1101–1113 (cit. on p. 41).

[12] Gideon Juve and Ewa Deelman. “Wrangler: Virtual Cluster Provisioning for the Cloud”. In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. HPDC ’11. San Jose, California, USA: ACM, 2011, pp. 277–278. URL: http://doi.acm.org/10.1145/1996130.1996173 (cit. on pp. ix, 35–37).

[13] Oliver Kopp and Frank Leymann. *Choreography Design Using WS-BPEL*. 2008 (cit. on p. 119).

[14] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000. URL: https://books.google.de/books?id=Xc8eAQAAIAAJ (cit. on pp. 15–17, 118).

[15] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDIs*. Pearson Education Corporate Sales Division, May 23, 2002 (cit. on pp. 8, 9).

[16] Peter Reimann, Michael Reiter, and et al. “SIMPL – A Framework for Accessing External Data in Simulation Workflows”. In: (March 2011) (cit. on pp. 2, 20, 21, 23, 24, 40, 113).

[17] Peter Reimann and Holger Schwarz. “Simulation Workflow Design Tailor-made for Scientists”. In: *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*. SSDBM ’14. Aalborg, Denmark: ACM, 2014, 49:1–49:4. URL: http://doi.acm.org/10.1145/2618243.2618291 (cit. on pp. 19–22).

[18] Peter Reimann et al. “Datenmanagement in der Cloud für den Bereich Simulationen und Wissenschaftliches Rechnen”. Deutsch. In: *Proceedings des 2. Workshop Data Management in the Cloud auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. Ed. by Gesellschaft für Informatik e.V. (GI). Stuttgart, Deutschland: Lecture Notes in Informatics (LNI), Sept. 2014. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2014-52&engl= (cit. on pp. 3, 32, 44, 65, 71, 119).

[19] *SDM Center Technologies for Accelerating Scientific Discoveries*. Boston, Massachusetts, 2007 (cit. on p. 40).

[20] Tao Sun and Xinjun Wang. “Research on Heterogeneous Data resource Management Model in Cloud Environment”. In: *International Journal of Database Theory and Application* 6.5 (2013), pp. 141–152 (cit. on p. 40).

[21] Michael Zimmermann. “Konzept und Implementierung einer generischen Service Invocation Schnittstelle für Cloud Application Management basierend auf TOSCA”. Bachelor Thesis. Stuttgart, Germany: University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, 2013 (cit. on p. 78).

Online sources

[22] URL: http://www.w3schools.com/ (cit. on pp. ix, 9, 17).

[23] URL: http://www.postgresql.org/docs/9.2/static/config-setting.html (cit. on pp. ix, 50, 51, 81, 82).

[24] URL: http://www.postgresql.org/docs/9.1/static/sql-copy.html (cit. on pp. ix, 61).

[25] URL: http://www.postgresql.org/docs/8.1/interactive/backup.html (cit. on pp. ix, 62–64).

[26] URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca (cit. on pp. 2, 10, 38).

[27] URL: http://searchcloudapplications.techtarget.com/definition/cloud-application (cit. on p. 10).

[28] URL: http://www.simio.com/applications/ (cit. on p. 18).

[29] URL: http://www.iaas.uni-stuttgart.de/forschung/projects/simtech/sim-workflows.php (cit. on pp. 18, 119).

[30] URL: http://www.engr.uvic.ca/~mech410/lectures/FEA_Theory.pdf (cit. on p. 19).

[31] URL: http://www.iaas.uni-stuttgart.de/forschung/projects/ODE-PGF/ (cit. on p. 26).

[32] URL: http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm (cit. on p. 26).

[33] URL: https://eclipse.org/bpel/ (cit. on p. 27).

[34] URL: http://www.soapui.org/ (cit. on p. 27).

[35] URL: http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/ (cit. on p. 27).

[36] URL: http://askstop.com/questions/518803/what-is-the-difference-between-aar-and-war-file (cit. on p. 29).

[37] URL: http://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/6356/Simulation-in-the-Cloud-is-Becoming-Mainstream.aspx (cit. on p. 31).

[38] URL: https://www.comsol.de/blogs/running-comsol-multiphysics-with-amazon-ec2/ (cit. on p. 33).

[39] URL: http://www.mechbau.uni-stuttgart.de/pandas/index.html (cit. on p. 33).

[40] URL: https://www.vagrantup.com/ (cit. on p. 37).

[41] URL: http://www.ogsadai.org.uk/ (cit. on p. 40).

[42] URL: http://www.gistutor.com/postgresqlpostgis/10-intermediate-postgresqlpostgis-tutorials/39-how-to-import-or-export-a-csv-file-using-postgresql-copy-to-and-copy-from-queries.html (cit. on p. 61).

[43] URL: http://www.codeguru.com/cpp/misc/misc/plug-insadd-ins/article.php/c3879/Plugin-Architecture-Framework-for-Beginners.htm (cit. on p. 68).

[44] URL: http://ddweerasiri.blogspot.de/2009/05/how-to-deploy-axis2-web-service.html (cit. on p. 71).

[45] URL: http://dev.winery.opentosca.org/winery/servicetemplates/ (cit. on pp. 72–74).

[46] 2007. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (cit. on p. 17).

[47] 2013. URL: http://www.iaas.uni-stuttgart.de/OpenTOSCA/ (cit. on pp. 14, 15, 113).

[48] 2013. URL: http://www.slideshare.net/OpenTOSCA/tosca-and-opentosca-tosca-introduction-and-opentosca-ecosystem-overview (cit. on p. 14).

[49] June 24, 2003. URL: http://www.w3schools.com/webservices/ (cit. on p. 27).

[50] Timothy Grance Peter Mell. September 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (cit. on pp. 7, 8).

All links were last followed on July 23, 2015.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

(Marzieh Dehghanipour)