Master Thesis

in the European Master on Software Engineering EMSE

A Traceability Model for Management of
Service-Oriented Landscapes

by

Gabriel Hernando Lozano Martínez

Examiners: Prof. Dr. Paul Müller
Prof. Dr. Oscar Dieste.

Supervisor: Dipl. -Inform. Aneta Kabzeva.

ICSY - Integrated Communication Systems
Technische Universität Kaiserslautern
Universidad Politécnica de Madrid

April, 2011
I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Por la presente declaro que he creado esta obra totalmente por mi cuenta y que no he utilizado ninguna otra fuente o instrumento de los que aparecen, y que he señalado en las citaciones en consecuencia.

Kaiserslautern, April 2011
Gabriel Hernando Lozano Martínez
# Contents

1 Introduction ......................................................... 1  
  1.1 Context .......................................................... 1  
  1.2 Scope and Structure of this Work .............................. 4  

2 Basics ........................................................................ 5  
  2.1 Capabilities to Solve or Manage People and Organization’s Problems ............... 5  
  2.2 SOA as an Organization Framework to Match Needs and Capabilities ................. 6  
    2.2.1 Definition of Service-Oriented Architecture ................................. 6  
    2.2.2 Service ............................................................. 7  
    2.2.3 Service-oriented Design Principles to Obtain the Benefits of Service-Oriented Solutions ....................................................... 8  
    2.2.4 Service Consumer and Service Provider ........................................ 9  
    2.2.5 Horizontal Layers Composing a SOA ............................................ 11  
    2.2.6 A Mechanism to Bind Service Consumers and Service Providers .......... 11  
    2.2.7 Which Information is Required to Interact with a Service ................... 12  
    2.2.8 SOA Implementation Technologies .............................................. 13  
  2.3 Governance and Management of Service Oriented Architectures ...................... 13  
    2.3.1 What Can Be Governed? ...................................................... 15  
    2.3.2 How to Implement SOA Governance? ........................................... 15  
    2.3.3 Motivations for Management ..................................................... 16  
    2.3.4 Domains of Interest for Management ........................................... 16  
    2.3.5 Management of Resources ....................................................... 16  
    2.3.6 Properties of Resources to Be Managed ....................................... 17  
  2.4 SOA Challenges ...................................................... 17  
    2.4.1 Service Descriptions to Support Interaction .................................. 18  
    2.4.2 Infrastructure Required for Governance ..................................... 18  
    2.4.3 Requirements for Management ............................................... 19  

3 Requirements for SOA, Governance and Management .......................... 21  
  3.1 A mechanism for Awareness .......................................... 21
A Solutions Review
A.1 Models and Tools for SOA Governance ................. 81
A.3 A Semantically Enhanced Repository ..................... 83
A.4 Software AG ........................................... 83
A.5 Oracle ................................................. 87
A.6 UDDI .................................................. 87
A.7 IBM ................................................... 87
A.8 SAP .................................................... 91

Bibliography .................................................. 93
# List of Figures

2.1 Enterprise layers. *Source: [Erl, 2006]* ........................................... 8
2.2 Enterprise layers from *The Open Group Reference Architecture*. *Source: [Harding, 2009]* ......................................................... 10
2.3 SOA Example. *Source: [web, 2011]* ....................................................... 14

3.1 Mediated Awareness *Source: http://www.w3.org/2003/Talks/0521-hh-usa/soa.png* 22

4.1 Artifacts Metamodel ................................................................. 30
4.2 Description Type Metamodel ..................................................... 33
4.3 Description Type Categorization Metamodel ................................. 34
4.4 Roles Metamodel ........................................................................ 34
4.5 Processes’ Metamodel ................................................................. 35
4.6 Static Process ............................................................................. 36
4.7 Relation of *Process and Process Instance* .................................... 37
4.8 Relation of *Process, Process_instance, Role and Participant* ............. 37
4.9 Relation of *Process, Process_instance, Artifacts and Artifact instances* . 38
4.10 Policy Model ................................................................................ 40
4.11 Policies and Contracts Model using the artifacts metamodel ............... 41
4.12 Contracts model .......................................................................... 42
4.13 Description Type Model ............................................................... 42
4.14 Associated Annotations Artifact .................................................. 43
4.15 Service Description Model ........................................................... 44
4.16 Service Functionality Model ........................................................ 46
4.17 Service Interface is composed by service behavior and information models . 47
4.18 Information model ....................................................................... 48
4.19 Information model example - *Year data artifact* ............................. 49
4.20 Service behavior model is composed by action model and process model . 49
4.21 Action model .............................................................................. 50
4.22 message, message exchange, participants, service, action and real-world effect [Oasis, 2009, p. 71] ......................................................... 51
4.23 Request/response and event notification message exchange patterns [Oasis, 2009, p. 73] ................................................................. 52
4.24 Process Model ........................................................................ 53
4.25 Service Reachability Model ..................................................... 54
4.26 Service Relational Model ........................................................ 56
4.27 Service policy model ............................................................... 57
4.29 Service Category Model .......................................................... 59
4.30 Roles Model ........................................................................... 60
4.31 Business Domain Representatives Model ................................. 61
4.32 Service Development Team Model .......................................... 62
4.33 IT Operations Model ............................................................... 62
4.34 Service Lifecycle ................................................................... 64
4.35 State chart diagram of Service States ....................................... 73
4.36 Review matrix of the solutions ................................................. 76
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Design-time phase activities, roles and artifacts</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>Run-time phase activities, roles and artifacts</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>Change-time phase activities, roles and artifacts</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary of the requirements to evaluate SOA solutions</td>
<td>75</td>
</tr>
<tr>
<td>A.1</td>
<td>Models and Tools for SOA Governance</td>
<td>82</td>
</tr>
<tr>
<td>A.2</td>
<td>STraS: A Framework for Semantic Traceability in Enterprise-wide SOA Life-</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>Cycle Management.</td>
<td></td>
</tr>
<tr>
<td>A.3</td>
<td>A Semantically Enhanced Service Repository for Service Oriented application</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>System Development</td>
<td></td>
</tr>
<tr>
<td>A.4</td>
<td>Software AG solution.</td>
<td>86</td>
</tr>
<tr>
<td>A.5</td>
<td>Oracle solution.</td>
<td>88</td>
</tr>
<tr>
<td>A.6</td>
<td>UDDI standard.</td>
<td>89</td>
</tr>
<tr>
<td>A.7</td>
<td>IBM solution.</td>
<td>90</td>
</tr>
<tr>
<td>A.8</td>
<td>IBM solution.</td>
<td>92</td>
</tr>
</tbody>
</table>
Abstract

During the work of this thesis, a set of meta-models, models and a life-cycle related to Service-Oriented Architectures was done. They contain the elements that are commonly found in this kind of architectures together with the relations among those elements. These elements and their relations were obtained based on the information provided in several reference models and reference architectures documents. These documents were developed by consortiums of organizations with experience in the design and implementations of systems based on Service-Oriented Architectures. The meta-models and models developed provide a unified view of the information contained in the different documents. To show the viability of these meta-models and models, the information contained in them is represented in a given implementation technology, like web services. A service-lifecycle, that associates participants, service states, artifacts and phases is provided. The meta-models, models and the service life-cycle form the basics for the management and governance of SOA based systems, which in turn increases their chance of success.
Chapter 1

Introduction

1.1 Context

Entities (people and organizations) create capabilities to solve or support a solution for the problems they face in the course of their business [et all. MacKenzie et al., 2006, p. 8]. A business in this context must be interpreted as any activity that entities are engaged in [Oasis, 2009, footnote on p. 9]. If a business of an entity changes over time, either new capabilities should be created or the existing ones should be adjusted in accordance. Let us illustrate these concepts with an example. A bank, let us call it The National Bank, have created the withdrawing cash capability that allows their clients to take money from their bank accounts in their country of residence. Let us suppose that The National Bank now allows customers to withdraw cash in more than 140 countries around the world. In that case, the withdrawing cash capability must be changed to support this new functionality.

For four decades, entities have accumulated IT resources. Those IT resources should have delivered a set of application capabilities with the potential to adapt efficiently to any business change [Group, 2007, p. 10]. This potential did not turned into reality. Why? Because application capabilities were not designed with those goals in mind. Continuing with our example, when The National Bank needed the withdraw cash capability, an application that implements this capability was developed. When the bank decided to allow customers to withdraw cash from several countries around the world, the initially created system had to be completely refurbished to be able to interact with other systems. Big amounts of development time and money were spent. To this respect, Software AG states: “The inability of inflexible, tightly coupled legacy systems to respond quickly and effective to business needs is a key reason companies have invested in flexible, loosely coupled SOA” [Castaldini, 2008, p. 3].

Service-Oriented Architectures (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under control of different ownership domains [et all. MacKenzie et al., 2006, p. 8]. It defines an abstraction layer that decouples businesses
from IT resources. This layer is composed by services. A service is a mechanism that joins needs and capabilities [et all. MacKenzie et al., 2006, p. 9]. Instead of directly linking businesses to IT resources, businesses are expressed in terms of services that encapsulates the interaction with the underlying application logic. Services should be designed in such a way that given a change in the business logic, they allow SOA-based systems re-accommodate quickly and efficiently. In the case of The National Bank, If the withdraw cash service is properly designed, it would be much easier to create a second one that deals with international cash withdraws and interconnect both of them.

Given the fact that business and IT resources are through services decoupled, several scenarios of service usage are expected. For example, if The National Bank decides to have an agreement with another bank, lets call it Bank of Progress, it would be easy to the Bank of Progress to use the services created by The National Bank. Or if the bank decides to offer on-line banking, the new system may be based on existing services. A third, fourth, fifth,... a priori unknown scenarios of service usage may emerge in the future. By decoupling business and IT resources, solutions are organized in a way that promotes reuse, growth and interoperability [et all. MacKenzie et al., 2006, p. 9].

The decoupling of customers and services raises some challenges. The first one is, how can a potential customer know the existence of a service that could be useful? A complete description of the services is needed. After potential consumers obtain the description of the services, they can evaluate if a service that fulfills their needs is offered. For instance, if the Research and Development department of The National Bank wants to develop the on-line banking system, they will search for services that, for example, allow to search the data of a client’s account, a service that allows transactions, etc. Complete description of services includes information about the mechanism to interact with them. For example, the description may specify that the service is access through web services technology together with the actions and operations to interact with it.

Now, let us assume that the Bank of Progress wants to close the agreement with the The National Bank that allows them to use some of the other bank’s services. But some executives from Bank of Progress are not sure about the reliability of the services and capabilities offered by the The National Bank. Moreover, they are not sure about the platform or infrastructure that supports the interaction with the services offered by The National Bank. Is there a reliable and secured communication channel? Is the performance of the services good enough? Are services available? A second challenge in SOA-based systems is to reach an agreement on the governance framework and the processes to manage it, in order to have a predictable interaction among the participants. Some policies, rules and regulations must be established between the participants. Those policies, rules and regulations are defined within the governance framework. The processes to manage the framework, i.e. the processes that control the creation, agreement, change, enforcement and retirement of policies, rules and regulations should also be defined. For example, in the case of the banks agreement, a process can define that to change a policy, the The National Bank must communicate this to the Bank of Progress 21 days before. In the case of the banks example,
the agreement should include a governance framework and the processes to manage the relation between them. A common way to represent the policies, rules, regulations and processes, understandable and agreeable by the different participants is needed.

The services provided by the *The National Bank* are so good, that not only *Bank of Progress* but also several organizations in the financial sector are using them. Furthermore, some of the clients have even asked for add-ons to the services. For example, there may be a client needing a better performance of the service or another client that asks for a slightly modification in the functionality of the service. To manage this, the *The National Bank* created several versions of the same set of services. For each version, several instances are there. As soon as there is the possibility of more than one instance of a resource, the issue of managing those resources becomes relevant, states the Oasis SOA Reference Architecture [Oasis, 2009, p. 102]. Manage of services includes manage of the description of the services, manage of the policies and contracts associated to a service, manage of the participants and also manage of the IT resources and applications that support the service. Some properties of resources that can be managed are their life-cycle, their configuration, events related to the resource monitoring, accounting (the measure of the use of the resources), quality of service and the policies associated to them [Oasis, 2009, p. 103]. All these are relevant aspects when managing a service. For example, if a database fails, the versions of the services that depend on the failing database must be identified together with the policies and contracts associated to them. If the policies say that in the case that a service stops working customers must be communicated, the customers of the service must be found and the failure on the given service must be communicated.

A Service executes its functionality through IT resources, like existing applications including J2EE and .Net applications, other services, legacy applications, databases, etc [Harding, 2009, p. 20]. Each of these resources may be under control of different organizations. For instance, a database can be controlled by one department, the code that runs the capabilities exposed by the service controlled by another department, and the service, i.e. the mechanism (e.g. a web service) that allows a client access to the capabilities offered by the service, controlled by another department or even another organization. The management of a given service depends on the management of each of the resources on which this service depends on together with the communication of the events associated to them. For example, the withdraw cash service offered by *The National Bank* may depend on a database that is not hosted by them. In this case, the administration of this service depends on the administration of the database and the communication of the events associated to it. To successfully manage SOA-based system, a mechanism that collects the events that occur in the system must be in place, that based on a model that relates the composing, distributed resources of the system, establishes the affected participants and inform them about these events. In some cases, properties related to the resources that compose the system may not be explicitly managed. For example, measures in the usage of a resource (e.g. measures on the usage of the database) may not be taken. A challenge in SOA-based system is to provide mechanisms that, in some way, collect the data needed
for the management of the resources. This would enable, in turn, the management of the dependent resources. Because of the dependency among the management of resources, policies in the governance framework may establish rules and regulations that force participants in charge of the management of the resources to communicate events of interest associated with those resources to the interested stakeholders.

1.2 Scope and Structure of this Work

In this work, a complete set of meta-models and models, describing all relevant aspects of services through related resources that may be under control of different participants of the system, together with the definition of processes that allow management and governance of SOA-based systems was developed. The resources that are described in the models that compose this work include services, service versions, service descriptions, policies, rules, participants, IT resources and applications that support the services. Models created during the development of this work include models for the description of the resources, models for the complete description of services, models of policies, contract and rules, models of the processes for management and governance of SOA-based systems, models of the participants together with the roles they play and finally models of dependencies among the resources. Those models should be able to depict a design-time and run-time SOA-based system.

In chapter 2 the basics on which this work is based are given. Chapter 3 summarizes the requirements for the successful implementation of a SOA-Based system. These requirements are the ones used to evaluate similar works and to base the created models in this work. In the first part of chapter 4 the approach to solve the requirements proposed before is introduced. Meta-models and models are created and described in detail in the second part of the chapter. In the final part of the chapter a comparison of similar works, from research -and commercial areas is shown. The work finishes with some conclusions in chapter 5, that summarizes the lessons learned during the realization of it.
Chapter 2

Basics

This chapter provides the fundamentals on which the work of this thesis is done. First, an introduction to the relationship and interaction between entities (i.e. people and organizations) and capabilities is given. Afterwards, SOA is introduced as a paradigm to organize this relationship in a novel way that makes entities and capabilities more effective. Then, governance and management as a requisite for successfully implementing SOA is presented. At the end of the chapter, a summary of the challenges to implement SOA, based on the all the previous concepts introduced, is done.

2.1 Capabilities to Solve or Manage People and Organization’s Problems

Entities (i.e. people and organizations) are involved in several businesses. In this context, a business refers to any activity that people are engaged in [Oasis, 2009, p. 9]. In the course of these businesses, people and organizations face problems. To solve or manage them, capabilities are in some cases created [et all. MacKenzie et al., 2006, p. 8]. It is also natural to think that these problems can be solved by a capability offered by some other person or organization. In computational terms, this is equivalent to say that one computer agent need can be solved or managed by a capability offered by some different agent [et all. MacKenzie et al., 2006, p. 8]. There is not necessarily a one-to-one relationship between entity needs and the corresponding capabilities. A given need can be solved or managed by several capabilities and also a given capability may cover several needs.

If those problems that entities face change over time, the associated capabilities to solve or manage them must be adjusted in accordance. This implies that capabilities must be flexible and agile enough to react efficiently to this change. In practical terms this means that the overall system should not be greatly affected by this new situation. In some cases, these capabilities take the form of software applications. In this case, the flexibility and agility rely in a great extend on having an architecture in which these software applications
can be based on, that support these requirements [Castaldini, 2008, p. 3].

Nowadays, the ability to answer efficiently to changes in the environment is important. For example, for commercial organizations in a highly competitive marketplace, the ability to satisfy ever-changing market demands in an efficient way is a necessary component to keep them in business [IBM, 2010, p. 2]. For organizations in general, the ability to express their businesses in terms of the capabilities associated and to be able to answer fast to changes in the environment is a key ingredient to achieve more flexible businesses [Braswell et al., 2007, p. 4]. It should not be forgotten that these capabilities may be owned by a different actor, which adds constrains to the solution.

2.2 SOA as an Organization Framework to Match Needs and Capabilities

In this section, Service-Oriented Architecture (SOA), as an way to organize the relation between entities and capabilities, is described. Definitions of SOA and Service are given. Then, the principles of service-orientation are given. The common participants, service consumers and providers, are introduced. Subsequently, the theoretical horizontal layers of the architecture are reviewed. The problem of service providers and service consumers matching is explained. In this scenario, a mechanism that matches both, providers and consumers, is needed. This need is depicted in this section. A matching mechanism includes the description of the required information for a service consumer to find an appropriate service. At the end, a brief discussion on SOA implementation technologies is given.

2.2.1 Definition of Service-Oriented Architecture

In the Oasis Reference Model document [et al. MacKenzie et al., 2006, p. 8] the following definition of SOA is found:

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.

The primary goal of SOA is to organize and utilize this distributed capabilities in a way that the business world and the world of information technology (IT) are more effective [Braswell et al., 2007, p. 4]. To this effect, SOA defines a set of design principles. When these design principles are applied to the capabilities, it results in standarized service-oriented capabilities. When a solution is comprised of units of service-oriented capabilities, it becomes what is referred to as service-oriented solution [Erl, 2006].

Benefits of a service-oriented solution includes, but are not limited to [Erl, 2006, Ch. 3]:

- improved integration;
• reusability;
• streamlined architectures and solutions;
• leveraging of the legacy investment;
• establishing standardized XML representation;
• focused investment on communications infrastructure;
• “best of breed alternatives” (due to the decoupling of the service interface and its implementation); and
• organizational agility.

Previously it was stated that there is not necessarily a one-to-one relationship between needs and capabilities. A need can be solve or manage by several capabilities and also a number of needs may be cover by a given capability. SOA is a paradigm for organizing capabilities because is perceived as a framework for matching needs and capabilities or needs and combined capabilities [et al. MacKenzie et al., 2006, p. 8]. The capabilities to solve or manage a problem can be owned by the actor having the corresponding need or by different actor(s). In the latter case the capability is under the control of a different ownership domain. Still, a component that brings together needs and capabilities is missing.

2.2.2 Service

There is a distinction between the capability and the ability to make it available. Therefore, a component that connects needs and capabilities is missing. The following definition of services is found in the Oasis Reference model [et al. MacKenzie et al., 2006, p. 9]:

**Services are the mechanism by which needs and capabilities are brought together.**

A service is considered a self-describing, independent unit of logic [Erl, 2006]. From a business perspective a service corresponds to real-world recognizable business functions or business activities that can be accessed given certain policies established. From a technical perspective, services are coarse grained reusable IT assets with well-defined interfaces [Newcomer and Lomow, 2004, p. 58]. Services define an abstraction layer between business and application logic that decouples the business design with the corresponding application implementation. This decoupling enables the business logic and the application logic to evolve independently as long a the service interface remains unchanged [Newcomer and Lomow, 2004, p. 58]. For this reason, the services layer enables enterprise-wide loose coupling [Erl, 2006, Ch. 9]. Figure 2.1 shows the conceptual enterprise layers. For a complete discussion on service, business and application layers, see [Erl, 2006, Ch. 9].
A service-oriented solution is comprised of units of service-oriented capabilities. If we say that a service-oriented solution is comprise of services, then services can be units of service-oriented capabilities. As previously stated, units of service-oriented capabilities are the ones in which service-oriented design principles are applied. Then, services can be redefined as units of service-oriented capabilities in which service-oriented design principles are applied. The benefits of service-oriented solutions were previously mentioned. What is left is to discuss about the service-oriented design principles.

2.2.3 Service-oriented Design Principles to Obtain the Benefits of Service-Oriented Solutions

The main objective of SOA is to define an efficient implementation that aligns business and application logic, providing at the same time a flexible and agile framework. In order to develop such an architecture, a set of principles have been established in the design of services. There is not an standardization organization that mandates which are the principles that constitute what is called Service-Orientation. They originated from different opinions, public IT organizations to vendors and consulting firms [Erl, 2006, section 8.3].
A common set of service-oriented principles are: (taken from [Erl, 2006])

**Services are reusable:** Services are designed to support potential reuse.

**Services share a formal contract:** Services, services users and potential users only interact through a formal contract.

**Services are loosely-coupled:** Services must interact without the need for tight, cross service dependencies.

**Services must be abstract:** Services must abstract users from the underlying logic.

**Services are composable:** Services may compose other service.

**Services are autonomous:** All the service logic resides within a specific boundary.

**Services are discoverable:** Services should allow their descriptions to be discoverable and understandable.

The application of these principles in the design of services is a necessary requisite to obtain an agile and flexible enterprise architecture. In [Seedorf et al., 2009] the following statement is found:

...A SOA not only promotes interoperable, loosely coupled enterprise applications; it also reduces the conceptual gap between business and IT.

The need for such an architecture has been the driving force for the development of SOA. *Legacy systems* are considered inflexible and tightly coupled and, because of this, unable to respond quickly and effectively to business needs [Castaldini, 2008, p. 3]. A deeper discussion on the principles of service-orientation can be found in [Erl, 2006, Ch. 9].

A service gives access to people or organizations with one or several needs to capabilities that can be locally owned or owned by a different person or organization. This introduces the concepts of service consumer and service provider.

### 2.2.4 Service Consumer and Service Provider

The abstraction layer that defines services also defines two different participants in the architecture: **Service Providers** and **Service Consumers**. The Oasis Reference Architecture provides the following definitions for *service provider* and *service consumer* [Oasis, 2009, p. 40-41]:

A Service Provider is a participant that offers a service that enables some capability to be used by other participants.
A Service Consumer is a participant that interacts with a service in order to realize the real world effect produced by a capability.

A real world effect produced by a capability can be one of the next possibilities [et al. MacKenzie et al., 2006, p. 12]:

1. information returned in response to a request for that information,
2. a change to the shared state of defined entities, or
3. some combination of the previous possibilities.

It can be thought that a service consumer is the one that always initiates the communication with the service provider, but that is not true in all cases. For example, in a publish-subscribe communication model, the service consumer only initiates the communication for the first time, further communications will be very probably started by the service provider.

Taking the layers model from Erl [Erl, 2006] (see figure 2.1), the service consumer deals with the service process layer and the service provider with the application logic. Taking the layers model proposed in the reference architecture by The Open Group [Harding, 2009] (see figure 2.2), a service consumer deals with consumer interfaces and business processes and the service provider with service components and operational systems layers.

A service consumer can find easier the matching services offered by a service provider if these services are well organized and properly described. The perceived value of SOA
is that it provides a framework to match needs and capabilities [et all. MacKenzie et al., 2006, p. 8]. It should not be forgotten that service consumer and providers may belong to different organizations. For this reason, a mechanism that allows the service consumer to know the existence of a service provider and vice-versa is required.

2.2.5 Horizontal Layers Composing a SOA

Section 2.2.4 discussed new participants in the architecture as the result of the new abstraction layer, i.e. the services layer: service consumer and providers. Nevertheless, this architecture is based on previous components of previous architectures. Traditional layers like the operational systems and consumer interfaces, as defined in The Open Group Reference Architecture [Harding, 2009] (see figure 2.2), play a roll in SOA’s. In that reference architecture document, the following definition is found:

A layer is an abstraction which contains a set of components such as Architectural Building Blocks (ABB’s), architectural decisions, interactions among the components and interaction with other layers. [Harding, 2009, p. 15]

In the same document, the following definition of Architectural Building Block is found:

ABB is a constituent of the architecture model that describes a single aspect of the overall model [Harding, 2009, p. 15](see also [Harding, 2006]).

Examples of ABB’s are: existing databases or legacy applications in the operational systems layer, the software components which provide the implementation for a service in the service components layer, or the services in the services layer. ABB’s define key responsibilities of the layer. This organization of the layers is called horizontal because the composing ABB’s are not distributed and they have a priori knowledge of each other. ABB’s have also relation to ABB’s in different layers as can be seen in figure 2.2, which constitutes a natural association across layers [Harding, 2009, p. 15]. These associations are important for the management of SOA-based systems, as the ABB’s affected by a given change in the system must be calculated and interested participants must be notified of these events.

2.2.6 A Mechanism to Bind Service Consumers and Service Providers

Service consumer and providers can be completely independent participants in the architecture, without knowledge of each other. It is still possible that the services the provider offers are of interest to the service consumer. A mechanism that informs the service consumer about the existence of a given service provider is required. This is also valid for the opposite case, i.e. a mechanism that informs the service provider about the existence of a service consumer is also needed. The following describes a list of possible mechanisms for the binding of service provider and consumers [et all. MacKenzie et al., 2006, p. 14]:
• **A Service Provider registry.** This is a special case of service mediator. A *Service Mediator* would be a participant in the enterprise architecture that facilitates the offering or use of services in some way [Oasis, 2009, p. 41]. In this case, the provider registry is an a priori known participant by providers and consumers in the SOA-based system.

• **Broadcasting to Service Customers.** A Service Provider can broadcast its existence, hoping that an interested service customer know its existence and may contact it in the future.

• **A Service Customer Broadcasting.** Within this model, a service customer broadcasts its needs to service providers, hoping that one provides a capability and wants the customer to make use of it.

• **A Service Consumer probes a network.** A service consumer may probe an entire network, in the hope that an appropriate service provider, providing an specific service, exists.

  Awareness of each other is not enough to interact. There are several concepts that a service consumer must know about a service before actually interact with it.

### 2.2.7 Which Information is Required to Interact with a Service

After the service provider and consumer have knowledge of each other, the next step is that the service consumer obtains the information about the services the provider offers, to decide if, among these services, there is one that is suitable for its current needs, and also to decide if the consumer satisfies the requirements established by the provider to access the service. This information about the services is called the *service metadata*. The Open Group reference architecture states:

> ... The meta-data underlying each layer and relationship between layers can further facilitate in bridging the gap between business and IT from solution modeling to solution realization. [Harding, 2009, p 10]

In the case of SOA, the additional layer is the services layer and the metadata associated is the services metadata. What information is required to satisfy these prerequisites?

The *Oasis refence model* [et all. MacKenzie et al., 2006] establishes the concepts that must be described for a service in order that the consumer can find an appropriate service. These concepts are *visibility*, *interaction* and *real-world effect*.

*Visibility* includes *awareness*, *willingness* and *reachability*. *Awareness* is not only the knowledge of the existence of each other, i.e. service consumer and provider, but also of the services the provider offers. The concepts explained here, that describes the different facets of services, must be contained in a document that from here on will be called *service*
Willingness defines the intent from service consumer and providers to interact. This is highly related to the policies associated to a given service that must be present in the service description. Reachability defines the extend to which a service consumer and provider are able to interact. This is associated to the service access information like the communication channel and location of the service, among others.

After service consumer and provider have knowledge of each other, they have the will to interact and are reachable, the interaction model must be established. This includes an information model and a behavioral model. The information model characterizes the information that service consumer and provider may exchange in the interaction with the service. In turn, the behavioral model relates the information model, i.e. the actions needed to interact with the service, with the temporal aspects for the interaction.

The real-world effect determines the response for a request of some information or the change in the state of some defined entities shared by the participants.

A big part of this work deals with the modeling of these concepts, models that are shown and described in subsequent chapters.

Previous concepts, i.e. needs and capabilities, people and organizations, services, service design principles, the binding mechanisms and the service descriptions, form the bases of Service-oriented Architectures. Problems arise in SOA’s given that the system may be used across ownership boundaries. New concepts and challenges arise and are described in following sections.

2.2.8 SOA Implementation Technologies

SOAs are commonly implemented with web services (see [et all. MacKenzie et al., 2006]). Web services is very useful to understand SOA (see [Newcomer and Lomow, 2004]), but services can be visible, interaction can be supported and real world effects can be obtained with many different technologies.

2.3 Governance and Management of Service Oriented Architectures

One of the most important benefits of SOA is its flexibility. This can be seen in the fact that services can be used in several unexpected situations by different users from different environments that transcends domains of ownership. Figure 2.3 illustrates flexibility in SOA-based systems. Services provide a set of functionalities to many potential clients, leveraging existing investments in code. The applications that access the services in this example include [web, 2011]:

- a Dashboard application,
- a CRM application,
• a Help-desk application,
• a Financial application, and
• an Infrastructure application to manage other services and applications.

This flexibility requires confidence not only in the services but also in the underlying capabilities. SOA Governance deals with the establishment of a stable, reliable and extremely robust platform in which participants can interact in a dependent way, i.e. a platform that promotes service visibility, facilitates interaction among participants and makes sure that the results of service interaction are the ones promised in the service description [Oasis, 2009, p. 91]. This imposes a challenge, due to fact that several participants, from different ownership boundaries interact. For example, the owner of the service, a provider, the host and the access mediator, all may belong to different domains. There are strong limits on the control and authority of any of the participants and this is the main challenge of SOA governance: to establish the policies for service interaction that all participants agree to comply with. These policies together with procedures must ensure standardization across internal and external organization boundaries to promote the effective use and creation of services [Oasis, 2009, p. 85].

Management, in turn, is concerned with the administration of the resources, the realization and enforcement of policies and the control of the relationships of the participants.
in SOA-based systems. Through governance, the policies for the interaction among the participants are established. Through management, the enforcement of the policies is done.

### 2.3.1 What Can Be Governed?

Three different aspects of SOA may be governed: **SOA Infrastructure**, **SOA Inventory** and **Participant Interaction** [Oasis, 2009, p. 91].

Governance of *service infrastructure* is in charge of making sure that the required SOA services that enable participants to interact exist. These infrastructural services provide access to important business services, like messaging, storage, discovery or mediation. SOA Governance makes sure that these services are provided in a reliable, stable and robust way under all possible operating conditions.

Governance of *service inventory* defines which SOA services are allowed on the system. It defines well-behaved services based not only on the characteristics inherited from experiences in distributed computing but also on previous SOA experiences. To this concern, it is more important that services are well described to support an informed decision from potential users that actually define well-behaved characteristics of services. For this purpose, governance must define which attributes and values should be present in appropriate descriptions of services. It would also be good when the values of the attributes that describe a service can be certified by a known and recognized authority.

Governance of *participants interaction* must ensure adherence of service interfaces and reachability to parameters previously established. Governance would also require that the real world effects obtained from the interaction with a service are consistent with the ones promised in the service description. Besides, the real world effects must be consistent with the policies agreed for the interaction with services. This assurance of consistency may deal with contractual and legal aspects.

### 2.3.2 How to Implement SOA Governance?

Governance is important to SOA because it sets the policies that allow change management and evolution, establishes strategies for change, resolves disputes that arise and ensures that SOA-based systems continue to fulfill the business goals [Oasis, 2009, p. 86]. All these tasks of governance must be established before they are in use.

Governance should include the definition of the processes that implement these tasks. These processes must be communicated to the participants and must be accepted by them. Each process defines a set of tasks. For each task, it is defined the participant responsible for carrying it out. It should not be forgotten that all these processes must be created, communicated and accepted by participants in an environment that transcends domains of ownership. That is a big challenge when implementing SOA-based systems.
2.3.3 Motivations for Management

In SOA-based systems, proper control of the artifacts is an important aspect for their implementation. Without it, the flexibility of SOA-based systems can turn into chaos [Castaldini, 2008, p. 3]. There are a considerable number of resources in this kind of architectures that should be managed. If the interdependencies are considered, the number of moving parts increases exponentially. Different aspects of these resources like the life-cycles, configurations, event monitoring, accounting among others must be managed.

2.3.4 Domains of Interest for Management

There are three domains of interest for management: (taken from [Oasis, 2009, p. 102])

- the first is the management of the resources,
- second is the promulgation and enforcement of policies which is related closely to governance and third is the management of the relationships of participants in SOA-based systems that allows to the interaction among the participants and between the participants and the services they offer or use.

2.3.5 Management of Resources

In systems like SOA-based ones many manageable resources exist. As soon as there is a possibility to have more than one instance of a given resource, the issue of managing the resource becomes relevant [Oasis, 2009, p. 102]. In SOA, manageable resources are:

- services,
- service descriptions,
- policies and contracts,
- roles,
- relationships (for example between customer or potential customers and providers),
  or
- infrastructure elements.

In the case of SOA’s, the management of the resources represents a special challenge given the different ownership boundaries of related resources. The same example for governance applies here, where the owner of the service, the provider, the host and the access mediator may belong to different domains.
2.3.6 Properties of Resources to Be Managed

The management of a resource is a capability that allows it to be managed with respect to some property. The following is a list of attributes that can be managed from the resources: [Oasis, 2009, p. 103]

- **Life-cycle Manageability.** The life-cycle of a resource typically refers to how the resource is created, how it is destroyed and which dependencies might exist. The life-cycle manageability includes the versioning of the resources. Several versions may exist for a given service. For a consumer and a provider it is important to know with which version of the service they are interacting with.

- **Configuration Manageability.** This capability allows the configuration of the given resources to be managed. In the particular case of services the configuration can be very complex.

- **Event-monitoring Manageability.** This capability allows the configuration of the events that are wanted to be monitored.

- **Accounting Manageability.** This capability allows the measure and accounting of the usage of the resources.

- **Quality of Service.** A manageable capability that allows the quality of service to be managed.

- **Policy Manageability.** This capability allows the association and configuration of policies associated to resources. In the case of policies, management includes promulgating policies, retiring policies, and ensuring that enforcement points of policies are working.

2.4 SOA Challenges

The first part of this section explains the challenges to interact with the services that belong to the services layer.

With the services abstraction layer, greater flexibility is introduced in SOA-based systems. As already seen, this flexibility adds complexity to these systems because services can be used by unexpected consumers. SOA governance is proposed as a solution to manage this added complexity. The second part of this section deals with the infrastructure required for governance.

Governance defines the tasks and the participants responsible for each of these tasks. In order to carry out these tasks, participants must manage the resources that they are in charge of. SOA Governance includes the enforcement of the policies, which rely on the information obtained from the management of the resources composing SOA-based
systems. The third part of this section deals with the requirements for management in SOA-based systems.

2.4.1 Service Descriptions to Support Interaction

Section 2.2.7 discusses the concepts associated to the interaction with services. These concepts must be documented and accessible to the interested participants, in this case, service consumers and service providers. A Service Description is the document within the architecture in which the concepts required to interact with services can be found. There, information about the service functionality, access points to the service, the syntax and semantic of the messages to interact, the temporal aspects of interacting with the service and the effects of interacting with the service must be found. This description must be accessible to potential service consumers and complete, so that this participant can decide whether a service solves his or her needs or not. Given the amount of information contained in this document, a service description can also be thought as a web of documents where the different aspects related to it are references to external locations. The information that a service description contains has certain characteristics that must be taken into account when designing a solution to hold and maintain it.

The information contained in the service descriptions is inherently incomplete but is considered sufficient when it allows participants to access and use the services [Oasis, 2009, p. 48]. The necessary information to allow participants access and use the services depends on the context in which it is used. It could be enough to describe a service as the service running on that machine. On the other extreme, a service description with machine processable description of the semantics of its operations and real world effects may be required by automated service discovery and planning systems [Oasis, 2009, p. 48]. To this respect, the challenge is to have the complete description for the given context.

Descriptions change over time. Descriptions evolve with time through the previous experiences of an organization implementing an SOA. The solution must be flexible to support constant changes to the structure of the given descriptions. Descriptions come from a basis of what is considered common knowledge [Oasis, 2009, p. 49]. For this reason, it is impossible to describe everything that is expected in these descriptions. Descriptions should evolve over time.

A mechanism that allows the creation and management of service descriptions with the complete information given the context, flexible enough to support fast changes and to accommodate to different environments is required.

2.4.2 Infrastructure Required for Governance

Governance is expressed through the policies and contracts established among the participants in SOA-based systems. They contain the conditions for the interaction with services. This requires:
• A way to define the processes that sets the policies that govern SOA-based systems.
• A mechanism to communicate the processes that set the policies.
• A mechanism to validate that the affected participants know and accept the processes that set the policies that govern SOA-based systems.
• Visibility of the policies associated to services. The policies must be completely described, including their functions and effects.
• A discovery mechanism that let the participants find the different policies that exist and are reachable as well as applicable to them.
• A mechanism that enables access to the policies that are of interest to the participants.
• A known information site with the information about governance including policies, rules and regulations. A mechanism to communicate to the interested stakeholders about events of interest related to governance is also needed.
• Because governance requires measures to ensure the application of the policies and contracts defined, metrics must be defined and the infrastructure to collect the data must be in place. This data must also be available for report generation.

2.4.3 Requirements for Management

Management has to do with three concepts: resources, policies and contracts and participants management (for more information see section 2.3.4). The requirements for successful management of SOA-based systems are:

• An information collection site, where participants can access and store information about management like manageable resources, policies, contracts and participants information.
• A mechanism to define the processes so that the participants in charge of resource management can describe the life-cycle of these resources. In SOA-based systems, especially the life cycle of services should be described.
• The definition of the life-cycle of the managed resources should be created, communicated and accessible to all participants.
• Access to the capabilities that allow participants to manage resources.
• A mechanism that enables access to the policies that are of interest to the participants.
• The definition of metrics and a mechanism to collect data is also necessary for management to ensure and measure the realization of the policies and contracts established.
• Enough information about the participants that allow the communication among them.
Chapter 3

Requirements for SOA, Governance and Management

3.1 A mechanism for Awareness

A mechanism that facilitates the service consumers to find the provider’s managed service descriptions is required. Possible awareness mechanisms are: (see [et all. MacKenzie et al., 2006, p.14])

1. A mediator that provides awareness to one or more consumers of one or more services. A mediator would then be a known or agreed facility or location that keeps service descriptions (see [Oasis, 2009, p. 66]).

2. Broadcasting by a service provider of service descriptions to all potential consumers.

3. Potential customers may broadcast their particular service needs.

4. A service consumer may probe an entire network to determine if a suitable service exists.

In the mediator approach, the mediator is commonly known as the Service Broker. Figure 3.1 shows the concept of mediated awareness.

3.1.1 A Mechanism for Awareness in the Context of SOA Governance

The concepts shown in section 3.1 also applies to the case of policies, as it was required on section 2.4.2. Governance requires that participants (like a governance body for example) in SOA-based systems know about the structures created to define and implement governance. For more information, see [Oasis, 2009, p. 95].
3.2 Support for User-defined and Evolving Artifacts to Describe the Resources of the Architecture

As discussed before, a mechanism that allows the creation and management of service descriptions with the complete information given the context, flexible enough to support fast changes and able to accommodate to different environments is required. This mechanism should in general enable entities to create and manage any kind of artifact to describe the resources present in the architecture, not only service descriptions. Examples of other artifacts are participants descriptions and IT resources descriptions. This mechanism should accommodate the composing elements of the artifacts and enable changes to them efficiently.

3.3 Complete Description of the Resources

As previously seen, descriptions are inherently incomplete, but they should be complete enough to enable participants to access and use the services, and this depends on the context. The description of the entities must include the elements necessary to allow participants to access and use services in any possible context. This includes not only the complete description of services but also the description of policies that apply to the context of execution of the service.

The resources of the architecture must be completely described so that they can be discovered and used by the interested participants of the SOA-based system.

Components of the architecture are Consumer, Participants and Needed Resources other than Services [Oasis, 2009, p. 48].

Figure 3.1: Mediated Awareness Source: http://www.w3.org/2003/Talks/0521-hh-wsa/soa.png
Complete Description of Services

A Service must be described so that it defines or references the information needed to “use, deploy, manage and otherwise control a service” [Oasis, 2009, p. 48].

A service description must take into account the concepts of visibility, interacting with services and real world effect introduced in section 2.4 SOA Challenges. In summary, a service consumer needs to know: [et al. MacKenzie et al., 2006, p. 21]

1. That the service exists and is reachable;
2. That the service performs a certain function or a set of functions;
3. That the service operates under a specific set of constraints and policies;
4. That the service will (to some implicit or explicit extend) comply with policies as prescribed by the service consumer;
5. How to interact with the service in order to achieve the required objectives, including the format and content of information exchanged between the service and the consumer and the sequences of information exchange that may be expected.

3.3.1 Policies

The first challenge established in section 2.4.2 on page 18 states the need for the visibility of policies associated to services. Visibility includes the complete description of policies so that they can be discovered by interested participants. Complete description of policies includes identifiers and representation of the meaning of terms used to describe the policy, its function and its effects. The representation of the meaning is preferable when it is machine-readable [Oasis, 2009, p. 95].

3.4 Support for Processes Descriptions

SOA includes the definition of the processes to implement governance (see section 2.3.2 - How to Implement SOA Governance?). Also, for the management of the resources of the architecture, definition of processes is important, specially for the management of the life-cycle of the resources.

Processes at design-time and run-time should be described. Description of the processes at design-time includes description of the processes itself, of the activities associated to the processes, of the subprocesses, of the input artifacts to the processes and output artifacts of the process, and of the roles in charge of the activities and sub-processes. Description of the processes at run-time includes description of the process instances, of the instances of the sub-processes, of the instances of the input and output artifacts, and of the association of the participants in charge of the task defined that play the role that is described in the static definition of the process.
3.5 An Information Collection Site

An a priori known site that collects information is needed. In the case of governance, participants should have access to governance information to “understand the intent of the governance, the structures created to define and implement governance and the processes to be followed to make governance operational” [Oasis, 2009, p. 95]. For example, there should exist policies that define which information is needed to describe a service, then a known place to obtain the corresponding policies is needed when a given actor wants to describe them.

3.6 A Communication Mechanism

A communication mechanism is needed to inform the participants in the SOA-based systems about events that are of interest for them. In the case of governance, participants can be interested in changes in policies, rules and regulations and these participants must be notified of these events.

3.7 SOA Infrastructure for Metrics and Measures

Governance relies on metrics and measures to ensure that services keep behaving well during run-time. For example, a given service that makes excessive use of resources may not behave well. The governance body may notice the use of resources through the measures taken, and understand that this usage is excessive through the definition of an excessive defined in the policies.

3.8 A Mechanism to Manage Resources

Section 2.3.5 on page 16 discusses about the importance of managing resources and summarizes the resources that are manageable in SOA-based systems. A mechanism to manage these resources, i.e. services, service descriptions, policies and contracts, roles, relationships and infrastructure elements is needed. The properties of the resources that should be managed where described in section 2.3.6 on page 17 an includes life-cycle manageability, configuration manageability, event-monitoring manageability, accounting manageability, quality of service and policy manageability. It should not be forgotten that the main challenge in the management of these resources is the different ownership boundaries, where, for example, the owner of the service is one participant, the host is another, the mediator another and the customer another.
3.9 A Traceability Mechanism

A traceability mechanism that allows change impact analysis is required. Given a change or a simulation of a change in a resource of the SOA-based system, a mechanism should exist that tells which other artifacts and participants are affected. Changes that must be tracked include:

life-cycle events: events associated to the life-cycle of the resources, for example, the creation of a service;

configuration events: when the configuration of a resource is modified;

policy events: when the policies are created, modified or retired;

Policy events are very important as a support for governance of the SOA-based system. This mechanism must use the communication one to propagate the event(s) to the interested participants.

The traceability mechanism should be also useful for a participant in charge of a given resource to see associated resources. This would allow a graphical user interface tool to show these dependencies. It is important to note that many times the associated resources belong to different layers of the SOA-based system. For a discussion about the possible layers and the relationship between its components, see section 2.2.5 on page 11.
Chapter 4

Work Development

This work proposes a set of models that enables the traceability of the different components of SOA-based systems, which are necessary for management and governance. The models must take into account that participants and the elements in SOA-based systems may belong to different ownership domains. The elements that compose SOA-based systems include: participants, services, service descriptions, service implementations and IT resources that support the implementation of the capabilities offered by the services. Given the distributed and dynamic nature of SOA-based systems, governance processes must be designed, implemented and enforced for a successful implementation of those kind of systems. To enable governance of those systems, the resources that compose them must be managed, and the events related to their management must be communicated to the interested stakeholders.

This chapter begins with a description of the approach taken in this work to develop a solution to the requirements outlined in the previous chapter. A metamodels and models approach is proposed in this work. Section 4.1, Approach to Model Resources Composing an SOA, explains this.

The use of metamodels is the solution chosen to develop models that are flexible and extendable. This is an important requirement for SOA-based systems where several participants, from different ownership domains, participate. The metamodels are shown in section 4.2, Metamodels to Represent SOA Models.

The models containing the necessary elements and relations are shown and described in section 4.3, Models of SOA-based Systems. These models are based on reference architectures documents to obtain the necessary elements. These models can be seen as instances of the metamodels developed. These models should help the different participants of the architecture to manage and govern SOA-based systems with success. Participants include: developers, IT administrators, system administrators, software architects, and project managers.

Having the models that contain the elements that compose SOA-based systems, a link
that associates them is needed. Through the Services lifecycle, the different stages, participants and artifacts are interconnected. A concrete service life-cycle is vital in a systematic approach which, in turn, is required in order to maximally achieve SOA’s promises and reduce disruptions [Gu and Lago, 2007, p. 1]. A life-cycle that relates the service’s life-cycle stages (design-time, run-time and change-time), participants, and the artifacts linked to services is presented in section 4.4 - Service Life-cycle.

Finally, last part of this chapter presents a review of other solutions for the requirements shown in a previous chapter. Solutions from educational and commercial sectors were evaluated. This evaluation was based on the public documentation obtained from the creators of these solutions, it was not an empirical one. This implies that the results may not be consistent with the actual behavior of the solutions evaluated. A summary of the evaluation is provided.

4.1 Approach to Model Resources Composing a SOA

This section shows the approach that was followed in this work to solve the requirements posed before. First, an analysis of how to model entities that are complete in SOA-based systems is done. Afterwards, a description of the metamodels to express the models required for SOA, SOA governance and SOA management is presented. Finally, the models, based on the metamodels, that contains the resources and relations among them, that allow all the participants of the architecture to do their tasks are shown.

4.1.1 Completeness of the Description of the Composing Resources

The approach to obtain and describe the resources composing an SOA together with the relations among them can vary. In the work “Models and Tools for SOA Governance” [Derler, 2007] an empirical approach is used. The models and tools developed in that work were developed for a particular organization in the financial domain. In the work “STraS: A Framework for Semantic Traceability in Enterprise-wide SOA Life-cycle Management” [Seedorf et al., 2009], a theoretical approach was used.

An empirical approach that is valid from an statistical point of view would be the best option but would be more complex in general. A theoretical approach, based on the concepts contained on books, is a good option, but some practical aspects of the actual implementation of SOAs can be omitted. One of this aspects is the cultural one, in the sense that the participants of the running SOA instance understand and agree with the models proposed. An empirical approach for the development of models may easier represent a running SOA-based system.

The option followed in this work is based on Reference Models and Architectures. A reference model specifically defines an abstract framework for understanding the important resources and the relations among them [et all. MacKenzie et al., 2006]. A reference architecture “models the abstract architectural elements in the domain independent of
the technologies, protocols, and products that are used to implement the domain” [Oasis, 2009]. The documents used in this work are: Reference Model for Service Oriented Architecture 1.0 [et al. MacKenzie et al., 2006], Reference Architecture Foundation for Service Oriented Architecture - Version 1.0 [Oasis, 2009] created by The Organization for the Advancement of Structured Information Standards (OASIS) and Draft Technical Standard - SOA Reference Architecture [Harding, 2009] and Technical Standard - SOA Governance Framework [Arsanjani et al., 2009] created by The Open Group.

These documents take experiences from previous implementations of SOA’s as a source. The Open Group SOA Reference Architecture “presents the results of abstracting and using a SOA Reference Architecture based on multiple projects in different industries, commencing from 2002.” [Harding, 2009, p. 9]. These characteristics make the approach here a kind of hybrid one. On one side, it is based on the concepts and relations established on the reference models. On the other side, it is based on reference architectures that were obtained from experiences in multiple SOA projects in different kinds of industries. If the reader is interested in reference models, reference architectures or other important open standards, like SOA maturity models, the document “Navigating the SOA Open Standards Landscape Around Architecture” by The Open Group [Kreger and Estefan, 2009] would be a good reading.

4.1.2 Metamodels to Express User-defined and Evolving Artifacts

One of the requirements of the solution is the support of user-defined and evolving artifacts that describe the resources composing the architecture (see section 3.2 Support for User-defined and Evolving Artifacts to Describe the Resources of the Architecture on page 22). The solution must enable entities to extend or personalize their artifacts. The use of metamodels is the solution proposed in this work for this requirement. The metamodels proposed provide the elements to represent resources and relations among them in the given models.

A difference between metamodels and models is that the former depicts the system in design-time while the latter in run-time. Metamodels should work for any entity implementing a SOA-based system. The models can be thought as an starting point to model a running SOA-based system. For example, the artifacts metamodel contains the elements to define any kind of artifact. Examples of these artifacts are the service description, policies descriptions, service functionality description, etc. The artifacts metamodel is useful as it is, i.e. it should not be changed, it must be useful to depict any possible scenario for any organization. But the artifacts depend on the specific requirements of each entity. Two or more organizations can based their artifacts on the same metamodel, but the resulting artifacts may be different.

Each of the metamodels comes together with an implementing model. For example, the Artifacts Metamodel comes together with the Artifacts Model. In this case, a metamodel defines how an artifact is compose and how artifacts are related to run-time instances of
them. Artifacts models contain the elements to describe artifacts like Associated Annotations (see figure 4.14) or Service Functionality (see figure 4.16). The metamodels created in the context of this work are: artifacts metamodel; description metamodel; roles metamodel and processes metamodel. Metamodels are described in section 4.2, Metamodels to Represent SOA Models.

4.1.3 Models to Represent Required Resources of the Architecture

Models to describe the resources that compose SOA-based systems and relations among them were done in the context of this work. They are expressed using the elements defined by the metamodels. For each of the metamodels developed a corresponding model exists. Among the models there are: service description model based on the artifacts metamodel, the service life-cycle model based on the processes metamodel, the services categorization model based on the categorization metamodel, the service functionality model and finally the roles model, based on the roles metamodel. The completeness of these models were discussed in section 4.1.1 Completeness of the Description of the Composing Resources.

4.2 Metamodels to Represent SOA Models

The metamodels here presented have the aim to support the models specifically developed to support a SOA-based system from design to run-time. These models are shown in section 4.3, Models of SOA-based Systems of this document.

The greatest difference between these metamodels and the specific SOA models is that the former are configured in design time while the latter are configured in run-time. This is because the SOA models can be personalized for the specific requirements of the users. The models shown later in this document can be considered as a starting point for an organization implementing an SOA.

The metamodels include: artifacts metamodel, description type metamodel, description type categorization metamodel, roles metamodel and processes metamodel.

4.2.1 Artifacts Metamodel

This metamodel provides the elements to represent any possible artifact in the architecture. Figure 4.1 shows the elements composing the artifacts metamodel. The Artifact element is a container for any kind of document. Artifacts are composed of a group of properties that are semantically related, each of them represented by the Property entity. This Artifact element has a name attribute with a short description of it. Service Requirements is a good example of one of these artifacts. Artifacts and properties are related through the Artifact_property entity. It tells which Property entities compose an Artifact and also the possible number of occurrences of a property through the cardinality attribute. To note is that a properties like name or description can belong to several artifacts.
Figure 4.1: Artifacts Metamodel
A Property element, besides defining a key and a value type, establishes a hierarchy through the artifacts. The value of a given property can also be another artifact, which in turn can define a set of properties or even reference other artifacts. In the model this entity is modeled by the Artifact_value entity, which is of type Artifact. For example, the Service requirements artifact previously mentioned is composed of one or more Service requirement artifacts, which in turn have a name, a description, and can also refer other artifacts like pre and post-conditions.

The Property_set_definition element describes the entity in charge of administering a given set of properties. This set can be domain specific. For example a set of properties to use in the financial domain can be defined. Each property element must belong to one of these Property_set_definition entities.

Examples of these are Associated Annotations (see 4.3.3), and Service Functionality (see 4.3.4).

Until now the previous entities define general artifacts and properties but some entities that model instances of the artifacts and the corresponding properties are needed. The most important one is the Artifact_instance entity. This entity models an instance of a given artifact. It is composed of several instances of the properties that describe the artifact instance. The index attribute of property instance establishes an order. Continuing with the previous example, a service requirements artifact contains several service requirements and the index attribute sorts them.

Because there are properties which values are other artifacts, the Artifact_instance_value entity, which is of type Artifact_instance was added to the model. Artifact_instance has a finished attribute that tells if this instance is already done or not. It must be noticed that for an Artifact_instance to be finished, all the composing property_instance elements that are required (i.e. the ones that in the corresponding Artifact_property the cardinality attribute is at least one) must be in place.

4.2.2 Description Type Metamodel

One of the requirements for the solution is the “Complete Description of the Resources” (see 3.3 on page 22). Basically, the resources that can be found in the architecture, like Services, Participants and others, must be fully described to be usable. For each of these resources, different kind of information is needed. This Description Type Metamodel describes the required resources and associates them to the information that is needed to describe them.

Figure 4.2 shows the elements and relations of the Description Type Metamodel. A hierarchy to describe the different resources that make SOA-based systems has been developed. That is the reason why the attribute Parent_Description_Type of the resource Description_type exists. A hierarchy example, taken from [Oasis, 2009, p. 50], can be seen. The resources Service_description, Participant_description and Other_description inherit from Description_type, and Consumer_description and Provider_description inherit from
Participant\_description. These kinds of hierarchies can be personalized by any organization to accommodate to their resources infrastructure in runtime.

The Description\_instance resource models actual instances of a given Description\_type resource. For example, a Customers database access service description instance models an actual instance of a Service\_description, which can also be consider a Description\_type resource.

To model the information associated to these resources, a relation with Artifact resources through the Description\_type\_Artifact one is given. For example, the most general Description\_type resource is described by the artifact representing associated annotations. Examples of these are best practices documentation, third-party certifications or previous users comments (see [Oasis, 2009, p. 51]). The Service\_description resource, which inherits from Description\_type, is described by service functionality documents, non-functional documents and contracts and policies just to mention some. The cardinality attribute of the Description\_type\_Artifact resource constrains the number of artifacts that can be associated to the given description type.

4.2.3 Description Type Categorization Metamodel

In order to complete the description of the entities introduced in the description type metamodel (see 4.2.2), categories are added to Description\_Type ones. This allow the classification of the entities of the architecture with predefined keywords or taxonomies from referenceable formal definitions and vocabularies (see [Oasis, 2009, p. 51]).

The Description Type Categorization Metamodel shown in figure 4.3 shows the entities, relations and attributes needed for the categorization of a Description\_instance entity. This entity can be associated to many categories. These categories have a name, a description and are organized in a hierarchical and taxonomical structure.

Several sets of categories can be configured. This is useful when a given organization wants to use or define a set of categories of a specific domain or defined by different standardizing organizations. An organization that wants to use some categories for web services defined by the W3C consortium is an example. A service can then be categorized with standard and well known terms even in a semantic way, facilitating the interaction with different, even unknown potential users of the service.

4.2.4 Roles Metamodel

The roles metamodel defines the participants and relationships to model the actors, their tasks and responsibilities in any given SOA-based system. The figure 4.4 is a graphical representation of this metamodel. It is based on the model for roles in social structures defined in the Oasis Reference Architecture [Oasis, 2009].

A social structure “embodies some of the cultural aspects that characterize the relationships and actions among a group of participants” [Oasis, 2009, p. 33]. The participants
Figure 4.2: Description Type Metamodel
can be part of more than one social structure. The social structure also defines the roles and the responsibilities attached to these roles.

### 4.2.5 Processes’ Metamodel

The purpose of the processes’ metamodel is to provide the elements so that any possible workflow can be described with it. This metamodel is related to metamodels introduced before. It is specially related to the roles and the artifacts metamodels. Figure 4.5 shows in a graphical representation the elements composing the processes metamodel.

This metamodel is composed of two parts. The first part consists of the elements to define the workflow. The second part consist of the elements to describe the workflow running.

The main element of the processes metamodel is the *Process* element. It describes a static process and summarizes all the elements related to a process. Figure 4.6 shows the elements composing a static process. It is composed of input and output artifacts that represent the inputs for the process and the results of its execution. Each *Process* element contains also a reference to the roles in charge of executing the activities contained within
Figure 4.5: Processes’ Metamodel
In this metamodel, any workflow is composed of activities or sub-processes, but not both. A process that is composed of activities can be considered an atomic process, i.e. as soon as the activities are done, the process is finished. The activities represent the tasks to be done in the process. The processes that are composed of sub-processes are finished when all the sub-processes that compose it are over. The sub-processes that compose a process has a type. The type can be ‘start’ for sub-processes that represent the starting point of the processes that contain them, ‘completion’ for sub-processes that represent the end of the processes that contain them, and ‘normal’ for sub-processes that are neither ‘start’ nor ‘completion’ nodes of the processes that contain them.

Another important part of the description of the workflow is the description of the order of execution of the sub-processes. This is done through the Activation rule element. After the completion of a given sub-process, another sub-process must execute. The Activation_rule element defines which sub-process from the list of possible sub-processes must be run next. This element is able to make this decision based on the value of some property or properties that are defined in an output artifact of a sub-process previously executed. In the SOA Models section, an instance of a process, using this metamodel, can be seen to illustrate these concepts. These elements are the basis for the definition of static processes. Now, the elements to describe the running processes must be defined.

The main element to describe a running process is the Process_instance element. This element represents an instance of a process of type Process (see figure 4.7). The process defines the roles in charge of executing the activities defined. When an instance of a process is created, participants responsible for implementing the activities defined by the process must be selected from those who adopt the roles specified. Figure 4.8 illustrates the relation among these elements.

The process also defines input and output artifacts. When an instance of a process is created, instances of the input and output artifacts should be selected. The types of the
Figure 4.7: Relation of Process and Process Instance

Figure 4.8: Relation of Process, Process_instance, Role and Participant
instances of the input and output artifacts are defined by the types of the input and output artifacts defined by the processes. Figure 4.9 shows these elements and their relations.

A composed process is a process composed of several sub-processes. When a composed process starts, the sub-process of type ‘\texttt{start}’ runs first. When this process finishes, the next process to run, with help of the ‘\texttt{Activation\_rule}’ element, is selected. This continues until the process of type ‘\texttt{complete}’ ends. Then it can be said that the process and all its sub-processes have finished. If the process is not composed, and there are activities associated with the process, when these activities are completed, the process that contains them will also be completed.

4.3 Models of SOA-based Systems

Given the challenges identified in section 2.4 - \textit{SOA Challenges}, the specific requirements identified in section 3 - \textit{Requirements for SOA, Governance and Management}, and the meta-models that were explained before, specific models for SOA will be described in this section. This set of models contains the resources, participants and their relationships that can be found in SOA-based systems. These models are the input needed for SOA management and Governance, therefore necessary for a successful implementation of a SOA-based system. A further question must be answered: How can a system architect know that the resources, participants and their relationships shown in these models fully describe a SOA-based system?

These models are based on documents issued by diverse consortia, like \textit{OASIS} or \textit{The}
Open Group. “These consortia attempt to standardize all the different aspects of the interaction, ranging from interface definition languages to message formats and interaction protocols” [Alonso et al., 2004, p. 133]. Those documents provide the necessary information to fully describe the resources that compose SOA-based systems. The documents referenced in this work are: the Oasis Reference Architecture [Oasis, 2009], Oasis Reference Model [et all. MacKenzie et al., 2006], The Open Group Reference Architecture [Harding, 2009] and The Open Group Governance Framework [Arsanjani et al., 2009]. This is done in order to make the resulting models as valid and complete as possible, as discussed in section 4.1.1 - Completeness of the Description of the Composing Resources.

This section starts with the Policies Model. It provides the elements needed to describe the policies related to some of the resources found in this kind of systems. These policies also place restrictions on the interaction among participants. It is worth noting that this Policies Model does not associate policies to services. This association will be shown later in the Service Description Model.

Following, the Contracts Model is shown. It relates contracts, i.e. a set of policies, to a set of services. The main characteristic of this model is that at least two stakeholders must agree to the restrictions set by the policies contained in the contracts.

The next model described is the Description Type Model. A Description_Type models the most abstract resource that can be found in SOA-based systems (see section 4.2.2 - Description Type Metamodel). The Description Type Model shows the elements and associated relations to describe Description Type resources. The Service Description model do the same as the Description Type one, but this time, a resource of type Service instead is described.

To complete the description of services, a Service Categorization Model is shown. This model is based on the categorization metamodel presented in section - 4.2.3. It is based on the categories defined by The Open Group Reference Architecture [Harding, 2009].

What is left is a model containing the role and participants in SOA-based systems. This is the Roles Model, which is related to the Roles Metamodel discussed earlier. This model is important because it relates each of the activities in the processes to the responsible participants that play a given role. Social structures that are typical in SOA-based systems are described in this model.

4.3.1 Policies Model

The elements in this model provide, in general terms, the structure to describe policies. This model forms the basis for a more refined one that associates policies to services. That model, called Service Policy Model, can be found under the Service Description Model section. Figure 4.10 shows a graphical representation of the elements that compose the Policies Model.

This model is basically composed of policies. A policy “represents some constrain or condition on the use, deployment or description of a resource as defined by a participant
or, more generally, a stakeholder” [Oasis, 2009, p. 81]. As any other managed resource, the policies must have a version. It must also have an id that enables other resources identify this policy. This will be important for the association of services with policies. A policy may include references to other policies. That’s why the Referred_policy element is in the model. The application of policies often requires the ability to compose one or more policies into a general one [Oasis, 2009, p. 82].

A policy is composed of policy constraints. A policy constraint is a “measurable proposition that characterizes the constraint that the policy is about” [Oasis, 2009, p. 81]. A proposition can take the form of a Permission or an Obligation. “A permission-style constraint is about the right to access some resource or perform some action” [Oasis, 2009, p. 81]. An obligation-style constraint requires a participant to carry out some action or keep the status of a resource. An example of the latter is a requirement to maintain certain balance in a bank account. A policy constraint must refer to a metric. A metric is a mechanism through which the policy’s constraints are predicated and are later enforced. A Metric can be a Performance or a Non-performance one. Performance metrics are associated to quantities like, for example, speed and accessibility of a service. The existence of a certification of a service is an example of a Non-performance metric.

Figure 4.11 shows the policies and contract’s model, using the elements defined in the artifacts’ metamodel. It is composed of Policy artifacts. This time, not only Policy, but also Metrics artifacts are defined independently, i.e. they are not attached to a particular Policy_constraint. These metrics may be reused in multiple policy constraints. A Policy artifact has a property version and a property Id. The latter allows external resources to
refer to this Policy artifact. In this model, a Compliance_Record artifact was added. This artifact stores, for each of the policies, operational values associated with them. Those records determine whether policies have been fulfilled.

### 4.3.2 Contracts Model

Figure 4.12 presents the elements that compose the Contracts Model in a graphical way, using the Artifacts Metamodel. The contracts model refers to a Services_group artifact. This artifact comprises a set of services, to which a group of policies apply. This set of services is grouped by the Contract artifact. The most important feature of the Contract artifact is that at least two participants must agree on the constraints defined in the policies contained by it. A contract may refer to other contracts. To model this relation, a Related_contracts artifact is in this model. The Related_contracts artifact groups a set of Contract_id artifacts that represent the contracts to which the original contract is related to.

### 4.3.3 Description Type Model

Description_type is the most abstract entity in the architecture, so all the characteristics described in this model apply to all the inheriting entities, like Service_description or Participant_description. Figure 4.13 shows this model.
Figure 4.12: Contracts model

Figure 4.13: Description Type Model
Associated Annotations

In figure 4.14 the associated annotations artifact is shown. This artifact is used to attach annotations to a given description type. It is not taken from any standardization source. It shows how the proposed architecture can be used for very specialized kinds of artifacts.

The Description_type entity can have multiple Associated_annotation artifacts attached to it. Any Associated annotation is composed of two properties: the User_description_property property and the Annotation_property property. The formers value is the User_description artifact while the latters value is the Annotation artifact. The User_description artifact is composed of two properties: the name_property and the role_property. The Annotation artifact is composed also of two properties: the Rating_property and the Comment_property.

4.3.4 Service Description Model

The service description’s model must contain the information necessary to use a service, as defined in section 3.3 Complete Description of the Resources on page 22.

The model in figure 4.15 describes the resource Service_description, which is of type Description_type and represents a service in a SOA (For information about the Description_type entity, refer to 4.3.3).

Several aspects of a service should be described. Those aspects of services are shaped by several models, as shown in figure 4.15. These are: the the Service Functionality’s Model,
which describes the expected real-world effect when interacting with a service; the Service Interface Model, which describes how to interact with services, including the temporal aspects of interacting with them; the Service Reachability Model, which describes how to access the service, i.e. information about the location of the service, and includes the description of the presence of a service; the Service Relational model, which contains the elements that relate services with resources in other layers; and finally, the service policy model, that associates policies and its constraints to the services and also to the actions that compose them.

Service Functionality

The service functionality is the most important characteristic when a service consumer is looking for a specific service, as it “describes what can be expected when interacting with a service” [Oasis, 2009, p. 54]. The functionality of a service can be described in the following ways:

- As natural language text.
- Reference to an existing taxonomy of functions.
- Reference to a more formal knowledge capture providing richer description and context.

When describing a service function through natural language text, a custom artifact that suits customer information’s needs must be defined. For the second case, a taxonomy of functions can be set using the category model. Then, a Description_type resource that describes a service can be associated to one of these categories that represents a function in a taxonomy.
An example of the third way to describe functionality is using a concept in a semantic model. This approach is the one used by the Semantics Annotations for WSDL mechanism. This mechanism, to describe a WSDL operation, “provides a high-level description of the operation, specifies its behavioral aspects or includes semantic definition” [W3C, 2006].

The model in figure 4.16 contains the information necessary to describe the functionality of a service based on the concepts previously introduced. The structure of this model was taken from the one proposed by the Oasis reference architecture [Oasis, 2009, p. 54]. The artifacts metamodel (see 4.2.1) was used.

Service Functionality artifact consists of functions. A function has a name, a service-level real world effect that states the purpose or objective of the function and technical assumptions which constrain the effects that can result from the service functionality [Oasis, 2009, p. 55]. These assumptions are domain specific and can express physical limitations like the access speed to a hard disk or the processing capability of a given computational node. A service-level real world effect is composed of at least one action-level real world effect. This two elements are described using natural language text (description property), a reference to a taxonomy of functions (external category) and referring a more formal knowledge capture providing richer description and context (semantic concept) following the advice of the Oasis reference architecture [Oasis, 2009] on how to express service functionality.

An example of an instantiation of this service functionality model is the Semantic Annotations for WSDL [W3C, 2006]. This annotations enhance the WSDL files with semantic about the services and operations they describe. The next excerpt of a WSDL file annotates an interface with a reference to an external category:

```xml
<wsdl:interface name="Order"
  sawsdl:modelReference="http://example.org/
categorization/products/electronics">...
</wsdl:interface>
```

And the following annotates an interface with a reference to a semantic concept:

```xml
<wsdl:operation name="order"
  pattern="http://www.w3.org/ns/wsdl/in-out"
  sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#RequestPurchaseOrder">
  <wsdl:input element="OrderRequest" />
  <wsdl:output element="OrderResponse" />
</wsdl:operation>
```
Figure 4.16: Service Functionality Model
Service Interface

The first step to use a service is to have awareness of it. After establishing awareness of the service, participants must establish visibility of the service and willingness to interact with the service before the interaction with it. “Service Interface is the means by which the underlying capabilities of a service are accessed” [Oasis, 2009, p. 71]. Next is the description of the elements that compose the interaction model that allow the interaction with a service. The interaction with a service can be characterized by a sequence of actions. Each of these actions conveys the goal or intent of interacting with a service. For example, the purpose or intent to interact with a service of an airline is to book a seat on a given flight. In the Oasis Reference Architecture, an action is defined as the “application of intent to achieve an effect (within the SOA ecosystem)” [Oasis, 2009, p. 27].

An important feature of the service interface is that the underlying details of the service implementation should not be visible to customers of the service. The interface should be an implementation-neutral one. Service interface is composed of information and behavior models as can be seen in figure 4.17.

The interaction with a service is usually mediated by a series of messages exchanges. The Information model must specify the syntax or structure and semantic of the messages that are used to communicate the actions to interact with the service. Figure 4.18 shows the elements that compose the information model.

The information model is based on the artifacts metamodel (see section 4.2.1 - Artifacts Metamodel). This model is composed of several Data artifacts. These artifacts describe the type and the meaning of the elements that compose a message. The Data_id is a property of type double that uniquely identifies the Data artifact. With this property, a Data artifact can be referenced by an external resource. The Label property simply provides a human-readable name for the Data artifact. The Type property sets the type of the data element that is being represented by this Data artifact. Finally, the Semantic_description property of type String describes the meaning or the expected use of the data element being represented by this Data artifact.

The concepts of information model described above are best illustrated with an example. Let’s describe a Year data artifact using the information model. Figure 4.19 show the
elements composing this artifact. As shown, the elements related to the artifacts metamodel, including artifact instance elements are used.

The Year data artifact is based on the information model defined earlier (see figure 4.18). It defines an instance of the property DataId with the value “18503”. It is of type double. It identifies the Year data artifact in a unique way. The value of the property Type, i.e. the instance of this property is integer. That means that the type of this data artifact is integer. The value of the property label is Year. This is the human-readable label for this data artifact. And finally, the semantic description value is “Represents a year”. This description is clearly done with the intent to be understood by humans. It could have included a reference to a semantic taxonomy, that could be processed by a non-human participant.

The Behavior Model includes the Action Model and the Process Model, as can be seen in figure 4.20. The Action Model contains a collection of the actions that can be performed against a given service. The Process Model references the actions in the Action Model and describes the sequence for performing these actions.

The Action Model is made up of a series of actions that may be carried out against a specific service. Every action has a number of related artifacts. A graphical representation of the elements of the Action Model and the relationships among them is shown in figure 4.21.

Every action has a name property to label each of them. As shown, this property is taken from the Artifacts Metamodel. This is because it is a property that is defined to use not only in the context of the description of actions. The property Description provides a human-readable description of the capability offered by the action.

The interaction with a service is conducted by performing actions against the service. The performance of those actions is mediated by a series of message exchanges [Oasis, 2009, p. 71]. “Message exchange is the means by which service participants (or their
Figure 4.19: Information model example - Year data artifact

Figure 4.20: Service behavior model is composed by action model and process model
agents) interact with each other” [Oasis, 2009, p. 72]. An action has one or more messages associated. A message conveys an action. These messages are modeled through the Message artifact.

The relation from message, message exchange, participants, service, action and real-world effect can be seen in figure 4.22, taken from [Oasis, 2009, p. 71]. A participant sends or receives messages. Participants include the provider of the service, the receiver of a service, brokers or mediators. The interaction among the participants is done through a message exchange. A message exchange describes the sequence of exchange of a given set of messages. As said before, a message conveys an action or an event. The actions cause real-world effects while the events reports them.

For a successful interaction among the service participants, the messages must be appropriately formatted. In addition, this format should be, a priori, agreed by the participants that interact. The successful interaction among participants includes that the participants understand the intention contained in the messages exchanged. This is the connection point with the information model presented above: it specifies the syntax and the semantics of the messages (see figure 4.18). The Pattern property of a Message artifacts characterizes the temporal aspects of a service interactions between participants [Oasis, 2009, p. 72]. Two patterns are the most fundamental ones for message exchanging:

- **Request/respose**, to communicate actions that cause a real-world effect and
- **Event notifications** to communicate real-world effects to stakeholders.

Figure 4.23, show in UML notation these two message exchange patterns.
The Action Model was already shown and described. The description of the Process Model is presented now to complete the description of the Behavior Model. The Process Model takes the actions contained in the Action Model and describes a sequence to interact with them. The elements and the relationships of the Process Model can be seen in figure 4.24.

The Process Model, as the other models of this work, is expressed using the terms defined in the Artifacts’s Metamodel. The Process Model is composed of one or more Service_Process artifacts. These artifacts join actions with the corresponding order to execute them. To do this, each Service_Process contains one or more Process_action artifacts. Each Process_action artifact in turn contains a property Action_Id and a property Action_order. The property Action_Id refers to one of the actions included in the Action Model of the service (see figure 4.21). In the action model, an Action artifact is uniquely identifies with the Id property. The value of the property Action_Id must match with the corresponding value of the property Id of the given action. The sequence of execution of the actions is modeled in the Process Model through the Action_order property. The artifact Service_Process is given a name through the property name and a unique identifier through the property Id.

The interaction between the Action Model and the Process Model can be seen in the following example. In a security-controlled database access service, the actions available to a service consumer are: presenting credentials, requesting updates to the database and reading results of queries [et al. MacKenzie et al., 2006, p. 17]. If a database customer wants to update the database or read the results of queries, he must be authenticated and authorized before. The database client must interact successfully with the presenting...
Figure 4.23: *Request/response* and *event notification* message exchange patterns [Oasis, 2009, p. 73]
credentials action before he or she is able to interact with the requesting updates or reading results of queries actions. In this case, the actions are included in the action model and the description of the temporal dependencies among actions are included in the process model of the service.

Service Reachability

The Service Reachability Model provides the necessary information on the location of the services and the protocols they support and require [et al. MacKenzie et al., 2006, p. 21]. This information complements the information in other models, like in the Service Functionality Model or the Service Interface Model, allowing the interaction with services. The Service Reachability Model also provides dynamic information about the current availability of the service.

The Service Reachability Model should basically indicate the endpoints to which a service consumer can direct messages to invoke actions and the protocol to be used for the exchange of messages using that endpoint. The location and protocol are the only information required in this model, but information on the presence of the service is also recommendable. Figure 4.25 shows the elements and relationships of the Service Reachability Model in a graphical way.

The Reachability Model is composed of multiple services. A service references an artifact called Service.presence. This artifact provides information about the availability of the service. One possible measure is the percentage of up time of the service, i.e. the percentage of the time that the service is running. The Service.availability property, attached to the Service.presence artifact provides this availability number. But, because a service is
Figure 4.25: Service Reachability Model
composed of several actions, the presence of a service is the aggregate of the presence of the actions that compose the service [Oasis, 2009, p. 70]. Some kind of calculation, to obtain the service presence, must be done based on the presence of each of the composing actions of the service.

Each Action artifact references an Action_presetce artifact, to describe the presence of the action. This is done in the same way as the service presence is described. The presence of an action depends on the presence of the endpoints that the Action artifact references. Every endpoint provides an address where the messages attached to the actions are sent to. Several heuristics can be used to calculate the presence of an action. Even if not all the endpoints of an action are available, the action, and in turn the service, could be available all the time.

An Action artifact references one or more Endpoint artifacts. Every endpoint has an address where the messages related to the endpoint are sent to. A protocol is also attached to every endpoint. It is important to remark that a Protocol artifact is attached to an Endpoint artifact, but is not created. The Protocol artifacts are associated to the Action artifacts. Only one Protocol artifact associated to an Action artifact can be attached to an Endpoint artifact. An Endpoint has also a human-readable name to be able to identify it.

The presence of an endpoint is modeled through the Endpoint_presetce artifact. The presence of an endpoint is determined by the interaction via a communication protocol associated with it [Oasis, 2009, p. 70]. The presence of a service may not be established, in many cases, before actually interacting with a service. A possible approach to test the presence of an endpoint is to test, in some time intervals, the availability of the endpoint. IT mechanisms may make use of presence protocols, to provide information about the up or down status of a given service.

Each time an endpoint’s presence is asserted, an Endpoint_presetce_log artifact is created and is associated to an Endpoint_presetce artifact. Each of these Endpoint_presetce_log artifacts saves time and result of the test. A summary of the presence of the endpoint is saved in the Endpoint_presetce_summary property that is associated to the Endpoint_presetce artifact. The Endpoint_presetce_summary should be updated every time the presence of the endpoints is tested. For example, if the endpoint presence was proven nine times, and nine times the endpoint was available, the presence or availability of the endpoint is 1.0. But if the tenth time the endpoint is evaluated and is not available, the presence of the endpoint can be modified to 0.9.

It is important to note how the presence of an endpoint is related to the presence of the corresponding action and the presence of the corresponding service. When the presence of an endpoint changes, the presence of the action must be re-evaluated and the action presence could be modified. It is important to note here that the action presence need not necessarily to be changed due to a change in the presence of an associated endpoint. As mentioned earlier, there could be an endpoint associated with an action that is not available, yet the action is available all time. Finally, when the presence of the action is
modified, the presence of the corresponding service must also be re-evaluated.

**Service Relations**

The *Service Relational Model* provides the necessary information on the resources on different layers that are related to services. In this case, each version of a service is related to a set of clients and service implementations. Figure 4.26 shows the elements and relationships of the *Service Relational Model* in a graphical way.

**Service Policy Model**

The *Service Policy Model* includes the elements to relate action, services and policies. Policies express the requirements, capabilities and preferences of participants that interact with a service [Alonso et al., 2004, p. 193]. Figure 4.27 presents the elements contained in the *Service Policy Model* in a graphical way. This model is composed of *Service* artifacts, that refer to *Action’s* artifacts which in turn refer to *Action_level_Policy* artifacts. *Service* artifacts refer to *Service_level_policy* artifacts, that represent policies related to the services. Those policies are defined in the *Policies -and Contracts’ Model*, in section 4.3.1. These policies relate actions and services to policies, policy constraints, metrics and compliance records.
The important fact in this model is the relation between action-level and service-level policies. Service-level policies must be consistent with the constraints described in action-level policies. For example, if an action-level policy states that it can not respond to a request in less than 200 milliseconds, a service-level policy can not dictate that the corresponding service must respond within 100 milliseconds.

A Comparison of Service Description Models and WSDL files

Figure 4.28 shows the structure of WSDL files, version 1.0 and 2.0. The types section describe the structure of the operations inside the interface (part of the Interface Model). The semantic description of the operations is not included. The Interface section describes the actions that compose the service (part of the Action Model). The temporal dependencies of the actions of the service, included in the Process Model, is not included in WSDL files.

WSDL files contain information about the location in the Concrete section, but not about the presence of a given service. The information of service’s presence is important in SOA-based systems because, based on it, this kind of systems can be governed. Based on this information, policies can be verified and, in the case the presence of a service does not fulfill what is required in those policies, the corresponding measures, in some cases legal measures, could be applied.
4.3.5 Service Categorization Model

This model uses the entities and relations defined in the category metamodel defined in section 4.2.3. It classifies services using the service types defined in the Open Group Reference Architecture [Harding, 2009, p. 23]. A UML representation of it can be seen in figure 4.29.

There are six possible categories of services defined in the model: Interaction service, Process service, Information service, partner service, business application service and access service. The next definitions are taken from [Harding, 2009, p. 23].

Interaction Services enable collaboration between people, processes and information. Process Services orchestrate and automate business processes. Information services manage diverse data and content in a unified manner. Partner services connect with trading partners. Business application services build on a robust, scalable and secure services environment. Access services facilitate interactions with existing information and application assets.

These categories are defined by the Open Group, and for this reason there is an OpenGroupServiceCategorization entity in the model of type Category_set_definition to which all the categories refer to.
Figure 4.29: Service Category Model
4.3.6 Roles Model

The roles model created in this work is based on the roles metamodel presented previously. It defines a set of roles, each belonging to one or more social structures. This roles model is divided into the following social structures: Business Domain Representatives, Service Development Team and IT operations. These structures are taken from the Governance Framework document written by The Open Group [Arsanjani et al., 2009, p. 24]. Figure 4.30 illustrates these social structures.

These social structures can be associated to the Service Consumer and Service Provider architectural roles, defined in section 2.2.4, on page 9. The roles that belong to the Business Domain Representatives social structure can be included in the Service Consumer architectural role. The roles that belong to the Service Development Team social structure and the ones that belong to the IT Operations social structure may be associated to the Service Provider social structure.

Moreover, these social structures can be associated to horizontal layers that generally compose an SOA (see section 2.2.5 on page 11). Based on the enterprise layers defined by The Open Group Reference Architecture [Harding, 2009], the roles that belong to the Business Domain Representatives social structure are associated to the Business Processes enterprise layer, the roles that belong to the Service Development Team social structure are associated to the Service and Service Components enterprise layers and the roles that belong to the IT Operations social structure are associated to the Operational Systems enterprise layer. Figure 2.2 on page 10 shows the enterprise layers defined by The Open Group.

Business Domain Representatives

The Business Domain Representatives social structure contains the roles in charge of determining the business service functionality, the communication of business requirements and identification of business services for each domain, and working on prioritizing requirements and services [Arsanjani et al., 2009, p. 25]. Figure 4.31 shows the roles that compose this social structure.
Service Development Team

The Service Development Team social structure comprises the roles in charge of the design, development, execution and delivery of the services. Figure 4.32 shows the roles that compose this social structure.

IT Operations

The IT Operations social structure comprises the roles in charge of the support of database administration services and network infrastructure services, support of the systems administration and support for central IT functions, among others [Arsanjani et al., 2009, p. 26]. Figure 4.33 shows the roles that compose this social structure.

4.4 Service Life-cycle

A systematic approach for the development of SOA-based systems is required, in order to maximally achieve SOA’s promises (i.e. a flexible and agile architecture) and reduce disruptions that are caused by conflicts to a minimum. To support this approach, a life-cycle that models participants, activities and states for the specification, development, running and evolution of services should be in place [Gu and Lago, 2007, p. 1]. A service life-cycle provides the participants the information about when and how they can interact with each other in the system in order to achieve their goals. A service life-cycle forms the basis for SOA-based governance because, based on it, the rights and responsibilities of
Figure 4.32: Service Development Team Model

Figure 4.33: IT Operations Model
each of the participants in each point of time can be established.

The service life-cycle developed in this work is driven by the architectural roles of SOA-based systems. These roles are: service provider, service consumer and brokers. This service life-cycle shows each of the participants together with the expected and possible interactions among them in different points of time.

The service life-cycle is also divided into three phases: design-time, run-time and change-time. The design-time phase refers to the period in the life-cycle of a service before it is operational. The run-time phase is the period of time when the service is operational. The change-time phase refers to the period of time in which adjustments to services should be done. When the change-time phase finishes, the service should enter into the design-time phase, completing the life-cycle of the service. The division of the service life-cycle into these phases facilitates the management of services [Gu and Lago, 2007, p. 2]. For this reason, the service life-cycle created here takes into account these stages.

The service states define the different artifacts produced and attached to them at different points of times in the life-cycle. These artifacts describe the different aspects of services. Those aspects and the related artifacts were described in section 4.3 - Models of SOA-based Systems. Those artifacts were created based on the assumption that they are the required ones to fully describe services. For each of the states and related artifacts, there are responsible roles. Those roles must be consistent with the ones defined earlier in the Roles Model. Artifacts that can be created and attached to services include: design-time artifacts, such as service requirements, design artifacts, policies and contracts, run-time artifacts, such as the service implementation, service interface files, and change-time artifacts, such as change requests or execution measures.

With the complete service life-cycle, it should be clear to each participant, which information must be provided by them in the interaction with other participants and when, or under which circumstances, this interaction may occur. These circumstances include restrictions for interaction, like authentication, authorization, previous payment to access some services, etc. The information that the participants should provide and the restrictions for the interaction among them are related to the different states previously defined for services. This is the sort of information that is required for the subsequent governance of this kind of systems.

Figure 4.34 shows an UML activity diagram that represents the service life-cycle developed in this work. It uses swim-lanes to differentiate the activities for each of the architectural roles. These roles are: Service Consumer, Service Provider and Broker. The different activities, through the service stages, are organized vertically. The activities in the Design-time phase are in blue color, the ones in the Run-time phase in yellow and the ones in the Change-time phase in red.
Figure 4.34: Service Lifecycle
4.4.1 Design-time Phase of the Service Life-cycle

This phase of the service life-cycle corresponds to the period of time when the need of a new or modified service appears but before it is operational. It includes collect service requirements, service realization planning, service design, service implementation, assembly or acquisition, testing and deployment.

Collect Service Requirements

The Collect Service Requirements activity consists on the collection of capability needs that form the basis for the creation or modification of services. In the case of a new service, the Define_new_service activity is chosen. Otherwise, in the case of the modification of a service, the Define_changed_service is the chosen one. In some instances, the source of these needs is a consumer request. In others, the source is ideas extracted from previous experiences or even a market scan.

Participants of the system playing roles from the business domain representatives and from the service development team are the ones in charge of this activity. Specific roles from the business domain representatives social structure include business architects and business subject-matter experts. From the service development team social structure, responsible roles include business analysts, service architects and even project managers. The roles contained in these social structures belong to different architectural roles. The former belong to service consumer while the latter to the service provider. This imposes a challenge for the interaction between participants playing those roles.

A first version of the service requirements is created. These requirements are fundamental for the description of the functionality of the service, that should be included in the service description model. The service should also be classified using a given taxonomy. This classification specifies predefined types of the service. It may be based on different aspects, like the logic it encapsulates, the re-use potential, etc. This classification is used to feed the service categorization model. A first version of the service contract should also be created. This contract must constraint the interaction between service consumers and providers. This contract can be specific for the specific service, or it can represent the restrictions for the interaction with a set of services. These contracts are later part of the policies and contracts model (see section 4.3.1 - Policies Model).

Service Realization Planning

In the Service Realization Planning activity, the requirements obtained from the previous Collect Service Requirements activity are analyzed to decide the way in which the service will be implemented. The choices to implement the service are [Arsanjani et al., 2009, p. 55]:

- a new implementation,
- use existing legacy functionality,
• use an external service,
• compose new services into a new one and,
• change an existing service.

This activity can also be seen as the modeling of a business process into low level processes, without going into technical details [Gu and Lago, 2007, p. 4].

Participants of the system playing roles from the Service Development Team are the ones in charge of this activity. Specific roles include business analysts and service architects. These roles belong to the service provider architectural role.

In this activity, the service realization plan artifact is created. This document defines the implementation strategies to develop the service in the following activities. These strategies include the service deployment platform.

**Service Design**

The Service Design activity consists on the design and specification of services. Basically, the goal of this activity is to create or modify a design that is consistent with the functional and non-functional requirements and with the realization planning generated before. The creation or modification of a service interface is carried out depending on whether the service is new or being modified. This includes:

• the specification of the parameters for the information exchange,
• in the case of we services, the creation or modification of the WSDL file and,
• in the case of any other technology, the creation or modification of the associated file.

Also, depending on the approach chosen on the service realization plan activity, an implementation model must be created. If the approach chosen there was to develop a new service, the local software development method must be selected to create the design. If the approach was to use existing legacy functionality, the patterns necessary for legacy integration must be identified. And, in the case the approach was to use an external service, the way and the feasibility of using this service must be defined. Finally, the interaction style between service and their clients must be defined. This makes part of the behavior model, more exactly part of the process model.

Participants of the system playing roles from the Service Development Team are the ones in charge of this activity. Specific roles include service architects like in the case of the service realization planning activity and Integration Specialists. These roles also belong to the service provider architectural role.

In this activity, the information model artifact is created. This document defines the semantic and structure of the operations to interact with services. A service implementation model that gives hints to the developers or integrators about how to implement the services, in the form of design and integration patterns, is created during this activity.
Service Implementation, Assembly or Acquisition

In the Service Implementation, Assembly or Acquisition activity is where services are implemented, assembled or acquired, according to the approach selected in the Service Design activity. This includes a deployment for testing in the development environment. After this testing, the service is ready to be handed over a testing team.

Participants of the system playing roles from the service development team and from the IT operations social structures are the ones in charge of this activity. Specific roles from the service development team social structure include project managers and developers. From the IT operations social structure, responsible roles include database administrators, network infrastructure architects and systems administrators. The latter roles are the one that support the IT infrastructure for the new or modified services.

In this activity, the resources from the service components layer are created. Also, the resources from the operating systems layer must be configured to support the execution of services.

Service Testing

In the Service Testing activity is where different aspects of services are tested to ensure that they behave as stated in the functional and non-functional requirements and that these behaviors are consistent with the restrictions defined in the service contracts and policies. Aspects of services to be tested include functional testing, not only fault detection but also quality control, performance testing, scalability and security. The test must be planned, executed and the results of the execution must be documented and available to the interested stakeholders. After services are completely tested they are ready to be deployed in a production environment.

Participants of the system playing roles from the Service Development Team are the ones in charge of this activity. Specific roles include Testers and Security Architects. The latter roles are important to plan, test and document the security aspects on services, that given the agility and flexibility of SOA’s, where potential clients are unknown, turns into very important aspects to be tested.

In this activity, service test results are generated. Again, given SOA characteristics, this information is important for potential users of the service to use them with a higher degree of confidence.

Service Deployment

The Service Deployment activity consists on the configuration of services to make them available to potential consumers in a production environment. It also includes the registration of the reachability information of the service in an appropriate repository. The specific deployment information must be documented. This implies giving the information required in the service reachability model, which is part of the SOA models defined before. At the
end, the IT environment must be described, including shared services and infrastructure. After the service is deployed, a final test of the service must be conducted.

Participants of the system playing roles from the service development team and from the IT operations social structures are the ones in charge of this activity. Specific roles from the service development team social structure include project managers, integration specialists and security architects. From the IT operations social structure, responsible roles include database administrators, and systems administrators, in charge of configuring the IT support for the execution of services. They also must provide the actual reachability information of the service, that makes part of the service reachability model.

In this activity, the information contained in the service reachability model is generated. A report of the certification of the deployed service, including the results of a test in the production environment is created. Finally a document with a description of the infrastructure required, including shared services must be done.

The Service Deployment activity is the final one in the design-time phase of the service life-cycle. When this phase is finished, a running service, based on the functional and non-functional requirements, restricted by the policies and contracts attached to it and tested in development and production environments, must be enable and accessible to its customers.

Table 4.1 is a summary of the activities, participants and artifacts that compose the design-time phase of the service life-cycle.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Roles</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect Service Requirements</td>
<td>From Business Domain Representatives social structure:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Business Architects.</td>
<td>• Service Requirements.</td>
</tr>
<tr>
<td></td>
<td>• Business Subject-matter Experts.</td>
<td>• Service Contract.</td>
</tr>
<tr>
<td></td>
<td>From Service Development Team social structure:</td>
<td>• Service Classification.</td>
</tr>
<tr>
<td></td>
<td>• Service Architects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Project Managers.</td>
<td></td>
</tr>
<tr>
<td>Service Realization Planning</td>
<td>From Service Development Team social structure:</td>
<td>From Service Development Team social structure:</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Business Analysts.</td>
<td>• Service Realization Plan.</td>
</tr>
<tr>
<td></td>
<td>• Service Architects.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Design</th>
<th>From Service Development Team social structure:</th>
<th>From Service Development Team social structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Service Architects.</td>
<td>• Information Model.</td>
</tr>
<tr>
<td></td>
<td>• Integration Specialists.</td>
<td>• Service Implementation Model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Implementation, Assembly, or Acquisition</th>
<th>From Service Development Team social structure:</th>
<th>From Service Development Team social structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Project Managers.</td>
<td>• Service Components.</td>
</tr>
<tr>
<td></td>
<td>• Developers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>From IT Operations social structure:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Database Administrators.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Network Infrastructure Architects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System Administrators.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Testing</th>
<th>From Service Development Team social structure:</th>
<th>From Service Development Team social structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Testers.</td>
<td>• Service Test Results.</td>
</tr>
<tr>
<td></td>
<td>• Security Architects.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.1: Design-time phase activities, roles and artifacts

<table>
<thead>
<tr>
<th>Service Deployment</th>
<th>From Service Development Team social structure:</th>
<th>From IT Operations social structure:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Project Managers.</td>
<td>• Database Administrators.</td>
</tr>
<tr>
<td></td>
<td>• Integration Specialists.</td>
<td>• System Administrators.</td>
</tr>
<tr>
<td></td>
<td>• Security Architects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Service Reachability Model.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Service Deploy Results.</td>
<td></td>
</tr>
</tbody>
</table>

#### 4.4.2 Run-time Phase of the Service Life-cycle

This phase of the service life-cycle corresponds to the period of time when the need of a new or modified service has been designed, developed, tested and deployed and is ready to be used in a production environment. It includes the *service management and monitoring* activity.

**Service Management and Monitoring**

The *Service Management and Monitoring* activity consists on the monitoring of the workload and system events associated to a given service that may cause problems or security incidents. To monitor the workload and system events, metrics defined in the design-time phase of the service are used. With measures collected, based on these metrics, a service provider can establish the availability, reliability and performance of a service. Besides, those measures support the enforcement of policies and contracts. Failing to meet what is stated in those policies and contracts may imply economic sanctions to the service provider.

Participants of the system playing roles from the *IT operations* and from the *service development team* are the ones in charge of this activity. The role from the *IT Operations* social structure responsible is the *system administrator*. From the *service development team* social structure, responsible roles include *operations architects* and also *integration specialists*. Roles from the *IT operations* social structure must ensure the technical aspects of obtaining measures of running services while roles from the *service development team* are responsible of the interpretation, i.e. of the semantics of those figures.
Periodically, while the service is in this stage, service performance reports must be generated. From those reports, characteristics of the services running, like its availability, reliability and performance must be extracted. Those values should be compared with those in the service policies and contracts to check whether the service is running within the allowed limits and restrictions.

Table 4.2 is a summary of the activities, participants and artifacts that compose the run-time phase of the service life-cycle.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Roles</th>
<th>Artifacts</th>
</tr>
</thead>
</table>
| Service Management & Monitoring | From IT Operations social structure:  
  • System Administrators.  
  From Service Development Team social structure:  
  • Operations Architects.  
  • Integration Specialists. | • Service Performance Reports. |

Table 4.2: Run-time phase activities, roles and artifacts

4.4.3 Change-time Phase of the Service Life-cycle

This phase of the service life-cycle corresponds to the period of time when running services present problems, incidents, or customers require changes in the functionality. When this happens, the service must go again into the design phase and, probably, a new version of the service will be generated. This new version includes a modified set of documents that describe the new configuration of the service. This phase includes the service support activity.

Service Support

The Service Support activity manages problems, incidents and interaction with the customers. This activity may conclude with the creation or modification of requirements, including functional and non-functional aspects associated to services. In that case, the resulting documents become the input for a new iteration in the service life-cycle.

Participants of the system playing roles from the Service Development Team and from the Business Domain Representatives are the ones in charge of this activity. The role responsible from the service development team social structure is basically the business analysts. From the business domain representatives social structure, responsible roles in-
clude process engineers and also business subject-matter experts. Roles from the service development team structure represents the service provider architectural role while roles from the business domain representatives social structure represents the service consumer one. They must interact in order to agree on the changes or modifications that must take place in services when that is needed. Those demands must be framed in the context of the contracts that were previously agreed among the parts.

In the case that the parts agree on the modifications, a service change proposals document may be created. As said before, this document becomes the input for a next cycle in the service-lifecycle. This implies that a new version of the service, with a number of modifications to the associated documents should be created.

Table 4.3 is a summary of the activities, participants and artifacts that compose the change-time phase of the service life-cycle.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Roles</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Support</td>
<td>From Service Development Team social structure:</td>
<td>• Service Change Proposals.</td>
</tr>
<tr>
<td></td>
<td>• Business Analysts.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>From Business Domain Representatives social structure:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Process Engineers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Business Subject-matter Experts.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Change-time phase activities, roles and artifacts

4.4.4 Service States

Figure 4.35 shows a state chart diagram with the states of services defined in the life-cycle. Like the service life-cycle, this diagram associate colors to the corresponding phases of the life-cycle.

4.5 Evaluation of Related Works

In this section, the result of the evaluation of some commercial and research solutions is presented.

Solutions from the educational and commercial sectors were evaluated. This evaluation was done based on the available documentation of each of the solutions. This implies that
A document may state that a solution addresses a requirement when it does not. The other way around is also possible. A document may not state that the solution addresses a requirement when it does. This is specially important for commercial solutions, as they mention what they do but not how.

First, a summary of the evaluated requirements are shown. These are based on the requirements analyzed in chapter 3 - Requirements for SOA, Governance and Management. Following, an introduction to the solutions evaluated and a matrix containing an evaluation of each of the requirements in each of the solutions is presented. This matrix summarizes the result of the evaluation. Finally, an analysis of the results is provided.

4.5.1 Summary of the requirements

Table 4.4 presents a summary of the requirements to evaluated the solutions.
<table>
<thead>
<tr>
<th><strong>Short Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>Identification and Description of artifacts excluding services in the sense of a registry in a standarized way.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>Search &amp; location capabilities (during run-time also involves the detection of unregistered services)</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management</td>
<td>Definition of services that allow potential customers discover and binding to services.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>Definition and support of a service life-cycle.</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>Extensible data model (i.e. capability to add user-defined new types of artifacts to ones model)</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>Classification of artifacts with personalized taxonomies for each domain and organization.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>A relationship between the artifacts that lie in different, distributed architectural layers or tiers in SOA-based systems.</td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>Repository functionalities, in the sense of providing a data store.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>Notification mechanisms for actors involved in SOA/BPM Governance processes.</td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>Validation capabilities, e.g. adherence to certain standards and policies.</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>Policy definition, monitoring, and enforcement mechanisms with emphasis on run-time monitoring.</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>General administrative aspects including accounting, monitoring of SLAs, etc.</td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>Reporting and analysis mechanisms personalized for stakeholders, supporting continuous improvement processes and optimization.</td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>Create, update or delete of dependencies between internal services and internal and external services.</td>
</tr>
<tr>
<td>Participants role management</td>
<td>Participants role identification and management, e.g. service provider and service requestor.</td>
</tr>
</tbody>
</table>
Use of an standard information model that allows much easier information sharing in service ecosystems.

Version management of the services.

Table 4.4: Summary of the requirements to evaluate SOA solutions

4.5.2 Solutions Evaluated

This matrix allow the reader to see, very easily, which and how the requirements are addressed by a given solution.

Figure 4.36 shows a matrix of the results of the evaluation of the solutions. In the rows are the requirements for the evaluations of the solutions. In the columns are the solutions that were evaluated. For each of the requirements, a value is associated to each of the solutions. When a solution covers entirely a requirement, ‘E’ is given, when the requirement is partially covered, ‘G’ is given and when is not cover or in not known if the requirement is cover, ‘N’ is given. To understand easier and faster the matrix, a color is associated to each value. For the ‘E’ value, black is associated, for ‘G’ gray and for ‘N’ a paler gray.

First three solutions come from the education sector. The first one is based on the document “Models and Tools for SOA Governance” [Derler, 2007]. In this work an specific model is developed and a service-lifecycle is developed, based on the financial domain.

The second one is based on the document “STraS: A Framework for Semantic Traceability in Enterprise-wide SOA Life-Cycle Management” [Seedorf et al., 2009]. This document has a proposal with use of ontologies. It provides a traceability model based on the semantic web. For example, to make queries, SparSQL is used, which is a language to use with ontologies expressed with Resource Description Framework (RDF).

The third one is based on the document “A Semantically Enhanced Service Repository for Service Oriented application System Development.” [Weiping et al., 2009]. Interesting proposal for a semantic-based repository. Given the fact that it is not governance driven, this solution does not address much functionality in this regard.

The following evaluated solutions come from the commercial sector. Next one is the solution developed by Software AG. The evaluation is based on the document “Comprehensive SOA Governance. Ensuring SOA success with effective, automated control throughout the life-cycle” [Castaldini, 2008]. A good description of governance and other related concepts is done, but not many details on the functionality is given.

For the Oracle solution, a set of documents was evaluated, but a good summary of the SOA governance strategy is found in “SOA Governance: Framework and Best Practices” [Afshar, 2007]. Oracle solution covers almost all of the requirements for the evaluation of
Figure 4.36: Review matrix of the solutions
the solutions.

Sixth solution is an special case: is not either from the educational nor commercial sector, it is an standard for a registry, UDDI [Tsai et al., 2007]. Given the fact that this standard is focused on the repository and not on the governance and management tasks, specific topics of SOA, the coverage of UDDI of the requirements is limited.

Next solution reviewed is the one developed by IBM. They provide a very good documentation about SOA, SOA governance and management. Solutions from IBM cover almost all the evaluated aspects. The main document, containing the description of the functionality provided by the IBM tools, is “WebSphere Service Registry and Repository Handbook” [Braswell et al., 2007], which provides a very complete and useful documentation.

Last commercial solution evaluated was the one provided by SAP. Not very good publicly available documentation was found. The documentation is in the form of slides. Three documents were reviewed: “SAP - Deep Dive into the Enterprise Services Repository” [SAP, 2007a], “Governance for Modeling and implementing Enterprise Services at SAP enterprise SOA solution Management” [SAP, 2007c] and “Enterprise Services Repository - an overview” [SAP, 2007b]. Public documentation of SAP is not very informative. In this case, an empirical evaluation of the solution would be clearly the best option to validate if the tools provide some kind of solution to the requirements.

Details on the results of the review of each of these solutions can be found in appendix A.

4.5.3 Evaluation of the Reviewed Solutions

From the evaluation of the solutions, the following conclusions can be made.

Research Solutions

In general, solutions coming from the education sector focus on specific requirements. The requirements are not production oriented. For example, only one (“Models and Tools for SOA Governance” [Derler, 2007]) provides endpoint and binding information. The advantage of this kind of solutions is that they show how they implement the requirements, which is good towards homogeneity and integration of distributed systems. One solution [Derler, 2007] tries to cover several aspects, but it is domain specific (this solution is specific to the financial sector). A clear example of a focused solution is “A Semantically Enhanced Service Repository for Service Oriented application System Development.” [Weiping et al., 2009], which proposes a repository with semantic information of services, leaving out governance aspects.

These solutions focus provide their results based on the architectural layers and the corresponding artifacts. Two out of three provide a service life-cycle. None of the solutions provide a notification mechanism. Neither management of policies nor a reporting or analysis mechanism is mentioned.
Almost all the solutions provide a notification mechanism. Only UDDI does not specify one. SAP documentation does not mention it explicitly, although one mechanism of this kind is most probable available. The same is the case for policy management. UDDI does not specify this and SAP documentation does not mention it explicitly. In the case of a reporting mechanism, only UDDI does not integrate one.

**Commercial Solutions**

The scope of the requirements addressed by this kind of solutions is broader than the research solutions. The focus of these solutions is more productive oriented. Concrete aspects of running systems, like the endpoint, binding and monitoring of services, are specified. The documentation of these solutions are, in general, neither based on the different layers of the architecture nor in the corresponding artifacts.
Chapter 5

Conclusions

In Chapter 2 - Basics, the fundamentals of SOA, in which this work was based on, were presented. It explained how SOA may bring flexibility and agility to systems implementing this architecture. Besides these advantages, the challenges associated to the implementation of SOA based systems were reviewed. Basically, those challenges consist on the complete service description, management and governance.

Chapter 3 - Requirements for SOA, Governance and Management summarizes the challenges defined in the end of the previous chapter, and presents them in the form of requirements. This requirements are the ones used as the source, not only to develop the solution proposed in this work, but also to evaluate similar solutions.

The solution proposed to the requirements described in the previous chapter is described in Chapter 4 - Work Development. Besides, the results of the evaluation of similar solutions addressing these requirements is shown at the end of this chapter. The solution proposed in this work consists on the usage of meta-models and models to describe the elements and the relations among them that must be defined in SOA-based systems. The meta-models set the framework and specify the information that should be provided to fully describe services. The models can be seen as instances of the meta-models proposed. To obtain the elements and relations that compose the meta-models and the models, several reference model and reference architecture documents were used. These documents were developed by different organizations that have experience in implementing SOA-based systems. In the evaluation of the related works, different solutions, from the educational and private sector, were evaluated. A matrix with the results is shown. Finally, an analysis of the results is given.

This work showed that SOA is an architecture that brings many advantages to systems implementing it. The services layer defines an important number of new elements associated to services. The number of these resources in a running system grows exponentially. Besides, these resources may depend on other resources that may lie in a different architectural layer. For this reason, a formal method to manage and govern these resources
is required. The meta-models, models and the service life-cycle shown in this work provide a publicly available and, to some extend, complete formal method for governance and management of these resources. They were developed in an abstract way, which decouples them from any implementation technology. To show viability of these meta-models, models, based on web services technology, were provided. The flexibility of these models allows any kind of organization, implementing SOA in any particular way, to adapt to the meta-models and models proposed in this work. The life-cycle tells participants when and how can they interact in the system. This information is very important for the governance of this kind of systems as it gives a frame to define the governance processes in the system.

This work gives a general view of the components and relations among the resources in SOA’s. This view may be very useful for a person with interest and no previous knowledge on this topic. The models contained in this work describe the different aspects that are related with services. Each of these aspects can be deeper investigated. Examples of them include semantic representation of services and complete workflows, machine-readable description of services, representation of constraints in policies and contracts, among others. Implementation of tools, based on the models and meta-models developed during this work, would be useful to check and improve them. If these tools were available, actual SOA-based systems could be instantiated. This would be an appropriate setting to validate research in the different areas related to SOA.

Recently, approaches are emerging to describe not only services but also workflows in a semantic way. Given an scenario where thousand of service providers are there, it will be difficult for a consumer to choose the best one (For a deeper discussion on this topic, see [Heuser et al., 2009]). A semantic description of services would allow an agent, acting on behalf of a given customer, at least to filter services so that when the customer must choose one, the number of candidates is, to a great extend, reduced. Another possible use of automatization is when a workflow is expressed in terms of services and one or more of them fail. Then it would be good for the organizations that the engine running the workflow can notice this and, given the semantic description of the failing service, replace it with another one that has the same behavior and terms of use. In this case, aspects like negotiation of policies and contracts for using services must also be taken into account in the automatization processes.
Appendix A

Solutions Review

Following are the results of the review of the solutions. This information can be valuable for a person or organization making a comparison of the solutions, as this can be used as an starting point for that kind of research. For a summary and an analysis of the review of the solutions, see section 4.5 on page 72.

A.1 Models and Tools for SOA Governance

Title
Models and Tools for SOA Governance.

Author
Patricia Derler and Rainer Weinreich.

General Comments
An specific model is developed. This model can be standarized for interaction in distributed environments.
See table A.1 for a summary.
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Eval</th>
<th>General Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>Done with the model. Not special or standard hierarchy specified</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>N</td>
<td>Static info of deployed services. Location is modeled as an attribute.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>G</td>
<td>An own lifecycle for the services is defined.</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>N</td>
<td>A fixed model is used.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>G</td>
<td>The model includes artifacts of several layers.</td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>N</td>
<td>Not runtime capabilities</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>E</td>
<td>Dependencies are modeled. Can I get dependencies info in runtime?</td>
</tr>
<tr>
<td>Participants role management</td>
<td>E</td>
<td>Is modeled as an attribute of the dependency concept.</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>N</td>
<td>An own model is defined</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Models and Tools for SOA Governance

Title

Author
Stefan Seedorf, Khrystyna Nordheimer,....

General Comments
Good Proposal with use of ontologies, lack of use of standards, distributed approach. No index of artifacts is specified for the query. Describes more an extraction tool than a design and creation tool. In section III b they say that in case a process has to be modified, all services to be adapted can be directly determined.
See table A.2 for a summary.

A.3  A Semantically Enhanced Repository

Title
A Semantically Enhanced Service Repository for Service Oriented Application System Development.

Author
Li, Weiping, et all.

General Comments
Interesting proposal for semantic-based repository. It is not governance driven, and because of this lacks a lot of functionalities to this respect.
See table A.3 for a summary.

A.4  Software AG

Title
Comprehensive SOA Governance.
Ensuring SOA success with effective, automated control throughout the lifecycle.

Author
Franco Castaldini. Director, Product Marketing, Software AG.

General Comments
A good introduction to governance, some interesting concepts, broad coverage. Because it is commercial, not many details are given.
See table A.4 for a summary.
<table>
<thead>
<tr>
<th><strong>Requirements</strong></th>
<th><strong>Eval</strong></th>
<th><strong>General Comments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>In page 5 some metadata is briefly (but very briefly) described. Another source for specifying this requirement should be found.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>G</td>
<td>Knowledge base can be queried but no query and index of services is specified.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>N</td>
<td>The work here extracts information but does not design or create new elements.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>G</td>
<td>But the service and process life-cycle is not standarized.</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>E</td>
<td>Based on ontologies..</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>G</td>
<td>The model can complemented with user-defined taxonomies.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>G</td>
<td>They define an own model with only defines two layers. There are standard models that define more layers.</td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>E</td>
<td>In the model a knowledge layer is defined where all the data extracted is stored.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>N</td>
<td>Is mentioned as a possible add-on but is not included in the implementation.</td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>N</td>
<td>The model and life-cycle is their own.</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>N</td>
<td>Nothing of these is implemented.</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>N</td>
<td>To be done in the future.</td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>E</td>
<td>Done in the knowledge layer and using the ontology.</td>
</tr>
<tr>
<td>Participants role management</td>
<td>E</td>
<td>All the artifacts has a responsible.</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>N</td>
<td>An own model is defined.</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Eval</th>
<th>General Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>The artifacts are described using an ontology. In this case it is only referred to services, not other artifacts.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>E</td>
<td>This work is done for improvement of this. Some measures of improvement would be nice.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>G</td>
<td>The description of services give some info to this respect.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Extensible data model</td>
<td>E</td>
<td>Based on ontologies for describing services.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>G</td>
<td>The repository is a semantic one, but by now is only for services.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Participants role management</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>G</td>
<td>Semantic Web Services using OWL-S is used.</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.3: A Semantically Enhanced Service Repository for Service Oriented application System Development
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Eval</th>
<th>General Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>No details of the definition of artifacts are given.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>G</td>
<td>Normal registry capabilities.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>G</td>
<td>The description of services give some info to this respect.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>E</td>
<td>A lifecycle is defined and supported.</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>E</td>
<td>CentraSite includes built-in taxonomies and enables users to create their own taxonomies.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>E</td>
<td>Can be seen as part of the previous point.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>G</td>
<td>The notification mechanism is mentioned but is not deep explained. SMTP and SNMP is used.</td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>E</td>
<td>One important point of these tools is Active Policy Management</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>E</td>
<td>related to the previous point.</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>E</td>
<td>Monitoring is in placed and with that service level agreement is possible.</td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>E</td>
<td>Users are able to create custom reports using X-Query.</td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>E</td>
<td>This is done in the change management phase, and is called impact analysis.</td>
</tr>
<tr>
<td>Participants role management</td>
<td>E</td>
<td>Multi-Role support is included.</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>G</td>
<td>UDDI v3 for federation is used.</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.4: Software AG solution.
A.5 Oracle

**Title**
Oracle SOA Governance Solution, et al.

**Author**
Unknown.

**General Comments**
Several components of Oracle are explained that interconnected give an enterprise-based solution to Oracle Governance
See table A.5 for a summary.

A.6 UDDI

**Title**
Roadmap to full services
Adding OWL-S Support to existing UDDI Infrastructure

**Author**
W.T.Tsai, Jim Luo

**General Comments**
The first consists of the general definition of a service broker and the expected functionality. The second gives a briefly overview of UDDI and explains how to add ontology support.
See table A.6 for a summary.

A.7 IBM

**Title**
SOA governance with WSRR (page 11 to 38)

**Author**
Hargrove, et all.

**General Comments**
Very complete description of Governance in general and in their solution in particular.
See table A.7 for a summary.
<table>
<thead>
<tr>
<th><strong>Requirements</strong></th>
<th><strong>Eval</strong></th>
<th><strong>General Comments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>UDDI v3 Registry is used</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>G</td>
<td>UDDI v3 Registry is used</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>G</td>
<td>Not explicitly mentioned.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>G</td>
<td>Lifecycle support is mentioned but not explained</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>G</td>
<td>Have an extensible system of record for all SOA asset info.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>E</td>
<td>Can be seen as part of the previous point.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td>Not explicitly mentioned.</td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>E</td>
<td>The solution includes a repository, the Oracle enterprise Repository.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>G</td>
<td>Oracle BAM (see Oracle SOA suited data sheet ) mentions alert conditions for communicating with the user.</td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>G</td>
<td>Not explicitly mentioned.</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>E</td>
<td>Monitoring of web services is done in OWSM. Also It has a Policy Manager and Policy Enforcement Points (PEP)</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>E</td>
<td>Mentioned.</td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>E</td>
<td>Oracle enterprise manager basically manage SOA assets dependencies.</td>
</tr>
<tr>
<td>Participants role management</td>
<td>G</td>
<td>Role based visibility is mentioned. Allthough is very probable that participants roles are managed, this is not mentioned.</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>G</td>
<td>UDDI v3 for federation is used.</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.5: Oracle solution.
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Eval</th>
<th>General Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>Is the standard registry used.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>G</td>
<td>A query is used for this approach.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>G</td>
<td>The core data model has a service binding part.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Extensible data model</td>
<td>G</td>
<td>TModel is used.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>N</td>
<td>But can be added with the tModel, that is what the second part uses.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>N</td>
<td>Is only a registry.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>N</td>
<td>Can be as an add-on, as in the other implementations.</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>N</td>
<td>Can be as an add-on, as in the other implementations.</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Participants role management</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>G</td>
<td>this itself is a standard by OASIS.</td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.6: UDDI standard.
<table>
<thead>
<tr>
<th><strong>Requirements</strong></th>
<th><strong>Eval</strong></th>
<th><strong>General Comments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>A metamodel is defined, and also ontologies can be used.</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>E</td>
<td>A registry is part of the solution.</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management</td>
<td>E</td>
<td>The core data model has a service binding part.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>G</td>
<td>Even a meta-lifecycle is defined.</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>G</td>
<td>Is not known, but an ontology can be used.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>G</td>
<td>An ontology can be established but the scope (i.e. if it is also for artifacts (see page 35) is unknown.</td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td>At least not explicitly mentioned</td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>E</td>
<td>Is also a repository the solution.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>E</td>
<td>Notification of events is included.</td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>E</td>
<td>Can be as an add-on, as in the other implementations.</td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>E</td>
<td>Can be as an add-on, as in the other implementations.</td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>E</td>
<td>Complete set of tools and plug-in architecture, which allows to add tools for managing.</td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>G</td>
<td>is not said that is personalized for stakeholders.</td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>G</td>
<td>A dependency is established but not very well explained. An interesting graphical approach is shown.</td>
</tr>
<tr>
<td>Participants role management</td>
<td>G</td>
<td>But not explicitly mentioned</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Versioning</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table A.7: IBM solution.
A.8 SAP

**Title**
SAP - Deep Dive into the Enterprise Services Repository
Governance for Modeling and implementing Enterprise Services at SAP enterprise SOA solution Management
Enterprise Services Repository - an overview.

**Author**
SAP

**General Comments**
Slides. Very superficial explanation but was the best document to find information about SAP governance
See table A.8 for a summary.
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Eval</th>
<th>General Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification and Description of Services and Artifacts</td>
<td>G</td>
<td>It is mentioned to do so</td>
</tr>
<tr>
<td>Search &amp; location capabilities.</td>
<td>E</td>
<td>Have a registry as the solution..</td>
</tr>
<tr>
<td>Service discovery, binding and endpoint management.</td>
<td>E</td>
<td>The core data model has a service binding part.</td>
</tr>
<tr>
<td>Service life-cycle management support</td>
<td>G</td>
<td>in second document, the Governance process is mentioned</td>
</tr>
<tr>
<td>Extensible data model</td>
<td>E</td>
<td>see slide 16 of the third document.</td>
</tr>
<tr>
<td>Classification within personalized taxonomies</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Association of artifacts across architectural layers</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repository functionalities</td>
<td>E</td>
<td>Is also a repository the solution.</td>
</tr>
<tr>
<td>Notification mechanisms for actors</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Validation capabilities</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Policy definition, monitoring, and enforcement mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>General administrative aspects</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reporting and analysis mechanisms</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Dependency management between services</td>
<td>G</td>
<td>In slide 19, cross references are mentioned</td>
</tr>
<tr>
<td>Participants role management</td>
<td>G</td>
<td>See slide 14 of the third document.</td>
</tr>
<tr>
<td>Use of an standard information model</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Versioning</td>
<td>G</td>
<td>see slide 14 of third</td>
</tr>
</tbody>
</table>

Table A.8: IBM solution.
Bibliography


