A SOA based approach to the Next Generation Networks management

Ing. Paolo Anedda

Advisor: Dott. Ing. Luigi Atzori
Curriculum: ING-INF/03- AREA INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

XXI Cycle
March 2009
To my girlfriend and my family
for their support.
Contents

1 Introduction 13
  1.1 Overview ......................................................... 13
  1.2 Aims of the Thesis ............................................ 16
  1.3 Outline .......................................................... 19
  1.4 Related Papers .................................................. 19

2 Background 21
  2.1 Service Oriented Architectures ............................... 21
    2.1.1 Introduction to Service-Oriented Computing .......... 21
    2.1.2 Service-Oriented Architecture .......................... 22
    2.1.3 Service-Orientation, Services, and Service-Oriented Solution Logic ............................................. 22
    2.1.4 Service Compositions ........................................ 23
    2.1.5 Service Inventory ............................................ 23
    2.1.6 Understanding Service-Oriented Computing Elements ........ 23
    2.1.7 Service Model ................................................. 25
    2.1.8 Service Inventory Blueprints .............................. 26
    2.1.9 Service-Oriented Analysis and Service Modeling ....... 27
    2.1.10 Service-Oriented Analysis and Design .................. 27
    2.1.11 Goals and Benefits of Service-Oriented Computing ... 28
  2.2 Web Services ..................................................... 28
    2.2.1 Overview ...................................................... 29
    2.2.2 WSDL .......................................................... 34
    2.2.3 WS-BPEL ...................................................... 37
    2.2.4 Jolie .......................................................... 54
3 State of the Art: SOA Network Management 65
3.1 Overview ......................................................... 65
3.2 Business context and technologies of web services in telecommunications 66
3.3 Dynamic composition of Web services .................................. 66
3.4 Performances .................................................................. 68
3.5 Deployed implementations of Web services-based systems .......... 69
3.6 Virtual resources management ............................................. 70

4 The Device Management System Software 73
4.1 Overview ..................................................................... 73
4.2 Software Architecture .................................................... 76
  4.2.1 Simple services: The proxy architecture ......................... 76
  4.2.2 Complex services: realizing a control plane for services orchestration .................................................. 79
  4.2.3 Applications ................................................................ 79
4.3 The device management library ........................................... 80
4.4 The VLan Creation Example .............................................. 87

5 Virtual Cluster Management 103
5.1 Overview ...................................................................... 103
  5.1.1 Virtual Computing Facilities ....................................... 103
5.2 Architecture design and implementation ............................ 104
  5.2.1 Technologies ............................................................. 108
  5.2.2 HaDeS ..................................................................... 109
  5.2.3 PFC ......................................................................... 109
  5.2.4 VMH ........................................................................ 111
  5.2.5 Machine Manager service ............................................. 112
  5.2.6 Virtual Machine Manager service ................................. 113
  5.2.7 Virtual Resource Manager Service ................................. 114
  5.2.8 Resources Manager .................................................... 114
  5.2.9 Vlan Manager Service ............................................... 115
  5.2.10 Switch Manager Service ............................................. 115
  5.2.11 User Application ...................................................... 116
  5.2.12 Notifier .................................................................... 117
CONTENTS

5.2.13 Account service ............................................ 117
5.3 Tests Evaluation .................................................. 118

6 Conclusions and Future Works .................................. 123
List of Figures

2.1 A conceptual view of how the elements of service-oriented computing can inter-relate .................................................. 24
2.2 Common service abstraction layers established by service models. . 25
2.3 Web service acting as a service provider. ............................. 30
2.4 Web service acting as a service consumer. .......................... 30
2.5 Web service acting as a service provider and consumer. .......... 30
2.6 WS-BPEL partnerLinkType construct ................................. 39
2.7 Request Dispatch Flowchart of the Active Bpel engine ............ 54

4.1 The services’ abstraction layers. ....................................... 74
4.2 The system’s architecture. ............................................... 76
4.3 The proxy’s architecture. ............................................... 77
4.4 The proxy’s implementation. ......................................... 78
4.5 The proxy’s pluggable architecture. .................................. 79
4.6 The architecture’s main services. .................................... 80
4.7 The Inversion of Control pattern. ..................................... 81
4.8 The devices class diagram. ............................................ 83
4.9 The connection class diagram. ....................................... 86
4.10 The vlan creation example. .......................................... 87
4.11 The vlan creation workflow. ........................................ 93

5.1 State diagrams for physical and virtual resources. .................. 105
5.2 The architecture’s main components. ............................... 106
5.3 The lifecycle of a virtual host. ....................................... 111
5.4 UI screenshot. ......................................................... 117
5.5 The testbed environment layout. .................................... 118
Chapter 1

Introduction

1.1 Overview

Nowadays we are experiencing a rapid evolution in the field of telecommunications. Driven by the evolution of mobile networks as well as by the diffusion of Internet and mobile applications, the convergence between the telco and the IT worlds is opening new applications’ scenarios. Even if creating new market-driven applications reusing an extensible set of existing service components has been a key aspect of telecom platforms for years, nowadays operators must deploy global IP-based networks to address the convergence of fixed and mobile telecommunications, cable networks, and the Internet. They must create scalable infrastructures to be able of cutting costs, stimulating service growth and innovation, and exceeding the customer’s quality of service (QoS) expectations.

From a business viewpoint, the ability for independent businesses to communicate in an automated, and reasonably close to real time manner (i.e. Just-In-Time systems), provides a key element in realizing the cost benefits of system automation. Also integration of business processes within an organization is often required before the organization can effectively supported extended integration (integration across corporate networks, or to external networks).

From a technology viewpoint, when connecting external entities, a significant challenge is the ability of independent business entities to exchange information correctly. Until now, the traditional implementation of this model, based on past technologies such as Electronic Data Interchange (EDI), private protocols, and the definition of
CHAPTER 1. Introduction

APIs, has lead to the creation of complex and monolithic network management architectures that are not capable of adapting themself to the changes imposed by the evolution of the telecommunication world.

Today, telecommunication infrastructure is more software intensive than ever, blurring the borders between IT (Information Technologies) systems and networking equipment. As such, new telecom equipment can benefit from generic software technologies and engineering techniques. Service Oriented Architecture (SOA) seems to be the most promising architecture for software systems. Many research activities, as well as some new commercial ones, have identified the SOA as the paradigm that could help to keep under control these complex scenarios and allow to follow the markets changes. According to SOA, systems are built orchestrating the services that different components expose through well defined interfaces.

SOA is an architectural model whose main idea is to loose couples services with operating systems, programming languages and other technologies which underlie systems or applications. This implies to adopt a new approach in business definition where each function is related to a specific business unit or service accessible over a network through standard communication protocols. This approach allows programmers to compose and reuse services for the developement of their systems and applications.

The main SOA principles establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing. Services acts as physically independent software programs with distinct characteristics. Each service is assigned its own functionalities that are suitable for invocation by external consumer programs are commonly expressed via a published service interface. Services can be thought as pieces of software that provide some functionalities in order to realize very specific tasks. Acting as black boxes, they expose these functionalities through a common interface that hides the internal details of the implementation logic that they realize.

The design of a traditional application, which is usually associated with the automation of a business process, following the service-orientation design principles, produces the creation of services with functional contexts that are agnostic to any
business process. These agnostic services are therefore capable of participating in multiple service compositions, thus realizing the specified business process. The adoption of the SOA’s principles allows to decompose complex and monolithic systems into applications composed by an ecosystem of simpler and well defined components. The use of common interfaces as well as standard protocols, gives an horizontal view of an enterprise’s system; thus the developing of business processes is the result of the process of designing workflows of coordinated services. This facilitates the interaction between the parts of an enterprise and allow to reduce the time necessary to adapt itself to the changes imposed by the markets evolution. A SOA approach also allows software as well as hardware reuse, because it doesn’t impose a specific technology for the services implementation.

SOA is an architectural model that is agnostic to any one technology platform. One of its most popular implementation is that of Web service. These are defined through a number of industry standards and are characterized by a plethora of XML-based specifications which deal with different aspects of such a kind of technology. A typical Web service is comprised of a physically decoupled technical service contract, a body of programming logic and a message processing logic. The core set of standards in web services include XML for data representation, XML Schema for data types, SOAP for data messaging, WSDL for service description and UDDI for service discovery.

SOAP essentially prescribes an XML based standard for messaging, and is based on the idea of sending one-way XML payload in a special envelope, independent of the underlying transport mechanism. Most commonly SOAP is assumed to ride over the popular HTTP transport mechanism, yielding a way to generate a request-response kind of SOAP message exchange pattern. Unlike this popular transport, SOAP can also ride over other transport mechanisms that offer asynchronous messaging capabilities. More often than not, SOAP messages are sent asynchronously in enterprise scenario. WSDL offers an implementation independent mechanism to represent the details of a service implementation. WSDL separates the specification of the service into two parts: an abstract part, and a concrete binding. The abstract part is specified by a portType, which is a collection of operations each of which in turn is a collection of messages, each message specifying a particular data exchange in either input or output mode. The abstract specification of WSDL in terms of port-
Types is important for typically separating web services interfaces from the actual implementation details.

1.2 Aims of the Thesis

The aim of this thesis is to investigate the real benefits that the adoption of the SOA principles can bring to the world of network and computing resources management. It aims at supplying a reasoning about the main concepts the Service Oriented Computing paradigm, that we believe are a valid and flexible way to manage the complexity of the resources management for a telecom organization which has to satisfy the requests for new services in an ever evolving technological environment. By means of formal models indeed, it is possible to define the peculiarities of services and services systems. Moreover, the thesis aims at supplying a new concrete model, based upon an architectural implementation, to network management.

Starting from the analysis of the theoretical concepts of the Service Oriented Programming, and ending with a software implementation, the present thesis could be considered a demonstration of how these principles could be applied to solve some of the complex tasks related to the network management, and to open new scenarios in the development of technological services.

Designing as well as managing a network is a very complex task. In order to achieve this goal several network’s management models have been proposed. The International Organization for Standards (ISO) addresses the five major functional area of the Network Management Model as:

- **Performance management.** Performance management means monitoring, assessing, and adjusting the available bandwidth and network resource usage in order to make a network run more efficiently.

- **Configuration management.** The primary goal of configuration management is to gather/set/track configurations of the devices.

- **Accounting management.** Accounting management monitors and assesses the usage of data and/or resources for the purpose of billing. This aspect of the network management is by Internet Service Providers to bill customers for the resources they use.
1.2. Aims of the Thesis

- **Fault management.** Fault Management is what most people think of when they think of network management. The purpose of this area of network management is to detect, log and alert the system administrators of problems that might effect the systems operations.

- **Security management.** Security Management deals with controlling access to resources and even alerting the proper authorities when certain resources are accessed.

In this thesis, we present a new approach to network management based on Service Oriented Architecture (SOA) and web services. We focused our work on the configuration and fault management areas through the definition of different services that participate in the creation of a very flexible and adaptable architecture for the deployment of next generation network based services. The use of SOA, and in particular the use of web services, can leverage the automatic network management as well as the interoperability. Our approach reduce the gap between the application management and the network management and tries to give an unique view on the management of the technological assets of an enterprise.

With our work we want to demonstrate how, a unified vision on the management of the network and the IT resources, supported by the use of a service oriented approach, can help to fill the gap between the creation of network services and IT services and leverage the deployment of next generation applications. In order to prove this reasoning, we develope a software, the Devices Management System (DMS), for the creation of network based services.

DMS is a software library that allows for the remote management of network as well as computing devices, through a web service interface.

DMS is organized in two separate logical layers: the device library and the service interface library.

The device library is the layer responsible for the device management. It is based on an object oriented model of the network's components. The devices model is organized into two separate packages. The former assembles all the objects related to the devices representation while the latter is composed by all the objects responsible for the low level communications with the devices.

The service interface library is the layer responsible for the web service communic-
tion management. Each device is associated to a specific web service that exposes, using the standard protocols for the web service interface definition, all the configuration functionalities of the specified device. The atomic functions of the device are grouped together, in a well defined sequence, to form complex operations.

The main idea behind the use of web service is to hide all the inner details related to the devices configuration as well as those related to the communication protocols used to access the device’s management console. Moreover, it allows to create a layer of abstraction that simplify the configuration operations and provides a common language for the administration of different vendor devices. One main advantage of a service approach to the network management is that it allows to benefit from the potentialities offered by the service composition. The composition of different services allows to create complex network administration operations such as the creation of vlans or the configuration of the QoS policies. To define and execute the processes for the services composition, in our implementation we use standard web services orchestration languages. The interactions among the different services are designed using a workflow and all the operations are carried on through web service methods invocation.

Even if using a service architecture that realizes a layer of abstraction over the network devices may seem pointless and onerous, the adoption of this approach is very powerful in the creation of complex and dynamic infrastructures that require the network to be very flexible and to adapt itself to the needs of the applications.

In this work we report on the adoption of this methodology for the implementation of an architecture for the creation of virtual computing clusters that has been conducted at the CRS4 in Cagliari, where we took part in a research initiative called Cybersar.

Cybersar is an high performance computing initiative, recently funded by the Italian Ministry of Research, that has as its goal the development of a Cyber-infrastructure for research in Sardinia based on high speed networks interconnecting the island research communities and computational facilities. Cybersar infrastructure is designed so that it can support innovative computational applications while providing an experimental platform for research on application driven orchestration of computing and network resources.

At the moment, some of the code developed for the DMS project, are being used at
the CRS4 to manage the network layer of an infrastructure for the dynamic deployment of virtual computing clusters. In the fourth chapter, we report on this activity, showing the use of our approach in a real context.

1.3 Outline

The outline of the thesis follows:

- In Chapter 2 we recall the main principles of the SOA paradigm with a particular regard to the features offered by the Web Service technology. We provide a brief introduction to the Web service specifications and its related technologies.

- In Chapter 3 we report on the actual research activities in the field of network management using the SOA paradigm. We present a brief overview of some works that are been conducted in this field.

- In Chapter 4 we present the DMS, a SOA based library for the management of network devices.

- In Chapter 5 we present a practical example of the application of the principles introduced in this thesis for the management of virtual clusters.

- In Chapter 6 conclusions and future works are reported.

1.4 Related Papers

The main contribution on which some of the chapters of this thesis are based, have already been published during the Ph.D. studies.

- The paper A general Service Oriented Approach for managing virtual machines allocation [AMG+09] deals with the discussion about the application of Service Oriented Computing principle for managing virtual machines and it has been published in the proceedings of the 24th Annual ACM Symposium on Applied Computing, Track on Service Oriented Architectures and Programming (SOAP’09).
• The paper VIDA: A Virtual Images Deployment Architecture [AGZ09] describes a new architecture for the deployment of virtual clusters using a SOA based distributed architecture and it has been published in the proceedings of the Workshop finale dei Progetti Grid del PON ”Ricerca” 2000-2006, Catania 10-12 February 2009.

• The poster Traffic Engineering in Next Generation Networks using Genetic Algorithms [OAA09] deals with the discussion about the traffic management in next generation networks and it has been published in the proceedings of the Workshop finale dei Progetti Grid del PON ”Ricerca” 2000-2006, Catania 10-12 February 2009.

• The paper SOA Based Control Plane for Virtual Clusters [AMGZ07] deals with the discussion about the services composition for the construction of a control plane for the management of virtual clusters and it has been published in the proceedings of the Workshop on Virtualization in High-Performance Cluster and Grid Computing as part of The 13th International European Conference on Parallel and Distributed Computing (Euro-Par 2007).

• The Poster Cybersar: a new computational infrastructure for research in Sardinia [AGZ+07] reports on a SOA based architecture for the management of virtual computational resources that has been developed in the Cybersar project and has been presented in the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science'07).
Chapter 2

Background

In this chapter we provide the background which is necessary to understand the remainder of the thesis. We introduce the Service Oriented Architectures’ principles and the Web Services technology with a particular regard to those specifications which are related to service design and composition. We recall the network management’s theory and how it is related to virtual computing facilities management. Furthermore we comment some necessary assumptions we made in order to develop such a work of thesis.

2.1 Service Oriented Architectures

2.1.1 Introduction to Service-Oriented Computing

Service-oriented computing represents a new generation distributed computing platform. As such, it encompasses many things, including its own design paradigm and design principles, design pattern catalogs, pattern languages, a distinct architectural model, and related concepts, technologies, and frameworks.

Service-oriented computing builds upon past distributed computing platforms and adds new design layers, governance considerations, and a vast set of preferred implementation technologies.

In the following we describe the fundamental parts of a typical service-oriented computing platform [Erl08]:

- Service-Oriented Architecture
• Service-Orientation
• Service-Oriented Solution Logic
• Services
• Service Compositions
• Service Inventory

2.1.2 Service-Oriented Architecture

SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing.

The service-oriented computing platform revolves around the service-orientation design paradigm and its relationship with service-oriented architecture. An SOA implementation can consist of a combination of technologies, products, APIs, supporting infrastructure extensions, and various other parts. Building a technology architecture around the service-oriented architectural model establishes an environment suitable for solution logic that has been designed in compliance with service-orientation design principles.

2.1.3 Service-Orientation, Services, and Service-Oriented Solution Logic

Service-orientation is a design paradigm comprised of a specific set of design principles. The application of these principles to the design of solution logic results in service-oriented solution logic. The most fundamental unit of service-oriented solution logic is the service.

Services exist as physically independent software programs with distinct design characteristics. Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Those capabilities suitable for invocation by external consumer programs are commonly expressed via a published service contract.
2.1.4  Service Compositions

A service composition is a coordinated aggregate of services. A composition of services is comparable to a traditional application in that its functional scope is usually associated with the automation of a parent business process. The consistent application of service-orientation design principles leads to the creation of services with functional contexts that are agnostic to any one business process. These agnostic services are therefore capable of participating in multiple service compositions.

2.1.5  Service Inventory

A service inventory is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise. Service inventories are typically created through top-down delivery processes that result in the definition of service blueprints. The subsequent application of service-orientation design principles and custom design standards throughout a service inventory is of great importance so as to establish a high degree of native inter-service interoperability.

2.1.6  Understanding Service-Oriented Computing Elements

In order to understand how the previously defined elements are related to each other, let’s revisit them:

- Service-oriented architecture represents a distinct form of technology architecture designed in support of service-oriented solution logic which is comprised of services and service compositions shaped by and designed in accordance with service-orientation.

- Service-orientation is a design paradigm comprised of service-orientation design principles. When applied to units of solution logic, these principles create services with distinct design characteristics that support the overall goals and vision of service-oriented computing.
Service-oriented computing represents a new generation computing platform that encompasses the service-orientation paradigm and service-oriented architecture with the ultimate goal of creating and assembling one or more service inventories.

Figure 2.1: A conceptual view of how the elements of service-oriented computing can inter-relate

The role and position of each element within a physical implementation are:

- Service-oriented solution logic is implemented as services and service compositions designed in accordance with service-orientation design principles.

- A service composition is comprised of services that have been assembled to provide the functionality required to automate a specific business task or process.

- Because service-orientation shapes many services as agnostic enterprise resources, one service may be invoked by multiple consumer programs, each of which can involve that same service in a different service composition.

- A collection of standardized services can form the basis of a service inventory that can be independently administered within its own physical deployment environment.
• Multiple business processes can be automated by the creation of service compositions that draw from a pool of existing agnostic services that reside within a service inventory.

• Service-oriented architecture is a form of technology architecture optimized in support of services, service compositions, and service inventories.

2.1.7 Service Model

Services can be categorized depending on:

• the type of logic they encapsulate

• the extent of reuse potential this logic has

• how this logic relates to existing domains within the enterprise

As a result, there are three common classifications that represent the primary service models:

• Entity Services

• Task Services

• Utility Services

Figure 2.2: Common service abstraction layers established by service models.

Entity Services

Examples of business entities include customer, employee, invoice, and claim. The entity service model represents a business-centric service that bases its functional boundary and context on one or more related business entities. It is considered a
highly reusable service because it is agnostic to most parent business processes. As a result, a single entity service can be leveraged to automate multiple parent business processes.

**Task Services**

A business service with a functional boundary directly associated with a specific parent business task or process is based on the task service model. This type of service tends to have less reuse potential and is generally positioned as the controller of a composition responsible for composing other, more process-agnostic services.

**Utility Services**

Each of the previously described service models has a very clear focus on representing business logic. However, within the realm of automation, there is not always a need to associate logic with a business model or process. In fact, it can be highly beneficial to deliberately establish a functional context that is non-business-centric. This essentially results in a distinct, technology-oriented service layer.

The utility service model accomplishes this. It is dedicated to providing reusable, cross-cutting utility functionality, such as event logging, notification, and exception handling. It is ideally application agnostic in that it can consist of a series of capabilities that draw from multiple enterprise systems and resources, while making this functionality available within a very specific processing context.

**2.1.8 Service Inventory Blueprints**

An ultimate goal of an SOA transition effort is to produce a collection of standardized services that comprise a service inventory. The inventory can be structured into layers according to the service models used, but it is the application of the service-orientation paradigm to all services that position them as valuable IT assets in full alignment with the strategic goals associated with the SOA project.

However, before any services are actually built, it is desirable to establish a conceptual blueprint of all the planned services for a given inventory. This perspective is documented in the service inventory blueprint. There are several common business and data models that, if they exist within an organization, can provide valuable
2.1. Service Oriented Architectures

input for this specification. Examples include business entity models, logical data models, canonical data and message models, ontologies, and other information architecture models.

A service inventory blueprint is also known as a service enterprise model or a service inventory model.

2.1.9 Service-Oriented Analysis and Service Modeling

To effectively deliver standardized services in support of building a service inventory, it is recommended that organizations adopt a methodology specific to SOA and consisting of structured analysis and design processes.

Within SOA projects, these processes are centered around the accurate expression of business logic through technology, which requires that business analysts play a more active role in defining the conceptual design of solution logic. This guarantees a higher degree of alignment between the documented business models and their implementation as services. Agnostic business services especially benefit from hands-on involvement of business subject matter experts, as the improved accuracy of their business representation increases their overall longevity once deployed.

Service-oriented analysis establishes a formal analysis process completed jointly by business analysts and technology architects. Service modeling, a sub-process of service-oriented analysis, produces conceptual service definitions called service candidates. Interactions through the service-oriented analysis and service modelling processes result in the gradual creation of a collection of service candidates documented as part of a service inventory blueprint.

While the collaborative relationship between business analysts and architects may not be unique to an SOA project, the nature and scope of the analysis process is.

2.1.10 Service-Oriented Analysis and Design

The service-oriented design process uses a set of predefined service candidates from the service inventory blueprint as a starting point from which they are shaped into actual physical service contracts.

When carrying out service-oriented design, a clear distinction is made between service candidates and services. The former represents a conceptual service that has
CHAPTER 2. Background

not been implemented, whereas the latter refers to a physical service. The traditional (non-standardized) means by which Web service contracts are generated results in services that continue to express the proprietary nature of what they encapsulate. Creating the Web service contract prior to development allows for standards to be applied so that the federated endpoints established by Web services are consistent and aligned. This "contract first" approach lies at the heart of service-oriented design and has inspired separate design processes for services based on different service models.

2.1.11 Goals and Benefits of Service-Oriented Computing

It is very important to establish why both vendor and end-user communities within the IT industry are going through the trouble of adopting the service-oriented computing platform and embracing all of the change that comes with it. The vision behind service-oriented computing is extremely ambitious and therefore also very attractive to any organization interested in truly improving the effectiveness of its IT enterprise. A set of common goals and benefits has emerged to form this vision. These establish a target state for an enterprise that sucessfully adopts service-orientation:

- Increased Intrinsic Interoperability.
- Increased Federation.
- Increased Vendor Diversification Options.
- Increased Business and Technology Domain Alignment.
- Increased ROI.
- Increased Organizational Agility.
- Reduced IT Burden.

2.2 Web Services

SOA is an architectural model that is agnostic to any one technology platform. By understanding this, an enterprise is given the freedom to continually pursue
the strategic goals associated with service-oriented computing by leveraging future technology advancements. In the current marketplace, the technology platform most associated with the realization of SOA is Web Services.

The Web Services platform is defined through a number of industry standards and is characterized by a plethora of XML [xml] based specifications which deal with different aspects of such a kind of technology. Some distinctions can be done in order to categorize the specifications into different layers. Here, we distinguish three main specification layers:

- Base-level specifications
- Quality of Service specifications
- Composition specification

It is worth noting that other specifications there exist but, for the sake of brevity, here we discuss only the most important ones which allow us to give a comprehensive overview of the technology.

### 2.2.1 Overview

A typical Web service is comprised of the following:

- A physically decoupled technical service contract consisting of a WSDL definition, an XML schema definition, and possibly a WS-Policy definition. This service contract exposes public functions (called operations) and is therefore comparable to a traditional application programming interface (API).

- A body of programming logic. This logic may be custom-developed for the Web service, or it may exist as legacy logic that is being wrapped by a Web service in order for its functionality to be made available via Web services communication standards. In the case that logic is custom-developed, it generally is created as components and is referred to as the core service logic (or business logic).

- Message processing logic that exists as a combination of papers, processors, and service agents. Much of this logic is provided by the runtime environment, but it can also be customized. The programs that carry out message-related
processing are primarily event-driven and therefore can intercept a message subsequent to transmission or prior to receipt. It is common for multiple message processing programs to be invoked with every message exchange.

A Web service can be associated with temporary roles, depending on its utilization at runtime. For example, it acts as a service provider when it receives and responds to request messages, but can also assume the role of service consumer when it is required to issue request messages to other Web services. When Web services are positioned within service compositions, it is common for them to transition through service provider and service consumer roles. Note also that regular programs, components, and legacy systems can also act as Web service consumers as long as they are able to communicate using Web service standards.

The popularity of Web services preceded that of service-oriented computing. As a result, their initial use was primarily within traditional distributed solutions wherein
they were most commonly used to facilitate point-to-point integration channels. As
the maturity and adoption of web services standards increased, so did the scope of
their utilization.

With service-oriented computing comes a distinct architectural model that has been
positioned by the vendor community as one that can fully leverage the open inter-
operability potential of Web services, especially when individual services are consis-
tently shaped by service-orientation. For example, when exposing reusable logic
as Web services, the reuse potential is significantly increased. Because service logic
can now be accessed via a vendor-neutral communications framework, it becomes
available to a wider range of service consumer programs.

Additionally, the fact that Web services provide a communications framework based
on physically decoupled contracts allows each service contract to be fully standard-
ized independently from its implementation. This facilitates a potentially high level
of service abstraction while providing the opportunity to fully decouple the ser-
vice from any proprietary implementation details. All this characteristics are desir-
able when pursuing key principles, such as Standardized Service Contracts, Service
Reusability, Service Loose Coupling, Service Abstraction, and Service Composabil-
ity.

For example, transformation avoidance is a key of Standardized Service Contracts.
This principle advocates the standardization of the data model expressed by the
service contract so as increase intrinsic interoperability by reducing the need for
transformation technologies. Services delivered via disparate component platforms
still require the transformation of technology regardless of whether data types are
standardized. Service expressed through Web service contracts have the potential
to avoid transformation altogether.

**Base-Level specifications**

In this category there are three main specifications which characterize Web Services:
WSDL, SOAP, UDDI.

- **WSDL:** *Web Service Description Language* [wsd]. This specification deals with
  the description language which allows for the standard definition of a Web Ser-
  vice interface. It is a fundamental specification which fixes the basic commu-
  nication primitives, the *operations*, exploited by a Web Service for exchanging
CHAPTER 2. Background

messages. There are four kind of operations: One-Way, Request-Response, Notification and Solicit-Response. In a One-Way operation a message is received, in a Request-Response operation a message is received and a response is sent to the invoker, in a Notification a message is sent and, finally, in a Solicit-response a message is sent to another Web Service and a response is received. Since WSDL specification is strictly related to service design, we will deeply discuss it in the following.

- **SOAP**: Simple Object Access Protocol [soa]. This specification describes the basic format of an exchanged message between two Web Services. A message is an XML document composed by two different parts: the header and the body. The former part is an optional part which can contain special control information which characterize the communication such as, e.g., security references for retrieving cryptographic keys or reliability tags for guaranteeing message delivery, whereas the latter part contains the information to be communicated. Since the SOAP specification does not directly deal with service design and composition, for the sake of this thesis, we do not present in a more detailed way such a kind of specification. The reader who is interested in this topic may consult [soa].

- **UDDI**: Universal Description Discovery and Integration [udd]. This specification deals with the programming interface exhibited by a discovery registry which is a particular kind of service that allows for the retrieving of Web Service depending on their functionalities. The UDDI specification introduces the concept of dynamic discovery of a Web Service. At run-time a Web Service can be potentially discovered by performing a query on the discovery registry. This thesis does not explicitly deal with dynamic discovery. For this reason, the UDDI specifications will not be commented in detail.

**QoS specifications**

This group of specifications are based upon the basic ones and they deal with other characteristics which can be implemented in order to allow the service to supply extra functionalities such as, e.g., security or transactional aspects. It is out of the scope of this thesis to supply an exhaustive list of such a kind of specifications and,
in the following, we list only the most important ones.

- **Specifications which deal with security aspects**: security is achieved by extending the SOAP specifications. The header part of a message indeed, can be enriched by introducing nonce and references to cryptographic keys. WS-Security [wssb] deals with the security aspects of a message exchange between two dialoguers. WS-Trust [wsta] and WS-SecureConversation [wssa] deal with the security aspects in a domain where security credentials must be distributed and trusted by participants.

- **Specifications which deal with transactional aspects**: transactions are fundamental for designing e-business applications. WS-Coordination [wsc] defines a Web Services framework where different participants can interact by exploiting specific transaction protocols which are defined within WS-Transactions [wstb] specifications.

- **Specifications which deal with message reliability**: In general, the HTTP protocol, which usually underlies the SOAP one, is not sufficient for guaranteeing the message reliability. The WS-Reliability specification [wsr] deals with such a kind of issues.

### Composition specifications

This group of specifications deal with the composition issue which allows for definition of systems where several services can interact each other in order to fulfill a specific task.

- **WS-BPEL** [wsb]. It is an Oasis specification where an XML-based language for dealing with Web Services orchestration is defined. WS-BPEL is the evolution of BPEL4WS [CGK+02] which was a first attempt, developed by Microsoft, IBM and other software industries, to define an orchestration language which merges together some features of WSFL (Web Service Flow Language) by IBM [Ley01] and XLANG by Microsoft [Tha01]. Such a kind of specification will be discussed deeply in the following.

- **Jolie** [MGZ07]. It is an open source project whose syntax is C/Java-like in order to provide a more programmer friendly development environment dif-
ferently from WS-BPEL which has a less human readable XML-based syntax. The peculiar and original characteristic of JOLIE is that it combines the solid mathematical basis provided by SOCK, a formal calculus language [GLG+06], with a programmer friendly development and execution environment.

- **WS-CDL** [Con04]. It is a W3C specification which defines an XML-based language for dealing with Web Services choreography. The development of such a language is followed by both industry and academic experts.

- **WS-Addressing** [ws-]. This specification deals with the definition of all the necessary information which represent a conversation end-point and it indirectly allows for the end-point reference exchange. Such a feature, from a system design point of view, allows for the design of Web Services which receives at run-time the address of a dialoguer by allowing a dynamically composition of services during their execution. Here, we do not comment deeply WS-Addressing specification because we are not interested in modelling all the communication details of Web Services message exchange.

### 2.2.2 WSDL

The WSDL specification deals with the definition of a Web Service interface by introducing an XML-based language which allows for the description of the access points of the service, called *operations*. Each Web Service must public its own WSDL document in order to allow other invokers to access its operations. In the following we present the structure of a WSDL document:

```xml
<wSDL:definitions name="nmtoken"? targetNamespace="uri"?>
  <wSDL:import namespace="uri" location="uri"/>*?
  <wSDL:documentation .... /> *?
  <wSDL:types>? 
    <wSDL:documentation .... /> *?
    <xsd:schema .... />*?
    <--- extensibility element --> *
  </wSDL:types>

  <wSDL:message name="nmtoken"> *
    <wSDL:documentation .... />*?
```
2.2. Web Services

```xml
<part name="nmtoken" element="qname" type="qname"/>
</wsdl:message>

<wsdl:portType name="nmtoken">
  <wsdl:documentation .... />?
  <wsdl:operation name="nmtoken">
    <wsdl:documentation .... />?
    <wsdl:input name="nmtoken" message="qname">?
      <wsdl:documentation .... />?
    </wsdl:input>
    <wsdl:output name="nmtoken" message="qname">?
      <wsdl:documentation .... />?
    </wsdl:output>
    <wsdl:operation name="nmtoken">?
      <wsdl:documentation .... />?
    </wsdl:operation>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="nmtoken" type="qname">
  <wsdl:documentation .... />?
  <wsdl:operation name="nmtoken">
    <wsdl:documentation .... />?
    <wsdl:input>?
      <wsdl:documentation .... />?
    </wsdl:input>
    <wsdl:output>?
      <wsdl:documentation .... />?
    </wsdl:output>
    <wsdl:fault name="nmtoken">?
      <wsdl:documentation .... />?
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```
• Tag `<definitions>` is the main tag of a WSDL document. It includes all the other tags of the document.

• Tag `<import>` allows for the inclusion of other WSDL documents. WSDL documents indeed, can be obtained by composing other documents.

• Tag `<types>` allows for the definition of data types by means of the definition of XML-Schemas. Such a kind of types can be exploited for defining the content of the exchanged messages.

• Tag `<message>` allows for the definition of the exchanged messages. Each message exchanged by the Web Service indeed, must be declared by means of this tag and it is formed by the so-called part. Each part has a specific content whose type can be defined by exploiting the XML-Schema declared in the previous tag.

• Tag `<portType>` describes an abstract collection of operations which will be deployed at the same location under the same protocol. The operation describes the access points exhibited by the Web Service. There are four kinds of operations:
  
  – *One-Way*: the Web Service receives a message
  
  – *Request-Response*: the Web Service receives a message and sends a response to the invoker
  
  – *Notification*: the Web Service sends a message to another service.
2.2. Web Services

- **Solicit-Response.** the Web Service sends a request message to another service and waits for its response.

Usually, the Notification and the Solicit-Response are not declared within a WSDL document. For each operation the input and the output messages are defined by referring to the ones declared within the tags `<message>` . Moreover, it is possible to define a fault message in the case an application error occurs.

- **Tag `<binding>`** allows for the binding of a specific protocol to each portType. For each message of each portType indeed, it is possible to define the message exchange protocol to follow. Usually, SOAP over HTTP is exploited but there are no restrictions about the protocol to use and, moreover, WSDL can be extended in order to introduce new protocols.

- **Tag `<service>`** allows for the declaration of the Web Services ports that are the real access points of the Web Service. Each port has its own portType, its binding and it is deployed to a specific address.

By specification, WSDL can be extended depending on the application context, some other tags, indeed, can be defined and added in order to enrich the interface description of a Web Service. This is the case, for example, of the WS-BPEL specification that introduces the `<partnerLinkType>` tag which will be discussed in detail in Section 2.2.3.

2.2.3 WS-BPEL

It is an Oasis specification where an XML-based language for dealing with Web Services orchestration is defined. WS-BPEL is the evolution of BPEL4WS [CGK+02] which was a first attempt, developed by Microsoft, IBM and other software industries, to define an orchestration language which merges together some features of WSFL (Web Service Flow Language) by IBM [Ley01] and XLANG by Microsoft [Tha01]. A WS-BPEL process is called *business process* as it describes the behaviour of an orchestrator by means of workflow constructs and communication primitives. A business process can actually be executed by the so-called *orchestrator engines*. An orchestrator engine is an execution environment which takes in
input a WS-BPEL specification and then animates it. In the following, we present the main characteristics of the WS-BPEL language, the reader who is interested in WS-BPEL details may consult ([wsb]).

The language

A WS-BPEL business process is formed by two main tags: `<definitions>` and `<process>`. The former represents the WSDL document of the business process which contains all the portTypes exhibited by it and all the portTypes exhibited by the other services it will interact with. The latter contains all the activities that have to be executed by the business process. In the following we discuss the two main tags by highlighting the most important features.

`<definitions>` Tag `<definitions>` contains the WSDL document related to the WS-BPEL business process and its document structure follows that defined within the WSDL specification. The WSDL of a WS-BPEL process is extended with some other tags in order to deal with specific aspects of the business process. These tags are: `<partnerLinkType>`, `<property>` and `<propertyAlias>`.

The tag `<partnerLinkType>` is exploited for describing all the peer-to-peer relationships where the business process is involved. In general, we call a `partner` a service which interacts with the business process. A business process can have several partners and, for each of them, can have several interactions. In particular, the tag `<partnerLinkType>` allows for the description of a two dialoguers relationship where each partnerLinkType is characterized by two roles that participate within the relationship and, for each of them, a portType where the message exchanges are performed is declared. In the following, we present a partnerLinkType example extracted from the specifications:

```xml
<plnk:partnerLinkType name="invoicingLT">
  <plnk:role name="invoiceService"
    portType="pos:computePricePT"/>
  <plnk:role name="invoiceRequester"
    portType="pos.invoiceCallbackPT"/>
</plnk:partnerLinkType>
```

Here, a partnerLinkType, named `invoicingLT`, is defined with two roles: `invoic-
2.2. Web Services

cService and invoiceRequester. The former role is joined with the portType computerPricePT whereas the latter is joined with the portType invoiceCallbackPT. In Fig. 2.6 we abstractly represent the partnerLinkType of the example where the dotted double arrow represents the relationship between the two roles. Such a kind of relationship is defined on the two portTypes represented as a black rectangle. Intuitively, the partnerLinkType defines an abstract relationship between two ports of two different dialoguers. Since the partnerLinkType definition does not allow for the joining of specific services to the roles, each business process must declare the role it plays within each partnerLinkType it is involved in. As we will see in the next section, such a declaration is performed within the tag <partnerLink> contained within the tag <process>. It is worth noting that, in a case where one of the two dialoguers is a priori unknown, a partnerLinkType may contain only one role. This is the case of a business process, for example, which supplies a service by means of a single Request-Response operation. Such a kind of service indeed, does not need to know the portType of the invoker because its Request-Response operation is sufficient to successfully complete the interaction with it. On the contrary, it is fundamental to declare the portType of both the participants when different messages are exchanged during the execution of the service (e.g. a callback configuration between two services).

The tags <property> and <propertyAlias> allow for the management of subparts of message data structures which will be used for correlating the incoming messages. The former tag allows for the definition of a unique name, a property, for

\footnote{For the sake of brevity we do not report the portType definitions.}

\footnote{The correlation mechanism will be presented in the following.}
an XML Schema type whereas the latter tag allows for the association of a subpart of a message with a property. When a propertyAlias is defined on a subpart of a message, within the business process it is possible to refer to that subpart by using the corresponding property. Let us consider the following example where the part identification of the message taxpayerInfoMsg is joined to the property taxpayerNumber by means of the propertyAlias tag.

```
<wSDL:definitions  ...>
  ...
  <wSDL:message name="taxpayerInfoMsg">
    <wSDL:part name="identification" element="txtyp:taxPayerInfoElem"/>
  </wSDL:message>
  ...
  <vprop:property name="taxpayerNumber" type="txtyp:SSN" />
  ...
  <vprop:propertyAlias propertyName="tns:taxpayerNumber"
    messageType="txmsg:taxpayerInfoMsg"
    part="identification">
  ...
  </vprop:propertyAlias>
</wSDL:definitions>
```

It is worth noting, that the type of the message part must be coherent with the type defined within the property.

**<process>** The tag `<process>` allows for the definition of the activities executed by the business process. The structure of a process tag is represented in the following where, for the sake of brevity, some details are omitted:

```
<process name="NCName"  ...>
  ...
  <partnerLinks>?
    <partnerLink name="NCName"
      partnerLinkType="QName"
      myRole="NCName" ?
      partnerRole="NCName" ?  ...>+
  </partnerLink>
</partnerLinks>
```
...<variables>
  <variable name="BPELVariableName" .../>
</variable>
</variables>

<correlationSets/>
  <correlationSet name="NCName" properties="QName-list"/>
</correlationSets>

<faultHandlers/>
  ...
</faultHandlers>

<eventHandlers/>
  ...
</event Handlers>

activity

In the following we comment each tag contained within the process one.

<partnerLinks> For each partnerLinkType where the business process is involved, it has to be declared which role is enroled by the business process and which by its partner. The <partnerLinks> tag allows for such a kind of declaration by means of the attributes myRole and partnerRole where the former specifies the role of the business process and the latter that of the partner. If we consider the partnerLinkType example presented in the previous section, the related partnerLink for an orchestrator which enroles the invoiceRequester role can be defined as follows:

<partnerLinks>
  <partnerLink name=" invoicing"
    partnerLinkType="lns: invoicingLT"
    myRole=" invoiceRequester"
    partnerRole=" invoiceService" />
</partnerLinks>

Each partner within a partnerLink is joined with an endpoint reference which
is the actual means for identifying an orchestrator and interacting with it. The endpoint reference association is differently managed depending on the engine which animates the specification.

<variables> The tag <variables> allows for the definition of the variables exploited within the business process. Variables will be exploited for storing the received messages and for manipulating data. In the following we present an example of variable declaration where the attribute messageType refers to a message defined within the WSDL and specifies that the type of the variable must be the same of that of the declared message:

```xml
<variables>
  <variable name="PO"
    messageType="lns:POMessage" />
  <variable name="Invoice"
    messageType="lns:InvMessage" />
  <variable name="shippingRequest"
    messageType="lns:shippingRequestMessage" />
  <variable name="shippingInfo"
    messageType="lns:shippingInfoMessage" />
  <variable name="shippingSchedule"
    messageType="lns:scheduleMessage" />
</variables>
```

<correlationSets> Different instances of a business process can be executed concurrently on the same engine. Each instance executes the same business process but with a different set of data. Usually, a business process instance is initiated by a message reception and it is identified by a particular set of the received data. The data which allows for the identification of each instance are defined within the so-called correlation set. Let us consider, for example, the case of a business process which starts its execution by receiving a message that contains the nickname of a user. Moreover, let us assume that the nickname is declared within the correlation set of the business process and that the nicknames univocally identify a user. If such a kind of business process is concurrently invoked by two users which have different nicknames as, for example, Micky Mouse and Homer Simpson, two instances will be created and each instance will be identified by the corresponding nickname. The tag <correlationSets> allows for the definition of all the data which are used for identifying a specific instance of the business process. In the following we present
an example of correlation sets declaration:

```xml
<correlationSets xmlns:cor="http://example.com/supplyCorrelation">
  <correlationSet name="PurchaseOrder"
  properties="cor:customerID cor:orderNumber"/>
  <correlationSet name="Invoice"
  properties="cor:vendorID cor:invoiceNumber"/>
</correlationSets>
```

It is worth noting that, in order to define correlation sets, WS-BPEL exploits the property and the propertyAlias constructs described in the previous section. In the example above, two correlation sets are defined: the former is named *PurchaseOrder* and it is joined to the properties *cor:customerID* and *cor:orderNumber* whereas the latter is named *Invoice* and it is joined with the properties *cor:vendorID* and *cor:invoiceNumber*.

**<faultHandlers> and <eventHandlers>** These tags allow for the definition of the handlers which manages faults and events. Faults can be logically generated during the process execution or received within a message exchange whereas some event reactions can be programmed when a message reception or an alarm occur. Since faults and events are out of the topic of this thesis, here we do not present such a kind of constructs in detail.

**activity** Each business process has an initial activity and a terminating one. In the following we list all the activities and we present some usage examples for the ones which are relevant to the end of the understanding of this thesis.

- Activities which deal with communication
  - *receive*: it allows the business process to receive a message on an One-Way or Request-Response operation. Within a receive it is possible to define a correlation set for identifying the right instance to which route the incoming message. In the following we present an example of the receive activity:

```xml
<receive partnerLink="purchasing"
  portType="lns:purchaseOrderPT"
  operation="sendPurchaseOrder"
  variable="PO"
/>
createInstance="yes">
</receive>

It is worth noting that the partnerLink, the portType and the operation on which the message reception is performed are declared. Moreover, the variable (PO) where the message content will be stored is defined. The attribute createInstance set to yes means that a new business process instance is created when the message is received.

- reply: it allows the business process to send a reply message in a Request-Response message exchange. The reply activities must be logically executed after a receive activity declared on the same operation. In the following an example of a reply activity is reported:

```xml
<reply partnerLink="purchasing"
      portType="lns:purchaseOrderPT"
      operation="sendPurchaseOrder"
      variable="Invoice">
</reply>
```

In this case, the attribute variable identifies the variable from which the message content will be taken.

- invoke: it allows the business process to invoke a One-Way or a Request-Response operation of another business process. In the case of a Request-Response invocation the invoke activity will be blocked until the response message will be received. In the following a Request-Response invocation is presented.

```xml
<invoke partnerLink="shipping"
       portType="lns:shippingPT"
       operation="requestShipping"
       inputVariable="shippingRequest"
       outputVariable="shippingInfo">
</invoke>
```

It is worth noting that the attributes inputVariable and outputVariable specify the variables where the incoming message will be stored and from which will be taken the outcoming one respectively.

In general, within a communication primitive a correlation set may be defined in order to route a received message to the right instance depending on the
2.2. Web Services

contained values. In the following, we present an example where a receive activity is correlated with the correlation set \textit{auctionIdentification}:

\begin{verbatim}
<receive name="acceptSellerInformation"
partnerLink="seller"
portType="as:sellerPT"
operation="submit"
variable="sellerData"
createInstance="yes">
<correlations>
  <correlation set="auctionIdentification"/>
</correlations>
</receive>
\end{verbatim}

- Activities which deal with computational aspects:
  - \textit{assign}: it allows the business process to assign a value or an expression result to a variable furthermore, the assign activity can also be used to assign an endpoint reference to a partnerLink. In the following we present an assign example where the tag \textit{<copy>} specifies that some values must be copied from a source specified within the tag \textit{<from>} to the target defined within the tag \textit{<to>}. In this example, the source is an endpoint reference whereas the target is a partnerLink.

\begin{verbatim}
<assign>
  <copy>
    <from>
      <literal>
        <sref:service-ref>
          <addr:EndpointReference>
            <addr:Address>
              http://example.com/auction/RegistrationService/
            </addr:Address>
            <addr:ServiceName>
              as:RegistrationService
            </addr:ServiceName>
          </addr:EndpointReference>
        </sref:service-ref>
      </from>
      <to partnerLink="auctionRegistrationService" />
    </to>
  </copy>
</assign>
\end{verbatim}
– validate: it allows the business process to check the validation of the values contained within a variable w.r.t. the joined WSDL data definition.

- Activities which deal with activity composition:

  – sequence: it allows for the sequential composition of activities. In the following we present a sequence example where an invoke activity is performed after two assign activities:

    `<sequence>
      <assign>
        <copy>
          <from>$sellerData.endpointReference</from>
          <to partnerLink="seller" />
        </copy>
        <copy>
          <from>
            <literal>Thank you!</literal>
          </from>
          <to>$sellerAnswerData.thankYouText</to>
        </copy>
      </assign>
      <invoke name="respondToSeller"
        partnerLink="seller"
        portType="as:sellerAnswerPT"
        operation="answer"
        inputVariable="sellerAnswerData" />
    </sequence>`

  – flow: it allows for the parallel composition of activities. In the following we present an example where a sequence of an invoke and a receive activity are composed in parallel with a sequence of two invoke activities.

    `<flow>
      <sequence>
        <invoke partnerLink="shipping"
          portType="lns:shippingPT"
          operation="requestShipping"
          inputVariable="shippingRequest"```
It is worth noting that the activities programmed to be executed in parallel can be synchronized by means of internal synchronization signals expressed by means of the tag `<links>`. The tags `<source>` and `<target>` allows for the specification of the activity which sends the synchronizing signal and the activity which receives it respectively. In the following we present the same example above where the signal `ship-to-invoice` is introduced. The source activity is the first invoke activity of the first sequence whereas the target is the last invoke activity of the second sequence.
It is important to highlight the fact that the tags <source> and <target> allow also for the specification of an inner guard (a boolean expression) which can block or not the sending and the receiving of the synchronizing signal.

– pick: it allows for the representation of non-deterministic choice among a set of events. Only the selected event will be executed whereas the other will be discarded. The events that can be programmed within a pick activity are the reception of a message, which resembles a receive activity, or an internal time alarm. In the following, we present an example where two message receptions and an alarm are programmed within a pick activity:

<pick>
    <onMessage partnerLink="buyer"
– if: it allows for the selection of exactly one activity among a collection of choices. It represents the usual if then else construct. An example where the if activity is used follows:

```xml
<if>
  <condition>
    bpel:getProperty('stockResult',
        'inventory:level') > 100
  </condition>
  <flow>
    <!-- perform fulfillment work -->
  </flow>
  <elseif>
    <condition>
      bpel:getProperty('stockResult',
        'inventory:level') >= 0
    </condition>
    <!-- perform elseif activities -->
  </elseif>
  <else>
    <!-- perform else activities -->
  </else>
</if>
```

– while: it allows for the expression of repeated activities depending on a
condition. The condition is evaluated before the execution of the inner activities. A usage example follows:

```xml
<while>
  <condition>$orderDetails > 100</condition>
  <scope>...</scope>
</while>
```

- `repeatUntil`: as the while activity it allows for the expression of repeated activities. The condition is evaluated after the first execution of the inner activities.
- `forEach`: it allows for the repetition of some activities. The inner activities are repeated a specified number of times.

- Activities which deal with activity scoping
  - `scope`: it allows for the definition of a nested activity scope where different partnerLinks, messageExchanges, variables, correlationSets, fault-Handlers, compensationHandler, terminationHandler can be defined.

- Activities which deal with faults
  - `exit`: it allows for the immediate termination a business process instance.
  - `throw`: it allows for the generation of a fault within the business process
  - `rethrow`: it allows for the re-generation of a fault within the fault handler it is processing it.
  - `compensate`: it allows the business process to start a compensation activity within all the inner scopes.
  - `compensateScope`: it allows the business process to start a compensation activity on a specific inner scope.

- Others:
  - `wait`: it allows the business process to wait for a specified amount of time.
  - `empty`: no activities are performed when this activity is executed.
  - `extensionActivity`: it allows for the extension of WS-BPEL activities with new kind of activities.
Abstract process

A WS-BPEL business process can be programmed as a so-called abstract process. An abstract process cannot be executed by an engine but it is exploited for representing a sort of behavioural skeleton of the business process. Such a kind of processes are usually exploited for supplying a view of a business process where only the observable actions are shown. Some opaque constructs are introduced for replacing executable statements with a non-observable ones. In particular it is possible to have an opaque construct for:

- an activity: an activity can be replaced with an opaque one in order to not specify its behaviour.
- an expression: an expression can be replaced with an opaque value. Such a kind of feature implicitly introduces non-determinism within choice constructs where conditions are evaluated.
- an attribute: tag attributes can be replaced with an opaque value.

In the following we present an abstract process example where all the three kind of opaque constructs are used:

```xml
<process ...>
<partnerLinks>
  <partnerLink name="homeInfoVerifier" partnerLinkType="#opaque" myRole="#opaque" partnerRole="#opaque">
  <documentation>
    Pick an appraisal request from one of 3 customer referral channels.
  </documentation>
</partnerLink>
</partnerLinks>
<variables>
  <variable name="commonRequestVar" element="#opaque" />
</variables>
<sequence>
  <opaqueActivity template:createInstance="yes">
    <documentation>
      Pick an appraisal request from one of 3 customer referral channels.
    </documentation>
  </opaqueActivity>
  <assign>
```
WS-BPEL engines

WS-BPEL engines architecture is not standardized and the implementations that exist are the result of a free interpretation of the WS-BPEL specification made by the different producers. A detailed description of these products and a comparison of the different features they supply, is out of the scope of this thesis. It is not clear, for example, if WS-BPEL orchestrators designed with different tools are compatible each other. Here we are interested, for the sake of completeness, to list the most credited ones and, in particular, we would to focus on their architecture. The documentation about this topic lacks in details and it is often not available. For this reason, in the following we comment a rough architecture extracted from the web pages of the Active BPEL project [act] in order to catch the basic characteristics an engine is based upon.

- Open source projects:
  - Active BPEL [act]. It implements an engine able to animate WS-BPEL processes which follows the specifications. It is written in Java and it
is also equipped with a visual support for designing BPEL processes by following a graphical approach.

- *Apache ODE* [Apaa]. It is a Java open source project which implements WS-BPEL specifications. It does not supply a visual support for designing processes.

- Commercial products:
  - *Oracle BPEL Process Manager* [ora]. It is a proprietary product developed by Oracle which supplies an execution environment for native WS-BPEL code and a visual tool for designing processes in a graphical way.
  - *IBM WebSphere* [web]. It is a complete infrastructure software for integrating web applications. It also supplies a WS-BPEL engine for executing Web Services orchestrators.

In [act] a very short description of the Active BPEL architecture is provided. Here, we try to summarize the basic concepts it is based upon:

- An active BPEL process is composed by different activities which correspond to those defined within the WS-BPEL specifications.

- Activities can be *Basic activities* (e.g. receive, reply, invoke) or *Structured activities* (e.g. sequence, flow, pick)

- Activities are joined by links and they have their own state which describe their current behaviour (e.g. inactive, ready-to-execute, executing)

- Each activity is defined within a scope which contains the values of the variables, fault handlers, event handlers, compensation handlers, etc. Each WS-BPEL process has a global scope as defined within the specifications.

- An activity is a *start activity* if it initiates a new active BPEL process. When a start activity is triggered a new process is created.

- The executing receive activities are queued in order to wait for an incoming message.

---

3The reader who is interested in details about this topic may consult [act].
• The receive queue contains also the incoming messages received from other services that did not match at the moment of the reception. These messages are queued until a timeout period passes.

• The engine dispatches incoming messages to the correct process instance.

• If there is correlation data, the engine tries to find the correct instance that matches the correlation data. On the contrary, if there is no correlation data and the request matches a start activity, a new process instance is created. In Fig. 2.7 we report the request dispatch flowchart provided by the web pages documentation of the project which is essentially self-descriptive.

![Figure 2.7: Request Dispatch Flowchart of the Active Bpel engine](image)

2.2.4 Jolie

JOLIE is an implementation of the service behaviour part of the SOCK language [GLG+06] and it provides a C-like syntax for designing orchestrator services. A C-like syntax makes the language intuitive and easy to learn for a programmer.

\[^4\]Such a feature is at the basis of the asynchronous communication which characterize Web Services. We discuss this topic in the next section.
2.2. Web Services

customed to it. In the following we introduce some basics of the JOLIE language, except expression and condition syntaxes which are similar to that of C language.

**Identifiers**

An identifier (often abbreviated to id) is an unambiguous name stored in the orchestrator shared memory which identifies a location, an operation, a variable or a link. An identifier must match the following regular expression: Some JOLIE statements require that the programmer provides a list of identifiers, which is formed by identifiers separated by commas (as identifier1, identifier2, a, b, c). In the following, we refer to the list of identifiers by using the name id list.

**Program structure**

A JOLIE program structure is represented by the following grammar:

```
program ::= 
locations { Locations-definition } 
operations { Operations-declaration } 
variables { Variables-declaration } 
links { Links-declaration } 
definition 
main { Process } 
definition 
definition := define id { Process }
```

where we represent non-terminal symbols in italic and the Kleene star represents a zero or more time repetition. For the sake of clarity, the non-terminals Locations-definition, Operations-declaration, Variables-declaration, Links-declaration and Process are separately explained in the following.

**Locations**

JOLIE communications are socket based: an orchestrator waits for messages on a network port (the default is 25552 ). In order to communicate with another orchestrator it is fundamental to know its hostname (or ip address) and the port it
is listening to: these information are stored in a location. A location definition joins an identifier to a hostname and a port. The non-terminal follows:

| Locations-definition := id= hostname:port |

where we do not define the hostname and the port non-terminals which must be intended as a representation of any hostname and any port respectively. In the following we present program fragment which shows a possible location declaration:

```java
locations {
    localUri = "localhost:2555",
    googleUri = "www.google.com:80",
    ipUri = "192.168.0.1:2556"
}
```

**Operations**

The operations represent the way a JOLIE orchestrator exploits for interacting with other orchestrators. We distinguish two types of operations:

- Input operations.
- Output operations.

The former represent the access points an orchestrator offers to communicate with it, whereas the latter are used to invoke input operations of another orchestrator. We distinguish two groups of input operations: One-Way and Request-Response. A One-Way operation simply waits for a message, while a Request-Response operation waits for a message, executes a code block and then sends a response message to the invoker. As far as output operations are concerned they can be a Notification or a Solicit-Response operation. The former is used to invoke a One-Way operation of another orchestrator, sending a message to it, while the latter is used to invoke a Request-Response operation. It is worth noting that a Solicit-Response operation, after sending the request message, is blocked until it receives the response one from the invoked service. The non-terminal follows:

| Operations-declaration := OneWay:id list |
| RequestResponse:id list |
By definition, input operations expect a list of identifiers, while the output ones expect a list of pairs \( id=id \) (we have identified such a list by using the notation \( id-assign \ list \)). As far as the output operations are concerned we distinguish between the operation name used within the orchestrator and the bound operation name of the invoked one. In a pair \( idA=idB \), \( idA \) represents the internal operation name whereas \( idB \) the bound name of the external one to be invoked. Such a language characteristic allows us to decouple the orchestrator code from the external operation name binding. In the following a program fragment shows an example of operation declaration.

```
operations
{
  OneWay:
    ow1
  RequestResponse:
    rr1, rr2
  Notification:
    n1 = serverOneWay1,
    n2 = serverOneWay2,
    n3 = serverOneWay3
  SolicitResponse:
    sr1 = serverRequestResponse1
}
```

### Variables

JOLIE variables are typeless. Implicit supported types are integers and strings. The variables declaration non-terminal requires only a list of identifiers which represent the shared memory variables. The definition follows:

```
Variables-declaration := id list
```
In the following example three variables, a, b and c are declared:

```plaintext
variables {
    a, b, c
}
```

**Links**

Links model the SOCK signals and are used for internal parallel processes synchronization. As for variables the links declaration non-terminal requires only a list of identifiers where the ids will represent internal links used for synchronization purposes.

```plaintext
Links-declaration:= id list
```

In the following example two links, link1 and link2, are declared:

```plaintext
links {
    link1, link2
}
```

**Definitions**

Definitions allows to define a procedure which will be callable by another one by exploiting the call statement. Each definition joins an identifier to a Process. Syntactically, a Process is a piece of code composed by JOLIE statements. Informally, the process defined within a definition can be viewed as the body of a C function. In the following we report an example where the procedure calc is defined. Its body is composed by two assignments on variables a and b:

```plaintext
define calc {
    a = 5*2−9;;
    b = a * (2−1)
}
```
2.2. Web Services

Main

The main block allows to define the process which will be run at the start of the program execution. Informally, it is comparable to the main function of a C program. In the following we report an example where, within the main procedure, the string "Hello, world!" is printed out on the user console and there is a call at the procedure calc defined above:

```plaintext
main
{
    out( "Hello, world!" );
    call(calc)
}
```

Statements

This paragraph shows a brief survey of JOLIE statements.

Program control flow statements

- `call(id)`: calls and executes the procedure which has been defined with the given identifier.
- `while(condition) {...}`: loop statement

Operation statements

- `id<id list>`: waits for a message for the OneWay operation declared in the operations block as id, and stores its values in the id list variables.
- `id<id list> <id list> (Process)`: waits for a message for the RequestResponse operation id, stores its values in the first id list variables, executes the code block Process and sends a response message containing the values of the second id list variables.
- `id@id<id list>`: uses the Notification operation represented by the first id to send a message which contains the values of the id list variables, to the orchestrator located at the second id. The second id can be a location declared
in the locations block, or a variable containing a string that can be evaluated as a location. It is worth noting that such a feature allows to implement the location mobility. It is possible, indeed, to receive a location which can be exploited for performing a Notification or a Solicit-Response.

- **id@id<id list> <id list>:** uses the SolicitResponse operation represented by the first id to send a message which contains the values of the first id list variables, to the orchestrator located by the second id (which can be, as for the Notification, a location or a variable). Once the message is sent, it waits for a response message from the invoked Request-Response and stores its values in the second id list variables.

### Synchronizing statements

- **linkIn( id ):** linkIn and linkOut are used for parallel processes synchronization and must be always considered together. In particular the linkIn waits for a linkOut trigger on the same internal link identified by id. In case there are already one or more linkOut processes triggering for the same internal link, it synchronizes itself with one of them by following a non-deterministic policy.

- **linkOut( id ):** triggers for a linkIn synchronization on the same internal link identified by id. In case there are already one or more linkIn processes waiting for the same internal link, it synchronizes itself with one of them by following a non-deterministic policy.

### Console input/output statements

- **in( variable id ):** waits for a console user input and stores it in the given variable.

- **out( expression ):** writes the evaluation of the given expression on the console (note that a variable can be considered as an expression).

### Others

- **sleep( n ):** makes the current process sleeping for n milliseconds where n is a natural.
2.2. Web Services


Statements composers

As the SOCK calculus, JOLIE provides three ways to compose statements: sequence, parallelism and non-deterministic choice.

**Sequence** Sequences are composed by exploiting the ;; operator. Let x1, x2, . . . , xn1, xn be statements. Then, the sequential composition

\[ x_1 ; ; x_2 ; ; \ldots ; ; x_{n-1} ; ; x_n \]

executes x1 and waits for it to terminate, then executes x2 and waits for it to terminate and continues with this behaviour until it reaches the end of the sequence.

**Parallel** Parallel processes are composed by exploiting the —— operator. The operator combines sequences (note a single statement is a sequence of one element). Let s1, s2, . . . , sn1, sn be sequences. Then, the parallel composition

\[ s_1 \ || \ s_2 \ || \ \ldots \ || \ s_{n-1} \ || \ s_n \]

executes every sequence in parallel. A parallel composition is terminated when all the sequences are terminated.

**Non-deterministic choice** A non-deterministic choice can be expressed among different guarded branches by using the ++ operator. A branch guard can only be an input operation or a linkIn statement, whereas the branch can be any process. Let

\[(g_1, p_1), (g_2, p_2), \ldots, (g_{n-1}, p_{n-1}), (g_n, p_n)\]

be branches where g is the branch guard and p the guarded process. The syntax of the non-deterministic choice follows:

\[[g_1]p_1++[g_2]p_2++\ldots++[g_{n-1}]p_{n-1}++[g_n]p_n\]

The guards are defined within square brackets. When a non-deterministic choice is programmed it makes the interpreter waiting for an input on one of its guards. Once an input has come, the related p process is executed and the other branches are deactivated.
Priority of the composers  The statement composers interpretation priority is: 

\[ \text{\texttt{Req1<\textless a\textgreater} A \texttt{\|\| B \\texttt{\|\| C ++ [\texttt{Req2<\textless b\textgreater}] D \texttt{\|\| C \\texttt{\|\| B \texttt{\|\| D})}} \]

In this code fragment there is a non-deterministic choice between two branches guarded by two One-Way operations \texttt{(req1\textless a\textgreater)} and \texttt{(req2\textless b\textgreater)}. By considering the operator priority the same code would be explicitized as follows.

\[ \texttt{[input1](A \texttt{\|\| (B \\texttt{\|\| C)) ++ [input2] ((D \texttt{\|\| C \texttt{\|\| B) \|\| D})}} \]

Example  As a practical example, consider a scenario in which we have an orchestrator which acts as a service provider. The orchestrator declares a Request-Response operation, named \texttt{factorialRR}, which has the purpose to receive a number and, as a response, to send its factorial. Moreover, the orchestrator has to interact with a logging server in order to communicate its activity for constructing a statistic of its usage. The following code snippet shows the part of a possible implementation. For the sake of brevity, only the main procedure is shown.

```java
main {
    while ( 1 ) {
        [ factorialRR< n \times result > ( call ( calcFactorial ) ) ]
        servedClients = servedClients + 1
        ++
        [ linkIn ( logLink ) ]
        notifyActivity@logServerUri< servedClients >; ;
        servedClients = 0
    }
    ||
    while ( 1 ) {
        sleep ( 60000 ) ; ; /* 60 seconds */
        linkOut ( logLink )
    }
}
The main is composed by two processes in parallel. The former defines a nondeterministic choice between the One-Way on which the service can be accessed for returning the factorial calculation and the linkIn process defined on the internal link logLink. The linkOut process which triggers the internal link logLink is defined in the second parallel process which, every 60 seconds, interrupts the service for sending the number of the served clients to the logging service located at logServerUri.
Chapter 3

State of the Art: SOA Network Management

3.1 Overview

The recent history of telecommunications technology may be characterized as an era of convergence between telephony systems and data systems. Many, and arguably most, of the points of convergence have occurred in the data space; when convergence has occurred, it is more often the case that telephony systems have adopted the technologies and protocols originated in data systems than vice versa. Along with adoption has come adaptation, as telephony systems often have requirements and constraints that are not common in data systems, resulting in extensions to and derivations from the data architectures and protocols, such as the various voice over IP (VoIP) protocols and Session Initiation Protocol (SIP).

In this decade the data world has evolved from the client-server architectural approach to the service-oriented architecture (SOA) approach for system organization. SOA and Web services have compelling business, financial, and technological benefits that may be readily translated into the telecommunications industry; and thus, predictably, in the past few years telephony systems have begun adopting and adapting the SOA approach and Web services protocols.

There are plenty of papers in literature describing the use of Service Oriented Architecture, and particularly the use of web services, for the management of heterogeneous systems in the world of Networking. The following sections present a brief
overview of the current areas of research in which researchers are conducting their work. For each of them a brief review of different articles has been reported. They have been collected into five main areas:

- Business context and technologies of web services in telecommunications.
- Dynamic composition of Web services.
- Performances.
- Deployed implementations of Web services-based systems.
- Virtual resources management.

3.2 Business context and technologies of web services in telecommunications

In [MBD07] the authors outline the evolution of SOA concepts in telecommunications. They report on the changes in the evolution of the telecommunication world, showing how the converge-driven need to deliver seamless services across different access networks has forced operators to embrace new approaches, starting from the Intelligent Network, until the IP Multimedia Subsystem. In [GP07] instead, the authors trace a brief description of how web services are been used today in the telecommunications. They show how the SOA and the web services have become a major agenda item for the majority of the telecommunication network operators. After presenting a brief survey of SOA and web services, they describe the key enablers in telecommunications: Web services, event-driven architectures, Parlay X/ECMA specifications, and the enterprise service bus.

3.3 Dynamic composition of Web services

Service composition is a technique that may help the development of management systems by aggregating smaller services to produce more sophisticated ones. As a technique, service composition can be used to address problems of several computer science disciplines, including network management, where composition is especially
interesting when a complex management process requires the execution of smaller activities in order to be successfully accomplished. For example, to track the number of routes an autonomous systems (AS) advertises through its different routers, a composition that combines the routers’ information exposed by their management agents is required, so one can be able to detect, for example, possible anomalies on an AS behavior.

Service composition can be realized by using traditional management technologies, although these technologies have not been conceived taking composition support as one of their main aspects. Current service-oriented architecture (SOA)-related efforts, however, define specific standards for Web services composition, such as the Web Services Business Process Execution Language (WS-BPEL).

In [REC+07] a review of service composition in the context of network management is presented. They have evaluated the use of traditional management technologies such as the IETF Script MIB, as well as technologies specifically defined to support workflow compositions, like WS-BPEL. Their evaluations have been executed considering a management environment composed of BGP routers that need to be monitored in order to detect anomalies related to the advertisement of BGP routes from remote autonomous systems. They have considered three main technologies: Script MIB, which is a flexible IETF solution for the deployment of management script on remote managers in a possible hierarchy of managers; ad-hoc compositions often implemented on Web systems such as meta-search engines; and WS-BPEL, a recent standard devoted to the specific creation and support of Web services-based compositions. Previously to their work, it was natural to believe that WS-BPEL - which requires a strong software infrastructure to be deployed - would perform poorer when compared with both Script MIB and ad-hoc compositions. However, from their evaluation results it is now evident that the performance issues of WS-BPEL compositions are not as critical as initially supposed. In addition to the performance results associated to it, WS-BPEL also has the advantage of being specifically created for service composition, thus more properly dealing with composition questions, such as native parallel execution support, better design and expressiveness of compositions, and an increasing set of tools available to automate service composition.
3.4 Performances

Today, comparing Web service with SNMP (the de facto Internet management protocol) is an intensive research area with very important, although preliminary, results. Some studies already show that Web service can consume less bandwidth than SNMP when a large number of management objects needs to be retrieved from a management entity [NVG+04]. On the other hand, other studies show that performance issues may prevent the WS usage [PFGL04], while one can argue that WS ease of use is an advantage that could surpass the performance aspect [DvdMP04]. The current Web service versus SNMP investigations are characterized by the fact that they are taken from a quite broad view, where the SNMP is considered in a general perspective, without paying closer attention to particular management situations or scenarios. We believe that such particular situations, however, can reveal aspects of the use of Web service for network management that are currently hidden by the broader, more general investigations. In [TLML05] the authors investigate the definition and use of SNMP to Web service gateways built considering SNMP services defined through management information bases (MIBs), instead of considering direct mappings of SNMP operations to WS operations, which is the approach usually taken in the current investigations. In their view, Web service can also play an important role in management by delegation (MbD) due to their flexibility and ease of use. Thus, they present a study on the use of WS in the context of MbD, and compare Web service with SNMP considering the specific case where the IETF Script MIB is used as a MbD tool. This particular situation serves as a case study for the proposed approach of building SNMP to Web service gateways based on services. They have evaluated the proposed approach observing the bandwidth consumed by the Web service traffic as well as the Web service execution time when Web service are providing the same set of services exposed by the Script MIB. They also present a prototype of a WS-based MbD system developed to allow the observation of Web service against SNMP.
3.5 Deployed implementations of Web services-based systems

Current network programmability implementations addressing network management are mostly based on legacy distributed programming models such as OMGs Common Object Request Broker Agent (CORBA), Microsofts Distributed Component Object Model (DCOM) and Suns Remote Method Invocation (RMI). To the best of our knowledge there are very few implementations that leverage the emerging Web services paradigm for distributed applications.

Hence, in [AS03], the authors concentrate on implementing programmable network management and service management applications based on the Web Services distributed technology. They aim at demonstrating that several characteristics of Web Services are perfectly tailored to supporting implementations of PNIs. One of their objectives is to provide a framework for implementing programmable interfaces that comply with the IEEE P1520 [JAJ+98] initiative and exporting them as W3C Web Services. Web Services are platform independent and enable sophisticated system interoperability, since they are invoked over the standard HTTP protocol. In addition, the HTTP based invocation bypasses all obstacles in network management imposed by proxies and firewalls. Furthermore, management information and commands are implemented and exchanged using XML. Thus, Web Services implementations are more lightweight compared to conventional CORBA and DCOM implementations. It is also important that Web Services allow programmable networks technology to be integrated with network policy-based management. Indeed, Web Services could interact with policy services through protocols like the widespread Light Directory Access Protocol (LDAP), or the more specialized Common Open Policy Service (COPS), or even implement stand-alone network policies using description, discovery and assembly techniques provided by the UDDI registry.

Using Web Services network management applications can be implemented at discrete layers/levels, based on the P1520 specification. Element level management (the P1520 Physical Element and Virtual Network Device levels) could be delivered as a Web Service. This delivery can exploit existing work in the area of XML/MIB mapping and exposing devices functionality through an XML interface. Using XML interfaces and through the Simple Object Access Protocol (SOAP) management of
each element could be delivered over the WWW. Thus, an element level API allowing potential development of multi-device network management functions based on a distributed computing paradigm, is created as a collection of Web Services. This collection constitutes the Element Level Web Services. Referring to the network level management level and the relevant Network Generic Services level of the P1520 initiative, they use Web Services to develop open, programmable interfaces. Indeed tasks as routing, resource management or call control could be addressed using a more complicate set of Web Services (i.e. higher level API), which could be based on the assembly of element level Web Services. They call Web Services implementing Network Generic Services management functions, Network Generic Web Services. Furthermore by exposing Network Generic Web Services functionality as another API, complex network and service management algorithms could be designed, developed and deployed in an easy and timely fashion. Note that a Web Services implementation provides direct support to Web based management applications given that the interface of the ubiquitous WWW Browser. Last but not least, the use of the WWW in the framework they propose implies inheritance of Web Based Management advantages, such as lightweight implementation, distribution of management functions or remote handling.

In [WTY07] the authors outline how the SOA and web services concepts can be a challenge to telecom service providers for the management of their infrastructures. They show how, the migration from the traditionally ITU-Ts FCAP-based model for network management systems to the next generation service management systems based on the eTOM model, can be leveraged from the adoption of SOA which provides a very robust integration technology.

### 3.6 Virtual resources management

A challenging issue facing high computing demanding communities is that while computing facilities providers can provide access to many heterogeneous resources, the resources to which access is provided often do not match the needs of a specific application or service. In an environment in which both resource availability and software requirements evolve rapidly, this disconnect can lead to resource underutilization, user frustration, and much wasted effort spent on bridging the gap between
applications and resources. These issues can be overcome by allowing authorized clients to negotiate the creation of virtual clusters made up of virtual machines configured to suit client requirements for software environment and hardware allocation. In [SH04] IBM researchers identify virtual computing and Service Oriented Architecture as the enabling technologies for the increasing request of on-demand operating environments and on-demand virtualization. They clearly illustrate how these technologies can help scientists as well as commercial companies to better satisfy their users requirements.

The authors of [KFF+05] present the concept of a virtual workspace, which allows a Grid client to define an environment in terms of its requirements (such as resource requirements or software configuration), manage it, and then deploy the environment in the Grid. Workspaces defined in this way can be implemented in a variety of ways such as for example dynamically creating Unix accounts and using system as well as software configuration tools to enforce the required properties. They focus on a particularly promising implementation of virtual workspaces: virtual machines (VMs). The use of virtual machines in Grid computing has been proposed before [3,4]. In addition to outstanding isolation properties, VMs can provide fine-grained enforcement; and by their very nature (virtualization of the underlying hardware) they enable instantiation of a new, independently configured guest environment on a host resource. They can be rapidly suspended and their state serialized, and thus easily migrated to remote resources. Moreover, as a result of recent progress in virtual machine technology, these advantages no longer come at a performance cost to either the application or the hosting resource: systems such as Xen demonstrate that they can be used with little or no performance degradation. They describe how virtual workspaces fit into the Grid architecture and present a prototype of this architecture based on the Globus Toolkit (GT), a Web service framework for Grid environment.
Chapter 4

The Device Management System Software

4.1 Overview

The starting point for the designing of a scalable system for the network management and its related services, is approaching the modeling using the concept of a stack of different abstraction layers.

The deployment of new services, starting from a new idea or a new user’s driven need, implies a sequence of actions and processes that span from the service designing to the network’s devices configuration. Each of this actions belong to a specific domain of knowledge and imply different processes belonging to different divisions inside the same company. Going from the bottom to the top, starting from the technical operations and ending with the deployment of complex services, the modeling activity proceeds defining different layers, each one with an higher level of abstraction.

The main functionalities of each layer can be associated to objects and those who share similar functionalities can be grouped together in order to create new independent components. These expose those functionalities as services.

This approach starts from the basic services until the most complex. The services at the higher levels can be obtained as a composition of other services in the lower layers using a composition as well as a coordination logic.

One of the most important concept on which the system’s architecture has been
designed is hiding the complexity of a single system inside a black box. Thus, the deployment of vertical services can be seen as a Lego construction and the final services themself as machines made of little bricks. This approach is very helpfull in the governance of the complexity for services deployment.

![Diagram of services abstraction layers]

**Figure 4.1: The services’ abstraction layers.**

As depicted in the figure above, in this work four different layers have been identified. The bottom layer is composed by the services that are responsible of providing access to the network’s devices functionalities. These are the basic functionalities of the common network devices and, more or less, can be mapped to the operations with which a network administrator deals with in his daily activity. A specific sequence of basic operations can determine a specific process like, for example, the creation of a VLAN inside a network. These operations can be associated to specific services belonging to the second layer.

The third layer is composed from more specific as well as complex services. These services carry out their job recalling the functionalities of the lower layers. Their behaviour as well as the logic that drive them, can be expressed using workflows of coordinated services. On the top of this pyramid, we found the applications. Applications are the primary interfaces to the final user. They expose the methods to access the system’s resources and allow to deploy vertical services with very specific functionalities.

The use of a common formalism for the services definition allow to deploy scalable as well as flexible system architectures that can seamless adapt to the growth and the evolution of a telecom operator’s network and, at the same time, constitutes an ideal environment for the creation of new applications.

Using common interface as well as protocols, operators without any knowledge of the
inner details of each single service, can compose new services tailored to their needs. For example, web programmers can deliver applications that make use of network’s devices, without boring with the protocols or primitives for the communication to the devices. Using this approach, integrating a short message service into a portal using an sms proxy equipped with a web service interface, would be very easy. Also designing can be easy using the same approach. Using the same formalism for the designing of all the components in the system pyramid, the designer has a general view on the whole system. Interactions among all the components are highlighted and thus, also plans for fault management can be developed in advance. Moreover, faults of components belonging to different layers can be outlined in advance. It becomes very simple to foresee the chain of errors correlated to the mulfanfunctioning of a specific device and locate the exact points of failure of the system.

Starting from these principles, in this thesis a framework, that has been developed for the construction of such an architecture, has been described. For the practical implementation of this concepts, we choose to use web services as well as standard Internet technologies. Every service is equipped with a standard web service interface. Also the access to the network devices is carried out through simple web services. Actually, a single as well as a group of devices are accessed using a Web Service proxy that is responsible for the dispatching of the commands to the devices. To expose those functionalities, the proxy make use of a new library that hide the inner details of the communications with the physical devices. Once again we designed this library using a black blox approach and an abstract layers composition methodology. We developed an Object Oriented model for the network devices description and, as we will describe in the next section, we implemented it using the Java language and an open source implementation of the Inversion Of Control (IOC) pattern [spr] that allow for the runtime components binding.

All the proxies constitute the simple services layer. On top of this, complex services are created. The logic behind the complex services, is designed using workflows of coordinated webservice. The implementation of the latter is made using standard languages for web services workflows as, for example, the BPEL language [wsb].
4.2 Software Architecture

4.2.1 Simple services: The proxy architecture

Almost every network environment is an evolving system in which bleeding edge devices can be used together with older apparatus. In such a situation it could be very easy to run into web-services enabled devices; nevertheless, in order to integrate newer as well as legacy devices, it could be very useful to have a proxy capable of communicating with bare bone hardware through a common and well known interface. The Web Proxy is the component that grants the access to the network resources. It is the main entry point for each hardware communication. The use of a web-service interface allows to have a common layer to communicate to each device in the network, regardless of its "age". The figure 4.3 displays the inner architecture of the Web Proxy. It is composed by two main components: the web-service interface and the device communication component. The former exposes the methods available, through a standard web service interface, for the device configuration; the latter is responsible for the device communication details using an ad-hoc communication library. The implementation code of the Web service interface is responsible for the mapping of the configuration operations into the specific commands of the device. Each method allows to exploit a single or a group of functionalities available in the device’s operative system. The number of methods available through the Web ser-
vice interface and their input parameters, as well as their output, depends on the level of abstraction chose during the interface’s design phase. Basically, two abstraction profiles have been supported: a very flexible but generic interface and a more specific but hard-shelled one. The first is basically composed by a single method. Using this method, an operator can send any device’s command, using the device’s language syntax; the proxy acts as a medium between the device and the operator. This solution is very flexible and let the operator insert any configuration he desire. The side effect of this approach is that the operator must know all the details of the device’s operating system. The latter exposes a more complex interface with specific methods that realize specific device’s functionalities; these are realized through the sending of different specific commands. This approach has the advantage of hiding all the details of the commands and their syntax. It allows to access all the device’s functionalities without knowing the inner details of the device’s system. On the contrary, the number of the methods available is fixed and the operator has a limited control on the device. The figure 4.4 depicts the inner details of the proxy architecture. It is composed by three main components: the web interface layer, the business logic layer and the device communication layer.

The interface layer exposes the methods available through a standard web service interface. It is responsible for the management of all the incoming as well as outgoing messaging operations involved in the proxy communication. It supports the SOAP and the REST messaging protocols. It has been implemented using the open source framework Axis2 [axi] from the Apache Software Foundation [apab].
The business logic layer is constituted by all the code for the realization of the logic behind the methods implementation. Each method of the web interface is translated into a list of specific procedures that are sent to the device using the device layer communication’s primitives. It is implemented using the Java language.

The device communication layer is the component used by the implementation layer to translate the interface’s methods into specific device’s command. It offers a complete set of primitives that maps all the functionalities of the device. The communication layer is also responsible for the communication’s session management. It opens a communication socket with the device’s console and sends all the commands to it using different communication protocols. As it is described later, it is organized as a java library that allows to communicate with a specific device through an abstract representation of it based on an object model. In this model each device is represented using a specific class. The device binding to its representation object is made at runtime using a java library that implements a common “Inversion Of Control” programming pattern. This library implements a class container that binds each device to its class driver. The binding rules are defined in a text configuration file used by the library’s container for the objects’ instantiation setup.

The web proxy’s description depicted above can be seen as a lego composition in which each network’s device can seamless be plugged to the proxy itself. To be controlled by the proxy, a new device must have a valid pluggable driver. No lines of code are necessary to put a new device under the proxy’s control, just a few configurations.
4.2.2 Complex services: realizing a control plane for services orchestration

The logic behind the creation and the management of complex services, can be expressed in terms of work-flows of business processes. A work-flow can be represented using a standard language. According to the philosophy to follow the SOA requirements, we choose the BPEL (Business Process Execution Language, [wsb]) language.

The creation of complex process can be represented as a sequence of coordinated actions performed by the components defined in the sections above.

4.2.3 Applications

Applications are the top of our architecture’s pyramid. They export all the system’s functionalities to the final user.

Applications make use of all the stacks of our architecture and can be run in any webservice compliant communication system. They can represent applications directly accessible by the final user or can offer a business service to other systems.

Through the use of standard web services protocols and services’ composition, applications can realize a perfect integration between the telecommunication and the IT worlds.
4.3 The device management library

As described in the previous section, the basic building block of our architecture is constituted by the network layer. This is the lower level of abstraction of the architecture’s pyramid model. In order to easily access the devices’ functionalities we have developed a Java library. This library is used by the web proxy for the communication with the network device. The main requirements of the library are:

- The designing of the network devices’ model should be Object Oriented.
- All the network devices should be accessible as well as manageable through the use of simple Java APIs.
- The APIs should abstract all the devices functionalities.
- All the functionalities should be organized into packages.
- For the sake of simplicity, the functionalities related to the communication with the devices should be grouped together in a separate package.
- The APIs methods should not depend on the device’s vendor or model.
4.3. The device management library

- The device’s drivers should be pluggable at runtime.

Starting from those requirements, we started to create an object model of the network apparatus. This is composed by two main packages: the connection package and the devices package. Because we wanted a live pluggable system capable of connecting running devices at run-time, we designed the library using a programming concept, the dependency injection, that is a direct application of the "Inversion Of Control" principle [JF88]. The dependency injection allows to manage external dependencies between classes. The methods access as well as the class’ fields modifications are carried out through a class proxy. The container is responsible for the physical binding of the objects using a factory. The binding rules are described by an XML configuration file that is read by the container at runtime. To implement such a system, we chose the Spring framework library [spr].

![Diagram of Inversion of Control pattern](image)

Figure 4.7: The Inversion of Control pattern.

Using this approach, several classes of devices were created. Each one implements a common device interface. The implementation code of the business logic layer can access the device’s functionalities without knowing the internal details of each device’s driver. These, as we mentioned earlier, can be plugged like lego’s bricks to the proxy.

The devices package

The use of an object-oriented representation to model the network devices has been done because of the intrinsic benefits of the OO model: it allows to design by
components, it encourages reusability, and so on. Actually tring to categorize the network devices isn’t a simple job and we must make a few assumptions. First of all, the vendors of the devices don’t adopt the same principle to categorize their products. For example, the Cisco divides them on the basis of the target market to which they are offering their products. So they have an enterprise bouquet, a SOHO one and so on even if, often the products belonging to these categories share the same functionalities. The Nortel’s offer, on the contrary, is based on the product’s technical features: wireless devices, optical apparatus, etc. We chose to categorize the devices on the basis of their functionalities, following the market’s classification, using a top-down approach. So at the root of the classes’ tree we found the GenericDevice interface, from which all the other classes are derived. This interface exposes simple methods, like attach and detach, that are common to all the devices. From this one two main groups of devices’ classes are derived: switches and routers. These two interfaces represents the basic operations common to all switches or routers. Starting from the principle that each interface represents an abstraction of a device, it exposes the methods common to all the devices that belong to it. In this way it is possible to represent different devices using the same interface. For each specific device a new class implementing this interface has been derived. When a device has functionalities that belong to different categories, its related class will implement both. So, for example, if a device has switches’ as well as routers’ functionalities, it will implement both these interfaces.

For the functionalities typical of a specific vendor, a series of specific classes has been designed. So, for example, the GenericCisco class represents an abstraction of the functionalities supported by all the Cisco devices. Finally, at the leaves, there are the devices that are available in the market. Using this approach, they are categorized by their functionalities as well as their brand. The figure 4.3 reports a simple extract of the class diagram.

As we stated previously, we used the dependency injection to manage the classes’ dependencies, so for each class a proxy is provided. The GenericSwitch interface has its GenericSwitchProxy class. Let’s see some code in order to clarify all these concepts.

Suppose that we want to setup a new VLAN on a Cisco6500 switch. We want to give the id 10 to the VLAN and we want to assign the GigabitEthernet 1/3/3 physical
interface to it, the GigabitEthernet 1/0/3 interface as trunk and assign an ip address for remote connections. Usually, the operator should type the following code on the device’s console:

```
configure terminal
vlan 10
name Prova
exit
interface GigabitEthernet 1/3/3
shutdown
switchport
switchport mode access
switchport access vlan 10
no shutdown
exit
interface 10
ip address 192.168.0.2 255.255.255.0
exit
interface GigabitEthernet 1/0/3
shutdown
switchport
switchport trunk allowed vlan add 10
no shutdown
```
The same operations could be obtained, invoking the methods of the Generic-Switch interface with the java language, as reported in the following fragment of code:

```java
int vlanID = 10;
String name = "Prova";
String vlanPort = "GigabitEthernet 1/3/3";
String vlanIP = "192.168.0.2";
String interf = "GigabitEthernet 1/0/3";
ctx = new FileSystemXmlApplicationContext("context.xml");
GenericSwitch obj = (GenericSwitch) ctx.getBean("switch");
obj.attach(ID, IP, port);
obj.createVLAN(vlanID, name);
obj.addPortVLAN(vlanID, vlanPORT);
obj.ipRemoteVLAN(vlanID, vlanIP);
obj.portTrunk(interf, vlanID);
obj.detach();
```

Looking at the code reported above, it can be noted that there is no direct reference to the class that implements the primitives. The methods’ signature are independent of the specific class that implements them, so the same code could be used to call any class that implements the GenericSwitch interface. Using the dependency injection the user can change the device’s class to control without changing a line of code. In the following, we report an extract of the code of the Cisco6500 driver class:

```java
public String createVLAN(int vlanID, String name){
    this.connectionManager.send_command("config t");
    this.connectionManager.send_command("vlan "+vlanID);
    this.connectionManager.send_command("name "+name);
    this.connectionManager.send_command("end");
    return "Create Vlan "+vlanID+" "+name;
}
```
public String addPortVLAN(int vlanID, String vlanPORT) {
    this.connectionManager.send_command("conf t");
    this.connectionManager.send_command(
        "interface "+vlanPORT);
    this.connectionManager.send_command("shutdown");
    this.connectionManager.send_command("switchport");
    this.connectionManager.send_command(
        "switchport mode access");
    this.connectionManager.send_command(
        "switchport access vlan "+vlanID);
    this.connectionManager.send_command("no shutdown");
    this.connectionManager.send_command("end");
    return "Vlan "+vlanID+" in "+vlanPORT;
}

public String ipRemoteVLAN(int vlanID, String vlanIP) {
    this.connectionManager.send_command("conf t");
    this.connectionManager.send_command(
        "interface "+vlanID);
    this.connectionManager.send_command(
        "ip address "+vlanIP);
    this.connectionManager.send_command("end");
    return "VLan: "+vlanID+"\nIP: "+vlanIP;
}

public String portTrunk(String interf, int vlanID) {
    this.connectionManager.send_command("conf t");
    this.connectionManager.send_command("shutdown");
    this.connectionManager.send_command(
        "interface "+interf);
    this.connectionManager.send_command("switchport");
    this.connectionManager.send_command(
        "switchport trunk allowed vlan add "+vlanID);
}
The connection package

All the details regarding the physical connection to the device, are hidden by the ConnectionManager interface. For each type of connection a new class has been developed. At the moment we have developed a TelnetConnectionManager class and a SSHConnectionManager class. Calls to a specific device are carried out using the ConnectionManagerProxy. The figure 4.9 reports a fragment of the connection package’s class diagram.

The class that implements a device’s functionalities can send commands to the physical device using the ConnectionManager’s methods. The following code reports the send_command method of a ConnectionManagerProxy implementation and show how this is possible:

```java
public String send_command(String command) {
    return this.connectionManager.send_command(command);
}
```

As the code reports, the class is unaware of the specific communication class used to call the device’s functionalities.
4.4 The VLan Creation Example

In the following we report an example that clarifies the use of our methodology and in particular the use of the DMS library for the creation of a vlan.

The figure 4.10 reports a network scenario where eight computer are connected to two switches. We want to show how our system can be used in order to create a service for the deployment of vlans. Each switch is controlled by a web service, the Switch Manager. This service allows to access the switch’s management functionalities through a standard WSDL interface. In particular it allows to set a new vlan giving an id and an optional name. It also allows to associate a particular port to the specified vlan. In the listing below we report an extract of its interface that shows the details related to those functionalities.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
  xmlns:ns1="http://org.apache.axis2/xsd"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
```
```
xmns:http=”http://schemas.xmlsoap.org/wsd1/http/”
xmns:ns0=”http://ws.apache.org/axis2”
xmns:xs=”http://www.w3.org/2001/XMLSchema”
xmns:mime=”http://schemas.xmlsoap.org/wsd1/mime/”
xmns:soap=”http://schemas.xmlsoap.org/wsd1/soap/”
xmns:soap12=”http://schemas.xmlsoap.org/wsd1/soap12/”
targetNamespace=”http://ws.apache.org/axis2”>
  <wsdl:documentation>SwitchManager</wsdl:documentation>
  <wsdl:types>
    <xs:schema xmlns:ns=”http://ws.apache.org/axis2”
      attributeFormDefault=”qualified”
      elementFormDefault=”qualified”
      targetNamespace=”http://ws.apache.org/axis2”>
      ...
    
    <xs:element name=”movePortToVlanAsUnTagged”>
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs=”0” name=”host_id”
            nillable=”true” type=”xs:string”/>
          <xs:element minOccurs=”0” name=”ip”
            nillable=”true” type=”xs:string”/>
          <xs:element minOccurs=”0” name=”port”
            type=”xs:int”/>
          <xs:element minOccurs=”0” name=”user”
            nillable=”true” type=”xs:string”/>
          <xs:element minOccurs=”0” name=”password”
            nillable=”true” type=”xs:string”/>
          <xs:element minOccurs=”0” name=”vlan”
            nillable=”true” type=”xs:string”/>
          <xs:element minOccurs=”0” name=”intef”
            type=”xs:int”/>
        </xs:sequence>
    ```
4.4. The VLAN Creation Example

```xml
<xs:element name="movePortToVlanAsUntaggedResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

...
<xs:element name="setVLANResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
</wsdl:types>

<wsdl:message name="setVLANRequest">
  <wsdl:part name="parameters" element="ns0:setVLAN"/>
</wsdl:message>

<wsdl:message name="setVLANResponse">
  <wsdl:part name="parameters"
    element="ns0:setVLANResponse"/>
</wsdl:message>

<wsdl:message name="movePortToVlanAsUntaggedRequest">
  <wsdl:part name="parameters"
    element="ns0:movePortToVlanAsUntagged"/>
</wsdl:message>

<wsdl:message name="movePortToVlanAsUntaggedResponse">
  <wsdl:part name="parameters"
    element="ns0:movePortToVlanAsUntaggedResponse"/>
</wsdl:message>

...
<wsdl:portType name="SwitchManagerPortType">
  <wsdl:operation name="setVLAN">
    <wsdl:input message="ns0:setVLANRequest">
      wsaw:Action="urn:setVLAN"/>
    <wsdl:output message="ns0:setVLANResponse">
      wsaw:Action="urn:setVLANResponse"/>
  </wsdl:operation>
  <wsdl:operation name="movePortToVlanAsUntagged">
    <wsdl:input message="ns0:movePortToVlanAsUntaggedRequest">
      wsaw:Action="urn:movePortToVlanAsUntagged"/>
    <wsdl:output message="ns0:movePortToVlanAsUntaggedResponse">
      wsaw:Action="urn:movePortToVlanAsUntaggedResponse"/>
  </wsdl:operation>
  ...
</wsdl:portType>
<wsdl:binding name="SwitchManagerSOAP11Binding" type="ns0:SwitchManagerPortType">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>
  <wsdl:operation name="setVLAN">
    <soap:operation soapAction="urn:setVLAN"
      style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
The creation of a vlan on such a network environment, where many network devices are connected together, requires the execution of numerous procedures for the devices configuration. First of all in our example the user has to set the vlan on both the switches and then add each port, on which the specified computers are connected, to the vlan itself.

When we want to create a vlan among different switches, actually on each switch we should also associate the new vlan to the trunking port of the switch. This would...
allow the switch to let the traffic belonging to the specified vlan to flow to the other switches. Actually, since on almost every switch each new vlan is attached to the trunking port by default, in our example we won’t care about this procedure. In the following, we summarize all the necessary steps for the creation of a new vlan:

1. Define a new Vlan ID.
2. Select the hosts to add to the vlan.
3. Retrieve the informations about the ports on which the hosts are connected.
4. For each switch:
   (a) Set the vlan giving the Vlan ID.
   (b) For each host:
       • Move the specified port to the Vlan.

These steps can be expressed as a workflow of different methods invocations on the Switch Manager proxies. This workflow defines the inner business logic of the Vlan Manager service as depicted in figure 4.11 and is implemented using the Jolie language. The execution of the workflow is triggered by an external user’s request,
who should only supply the list of the nodes to connect to the new vlan to the Vlan Manager service.

In the following we report an extract of the code of the Vlan Manager service. In the first part of the code all the interfaces are defined. In particular we found:

- DatabaseInterface: The interface to connect to the local DB.
- SwitchManagerInterface: The interface of the Switch Manager Proxy.
- VLanManagerInterface: The interface of the VLanManager.

```c
#include "console.iol"
#include ".../locations.ol"

evaluation{sequential}

constants {
    min_vlan_id = 1100,
    max_vlan = 20,
    ip = "10.0.0.12",
    user = "libsc",
    password = "RicercA",
    port = 23
}

//--- INTERFACES ----------------------------------------

interface DatabaseInterface {
    RequestResponse:
        connect,
        query,
        update
}
```
interface SwitchManagerInterface {
  RequestResponse:
    setVLANRequest,
    deleteVLANRequest,
    deleteVLANportRequest,
    movePortToVlanAsUnTagged
}

interface VLANManagerInterface {
  RequestResponse:
    /**
     * @request: void {
     *  .name:string      vlan name
     *  .list []:string   vlan hosts
     * @response:void
     *  .id:int           vlan id
     *}
    */
    createVLAN throws VlanNotAvailable DBFault,
    /**
     * @request: void {
     *  .id              vlan id
     *}
    */
    deleteVLAN throws DBFault,
    /**
     * @request: void {
     *  .id              vlan id
     *  .port            port to delete
     *}
    */
After the interfaces, we define the ports associated with the services. In particular we can have output as well as input ports. Each port must be associated to one or more interfaces.

//---- OUTPUT PORTS ----------------------------------

outputPort Database {
Interfaces: DatabaseInterface
}

outputPort SwitchManager {
Location: Location_SwitchManager
Protocol: soap {
    .namespace = "http://ws.apache.org/axis2";
    .schema = "/usr/local/src/JolieServices/Schemas/vlan.xsd";
    .debug=1
}
Interfaces: SwitchManagerInterface
}

//---- EMBEDDED ---

embedded {
Java:
    "joliex.db.DatabaseService" in Database
In the last part of the code list, after some initial definitions, the execution code of the service is reported.

```java
init {
    println@Console("Running...");
    scope( connection ) {
        install( InvalidDriver =>
            println@Console("-fault:InvalidDriver "+ connection.InvalidDriver.stackTrace);
            throw( DBFault )
        );
        install( ConnectionError =>
            println@Console("-fault:ConnectionError");
            throw( DBFault )
        );

        with( request ) {
            .driver = "mysql";
            .host = "localhost";
            .database = "vlan";
            .username = "tomcat";
            .password = "tomcat"
        }
    }
}
```
CHAPTER 4. The Device Management System Software

connect@Database( request )( response )
}
}

define perform_query
{
  scope( sql ) {
    install( SQLException =>
      println@Console("-fault: SQLException ");
      println@Console( sql.SQLException.stackTrace );
      throw( DBFault )
    );
    query@Database( query )( query_resp )
  }
}

define perform_update
{
  scope( sql ) {
    install( SQLException =>
      println@Console("-fault: SQLException ");
      println@Console( sql.SQLException.stackTrace );
      throw( DBFault )
    );
    update@Database( query )( query_resp )
  }
}

main
{

  install( VlanNotAvailable => println@Console(}
4.4. The VLAN Creation Example

```java
"-fault:VLANNotAvailable")
install ( DBFault => println@Console("-fault:DBFault"));

[createVlan( request )( response ){

  // looking for a free vlan_id. min_vlan_id <= vlan_id
  // <= max_vlan + min_vlan_id
  query = "SELECT DISTINCT vlan_id FROM " + 
  "vlan ORDER BY vlan_id ASC";
  perform_query;
  i = 0;
  found = 0;
  while ( (found == 0) && (i < max_vlan ) ) {
    println@Console(query_resp.row[i].vlan_id + 
    "," + int(i + min_vlan_id) );
    if ( int(query_resp.row[i].vlan_id) == 
      ( i + min_vlan_id ) ) {
      i++
    } else {
      found = 1
    }
  }
  if ( found == 0 ) {
    throw( VlanNotAvailable )
  }

  // found a free vlan, starting creation to switchManager

  free_id = i + min_vlan_id;

  with ( request.create ) {
    .host_id = "";
  }
```
.ip = ip;
.port = port;
.user = user;
.password = password;
.vlan_id = free_id;
.vlan_ip = "";
.vlan_name = request.name

setVLANRequest@SwitchManager( request_create )
( response_create );
println@Console( response_create.return + request.name );
// store the new vlan_id within the database
query = "INSERT INTO vlan ( name , vlan_id ) VALUES (""
 + request.name + "," + free_id + ")";
perform_update;

response.id = free_id;

// looking to hostname ports into the DB and
joining these ports to the VLAN
// calling the SwitchManager

undef( request_create.vlan_id );
undef( request_create.vlan_name );
undef( request_create.vlan_ip );

for( i=0, i<#request.list , i++ ){

  query = "SELECT port FROM ports WHERE hostname = ""
 + request.list[i] + ";"
  perform_query;
  println@Console( request.list[i] + ":" )
4.4. The VLAN Creation Example

```java
+ query_resp.row[0].port);
request_create.vlan = free_id;
//request_create.interf = query_resp.row[0].port;
request_create.interf = 30;
movePortToVlanAsUntagged@SwitchManager
    ( request_create ) ( response_create );

query = "INSERT INTO vlan_port (vlan_id, port_id)" +
    " VALUES (" + free_id + ","
    + query_resp.row[0].port + ")";
perform_update;
println@Console(response_create.return)

response.msg = "Operazione completata"
```

] { nullProcess }

[deleteVLAN( request )( response ){
    // first step: delete teh Vlan from the Switch manager

    with ( request_create ) {
        .host_id = "";
        .ip = ip;
        .port = port;
        .user = user;
        .password = password;
        .vlan_id = request.id
    };

deleteVLANRequest@SwitchManager(request_create)
    ( response_create );
```
Once the above code is executed, using the Jolie language interpreter, a new process listening to the specified port is activated. When the Vlan Manager service receives a new request, the code associated with the method defined in the request is executed. If a request to an external service is necessary, it sends a message request to the external service specified in the workflow.

The code reported above constitutes a simple example of different services composition. For the sake of simplicity we reported only the pieces of code useful to understand the logic behind the methodology described in this chapter. For a better understanding of the code’s details please refer to the second chapter of this work whereas, if you need a wider understanding of the technologies used in this example, please refer to the biography.
Chapter 5

Virtual Cluster Management

5.1 Overview

In this chapter we describe an architecture for the deployment of virtual clusters that has been conducted at the CRS4, Cagliari as a part of the Cybersar project. Cybersar is a high performance computing initiative, funded by the Italian Ministry of Research, that has as its goal the development of a Cyber-infrastructure for research in Sardinia based on high speed networks interconnecting the island research communities and computational facilities. Cybersar infrastructure is designed so that it can support innovative computational applications while providing an experimental platform for research on application driven orchestration of computing and network resources.

5.1.1 Virtual Computing Facilities

Virtual machines, such as VMWare [War05], Xen [DFH+03], and KVM [kvm], bring to HPC two main gifts. On one hand, they enable the specialization of operating systems to particular tasks with hardware resources safely and transparently multiplexed by the vm hypervisor [MUKX06]. On the other hand, virtual machines allow the decoupling of the physical computational infrastructure management from the management of the user-visible virtual computing facility [KFF+05], [ACC+05], making the former essentially homogeneous and agnostic with respect to the specific applicative context, while putting the latter in the hands of the virtual organization.
managing the virtual resource [FDF03], [RIG+06]. This separation removes many of the difficulties that oppose the effective sharing of computational resource between organizations that are not adhering to very strict common standards for their software stacks.

Virtualization guarantees a high level of flexibility in the dynamic configuration and use of (atomic) computational resource, which can then be organized and orchestrated to provide virtual clusters that, differently from standard, general purpose, HPC computing facilities, can be tailored to satisfy the needs of specific virtual organizations [FFK+06].

Virtual clusters can encompass computational resources that are distributed on different, potentially remote sites. Current trends on ever increasing available network bandwidth [FG03] make it possible (at least when the structure of the applications run allows to trade latency for pipeline depth) to tightly couple geographically distant computational resources in a single computational facility that appears as a coherent entity to its users. Such a constructions, however, requires a much closer coordination between computational and network resources with dynamic network configuration mechanisms, similar to the ones described in [LSJ06], being activate in parallel to the virtual cluster construction.

5.2 Architecture design and implementation

The main functionalities required for deploying virtual clusters are dynamic partitioning of a pool of high performance machines and dynamic creation of specialized virtual resources. In order to achieve this, the software manager must provide the following abstractions:

- It has to allow for the selection of a pool of physical machines where each of them can be initialized in a different way from the other ones. The state diagram reported in figure 5.1.a, which represents each machine, contains two different states: not-initialized and initialized. A machine can be initialized by performing an init action.

- For each physical machine it has to allow for the creation of a pool of different virtual machines which could have different setups. Virtual machines can be
started and stopped dynamically. The state diagram which represent them is reported in figure 5.1.b and it contains four states: not-initialized, stopped, running and pause. State change is triggered by the create, start, stop, pause and unpause actions.

![State diagrams for physical and virtual resources.](image)

Figure 5.1: State diagrams for physical and virtual resources.

This architecture addresses these issues by means of a collaborative service oriented system in which different kinds of services are implemented:

- **physical resources services**: Hardware Deployment System (Hades), PlatForm Controller (PFC), Resource Manager Service, Machine Manager Service.

- **virtual resources services**: Virtual Machine Host (VMH), Virtual Machine Manager Service, Virtual Resource Manager Service.

- **virtual network services**: Vlan Manager Service, Switch Manager Service.

- **support services**: Account Service, Notifier Service (N).

- **user service**: User Interface (UI), Client Service.

All architectural components expose their functionalities through a service interface, thus participating in the creation of a SOA-based architecture. As far as the physical resources are concerned, Hades deals with the deployment of the physical images whereas PFC allows for remotely booting the machines via IPMI by means of power on/off operations. Virtual resources are managed by the VMH service which is replicated in each physical machine; VMH provides operations for creating, starting, pausing and stopping virtual machines on a given physical machine. The Switch Manager is the component responsible for the management of the network devices.
Its primary role is to create and maintain the vlans useful for the communication among the virtual machines. The Machine Manager, the Virtual Machine Manager, the Notifier and the Vlan Manager assemble other services in a workflow structure in order to provide creation of physical and virtual resources.

Finally, Account Service allows for the authentication of users, the Resource Manager as well as the Virtual Resource Manager provide functionalities for physical as well as virtual machine reservation and booking, and the User Interface gives the user a graphical interface for inter-operating with the architecture. It is worth noting that the User Interface and the Client Service must be considered together as the client application. By following a pure service oriented approach, in fact, the user interface becomes a service which only provides the graphical representation of the client application whose logic is coded within the Client Service. In the following we refer to both of them as User Application.

Figure 5.2: The architecture’s main components.

In order to clarify the relationship between the services in the architecture, let us consider all the necessary steps for creating a virtual machine.

1. First of all, the user must log on to the user interface by inserting his username and password. The User Application requests for an authentication to the Account service. If the authentication is verified the user can access the architecture’s functionalities.
2. The user inserts the required virtual machines description into the User Interface and then requests its allocation. The User Interface sends a request to the Virtual Resource Manager which calculate the amount of physical nodes needed in order to support the user’s request. Then it asks the Resource Manager service for retrieving the free machines to allocate. If the requested machine are available, the Resource Manager returns a reservation ticket.

3. The reservation ticket is then used for opening a session on the Virtual Resource Manager that will store all the virtual machine informations for the physical resources allocated within that ticket. The Virtual Resource Manager initializes its inner database by invoking the resource manager for retrieving all the physical resources related to the reservation ticket. From now on, every operation performed by the Virtual Resource Manager on the virtual cluster will be managed through the Virtual Machine Manager.

4. The Virtual Resource Manager asks the Vlan Manager to create a new vlan for the communication among the Virtual Machines. It sends all the hostname of the physical nodes associated with the ticket reservation.

5. The Vlan Manager retrieves the informations regarding the physical topology of the network from its inner database. For each physical node, it finds the switch and the relative port on which the node is connected and then, it asks the Switch Manager to create a new vlan with those informations.

6. The Virtual Resource Manager initializes every selected machine by specifying the setup information. For each of them an initialization request is sent to the Virtual Machine Manager which forwards the request to the Machine Manager. Concurrently, since the initialization procedure could take several minutes, the Virtual Machine Manager subscribes itself to the Notifier in order to be notified when the VMH for the new machine is ready to be used. In the meanwhile the user can perform other operations on the cluster.

7. The Machine Manager starts the initialization for the given machine by invoking Hades, then it sends a request to Hades for the IPMI address of the given machine and, with that information, sends a request to the PFC service in order to reboot the machine.
8. At reboot the machine sends a request to Hades for the image that needs to be installed, downloads it and then installs it. The installed image contains the VMH for that machine which sends a ready notification to the Notifier when started.

9. The Notifier matches the received VMH notification with the previous Virtual Machine Manager request and then sends back a notification to it.

10. The Virtual Machine Manager updates the state of the physical machine and then forwards the notification to the Virtual Resource Manager.

11. Now, the Virtual Resource Manager can install virtual machines on top of the physical one. It specifies all virtual machine information and then sends a request to the Virtual Machine Manager which will call the related VMH for creating the virtual machine.

12. When the virtual machine is created the VMH sends back an identifier for it to the Virtual Machine Manager that updates its inner database and then notifies the Virtual Resource Manager that the new virtual machine is available.

13. The user can now operate on the virtual machines by starting, stopping or pausing them.

5.2.1 Technologies

In this architecture, the basic features of the SOA approach have been exploited in order to integrate different technologies for implementing the involved services depending on the functionalities they have to provide.

Since Hades, PFC and VMH deal with machine images deployment, we have chosen the Gentoo Linux distribution [gen] as the operating system for the machines which will run that services. Then, in order to easily exploit Python Gentoo system utilities, Hades, PFC and VMH have been implemented with the pyGridWare libraries [Bov] which allow for the management of Web Services SOAP and WSDL specifications in Python. For the workflow services (Machine Manager, Virtual Machine Manager, Virtual Resource Manager, VLAN Manager, Client Service and Notifier) we chose JOLIE, which provides easy non-XML primitives for managing incoming
and outcoming data, sending and receiving requests and correlating sessions. Finally, since the Account Service, the Resource Manager, the Image Repository, the Switch Manager and the User Interface will be employed in several architectures other than CyberSar, they have been implemented in Java, a well known technology which allows for an easier service management by different developers not directly involved with CyberSar.

### 5.2.2 HaDeS

To be able to deploy complete systems starting from the bare hardware, we have developed a new deployment system called HaDeS (Hardware Deployment System). Using standard SOA mechanisms, HaDeS is able to communicate with the orchestrator to properly control the critical steps of the deployment process such as creation of a specific disk partitioning, filesystems, and the population of partitions with the operating systems. Moreover, it is agnostic with respect to the operating system to be deployed. This is a major benefit for two reasons: first, we want to be able to support a large number of OS types with different installation methods; secondly, we want to be able to deploy a pre-assembled image of an operating system that can't be installed with traditional methods. These are requirements that systems such as RedHat Kickstart [kic], or Suse Yast2 [yas], or even Debian FAE cannot meet, while other deployment systems, such as SystemImager [sys], are not designed to negotiate with an orchestration system. HaDeS is similar in spirit to kadeploy [kad], albeit with a different implementation, but, in contrast with the latter, it has been designed since the beginning for integration with a SOA architecture.

### 5.2.3 PFC

Operating system deployment requires that physical nodes are rebooted at least two times: first to load the deployment tool, and then to boot the deployed image. In fact, the deployment operation needs not only the node's reset or power off/on: reboot is also required to load a different, pre-deployed operating system on another partition, or on a network-mounted one, or on a diagnostic tool resident on a different storage media. Machine reboot is required every time the operating system needs to be changed. For this reason a dedicated component of the CyberSar Architecture
has been developed: PFC (PlatForm Control). Reboot operations could be easily performed using different kinds of services running on the node, such as SSH or a dedicated reboot-service, but all of these require the operating system to be up and running. Moreover, node reboot operations cannot be done via software invocation if a machine is powered off or its operating system is hanging. A solution that doesn’t require any loaded operating system to manage power operations (like turning on-off and resetting a node) is offered by IPMI (Intelligent Platform Management Interface) [ipmi] interfaces. IPMI is a protocol that allows administrators to talk to an onboard hardware controller, the BMC (Board Management Controller). In order to do so remotely, the IPMI card should be used over a LAN. This would allow to issue commands to the BMC in order to toggle power to the node, even if it is powered off or the operating system is hanging. Moreover, it allows to read all the sensors’ values on the board, from voltage and CPU/chassis temperature to fan speed and operational status. To use the IPMI protocol over LAN, an IPMI-enabled Network Interface Card (NIC) is connected to a control plane network. This NIC could be dedicated to manage only IPMI traffic or could be shared with the OS. In the latter case, IPMI traffic is usually not visible to the overlying network stack, and the OS is not aware of it. There are two ways to communicate with the BMC via IPMI: by means of the OpenIPMI library or with a tool called ipmitool. Both methods can be used for IPMI over LAN and cover most of IPMI specs, both in version 1.5 and 2.0. In CyberSar we chose to adopt the ipmitool solution because it’s easier to setup and use than OpenIPMI. As stated before, all the components in the CyberSar Architecture are implemented as Web Services. The Web Service that handles node power switching via IPMI is called PFC (PlatForm Control). PFC implements a WSDL interface that allows the user to issue the reboot command using SOAP messages. The most important operations are:

- set_power_on
- set_power_off
- set_power_reset
- issue_multi_command
As can be easily guessed, the first three operations are used to send a single command to a single node, while the fourth one can be mainly used in the deployment phases; it accepts an array of IPMI IP addresses as an argument and one command. These are issued simultaneously to all the BMCs, thus allowing the entire cluster to be rebooted and deployed.

### 5.2.4 VMH

VMH (Virtual Machine Host) is the service responsible for the instantiation and control of the virtual images inside a real host. Each computing node in the architecture is equipped with a web service. When the VMH web service receives a request for the creation of a new node, it calls the virtual machine hypervisor’s console to create a new virtual machine instance. It is also responsible for the management of all subsequent requests for virtual nodes lifecycle management.

From a theoretical point of view, the lifecycle of a virtual host can be represented as a state diagram characterized by five states.

![State Diagram](image)

**Figure 5.3: The lifecycle of a virtual host.**

As depicted in the figure above, the lifecycle of a virtual host starts from the Initial state in which the virtual host is only a representation of a potential running host. When the virtual host is in this state, we call it a Virtual Host Image (VHI). A VHI is characterized by an operating system, a configuration and some representation of the initial conditions. We can represent it with a triple in the space of the virtual hosts:

\[
(OS_i(t_0), C_i(t_0), Ic_i)
\]

(5.1)

where we use \( OS_i, C_i \) and \( Ic_i \) to indicate, respectively, the operating system, its
configuration and the initial state (initial conditions) of the virtual host. When the user decides to create a new virtual host, the VMH turns a VHI into a running virtual host (VH) by instantiating it on a physical host. This transition is initiated by a create event. After its creation, a virtual host evolves into the running state. A VH is represented by the same triple of the starting VHI at the time $t$, plus the reference to the VHI from which it was created:

$$ (OS_i(t), C_i(t), Ic_i, Ref(VHI_i)) $$  \hspace{1cm} (5.2)

From this state, a VH can be destroyed, paused or frozen. When it is paused, all of its operations are suspended, while when it is frozen, the controller serializes it into a file and releases all the resources associated with that particular VH. From this state it can be instantiated again in the same machine or, after a migration procedure, in another one. Since the freeze method and the migration functionalities are not supported at the moment, the previous state diagram can be reduced to the one reported in figure 1.b.

Every operation is mapped into a web service method that allows to completely control each virtual machine via SOAP invocations. When a service method is invoked, the VMH asks the hypervisor, using the Xen API, to execute the corresponding operation on the specified virtual machine. Since Xen’s hypervisor console API is written in Python, it was natural for us to also develop VMH methods in Python.

### 5.2.5 Machine Manager service

The Machine Manager service provides a unique operation called $createMachine$ which allows for the creation of a physical machine by invoking both Hades and PFC services.

The $createMachine$ operation receives the host name of the machine that needs to be installed and the name of the setup file which contains installing information for Hades. Hades is invoked by means of the $Control_install_host$ operation in order to prepare machine initialization setup data. After that, Hades is invoked again by means of the $Control_Gethost_Info$ operation for retrieving the IPMI address of the installed machine.
5.2.6 Virtual Machine Manager service

The Virtual Machine Manager is the core service of the architecture and it is also the most complicated one. For each allocation ticket it stores all the information related to the physical and virtual resources and communicates with four different services: the Resource Manager, the Machine Manager, the Notifier and the VMH. Each Virtual Machine Manager session manages one ticket information and it is opened by means of the open operation which receives the ticket value from the User Application. The ticket value is used by the JOLIE interpreter as a correlated variable in order to identify each running session. When a session is opened, the VMM initializes its machine data by sending a request to the Resource Manager for the list of the allocated machine for that ticket. After that the Virtual Machine Manager provides a set of available operations for the User Application:

- get_machineList: returns the updated data of physical and virtual resources

- install_machine: allows for the initialization of a physical machine by receiving the host name and the setup file name. This is performed by invoking the createMachine operation of the Machine Manager. After that it registers itself to the Notifier in order to be notified when the corresponding VMH is available.

- create: allows creation of a virtual machine. It receives a hostname and a virtual machine description, invokes the create operation of the related VMH and finally returns the id which identifies the created virtual machine on that physical resource

- start, stop, pause and unpause: receives a hostname and a virtual machine identifier and allows for the starting, stopping, pausing and unpausing of the virtual machine identified by the received identifier respectively. This is achieved by invoking the start, stop, pause and unpause operations of the VMH installed on the given host. If the operation is performed correctly, it updates the state of the virtual machine (running, stopped and paused) coherently and then returns the updated data of the cluster to the User Interface

- notify_listener: receives a notification from the Notifier that the requested VMH is started up
5.2.7 Virtual Resource Manager Service

The Virtual Resource Manager service is responsible for the virtual resources allocation. It allows the user to book resources and maintains all the information regarding the resources’ allocation status.

When a user asks for some virtual resources creation, the Virtual Resource Manager calculates the number of physical nodes needed in order to support the user’s request according to a predefined allocation policy. After this step, it asks the Resource Manager to allocate the physical machines. The Resource Manager reserves the devices and return a ticket for that reservation. The Virtual Resource Manager stores the ticket with the relative physical machines details into its inner database. If this operation succeeded, it creates another ticket and sends it back to the user.

After the physical resources reservation, the Virtual Resource Manager service executes all the operations for virtual machines’ creation. For each physical node, it asks the Machine Manager to create the virtual machines according to the user’s virtual machine blueprint. After the virtual machines creation, the Virtual Resource Manager stores all the informations related to the virtual machines in its database.

5.2.8 Resources Manager

The Resources Manager service is responsible for physical resources allocation. Before any other operation is performed, such as virtual machine creation or destruction, the Virtual Resource Manager must obtain a ticket from the Resources Manager. This is necessary for the creation of a virtual cluster. First of all, the Virtual Resource Manager has to ask the Resources Manager to reserve the nodes, specifying the desired hardware features according to a cluster blueprint. If the resources are available, the Resources Manager reserves them, associates them with a ticket and finally returns the ticket to the Virtual Resource Manager. The ticket is needed for all subsequent operations on the system. The Resources Manager is also a helpful tool for other services that wish to know about resource informations. It exposes methods to list the resources associated with a ticket and to extend their allocation period. At the moment, the Resources Manager implements a simple resource allocation algorithm based on the verification of their availability, but our future plans include the support for more sophisticated algorithms as well as the possibility to
reserve resources’ allocation according to some billing mechanism.

5.2.9 Vlan Manager Service

The Vlan Manager is the service in charge of administering the network to support the communications among the virtual machines.

In order to make the management of the virtual clusters communications easy and secure, also networking resources has been virtualized. All the virtual machine belonging to the same virtual cluster can communicate with each other by participating in the same vlan.

The Vlan Manager Service supports the following operations:

- \texttt{createVlan}: This operation allows to create a new vlan, giving the list of the machines that are hosting the virtual machines.

- \texttt{deleteVlan}: This operation allows to delete a vlan.

Once the virtual machines are deployed, a vlan has to be created and each virtual node must be connected to it. The Virtual Machine Manager sends a \texttt{createVlan} request to the Vlan Manager service, giving the list of the machines on which virtual machines are hosted. The Vlan Manager has an internal database with the list of all the machines, the devices under its control and all the informations regarding the network’s topology. When it receives a request for a new vlan, it matches the list of servers against its database and retrieves all the informations regarding the switches on which the nodes are attached and on which port. After have these informations been collected, the Vlan Manager call the Switch Manager Service to configure the network’s devices.

5.2.10 Switch Manager Service

The Switch Manager is the component responsible for the management of the switches. Using the DMS software, as described in the 4th chapter of this thesis, it exports all the functionalities for the creation and the management of vlans using a web service interface.

The Switch Manager is a web proxy that allow to execute complicated sequence of operations for the configuration of the devices. Each device is referenced using
an unique identifier that is passed by the Vlan Manager to the Switch Manager to execute the operation on the specific switch.

The operations available through the Switch Manager interface are:

- \textit{setVLANRequest}: This operation allows to create a new vlan giving a number and an optional name.

- \textit{deleteVLANRequest}: This operation allows to delete the specified vlan.

- \textit{movePortToVlanAsTaggedRequest}: In order to attach a device to a vlan, the port on which the device has been connected must be configured. To allow the device to see the vlan, its port has to be added to the specified vlan as a tagged port. The use of tagged port is useful for the isolation of the traffic shared among the virtual machines. In fact, each virtual machine is equipped with a virtual network interface that is bridged to the real network interface of the hosting machine; when a new packet arrives, if it is tagged the network interface forwards it to the virtual network interface, otherwise it treat it as a normal network packet. This solution allows the vlan of the virtual machines to be completely isolated from the rest of the network’s traffic, thus assuring an high level of security.

- \textit{deleteVLANportRequest}: This operation allows to delete the specified port from a vlan.

- \textit{getInfoRequest}: This is an utility operation that allows to retrieve some useful informations about the state of the switch as well as its running configuration. These informations are useful to maintain the status of the devices under control and to signal potential malfunctionings of the network.

The Switch Manager Service makes use of the multi-vendor support for different devices of the DMS library, as reported in the 4th chapter, through the use of different specific device drivers. For each different model of the switches in the network, a specific driver is used.

5.2.11 User Application

The User Interface provides the user with an easy and intuitive graphical tool for operating on both the physical and the virtual cluster. The interface is launched
5.2. Architecture design and implementation

together with a Virtual Machine Manager Client developed in JOLIE which takes care of catching events from the user window and sending messages to the other services. In particular, window commands cannot be enabled until the user authenticates herself on the Authentication service by sending her credentials. After this, the other panes are unblocked and the user can start creating both the physical and the virtual cluster.

Figure 5.4: UI screenshot.

5.2.12 Notifier

The Notifier simply receives notification from each started VMH on the subscribeVMH operation and then forwards it to the Virtual Machine Manager session which has been registered for receiving it to the notify_listener operation.

5.2.13 Account service

The Account Service is the service responsible for user authentication. It has only one method, "authenticate", which takes the username and password parameters and checks them against the corresponding values stored in the users’ database. At the moment the Account Service uses an internal standard relational database to store user information but, in the future, it will also support integration with an existing directory service.
5.3 Tests Evaluation

In order to prove the feasibility of this solution and to collect some data for performance evaluation, several tests have been conducted.

As depicted in the figure 5.5, the environment layout for test execution was composed by a pool of seventy-two computing nodes, organized on three separated clusters. Each computer has two network interfaces. While the first one was used for the usual operations of cluster communications and data transfer, the second one was plugged to the control network plane via the IPMI interface, and was accessible only for management purposes. Each component has been first tested singularly, in order to verify its proper functioning:

- *HaDeS and PFC*: The first component tested was the deployment system. In order to evaluate the system’s response to a request for a massive deployment, a complete reconfiguration of the entire cluster has been done. First of all the machines were turned off, and then, using the procedures described in the previous sections, a new cluster setup was installed. At the end of this process all the machines were rebooted. Sixty-four of the seventy-two machines went up correctly while the remaining eight had some problems related to hardware
and network outages.
The average total time of deployment was about ten minutes, from the time
at which the first machine was forced to reboot itself, until the last one was
up and running.
Most of the deployment time was spent transferring images between the image
repository and the target node. Since this operation was done via a com-
mon client-server approach, using a standard sequential download mechanism,
when the reconfiguration command started, the image repository was flooded
by requests for image download. Since the result of these requests, that were
treated sequentially, are chunks of data with an order of magnitude of a gi-
gabyte, the image repository was over-stressed and quickly became unable to
handle all the requests. Moreover, since all the nodes were downloading a big
image, the network quickly saturated.
Even though test results demonstrate the feasibility and manageability of our
solution, they also pointed out that the main bottleneck is the disk image
download system. At the moment we are working on the development of a
new approach based on the use of a peer to peer torrent mechanism. The
preliminary results seem very promising and encourage us to further explore
this solution.

- **VMH**: During deployment tests, an image with VMH was installed on every
node. We created a shell script to feed each node with a request for the
creation of four virtual machines. This operation can be split into two different
operations. The first one is the virtual image download. The second one is the
creation of the virtual machine over the hypervisor software. During our tests,
using the system as it was, we measured an average total time of several hours.
This was caused by the inner slowness of ftp, that is the mechanism adopted
by VMH to download the images. To reduce total download time there are two
possible solutions: the first one is to multiply the available ftp servers, thus
complicating the system’s architecture; the other one is to manually pre-fetch,
via a parallel manual script invocation, the virtual image on the node’s disk.
This solution took about twenty minutes but it can’t be done automatically.
We expect to be able to drastically speed up this process by using the same
peer to peer mechanism described for disk images download.
• **Workflow services:** In order to test the functionalities of the workflow services, we created two virtual clusters using the UI, starting from a cluster blueprint, using an XML document based approach for nodes description. For each virtual cluster, we deployed four virtual nodes on three physical ones in order to obtain a twelve virtual machine cluster.

We tested all the functionalities of each workflow service starting from authentication on the User Application to the start, stop and pause operations on each virtual machine.

In particular, after user logon, we requested a ticket to the Resource Manager for a three physical node cluster. Once we received the list of available machines we initialized each of them by invoking the Machine Manager. When each machine was successfully initialized, we installed four virtual machines on each physical one. In the end, for each virtual machine, we tested the start, stop and pause functions.

This preliminary test allowed us to successfully verify the correctness of the SOA logic as described in the previous sections. At the moment we implemented the workflow services in the JOLIE language. In the near future, we are planning to develop the same services using the BPEL language in order to be able to make comparisons between these two languages. We will evaluate performance in terms of speed of execution as well as scalability.

• **Applications:** Once the system was up and running, some applications tests were carried on. These tests have been done with two goals in mind: first of all to test different algorithms for scientific purposes and, secondly, to outline the main problems and the bottlenecks of the entire architecture, from the application’s point of view.

At the end of the previously described tests, we assembled everything together and run a complete test. We created, using the graphical user interface, a virtual cluster starting from the bare hardware, thus proving the feasibility of our ideas.

Beyond the proof of the concept, test results have emphasized the presence of some bottlenecks in the architecture’s design, the main one being the sequential file transfer mode used to download the physical as well as the virtual images. This caused the network’s saturation which had a tremendous impact on the total average
time of deployment.
CHAPTER 5. Virtual Cluster Management
Chapter 6

Conclusions and Future Works

We have started this work presenting how the Service Oriented Computing, an emerging programming paradigm for the development of distributed software architectures which has in the Web Services its most promising implementation, can overcome some of the limitations of the actual architectures for network management. In particular, we have focused our work on the automatic network administration by means of different network web services composition, and on the integration between the network and IT applications, by means of coordinated services workflows that expose their interface using a common formalism. We have investigated the main uses of this approach in the field of telecommunication and we presented a brief overview on the state of the art in the application of SOA principles in the telecommunication world. In the following we remind the most important contribution of this thesis:

- We have presented a new approach in the modeling of a network administration architecture. It is based on four abstraction layers. On the bottom there are the network resources. In the second layer, the simple service layer, the network devices are wrapped by web services that exposes, through a standard WSDL interface, their main administration functionalities. The composition of these services, allow to create more complex services such as the creation of virtual networks. On the top of this structure there is the application layer that exposes the functionalities of the network services to the outer world.

- Starting from the development of an object oriented model representation of the network devices, we have developed a software library, the Devices Management System (DMS), for the management of network devices. Using a
CHAPTER 6. Conclusions and Future Works

component based approach, we grouped together the network functionalities and define different objects for each network class of devices. Then we created the web services that export the network devices functionalities, using the DMS library for the development of their methods implementation.

• We have showed how our approach can be applied in a real context to the management of virtual computing clusters. We analyzed the main services involved in the creation and the management of virtual clusters and we reported on the use of the DMS library for the management of the vlan associated with the virtual clusters. Though this work, we practically proved how a SOA based approach in the designing of such a system, can easily help to integrate network services with application services.

In the future we intend to continue our investigation by focusing on two different directions: On one hand, we intend to continue to develop our reasonings in the application of the SOA concepts to the main network management topics. In particular we will focus our attention on the development of a SOA based approach for the deployment of new QoS services. We also want to collect some tests results on the performance of our approach in various network deployment scenarios.
On the other hand, we will continue to develop our network management architecture by adding the following features:

• Errors handling. We want to define an unique XML based language for the representation of the network services faults as well as for the application services ones. This will help to easily integrate the network faults management processes into the application workflows.

• Automatic devices interface generation. Starting from the object model representation of a device, we would like to have the system to automatically generate the corresponding web service interface. This will help in the development of the simple services and will let the designer to focus his attention on the development of the workflows for the services composition and orchestration.

• Support for different orchestration languages. At the moment our software only support the BPEL and the Jolie languages but we want to investigate the
use of different orchestration languages.

- Extend our network object model. We will represent more devices functionalities and we will support more devices coming from different vendors. We also want to include in our model also application devices such as printers or cellular phones.
Bibliography


[AMGZ07] Paolo Anedda, Simone Manca, Massimo Gaggero, and Gianluigi Zanetti. Xhpc: Soa based control plane for virtual clusters. In VHPC’07: Proceedings of the Workshop on Virtualization in High-


BIBLIOGRAPHY


# Index

Background, 21

Service Oriented Architectures
  Goals and Benefits of Service-Oriented Computing, 28
  Introduction to Service-Oriented Computing, 21
  Service Compositions, 23
  Service Inventory, 23
  Service Inventory Blueprints, 26
  Service Model, 25
  Service-Orientation, Services, and Service-Oriented Solution Logic, 22
  Service-Oriented Analysis and Service Modeling, 27
  Service-Oriented Architecture, 22
  Understanding Service-Oriented Computing Elements, 23
  Web Services, 28
    Jolie, 54
  Overview, 29
  WS-BPEL, 37
  WSDL, 34

Conclusions and Future Works, 123

Introduction, 13
  Aims of the Thesis, 16
  Outline, 19

Overview, 13

Related Papers, 19

State of the Art: SOA Network Management, 65

Business context and technologies of web services in telecommunications, 66

Deployed implementations of Web services-based systems, 69

Dynamic composition of Web services, 66

Overview, 65

Performances, 68

Virtual resources management, 70

The Device Management System Software, 73

Overview, 73

Software Architecture, 76

The device management library, 80

The VLan Creation Example, 87

Virtual Cluster Management, 103

Overview, 103

  Virtual Computing Facilities, 103

Tests Evaluation, 118