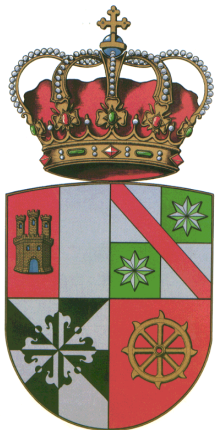
Verification and Validation of Web Service Compositions using Formal Methods



Dissertation for the degree of Doctor of Computer Science to be presented with due permission of the Department of Computer Science, for public examination and debate

---

Author: *José Antonio Mateo Cortés*

Supervisors: *Dr. Valentín Valero Ruiz*

Dr. Jiří Srba

Acknowledgements

Abstract

Nowadays, most of the computing systems are based on service-oriented computing (SOC). This paradigm aims at replacing complex monolithic systems by a composition of interacting systems called services. A service encapsulates self-contained functionality and offers it over a well-defined and standardized interface. It allows cross-organizational collaborations in which each participant is in charge of a particular tasks leading to the development of scalable, flexible and low-cost distributed applications. Each service works as an autonomous component, performing only the tasks for which it has been implemented. As the development of such services is independent, companies can reuse a considerable amount of components, thus saving money and time. Moreover, these technologies are widely used due to their ability to provide interoperability among services from different companies since all the participants know the services offered by the others as well as how to access them.

Due to privacy concerns or commercial policy, entities participating in one of these architectures have no access to complete information, that is, he code implementing the services the particpants consume is hidden, thus being impossible to examine or verify the implementation of the consumed services. Another issue is that web services are usually *stateless*, which means that no state is stored from the clients viewpoint. However, some new applications and services have emerged, which require to capture the state of some resources. Thus, new standards to manage the state of a web service have appeared. For instance, Open Grid Services Infrastructure (OGSI) was conceived to allow designers to manage resources when using web services, and this standard became Web Services Resource Framework (WSRF) when new improvements were introduced.

Obviously, in this scenario the probability of making errors is higher than working in a monolithic scenario. Therefore, there is a clear need of apply-

ing any kind of technique to ensure the correctness of each participant and their composition. In this Thesis, we present a formal language called BPELRF and its semantics. The aim of this language is to model a set of bussiness processess implemented in the de-facto standard modelling language, WS-BPEL, but enriched with the ability to manage distributed resources. These distributed resources are managed according to the guidelines provided by the standard WSRF. Moreover, we provide a visual model of this language in terms of coloured Petri nets in order to ease uninitiated people to deal with it, and we use the well-known toolbox, CPNTools, to verify the composition of web services with distributed resources expressed in BPELRF. As usual, the process of building manually the Petri nets model of large scenarios is time-consuming and error-prone. Therefore, we have implemented a tool to support web designers that, given a BPELRF specification, it extracts automatically the coloured Petri nets of the scenario. Finally, this model can be verified using CPNTools.

On the second part of the Thesis, we extend the classical definition of Workflow nets with time features. Workflow nets were introduced by Wil van der Aalst as a formalism for the modelling, analysis and verification of business workflow processes. The formalism is based on Petri nets abstracting away most of the data while focusing on the possible flow in the system. Its intended use is in finding design errors such as the presence of deadlocks, livelocks and other anomalies in workflow processes. Such correctness criteria can be described via the notion of *soundness* that requires the option to complete the workflow, guarantees proper termination and optionally also the absence of redundant tasks.

After the seminal work on workflow nets, researchers have invested much effort in defining new soundness criteria and/or improving the expressive power of the original model by adding new features and studying the related decidability and complexity questions. In this Thesis, we define a quantitative extension of workflow nets with timing features called timed-arc Workflow nets. It allows us to argue, among others, about the execution intervals of tasks, deadlines and urgent behaviour of workflow processes. Our workflow model is based on timed-arc Petri nets where tokens carry timing information and arcs are labelled with time intervals restricting the available ages of tokens used for transition firing. Here, we consider both discrete and continuous time semantics, thus conforming a whole theory of workflow nets. Finally, all the theory presented in this

thesis have been introduced in the tool Tapaal, offering researchers a mean to model timed-arc workflow nets and to automatically verify (strong) soundness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The development of software systems is becoming more complex with the appearance of new computational paradigms such as Service-Oriented Computing (SOC), Grid Computing and Cloud Computing. In these systems, the service provider needs to ensure some levels of quality and privacy to the final user in a way that had never been raised. It is therefore necessary to develop new techniques to benefit from the advantages of recent approaches, as Web service compositions. Formal models of concurrency have been widely used for the description and analysis of concurrent and distributed systems. Grid/Cloud environments are characterized by a dynamic environment due to the heterogeneity and volatility of resources. To composite web services, there are two complementary views: Choreography and Orchestration. The choreography view describes the observable interactions among services and can be defined by using specific languages such as Web Services Choreography Description Language (WS-CDL) or by using more general languages like UML Messages Sequence Charts (MSC). On the other hand, orchestration concerns the internal behaviour of a Web service in terms of invocations to other services. Web Services Business Process Execution Language (WS-BPEL) [1] is normally used to describe Web service orchestrations, so this is considered the de-facto standard language for describing Web services workflows in terms of web service compositions. Later on, a brief

introduction of WS-CDL is provided, and a deep description of WS-BPEL is introduced since it will be used as part of this work.

To facilitate additional interoperability among services, more standardization is required to deal with distributed resources. In January of 2004, several members of the *Globus Alliance* organization and the computer multinational *IBM* with the help of experts from companies such as *HP, SAP, Akamai, etc.* defined the basis architecture and the initial specification documents of a new standard for that purpose, Web Services Resource Framework (WSRF) [**?**]. Although the Web service definition does not consider the notion of state, interfaces frequently provide the user with the ability to access and manipulate states, that is, data values that persist across, and evolve as a result of Web service interactions. The messages that the services send and receive imply (or encourage programmers to infer) the existence of an associated stateful resource. It is then desirable to define Web service conventions to enable the discovery of, introspection on, and interaction with stateful resources in standard and interoperable ways [**?**].

The main motivation of the first part of the Thesis is to provide a formal semantics for WS-BPEL+WSRF to manage stateful Web services workflows by using the existing machinery in distributed systems, and specifically a well-known formalism, such as prioritised-timed coloured Petri nets, which apart of being a graphical model, but they also provide us an easier way to simulate and analyse the modelled system. Thus, our aim is not to provide just another WS-BPEL semantics. In order to deal with the integration of BPEL plus WSRF in a proper way, we have realized that it is more convenient to introduce a specific semantic model, which covers properly all the relevant aspects of WSRF such as notifications and resource time-outs. The integration of WS-BPEL and WSRF is not new; in the literature, there are a bundle of works defining this integration, but none of these works define a formal semantics in terms of Petri nets.

1.2 Objectives

Next, we describe the general and specific objectives of the Thesis.

Main Objective

The main objective of the Thesis can be splitted into two main objectives. The first one is the definition of a formal language to model Grid/Cloud Computing environments, reusing some of the web technologies presented to date. The second one is to propose a new extension of workflow nets in terms of timed-arc Petri nets, thus providing two formal models in which the time semantics is discrete or continuous. Obviously, these objectives are too general and, therefore, we list below a set of subobjectives that are required to achieve these overall objectives.

Specific Objectives

To meet this overall objectives the following specific objectives should be achieved:

- **Objective 1: State-of-the-art**
  1. Study of different formalisms for the modelling and analysis of Grid/-Cloud Computing applications using web services.
  2. Summarise the current definitions of soundness and the different extensions of workflow nets presented to date.
- **Objective 2: Technological framework definition**
  1. Study of the current techniques for modelling and implementing Web service composition and Grid/Cloud Computing applications.
  2. Analyse the different tools for modelling workflow nets as well as possible target applications of the theory presented in this Thesis.
- **Objective 3: Development of the proposal**
  1. Define the specific models, either the operational semantics or the PTCPNs semantics of the language BPELRF.
  2. Extend the current definition of workflow nets with a time semantics.
  3. Adapt the definition of soundness to this timed scenario.
  4. Develop tools within the theory presented here.

5. Analyse and evaluate both proposals.

- **Objective 4: Examples and Case studies**

  1. Propose a set of simple examples where the main features are displayed.
  2. Study a set of theoretical examples where the power of both proposals and its main aspects is characterized .
  3. Demonstrate the applicability of this work applying it to real (industry-based) case studies.

1.3 Dissertation Structure

This Thesis is organised in five different chapters as follows.

Chapter 1 makes a brief introduction of the Thesis, showing the motivation, the main objectives and the scope of it.

Chapter 2 shows the state of art of the contents included here. This chapter includes a brief description of Service-Oriented Computing (SOC) and distributed computing, e.g. Grid and Cloud computing and the use of formal methods for the analysis of web service compositions and the benefits of using formal techniques in the development of software and hardware. Moreover, a deep introduction of the standards used in this work is presented. Finally, we get into workflow nets and the possible extension of this formal model as well as its main properties.

Chapter 3 presents a formal specification language called BPELRF, which takes two well-known standards (WS-BPEL and WSRF) as basis, to model synchronous and asynchronous stateful interactions. This language is enriched with a publish-subscribe architecture, service discovery, event and fault handling and time-outs. As usual, an operational semantics for this language is defined. Moreover, we define a visual model of it in terms of coloured Petri nets and a tool to verify some properties of the specifications written in BPELRF.

In **Chapter 4** we suggest a workflow model based on timed-arc Petri nets and study the foundational problems of soundness and strong (time-bounded) soundness. We explore the decidability of these problems and compare the discrete and continuous semantics of timed-arc workflow nets.

Chapter 5 shows the main conclusions, contributions and future works of this Thesis.

Chapter 2

State of the Art

In this chapter, it will be introduced the state-of-the-art related to the specification, formalization and verfication of stateful web services and their composition as well as the use of formal methods in this topic. The aim of this chapter is to provide the reader with the basic notions about formal methods and stateful web service compositions in order to help he/she in the understanding of the Thesis. To begin with, a brief introduction of formal methods and why they are needed is presented. Second, a survey about the diffent technologies used to model web services and the different approaches to compose them are introduced and, next, the different mechanisms available to improve these web services with distributed resources. Finally, the different formal models used here are defined. On the other hand, we introduce workflow nets and why they are useful to model business processess. Some informal definition about the properties can be studied with this formal model is also provided.

2.1 Motivation

Throughout the history of computing , engineers and researchers have used different formal methods to improve the quality of hardware and software. These systems with continuous technological progress in integration techniques and programming methodologies inevitably grow in scale and complexity. Because of this complexity, the probability of error is higher and, in addition, some of

Figure 2.1. Example of systems where formal methods are (can be) used .

these errors can cause incalculable economic losses, time or even the loss of human lives. Therefore, the main aim of designers should be to provide developers with the required tools to build systems with a negligible error rate and with the lowest cost. However, this task is far from trivial since one needs to ensure the correctness of the specifications and needs to provide techniques that ease error detection and the verification of the developed models without consuming so much time of the development process. One of the ways that engineers have been used to achieve this goal is the use of formal techniques to ensure the correctness of the development process as well as the product under construction. These formal methods can be defined as the set of procedures and tools based on mathematical languages that virtually ensure the correctness of a system [4] since they increase the level of knowledge that the participants have about the system, revealing inconsistencies and ambiguities that could not be detected using other techniques, i.e., the use of formal methods provides a greater degree of refinement of the model than other methods.

In the past, the use of formal techniques in practice seemed to be utopian and unrealizable. Among other causes, the notations used to require a high mathematical background in mathematics and, therefore, they were too complicated for the uninitiated in the topic. The techniques did not allow the system

to be scalable and the existing tools were too difficult to use or understand or even there were no tools for a particular technique or formalism. In addition, case studies were not convincing enough and, therefore, developers could not appreciate the usefulness of formalization. However, in the early 90s, it started to glimpse a new way in this area. For the specification of software, the industry began to use the language Z [8] in order to obtain rigorous specifications. For hardware verification, major companies such as Intel and AMD started to use formal techniques such as *model checking* or *theorem proving* to supplement tests on simulators. This led to the description of larger case studies, which was benefitial for the advance of this area since other developers started to consider the possibility of introducing the use of formal techniques into their development processes. In Figure 2.1, one can observe different systems in which these techniques are currently used to ensure proper operation. For instance, big companies (e.g Boeing and Airbus) use formal languages to specify the requirements of the equipment as well as they use formal methods to verify the most critical systems in the aircrafts. Moreover, automotive companies verify the most critical systems (e.g. brake or airbag systems) using *model checking*.

The main advantages of using formal methods are:

- The use of mathematics as a base gives this approach a certain rigour.
- Identify ambiguity and inconsistencies.
- Facilitates the construction of consistent and *deadlock-free* systems.
- Provides customer confidence in the system.
- There are many tools that support the existing techniques.
- Find bugs early should save money.

The main disadvantages (or beliefs) that slow the progress of this area are:

- It is believed that the use of formal methods slows the development process.
- Many developers think it is difficult to work with formal specifications.
- It does not guarantee the correctness of the implemented code (only the model it is based).

- Increasing system complexity causes an exponential increase the complexity of the verification.

As commented previously, companies can use formal methods along the entire development lifecycle of a system, both hardware and software. Here, we will focus on software since this Thesis studies different standards for building software components. Next, we describe the different phases in which designers can apply any formal technique.

One of the most important part in the development of a system is the requirements specification. A specification can be seen as a technical document where the features and services needed to build a product are stated. Nevertheless, it can also include information on subsequent steps such as verification, validation, testing, etc. Therefore, this should be the first part in which the participants should apply formal methods, taking the required time to correctly specify the system since a neat and correct specification will influence the rest of the proccess. Anyway, make a proper specification does not guarantee the absence of errors because the presence of faults is an intrinsic characteristic of the systems. In this sense, the simple act of writing the document helps engineers to find errors in the early stages of the development process, helping the company to save money and time. In Figure 2.2, one can observe what is the effect (in money) of finding a bug in the different phases. As can be observed, the cost of fixing a bug increases as we advance in the lifecycle and, therefore, it is recommended to find these bugs as soon as possible. In this Thesis, we propose a formal language and its visual model to specify web service compositions with distributed resources, but this will be presented in Chapter 3.

In the classic life cycle, the verification and validation phases are performed after the implementation phase, but as we have seen in Figure 2.2, it is advisable to detect these errors as soon as possible. As expected, it is practically impossible to verify completely all the behaviour of a complex system so that the goal of researchers in this area is to check whether certain properties hold in the model. The properties of interest will be related to the classical problems of concurrency (*deadlock, mutual exclusion,...*) and some aspects directly related to the system itself such as check the adherence of it to certain time constraints. For example, in a banking system, it is mandatory to ensure that transactions

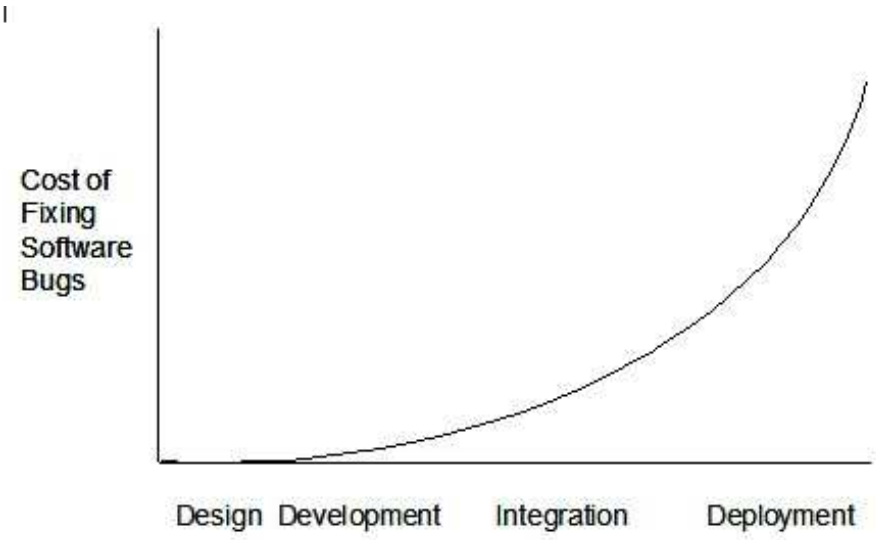


Figure 2.2. Cost evolution of fixing a bug.

meet the stipulated time for completion because if you exceed these restrictions some security issues could come out.

In this sense, one can follow two different ways to perform the verification of a system: *Human-directed proof or Automated proof*. The first one is used when you want to strengthen the knowledge of the system rather than completely ensure the correctness of it, and, therefore, it is a person who check the properties manually. This variant improves the knowledge of the system, but it is time-consuming and error-prone due to the entire process is conducted for a human being. In the second approach (*automated proof*) there are also two variants: *automated theorem proving and model checking*. The *automated theorem proving* is conducted by a program that tries to produce a formal proof of a system from scratch, giving a description of it, a set of logical axioms and a set of inference rules. On the other hand, model checking [5] is a technique for verifying finite state concurrent systems. It has a number of advantages over traditional approaches that are based on simulation, testing, and deductive reasoning. In particular, model checking is normally automatic and usually quite fast. Also, if the design contains an error, model checking will produce a counterexample that can be used to pinpoint the source of the error. Here, the specification can be expressed in propositional temporal logic propositional normally LTL [10] or CTL [12] or some of its variants, and the system is represented as a graph of

transitions between states. The main challenge in model checking is dealing with the state space explosion problem. When dealing with web systems, this problem occurs in systems with many components that can interact with each other or systems with data structures that have many different values. In such cases the number of global states can be enormous. Researchers have made considerable progress on this problem over the last ten years.

2.2 Web Services modelling

Although the Web was initially intended for the exclusive use of human beings, many experts believe that it needs to evolve (probably through modular design and construction services) to better support for the automation of many tasks. The concept of *service* provides a higher level of abstraction to organize large-scale applications and build more open environments, helping to develop applications with improved productivity and quality with respect to other approaches. As services are only a mean for building distributed applications, it is required to evaluate the different existing approaches in this area. Figure 2.3 shows an example of service-based architecture, where there are three main parts: a consumer, a provider (the servers) and a set of records, where the services are stored. The role of the providers is to publish and/or advertise the services offered in the records, where consumers can find and invoke them. Current standards that support interactions between web services provide a solid foundation for service-oriented architecture. The web architecture is a framework that can be reinforced with more powerful representations and techniques inherited from other approaches.

In this way, Service-Oriented Computing (SOC) paradigm promotes the use of services for the development of massively distributed applications, trying to achieve the creation of fast, low-cost, flexible and scalable applications [**?**]. Services are the main building block of this paradigm, being these services self-describing and platform-independent. Thanks to the use of standards for the description, publication, discovery and invocation, the services can be integrated without taking care of the low-level implementation details of each service. The aim of SOC is to make possible the creation of dynamic business processes and

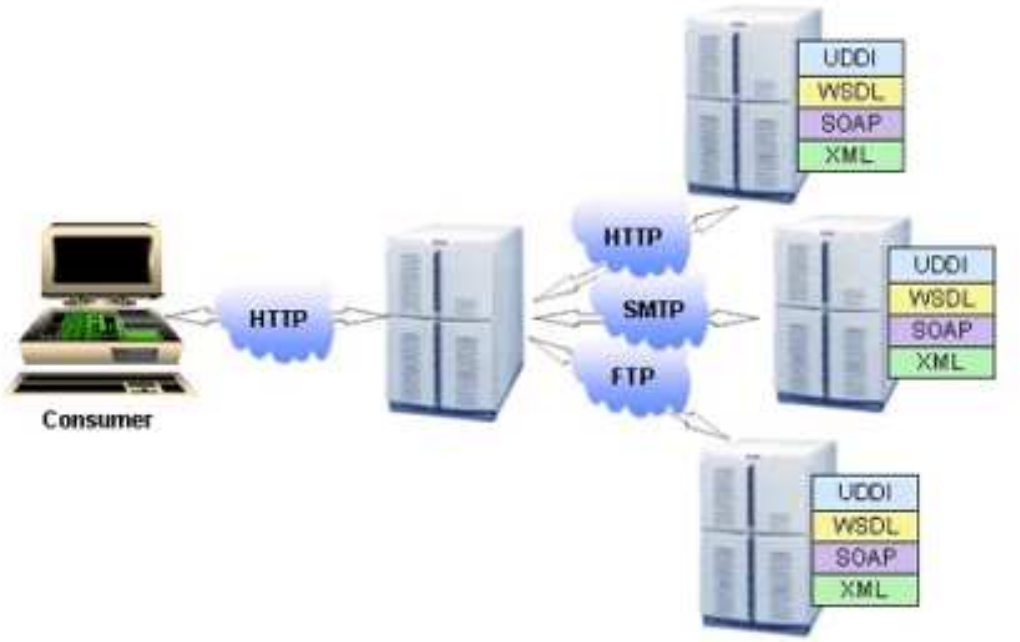


Figure 2.3. Client-server web architecture

agile applications by providing an easy way to assemble application components into a loosely coupled network of services.

To reach the goals of SOC, a Service-Oriented Architecture (SOA) is defined. SOA is a software architecture based on the utilization of services, being these services provided to the user of the application or to other services in the network. This is possible by the use of service interfaces that can be published and discovered. SOA is based on a model of roles where every service can play multiple roles. For example, a service can offer certain functionality to a user and, at the same time, being the consumer of the functionality provided by some other services. Such model reduces the complexity of applications and increases their flexibility. Although at the beginning of SOA there were several architectures aspiring to become SOA standards [**?**, **?**], the most successful one was the architecture based on Web Services.

W3C defines a Web Service (WS) in the following way:

> "A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an inter-

face described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

We can see in this definition that there are two basic standards related to Web Services: Web Service Description Language (WSDL) for the definition of the service functionality and its properties [**?**], and Simple Object Access Protocol (SOAP) for the exchange of XML messages between services [**?**]. There is also an additional standard called Universal Description, Discovery and Integration (UDDI) used to create Web Service directories and to search for services in the network [**?**], but this is a bit out of date. The use of these standard protocols is the key point to improve the integration between different parties in a web service architecture.

In Figure 2.8 a possible representation of the web service architecture stack is shown. One can see that the three standards described above are only a small part of the stack. One also need protocols to define security aspects (ensuring that exchanges of information are not modified or forgotten in a verifiable manner and that parties can be authenticated), to provide reliable messaging for the exchange of information between parties, to specify the collaboration between services when we compose them, to individually describe the behaviour of each service in a business process, etc. The problem is that whereas the standards for basic services (WSDL and SOAP) are widely adopted for their respective purposes, the situation is not very clear when we talk about composing services, having multiple protocols aspiring to become a standard in this layer.

Two different approaches can be followed when we designing web service compositions. They are called *orchestration* and *choreography*. The former describes the individual business process followed by each one of the participants in the composition, while the latter describes the composition from a global viewpoint, defining the interactions (exchange of messages) happening between the parties, that is, how they collaborate in the composition. However, the ideal solution would be fusing both approaches in a single language and environment [**?**].

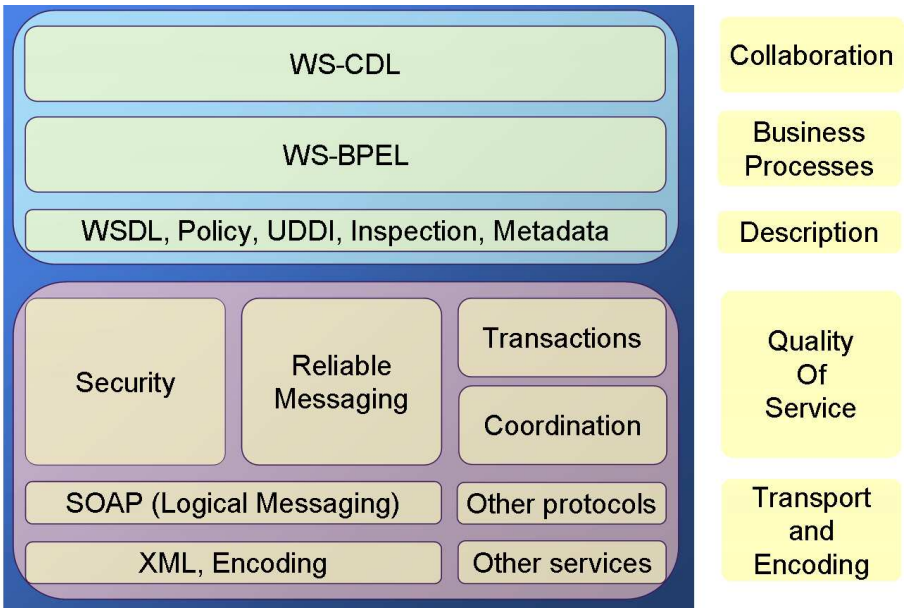


Figure 2.4. Web Service architecture stack.

Anyway, the languages we can use in both cases should accomplish some common goals: (i) the capacity of modelling service interactions, including control flow and data constraints, (ii) the possibility of specifying exceptional behaviour, indicating which errors can happen in the execution of the composition and the way of handling these errors, and (iii) the ability to model web service compositions at a high level, without taking care of the implementation details of each one of the services.

Regarding the choreography approach, there are several languages that have been designed for that purpose. One of the most popular languages is Web Services Choreography Description Language (WS-CDL), which specifies the common and complementary observable behaviour of all participants in a composition [**?**]. It is based on XML and describes the peer-to-peer collaborations between the composite web services from a global point of view, that is, the exchange of messages to achieve a common business goal. The aim of this language is allowing the composition of any kind of web services, regardless of the platform hosting the service or the implementation language. Figure 2.5 is an example of how WS-CDL can be useful for the integration of different kinds of web services.

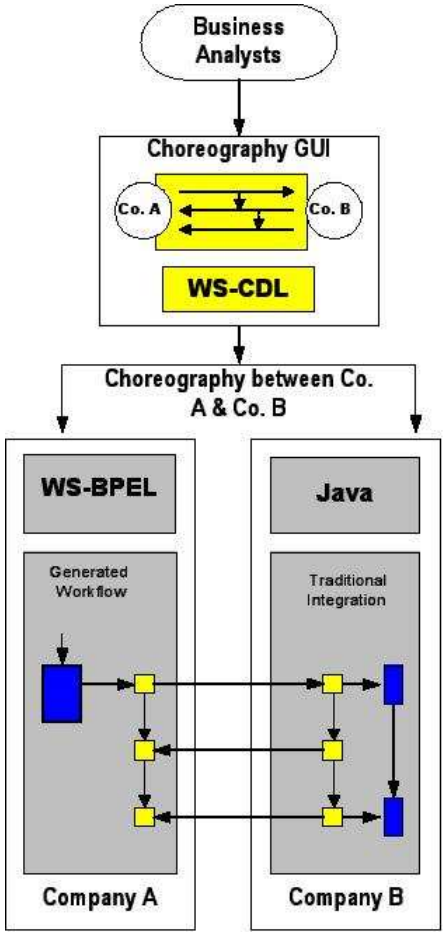


Figure 2.5. Integration of Web Services using WS-CDL.

A WS-CDL document defines a hierarchy of choreographies, where there is only one top-level choreography, marked explicitly as the *root choreography*. The basic building block of a choreography is the *interaction* element. It indicates information exchanges between participants, possibly including the synchronization of some information values. These interactions are performed when one participant sends a message to another participant in the choreography. When the message exchanges complete successfully, the interaction completes normally.

We can distinguish two different kinds of *complex activities* inside a choreography: the workunit element and the ordering structures. The *workunit* element specifies a condition that must be fulfilled in order to perform some work and/or the repetition of some work. It completes successfully when the set of activities

inside completes successfully. *Ordering structures* are used to combine basic activities and other complex activities in a nested way, expressing the order in which actions are performed within the choreography. There are three ordering structures: The *sequence* ordering structure expresses that the set of activities inside must be executed sequentially. The *parallel* ordering structure indicates that the set of activities inside must be executed concurrently. It completes successfully when all the concurrent activities complete successfully. And the *choice* ordering structure specifies that only one of multiple activities can be executed. If the choice have workunits inside, only the first one in lexical order with a "true" guard condition is selected. If there are other activities, there is no way to know which one is selected; it is considered as a non-observable decision.

Different types of exceptions are considered in WS-CDL. Exception workunits can be defined to handle all these exceptions. They may also be used as the mechanism to recover from the exceptions. At least one exception workunit must be defined. The guard of the workunit can be used to specify the particular type of exception we want to handle. Only one exception workunit can match each exception. If multiple exception workunits are defined, the order of evaluating them is based on the order in which the workunits have been defined. When the matching happens, the actions of the matched workunit are executed. If no matching happens and a default exception workunit exists, then the actions of this workunit are executed. Otherwise, the exception is raised in the parent choreography. WS-CDL also allows us to define finalization actions within a choreography that can confirm or cancel the effects of this choreography, so we can use this actions for compensation.

==========Hasta aqu==========

2.3 Cloud Computing/Grid Computing

Gracias a la rpida evolucin que ha tenido la sociedad, servicios bsicos para el desarrollo de la vida cotidiana son comnmente suministrados a los ciudadanos, de tal manera, que cualquier persona puede tener acceso inmediato a ellos de forma fcil. Hoy en da, estos servicios, conocidos en el mundo anglosajn como "utility services", engloban el suministro de agua, electricidad, gas y telfono, pero en los

ltimos tiempos est cobrando fuerza una vieja idea que se intent llevar a cabo sin xito a finales de los aos 60 y principios de los 70, el “Utility Computing”. Este nuevo paradigma de computacin es normalmente confundido con Cloud o Grid Computing, pero hay ciertos matices que los diferencian. “Utility Computing” se puede entender como el modelo de negocio que subyace en una infraestructura Cloud o Grid, es decir, puede ser entendido como el medio de cobro de servicios computacionales similar al que se hace con la electricidad, por lo que el usuario pagar slo por su consumo, mientras que los costes asociados a la produccin y distribucin de potencia de cmputo sern sufragados por las compaas suministradoras. As, las pequeas y medianas empresas podran competir en igualdad de condiciones con las grandes empresas, ya que no sera necesario hacer una gran inversin en datacenters para poder ofrecer un determinado servicio y se fomentara la creacin de empresas, ya que estos datacenters suponen una fuerte inversin inicial que muchos emprendedores no pueden acometer. Adems, el usuario final de las aplicaciones, servicios o infraestructuras tambin se beneficiara porque al reducir costes de produccin se reduce el precio de los productos. Por seguir con el smil de la energa, podemos ver el cloud computing como una gran central generadora de energa que da suministro a millones de usuarios y que evita que dichos usuarios tengan que tener su propia central en casa para poder encender sus aparatos elctricos, mientras que el “utility computing” es la forma de tarificar el gasto de los usuarios o, de una forma ms abstracta, podemos verlo como el contador que muestra el consumo energtico. Al igual que pasa con el software, los protocolos o cualquier paradigma relacionado con la informtica, Cloud Computing debe atravesar una serie de etapas para poder comprobar si toda esta publicidad que le estn dando las empresas sirve de veras para ahorrar costes y favorecer la competitividad o solo es ms una forma de aumentar ingresos o, en el caso de la investigacin, obtener nuevos fondos. En este sentido, algunos autores consideran que Cloud Computing no es mas que una nueva forma de nombrar lo que toda la vida se ha llamado Grid o Web Services y que realmente no supone ningn avance en el campo de la informtica. Este artculo tratar de presentar ms ampliamente la arquitectura y conceptos para comprender un sistema de computacin basado en la nube y mostrar las diferencias entre dos enfoques clsicos de computacin (Grid y Web Services) y Cloud. Finalmente, se propondr una serie de ideas que pueden llegar a convertirse en trabajos de investigacin en un futuro.

2.4 Introduccin

En 1943, el presidente de IBM, Thomas J. Watson, predijo:

"I think there is a world market for about five computers"

Esta frase, tan comentada en el mundo de la informtica en los ltimos tiempos, ha pasado de ser una prediccin con poco fundamento a ser una realidad en la actualidad.

Cloud Computing, el viejo sueo de ofrecer servicios de computacin como utilidad, tiene el potencial de transformar gran parte de la industra informtica, haciendo el software ms atractivo al ofrecerlo como servicio y moldeando la forma en que se disea y compra el hardware. Con este nuevo enfoque, cualquier emprendedor con buenas ideas para ofrecer servicios a travs de Internet no necesitar realizar grandes inversiones en equipamiento para llevar a cabo su proyecto ni necesitar contratar inicialmente mucho personal que gestione y mantenga dicho equipamiento. Adems, no tiene que realizar complicados estudios previos para calcular el nmero de usuarios potenciales y evitar, as, unos de los principales quebraderos de cabeza de los jefes de proyecto: el sobre-aprovisionamiento o el infra-aprovisionamiento. Estos dos conceptos junto con la elasticidad de recursos pueden ser considerados como claves en computacin en la nube, ya que el objetivo de reducir costes es directamente proporcional a la correcta estimacin de recursos en "tiempo real" y esta correcta estimacin slo se puede proporcionar si el sistema cumple la propiedad de elasticidad, es decir, que en un intervalo de tiempo relativamente corto aumentas y disminuyes los recursos dedicados a una tarea con un coste econmico bajo. Este enfoque puede hacer que, a primera vista, no se perciba la posibilidad de utilizar mtodos formales con este tipo de sistemas, puesto que normalmente se utilizan tcnicas formales en el diseo de sistemas con tiempos de respuesta crticos como sistemas de navegacin de un avin, transporte de materiales peligrosos, etc. De esta manera, se hace inviable el uso de la computacin en la nube cuando se exijan tiempos de respuesta muy bajos, ya que, por ejemplo, un sistema de navegacin de un avin no puede esperar varios minutos a que se le asignen nuevos recursos para tomar una decisin. Sin embargo, si que hay otro tipo de sistemas en los que los mtodos formales y el

cloud computing pueden converger, los sistemas de alta disponibilidad.

As, utilizando tcnicas formales se pueden disear este tipo de sistemas y verificar la ausencia de fallos en su construccin. Por ejemplo, una tienda de venta online podra pasar de tener cientos de usuarios simultneamente a miles de ellos en periodos como las vacaciones de Navidad, de manera que necesitara mucha mayor potencia de computacin si quiere satisfacer a todos los clientes y no perder ingresos ni nuevos clientes por no poder atender esa demanda. Para satisfacer esta necesidad, hara un estudio preliminar de cuantas visitas como mximo puede tener en ese perodo y comprara los datacenters necesarios para no tener problemas de congestin, lo que supone una inversin grande en infraestructura por parte de la compaa, sin embargo, una vez acaba la campaa navidea la demanda de usuarios vuelve a ser de unos cientos y la empresa se encuentra con que tiene una potencia de cmputo que no va a necesitar y, por tanto, no est amortizando econmicamente la inversin realizada. En este sentido, si la compaa en lugar de comprar los datacenters hubiese comprado capacidad de cmputo a un proveedor entonces habra amortizado en mayor medida el dinero invertido y no tendra mquinas en sus oficinas que ocupan bastante espacio y que tienen unos niveles de carga de trabajo muy bajos.

Cloud Computing se refiere tanto a las aplicaciones que se ofrecen como servicios a travs de Internet como al hardware y software que est presente en los datacenters que proveen dichos servicios. Estos servicios se han referido normalmente como Software as a Service (SaaS). El datacenter en s es lo que se considera la nube (o cloud). Desde el punto de vista del hardware, hay tres aspectos novedosos en Cloud Computing:

1. La ilusin de tener recursos ilimitados bajo demanda eliminando a los usuarios la necesidad de aprovisionarse antes de acometer una tarea.

2. La eliminacin de la inversin inicial en equipamiento permitiendo a las compaas empezar con pocos recursos e ir aumentndolos cuando las necesidades aumenten.

3. La posibilidad de pagar por el uso de recursos de computacin a corto plazo segn se necesiten (por ejemplo, procesadores por hora o capacidad de almacenamiento de datos por da) y poder liberarlos cuando no sean necesarios.

Cloud Computing podra tener el mismo impacto en la produccin de software que el que tuvieron las fundiciones de metal en la industra del hardware. En principio, las compaas fabricantes de hardware necesitaban tener sus propias instalaciones donde fabricar los componentes que componan sus productos, lo que les supona un gran esfuerzo econmico para construir y operar estas instalaciones y, por consiguiente, haca que el precio de los equipos se doblase en cada nueva generacin. Sin embargo, la aparicin de compaas que fabricasen componentes favoreci que empresas ms pequeas pudiesen entrar en el mercado del hardware, copado hasta aquel entonces por Intel o Samsung, que eran las nicas que podan hacer frente a este gran esfuerzo econmico. De manera similar, la computacin en la nube podra jugar el papel que hicieron las fundiciones, favoreciendo la competencia y evitando monopolios de grandes empresas.

Debido a que muchas empresas usan servicios software como base para su modelo de negocio, se presentan, a continuacin, los actores que formarn parte de este escenario. Los *Services Providers*(SPs) hacen accesibles los servicios a los *Service Users* por medio de interfaces que se comunican a travs de Internet. Dado que uno de los objetivos de la nube es externalizar la provisin de servicios, se necesita la aparicin de otro actor que ofrezca esta infraestructura como "servicio" llamado *Infrastructure Provider*, migrando los recursos desde los SPs al IPs y, as, los SPs pueden ganar flexibilidad y reducir costes como se puede ver en la figura 2.6.

2.5 Comparacin entre servicios web y Grid computing/Cloud computing

Como es sabido, nuestro grupo de investigacin ha centrado su investigacin en el desarrollo de una metodologa que permita construir y verificar sistemas con restricciones temporales mediante el uso de tcnicas formales. En los ltimos aos, se ha aplicado esta metodologa en el rea de los servicios web, ms conc-

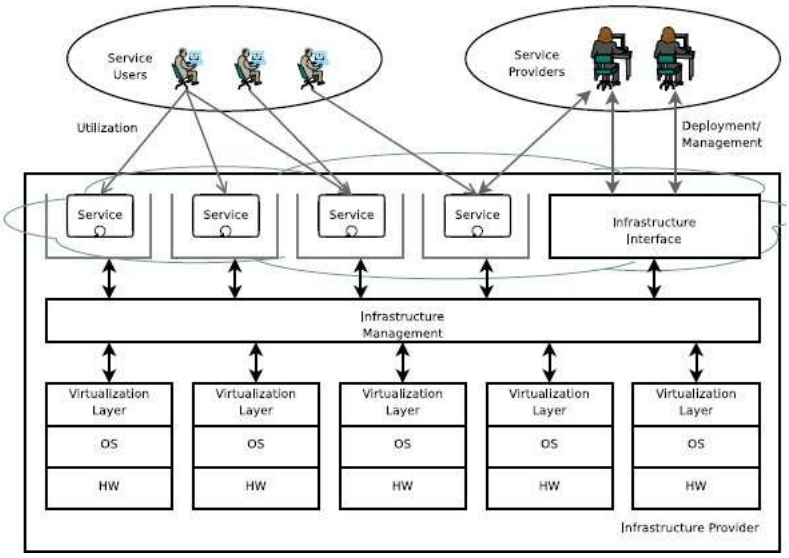


Figure 2.6. Actores en un sistema en la nube.

retamente, en que estos servicios cumplan la tarea que se les encomienda y que se coordinen automticamente para conseguir llevar a cabo un trabajo ms general. El problema que estn teniendo los servicios web es que como tuvieron un gran auge hace pocos aos, muchos grupos de investigacin centraron sus estudios en este campo y, por tanto, hay muchos investigadores proponiendo nuevas aproximaciones y sto ha llevado a que existen ciertas partes como BPEL o WS-CDL que estn bastante estudiadas. De esta manera, esta surgiendo un sistema donde los mtodos formales pueden jugar un papel muy importante y donde nuestro grupo puede beneficiarse de su amplia experiencia tanto en formalizacin como en servicios web, el cloud computing. Este nuevo paradigma, como se ha expuesto anteriormente, est viviendo su poca de plenitud en este momento y grandes empresas como Google, IBM, Microsoft han decidido dar un paso al frente y apostar fuertemente por la computacin en la nube. Adems, muchos gobiernos estn interesados en migrar sus servicios a la nube para abaratar costes y permitirles escalabilidad cuando les sea necesaria. Por ejemplo, hay que preguntarse si es necesario para la Agencia Tributaria tener grandes centros de datos cuando la demanda de servicios por parte de los ciudadanos slo crece en la poca de la declaracin de la renta. Probablemente la respuesta sea afirmativa porque s necesita almacenar todos esos datos y dar cierta confianza de que tus datos fiscales no van a caer en manos de gente con no muy buenas intenciones, pero toda la necesidad de clculo si que se puede externalizar para

ahorrar costes en equipamiento o incluso podran crear una nube privada entre todos los organismos que colaboren con la agencia pblica para compartir recursos e informacin. En este sentido, este tipo de sistemas donde la seguridad, privacidad y la disponibilidad son un requisito innegociable es donde podemos centrar parte de nuestras investigaciones e intentar mejorar alguno de los componentes de la arquitectura cloud expuesta en el apartado anterior. Por ejemplo, la mayora de grupos de investigacin desarrollan herramientas, pero la fase de pruebas o no existe o se le dedica poco tiempo. La semana pasada nos reunimos con uno de los grandes investigadores en el campo del Grid/Cloud Computing, Karim Djemame, y nos cont que el principal problema que tena era ese que no saban concretamente porque funcionaba bien su herramienta y que estaba bastante interesado en la verificacin de su herramienta.

Por otro lado, a continuacin se enumeran algunas diferencias entre servicios web y grid/cloud computing para ver donde es posible aplicar nuestra experiencia en este sistema. En primer lugar, podemos considerar que los servicios web son en s software que se ofrece como servicio (SaaS), aunque existan ciertas diferencias entre ambos enfoques, por ejemplo, la estandarizacin. Por tanto, este software podra estar compuesto de un conjunto de servicios, probablemente comunicados a travs de Internet, y que se coordinan para realizar una determinada tarea. Hasta el momento, nada nuevo, pero la principal diferencia reside en la virtualizacin, ya que los diferentes servicios que ofrece la nube se realizan en mquinas virtuales en lugar de directamente sobre un servidor como puede ser el caso del servicio web, de manera que la concurrencia en el sistema es mayor.

Otra diferencia es la persistencia de los datos. Si queremos coordinar varios servicios web para que realicen sumas la nica posibilidad de que stos puedan almacenar el resultado es guardndolo en la base de datos, sin embargo, existe una aproximacin llamada WSRF (Web Services Resources Framework) que ha sido estandarizada y que resuelve este problema. En este framework cada servicio web lleva asociado un recurso o varios del sistema de manera que puedes interactuar con el servicio y decidir a que recurso acceder. La principal ventaja que tiene es que todos los servicios se definen con WSDL (Web Services Descrip-

tion Language) y que la comunicacin, direccionamiento, etc. est estandarizado, de manera que la colaboracin entre sistemas de este tipo es sencilla. Otra ventaja es que el usuario tiene la posibilidad de decidir con que recursos interacta. As, podramos aadir una capa inferior en nuestra metodologa que permitiese la definicin de servicios web con recursos y una vez verificado que el sistema es correcto, desplegar estos servicios web en las mquinas fsicas. Este enfoque encajara perfectamente con nuestra investigacin, ya que utiliza servicios web con recursos y estos recursos tienen restricciones temporales para evitar que un usuario abarque todo el sistema.

Tambin, podemos observar que cloud computing podra verse como una capa que se colocara debajo de los servicios web, ya que se puede utilizar stos para acceder a los recursos, pero hay que resaltar que la nube no es solo ofrecer software como servicio, sino que tambin hay infraestructura y plataforma como servicio, cosa que los servicios web no pueden abarcar. Es decir, una parte del cloud computing (SaaS) puede compararse directamente con los servicios web, pero las otras dos partes no tienen nada que ver, por lo que sera como comparar el protocolo TCP/IP con la arquitectura de un PC, aunque es necesario que ambas aproximaciones (servicios web y cloud computing) converjan para el crecimiento de ambos paradigmas, igual que grid computing y servicios web convergieron en WSRF.

Por ltimo, a modo de curiosidad la principal diferencia entre un sistema grid y uno cloud reside en la virtualizacin, ya que en grid el usuario no comparte en tiempo real los recursos que tiene asignados, mientras que en cloud es indispensable la virtualizacin de recursos para conseguir dar servicio a ms clientes y conseguir ese ahorro que prometen los proveedores.

2.6 Web Services Resource Framework(WSRF)

La arquitectura que presentan los servicios web ha sido ampliamente aceptada como medio para estructurar las interacciones existentes entre los servicios que forman parte de un sistema distribuido y que colaborar para conseguir un ob-

jetivo comn. En la actualidad, los desarrolladores requieren a los entornos una mayor estandarizacin para facilitar interoperatividad adicional entre dichos servicios, pero hasta mediados de 2004 ningn grupo de investigacin o grupo de expertos se haba planteado seriamente la idea de proponer un estndar para modelar la comunicacin entre servicios web que poseen recursos persistentes asociados. As, en Enero de ese ao, varios miembros de la organizacin *Globus Alliance* y de la multinacional informtica IBM definieron, con la ayuda de expertos de empresas como HP, SAP, Akamai, etc., la especificacin de los documentos que deberan producirse en este modelo y la base de una arquitectura inicial. Estos documentos fueron enviados a la organizacin encargada de su estandarizacin, OASIS, en Marzo de 2004. En un principio, se formaron dos comits que se encargaran del estudio y desarrollo de ciertas partes de este nuevo estndar. Por un lado, estaba el *WSRF Technical Committee* que gestionaba cuatro especificaciones: *WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, y WS-BaseFaults*. Por otro lado, el *WSN Technical Committee* se encargaba de las especificaciones: *WS-BaseNotification, WS-Topics, y WS-BrokeredNotification*.

WS-Resource Framework est inspirado en el trabajo realizado previamente por el *Global Grid Forum's Open Grid Services Infrastructure (OGSI) Working Group* [11]. Ms concretamente, puede ser visto como una sencilla refactorizacin de los conceptos e interfaces desarrollados en la especificacin *OGSI V1.0*, de manera que explota los recientes desarrollos en el rea de los servicios web (por ejemplo, WS-Addressing).

El objetivo de este trabajo es introducir los conceptos fundamentales para la gestin y destruccin de servicios web persistentes, es decir, servicios web que llevan asociados recursos donde guardar los estados de los mismos, ya que hasta la aparicin de esta aproximacin, los servicios web eran considerados "*stateless*" y, por tanto, no podan almacenar temporalmente datos o resultados de sus operaciones de una manera sencilla para el usuario, ya que era necesario almacenarlos en una base de datos ajena al servicio. En este enfoque, es necesario codificar la relacin entre el servicio y el recurso en trminos de patrones utilizando una serie de tecnologas ampliamente estudiadas, como, por ejemplo, el WS-Addressing y, tambin, ser necesario hacer sus propiedades accesibles desde

el exterior a travs de un interfaz. En este sentido, llamaremos *WS-Resource* a la asociacin entre un servicio web y un recurso persistente.

2.6.1 Introduccin

WS-Resource Framework [3] es una especificacin, desarrollada por OASIS y algunas de las empresas informticas ms pioneras, cuyo propsito es definir un marco genrico para el modelado y acceso a recursos asociados a servicios web, as como las relaciones entre dichos recursos en un entorno Grid/Cloud. Esta aproximacin est compuesta por un conjunto de especificaciones que definen la representacin del WS-Resource en los trminos que especifican los mensajes intercambiados y los documentos XML relacionados. Asimismo, incluye mecanismos que describen el medio para consultar el estado de un recurso y la descripcin del servicio, que forman conjuntamente la definicin de un WS-Resource. Adems, definen los pasos necesarios para hacer el estado de un servicio web accesible a travs de su interfaz (descrita en WSDL).

Normalmente, las interfaces de los servicios web proporcionan al usuario la posibilidad de acceder y manipular el estado del mismo, como, por ejemplo, valores de datos que evolucionan por la interaccin entre varios servicios. En otras palabras, los intercambios de mensajes que se implementan en el comportamiento de los servicios tienen como objetivo permitir el acceso a estos recursos persistentes. Sin embargo, la nocin de recursos persistentes que subyace en la implementacin de los servicios no es tan evidente en la definicin de la interfaz [7]. Los mensajes que estos servicios envan y reciben implican (o animan al programador a inferir) la existencia de un tipo de recurso asociado. Por tanto, es deseable que se definan estndares que permitan el descubrimiento, creacin, introspeccin, interaccin y destruccin de dichos recursos y que la forma elegida para llevar a cabo esta misin sea lo ms interoperable posible. Estas observaciones han motivado la aparicin de la propuesta comentada anteriormente, WS-Resource, para modelar estados en el contexto de los servicios web. Un WS-Resource se define como la composicin de un servicio web y sus recursos persistentes asociados, esto es, *(i)* expresado como una asociacin de un documento XML con un tipo definido con uno o varios *portTypes* (un servicio podr jugar un

determinado rol si implementa todos los *portTypes* que comprenden ese rol) y *(ii)* direccionado y accedido de acuerdo al patrn del recurso implcito, una derivacin de las *Endpoint References* del WS-Addressing. Una *Endpoint Reference* estar compuesta por: Uniform Resource Identifier (URI), parmetros del mensaje que se envi para solicitar el envo de la *Endpoint Reference* y datos relativos a la interfaz que se usa. En este intercambio, el identificador del recurso persistente es encapsulado en una *Endpoint Reference* y usado para identificar al recurso en cualquier intercambio de mensajes entre los servicios que formen la coreografa. As, WSRF permite declarar, acceder, monitorizar y destruir WS-Resources mediante mecanismos convencionales, lo que facilita la tarea de gestin, ya que no es necesario hacer ms difcil la lgica de decisin del servicio propietario del recurso para procesar los mensajes de gestin. Estos mecanismos convencionales componen cinco especificaciones tcnicas que definen los medios por los cuales:

- Se destruye un WS-Resource, ya sea de manera sncrona con respecto a una peticin explcita de destruccin o, a travs de un mecanismo basado en tiempos (scheduled). Adems, es posible declarar unas caractersticas especficas de los recursos (WS-ResourceProperties) que podran ser utilizadas para inspeccionar y monitorizar el tiempo de vida de dicho WS-Resource (WS-ResourceLifetime).

- Se definen los tipos de WS-Resource, que estn compuestos por la interfaz de la descripcin del servicio web (WSDL) y por un documento XML de propiedades del recurso. Por otro lado, el estado del WS-Resource puede ser consultado y modificado a travs del intercambio de mensajes (WS-ResourceProperties)

- Un Endpoint Reference (WS-Addressing) puede ser renovado cuando su informacin de direccionamiento ha caducado o ha dejado de ser vlida por algn error (WS-RenewableReferences).

- Adems, se define la capacidad de implementar entornos heterogneos como colecciones de servicios web, sean o no WS-Resources (WS-ServiceGroups).

- La notificacin de errores puede ser ms estandarizada al usar tipos XML Schema para definir los fallos base y definir reglas que muestren cmo esos fallos son usados y extendidos (WS-BaseFaults).

2.6.2 WS-ResourceProperties

Como se ha comentado anteriormente, WSRF utiliza una especificacin concreta para definir las propiedades del WS-Resource. Este recurso estar compuesto por la definicin de la interfaz en WSDL y un documento XML (Resource Properties Document) que especifica las propiedades del mismo, por ejemplo, el tamao de disco, la capacidad del procesador, etc., de tal manera que si queremos acceder, modificar o actualizar este documento debemos utilizar una serie de mensajes preestablecidos en la especificacin. Las operaciones que se pueden hacer son las siguientes:

GetResourceProperty

Esta operacin como su propio nombre indica permite al servicio web que realiza la peticin recuperar el valor de una **nica** propiedad del documento de propiedades. Para aclarar ms los conceptos se define el siguiente ejemplo.

Dado el documento de propiedades:

```
...
<GenericDiskDriveProperties
xmlns: tns=``http://example.com/diskDrive'' >
  <tns:NumberOfBlocks>22</tns:NumberOfBlocks>
  <tns:BlockSize>1024</tns:BlockSize>
  <tns:Manufacturer>DrivesRUs</tns:Manufacturer>
</GenericDiskDriveProperties>
...
```

Una posible peticin puede ser:

```
...
<s12:Body>
  <wsrp:GetResourceProperty
    xmlns:tns=``http://example.com/diskDrive''>
     tns:NumberOfBlocks
  </wsrp: GetResourceProperty>
</s12:Body>...
```

GetMultipleResourceProperties

Este mtodo es equivalente al anterior, pero para acceder a ms de una propiedad del documento en el mismo mensaje, es decir, se utiliza para evitar congestionar la red. El mensaje enviado sera:

```
...
<wsrp:GetMultipleResourceProperties
 xmlns:tns=''http://example.com/diskdrive''>
 <wsrp:ResourceProperty>tns:NumberOfBlock</wsrp:ResourceProperty>
 <wsrp:ResourceProperty>tns:BlockSize</wsrp:ResourceProperty>
</wsrp:GetMultipleResourceProperties>
...
```

SetResourceProperties

Este mtodo se utiliza para realizar cambios en el documento de propiedades. Existen 3 tipos de cambios:

- Insert: Permite aadir nuevas propiedades en el documento.
- Update: Se utiliza para actualizar el valor de alguna propiedad.
- Delete: Elimina propiedades del documento.

Un posible ejemplo de peticin sera:

```
...
<s12:Body>
 <wsrpw:SetResourceProperties
        xmlns:tns=''http://example.com/diskdrive''>
   <wsrp:Update>
    <tns:NumberOfBlocks>143</tns:NumberOfBlocks>
   </wsrp:Update>

   <wsrp:Delete resourceProperty=''tns:Manufacturer''/>

   <wsrp:Insert>
    <tns:someElement>42</tns:someElement>
   </wsrp:Insert>
```

```
 </wsrp:SetResourceProperties>
</s12:Body>
...
```

El documento de propiedades quedara con el siguiente formato:

```
...
<GenericDiskDriveProperties
  xmlns:tns=''http://example.com/diskDrive''>

  <tns:NumberOfBlocks>143</tns:NumberOfBlocks>
  <tns:BlockSize>1024</tns:BlockSize>
  <tns:someElement>42</tns:someElement>

</GenericDiskDriveProperties>
...
```

QueryResourceProperties

Como su propio nombre indica, este mtodo se utiliza para realizar consultas sobre propiedades del recurso. Por ejemplo si queremos saber si el nmero de bloques es mayor que 20 y el tamao de bloque es 1024 realizaramos la siguiente consulta:

```
...
<s12:Body>
 <wsrp:QueryResourceProperties>
  <wsrp:QueryExpression
    Dialect=''http://www.w3.org/REC-xpath-19991116''>
     boolean(/*/NumberOfBlocks>20 and /*/BlockSize=1024)
  </wsrp:QueryExpression>
 </wsrp:QueryResourceProperties>
</s12:Body>
...
```

La respuesta que enva el otro servicio es:

```
...
<s12:Body>
 <wsrp:QueryResourcePropertiesResponse>
   true
 </wsrp:QueryResourcePropertiesResponse>
</s12:Body>
...
```

2.6.3 WS-Base Faults

El diseador de un servicio web normalmente utiliza interfaces definidas por otros, por lo que un mtodo que estandarizase el formato de los mensajes de notificacin de errores facilitara la labor de los desarrollares. ste es el objetivo de WS-BaseFaults. Los mensajes de fallos en WSRF tienen el siguiente formato:

```
...
<BaseFault>
  <Timestamp>xsd:dateTime</Timestamp>
  <OriginatorReference>
    wsa:EndpointReferenceType
  </OriginatorReference> ?
  <ErrorCode dialect=''anyURI''>xsd:string</ErrorCode>?
  <Description>xsd:string</Description> *
  <FaultCause>wsbf:BaseFault</FaultCause> *
</BaseFault>
...
```

donde:

- Timestamp: Hora exacta cuando el fallo ha ocurrido.
- OriginatorReference: Direccin en formato WS-Addressing del servicio que ha generado el fallo.
- ErrorCode: Cdigo de error para ser utilizado por sistemas de informacin de fallos, por ejemplo, POSIX errno.

- Description: Explicacin de la causa del fallo (en lenguaje natural).

- FaultCause: Causa tcnica del fallo.

2.6.4 WS-ServiceGroup

Esta especificacin permite crear grupos que comparten una serie de propiedades en comn, es decir, agrupar diferentes servicios web que tienen comportamientos similares.

2.6.5 WS-ResourceLifetime

El tiempo de vida de un WS-Resource se define como el perodo que transcurre entre su instanciacin y su destruccin. La misin de esta especificacin es estandarizar el proceso de destruccin de un recurso y definir mecanismos para monitorizar este ciclo de vida, pero lo que no se define es cmo crear el WS-Resource. Generalmente, en los sistemas distribuidos, los clientes slo quieren tener un recurso por un determinado intervalo de tiempo, aunque en muchos escenarios es ms apropiado para el cliente que se produzca la inmediata destruccin del recurso. Otro ejemplo claro de uso se presenta cuando el cliente quiere suscribirse a un servicio por un cierto tiempo y quiere que despus de este tiempo se destruya dicha unin. Como se coment en la introduccin, existen dos formas de destruir un recurso: inmediata, mediante un mensaje explcito o temporizada, mediante un mensaje que activa o gestiona un timer.

Destruccin inmediata

Para la destruccin inmediata slo hace falta poner $< wsrl : Destroy/ >$ dentro del cuerpo ($< Body >$) del mensaje SOAP que se enva al servicio que gestiona el recurso y dicho servicio responder con $< wsrl : DestroyResponse/ >$ dentro del cuerpo ($< Body >$) del mensaje SOAP de respuesta.

Destruccin temporizada

En este caso, el WS-Resource tiene asociado un tiempo de terminacin que define el tiempo despus del cual se espera que el recurso haya sido destruido y, razonablemente, se espera que antes del mismo el recurso est disponible. A continuacin se muestra un ejemplo de cmo determinar el tiempo de terminacin de un recurso:

```
...
  <s12:Envelope
      <ex:ResourceDisambiguator>
       uuid:ba32-8680cace43f9
      </ex:ResourceDisambiguator>
      <s12:Body>
       <wsrl:SetTerminationTime>
        <wsrl:RequestedTerminationTime>
         2001-12-31T12:00:00
        </wsrl:RequestedTerminationTime>
      </wsrl:SetTerminationTime>
      </s12:Body>
  </s12:Envelope>
...
```

Como podemos observar el servicio que solicita la destruccin puede indicar la hora de destruccin y la hora actual (para evitar desajustes por la forma de representar la zona horaria). Una vez que CurrentTime alcanza el valor TerminationTime, el recurso se destruye sin ninguna intervencin ms y se notifica al emisor del mensaje de destruccin que el recurso deja de estar disponible. Existe otro mensaje que se manda desde el receptor al emisor para comunicarle que ha recibido la peticin de cambio.

Sin embargo, puede darse la situacin de que haya ms de un servicio utilizando el recurso que vaya a destruirse por lo que el propietario del recurso puede decidir o no (se deja a libre eleccin del programador) implementar los mensajes WS-Notification para informar a los interesados que el recurso deja de estar disponible. Para llevar a cabo esta tarea debe crear este Topic:

```
...
<wstop:TopicSpace name=''ResourceLifetime''
    targetNamespace=
    http://docs.oasis-open.org/wsrf/2004/06/
    wsrf-WS-ResourceLifetime-1.2-draft-01.xsd

 <wstop:Topic name=''ResourceTermination''>
   <wstop:MessagePattern>
     <wsrp:QueryExpression
        dialect= http://www.w3.org/REC-xpath-19991116 >
         boolean(/*/TerminationNotification)
     </wsrp:QueryExpression>
 </wstop:MessagePattern>
...
```

Adems, el mensaje de notificacin asociado debe contener los siguientes campos:

```
...
<wsrl:TerminationNotification>
 <wsrl:TerminationTime>xsd:dateTime</wsrl:TerminationTime>
 <wsrl:TerminationReason>xsd:any</wsrl:TerminationReason>?
</wsrl:TerminationNotification>
...
```

donde *TerminationTime* informa de la fecha de destruccin y *TerminationReason* contiene la explicacin de la destruccin.

2.7 WS-Notification

Esta especificacin permite a un *NotificationProducer* enviar un mensaje de notificacin a un *NotificationConsumer* de dos maneras diferentes:

1. El *NotificationProducer* enva un mensaje de notificacin al *NotificationConsumer* sin seguir ningn formalismo.

2. El *NotificationProducer* utiliza el formalismo que se describe a continuacin para enviar las notificaciones.

La opcin a utilizar la elegir el suscriptor cuando mande la peticin de suscripcin. En este sentido, la segunda opcin permite al usuario recibir un amplio rango de mensajes de notificacin, ya que la informacin que se enva en estos mensajes se obtiene de un rbol de Topics (temas) y, por tanto, se permite enviar subrboles en un mismo mensaje para informar de diferentes Topics. En la Figura 2.7 vemos un ejemplo:

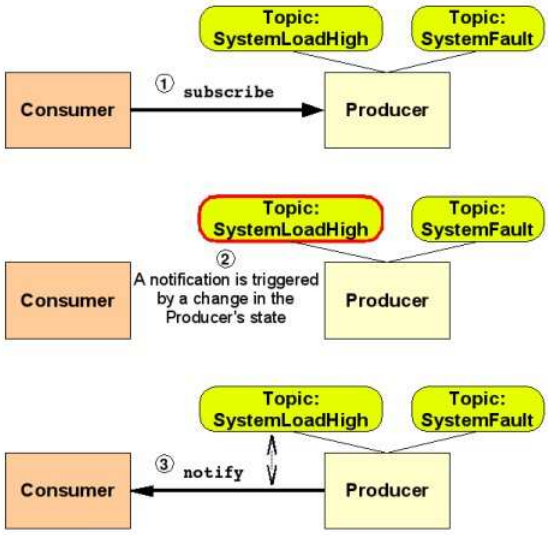


Figure 2.7. Ejemplo de uso de WS-Notification sin broker.

Este caso muestra un ejemplo de interaccin entre un consumidor y un productor de notificaciones, en el caso de que el suscriptor y el consumidor sean la misma entidad. El sistema es simple ya que tenemos un consumidor y un productor que publica 2 topics: SystemLoadHigh y SystemFault. Los pasos necesarios son:

1. En primer lugar, el consumidor se suscribe al topic SystemLoadHigh, por lo que internamente se crea un *Subscription resource* con la informacin de la suscripcin. El productor debe implementar un mtodo *Subscribe* y el consumidor un mtodo *Notify*.

2. Despus, el productor debe enviar una notificacin cuando el sistema sobrepase una determinada carga de trabajo. Por ejemplo, nuestro sistema enviar notificaciones cuando la carga de trabajo sea mayor de 50%.

3. Por ltimo, el productor enva la notificacin invocando la operacin *Notify* en el consumidor.

Un ejemplo de mensaje *Notify* es:

```
...
<wsnt:Notify>
    <wsntw:NotificationMessage>
     <wsnt:Topic Dialect= xsd:anyURI >
       {any}
     </wsnt:Topic>
     <wsnt:ProducerReference>?
      wsa:EndpointReference
     </wsnt:ProducerReference>
     <wsnt:Message>xsd:any</wsnt:Message>
    <wsnt:NotificationMessage>+
</wsnt:Notify>
...
```

Como podemos observar el mensaje *Notify* contiene uno o varios mensajes de notificacin (*NotificationMessages*). Los campos dentro de stos son:

- Topic: La informacin del topic que se enva.
- Dialect: El dialecto usado para expresar el topic anterior, es decir, el lenguaje utilizado para expresarlo.
- ProducerReference: Direccin del productor.
- Message: Una copia de la carga til (payload) del mensaje actual.

A continuacin, se muestra el mensaje que manda el suscriptor para registrar su inters en uno o ms topics:

```
...
<wsnt:Subscribe>
```

```
  <wsnt:ConsumerReference>
    wsa:endpointReference
  </wsnt: ConsumerReference>
  <wsnt:TopicExpression Dialect = xsd:anyURI >
    {any}
  </wsnt:TopicExpression>
  <wsnt:UseNotify>xsd:boolean</wsnt:UseNotify>?
  <wsnt:Precondition>wsrp:QueryExpression</Precondition>?
  <wsnt:Selector>wsrp:QueryExpression</wsnt:Selector>?
  <wsnt:SubscriptionPolicy>{any}</wsnt:SubscriptionPolicy>?
  <wsnt:InitialTerminationTime>
    xsd:dateTime
  </wsnt:InitialTerminationTime>?
</wsnt:Subscribe>

...
```

Los conceptos importantes en este mensaje son *UseNotify* que se utiliza para decidir si el mensaje de notificacin sigue el formalismo WS-Notification o se manda sin formato, *Precondition* que es la condicin que genera mensajes de notificacin, es decir, si se cumple esta condicin se generan mensajes, pero debe cumplirse tambin la condicin *selector* para enviarlos a los destinatarios que es la que se usa para decidir si se transmiten o no los mensajes generados. Adems, *SubscriptionPolicy* se podra utilizar para controlar el ratio de envo de mensajes(por ejemplo, no ms de 3 por segundo) y *InitialTerminationTime* contiene una sugerencia del tiempo de vida de la suscripcin. WSRF tambin incluye mensajes para detener la suscripcin, reanudarla o para que un servicio que acaba de unirse a una suscripcin pueda obtener un historial de notificaciones sobre un determinado topic.

2.7.1 WS-BrokeredNotification

Un *NotificationBroker* es un intermediario que, entre otras cosas, permite el envo de mensajes entre uno o varios *Publishers* y uno o varios *NotificationConsumers*. La misin del *Publisher* es observar ciertas situaciones y crear mensajes de notificacin para informar de esas situaciones, mientras que el broker es el encargado

de distribuir estos mensajes.

En este caso, se pueden dar tres relaciones entre las partes: *simple publishing*, *composable publishing* y *demand-based publishing*. En el primer caso, el *Publisher* es el encargado de observar las situaciones y notificarlas al broker que ser el encargado de transmitirlas a los interesados. En el segundo caso, el papel del *Publisher* lo realizar una entidad que implementa una serie de servicios especificados en WS-Notification (NotificationProducer). En este caso, el mensaje de notificacin puede llegar a otros consumidores que estuviesen suscritos al productor. En ambos casos, el broker puede pedir al *Publisher* que se registre para poder publicar mensajes sobre un topic determinado. El ltimo enfoque (*demand-based publishing*) requiere que el *Publisher* sea un *NotificationProducer* y, as, acepte mensajes de suscripcin. El objetivo es reducir el nmero de mensajes de notificacin haciendo que stos solo se manden cuando se soliciten expresamente.

This chapter is about the state of the art of Service-Oriented Computing, the use of formalisms for the analysis of Web Service compositions, and the specification of electronic contracts (e-contract). We specially focus on the application of e-contracts to Web Service compositions, describing some of the existing approaches for that purpose. We also review the related work in the field of formal specification and verification of e-contracts.

2.8 Service-Oriented Computing (SOC)

The Service-Oriented Computing (SOC) paradigm promotes the use of services for the development of massively distributed applications, trying to achieve the creation of fast, low-cost, flexible and scalable applications [**?**]. Services are the main building block of this paradigm, being these services self-describing and platform-independent. Thanks to the use of standards for description, publication, discovery and invocation, the services can be integrated without taking care of the low-level implementation details of each service. The aim of SOC is to make it possible the creation of dynamic business processes and agile applications by providing an easy way to assemble application components into a loosely coupled network of services.

2.8.1 Service-Oriented Architecture (SOA)

To reach the goals of SOC, a Service-Oriented Architecture (SOA) is defined. SOA is a software architecture based on the utilization of services, being these services provided to the user of the application or to other services in the network. This is possible by the use of service interfaces that can be published and discovered.

SOA is based on a model of roles where every service can play multiple roles. For example, a service can offer a certain functionality to a user and, at the same time, being the consumer of the functionality provided by some other services. Such model reduces the complexity of applications and increases their flexibility.

Although at the beginning of SOA there were several architectures aspiring to become SOA standards [**?**, **?**], the most successful one was the architecture based on Web Services.

2.8.2 Web Services (WS)

W3C defines a Web Service (WS) in the following way [**?**]:

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

We can see in this definition that there are two basic standards related to Web Services: Web Service Description Language (WSDL) for the definition of the service functionality and its properties [**?**], and Simple Object Access Protocol (SOAP) for the exchange of XML messages between services [**?**]. There is also an additional standard called Universal Description, Discovery and Integration (UDDI) used to create Web Service directories and to search for services in the network [**?**]. The use of these standard protocols is the key point to improve the integration between different parties in a Web Service architecture.

In Figure 2.8 a possible representation of the Web Service architecture stack is shown. We can see that the three standards described above are only a small part of the stack. We also need protocols to define security aspects (ensuring that exchanges of information are not modified or forged in a verifiable manner and that parties can be authenticated), to provide reliable messaging for the exchange of information between parties, to specify the collaboration between services when we compose them, to individually describe the behaviour of each service in a business process, etc. The problem is that whereas the standards for basic services (WSDL and SOAP) are widely adopted for their respective purposes, the situation is not so clear when we talk about composing services, having multiple protocols aspiring to become a standard in this layer.

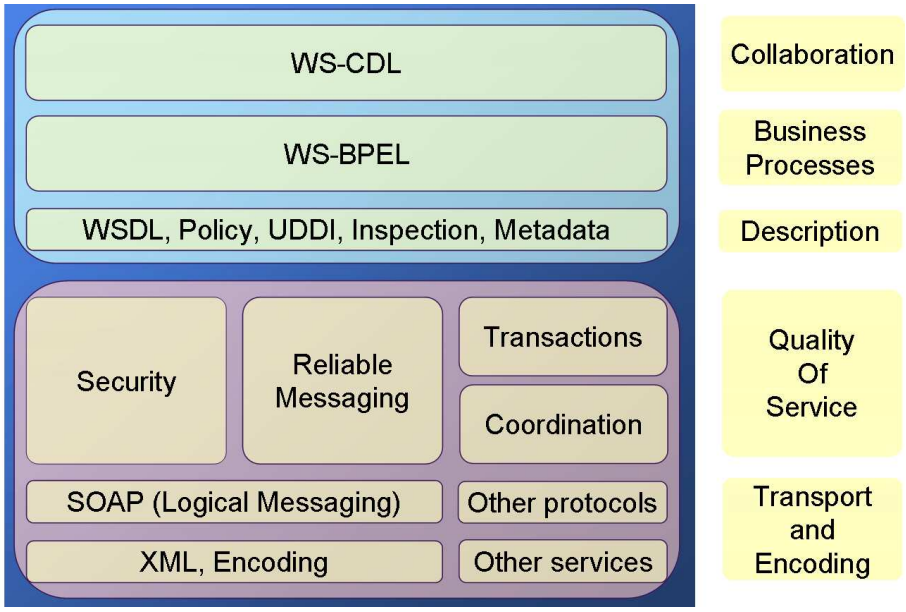


Figure 2.8. Web Service architecture stack.

2.8.3 Web Service Composition

Two different approaches can be followed when we talk about composing Web Services. They are called *orchestration* and *choreography*. The former describes the individual business process followed by each one of the participants in the composition, while the latter describes the composition from a global viewpoint, defining the interactions (exchange of messages) happening between the parties,

that is, how they collaborate in the composition. However, the ideal solution would be fusing both approaches in a single language and environment [**?**].

Anyway, the languages we can use in both cases should accomplish some common goals: (i) the capacity of modelling service interactions, including control flow and data constraints, (ii) the possibility of specifying exceptional behaviour, indicating which errors can happen in the execution of the composition and the way of handling these errors, and (iii) the ability to model Web Service compositions at a high level, without taking care of the implementation details of each one of the services.

Business Process Execution Language (BPEL)

Considering the orchestration approach, one of the most used languages for this purpose is Business Process Execution Language (BPEL), also known as BPEL4WS or WSBPEL [**?**]. A BPEL specification models the behaviour of the different services taking part in a business process, defining a BPEL source file for each service. BPEL provides an XML syntax to describe the control logic necessary to coordinate all the services participating in the process. An orchestration engine is responsible for the execution of the processes, coordinating all the activities and compensating any error that could happen. BPEL can be seen as a layer over WSDL, providing this latter language the available functionality while BPEL defines how to sequence the operations [**?**].

The components of a BPEL specification are the following:

- **Events**, which describe the flow execution in an event-driven way.
- **Variables**, which are defined by using WSDL schemes for internal or external purposes, and are used in the message flow.
- **Correlations**, which identify the processes interacting by means of messages.
- **Fault handling**, defining the behaviour when an exception is thrown.
- **Event handling**, defining the behaviour when an event happens.
- **Activities**, which are the basic unit of behaviour of a Web Service.

For example, in figure 2.9 we can see part of a BPEL specification corresponding to shipping service.

2.9 Formal Analysis of Web Service Compositions

In last years the formal analysis of Web Service compositions has been recognized as an important problem that needs to be solved [**?**, **?**, **?**, **?**]. A strict analysis is necessary to guarantee the correct composition of the services for multiple reasons: the integration of several independent applications, the possibility of having incomplete specifications of some services, the impossibility of using traditional evaluation techniques such as software testing, ...

The analysis of Web Service compositions usually consists of checking its specification correctness by analysing if a set of functional and non-functional requirements are fulfilled. The behavioural requirements are especially interesting, as they specify the properties required to reach a concrete business goal. These requirements can include general properties such as deadlock freeness or correct termination, but they can also be defined explicitly, for example defining the conformance to a concrete model. This formal analysis is usually done by means of formal methods.

Formal methods [**?**] are a collection of notations and techniques for describing and analyzing systems [**?**]. They are called *formal* because they are based on mathematical theories (logics, automata, sets, ...). In formal methods we can distinguish between the *formal specification*, which describes unambiguously a system or its properties, and the *formal analysis* or *formal verification*, which serves to verify if a system satisfies its specification.

Formal methods usually cannot guarantee the correctness of the implementation and, if the specification is not correct, the verification results will also be wrong. Despite these and some other limitations, formal methods are still needed because they increase the confidence on system reliability, minimize the number of errors in the implementation and can find out errors impossible to find with other techniques such as testing.

When we are going to apply a formal method it is very important to choose the right level of abstraction in the specification of the system depending on what we

```
<process name="shippingService"
...
   <sequence>
      ...
      <if>
         <condition>
            bpel:getVariableProperty('shipRequest',
             'props:shipComplete')
         </condition>
         <sequence>
            ...
            <invoke partnerLink="customer"
               operation="shippingNotice"
               inputVariable="shipNotice">
               <correlations>
                  <correlation set="shipOrder"
                   pattern="request" />
               </correlations>
            </invoke>
         </sequence>
         <else>
            <sequence>
              ...
              <while>
                <condition>
                  $itemsShipped
                  &lt;
                  bpel:getVariableProperty(
                  'shipRequest','props:itemsTotal')
                </condition>
                <sequence>
                  ...
                  <invoke partnerLink="customer"
                    operation="shippingNotice"
                    inputVariable="shipNotice">
                    <correlations>
                       <correlation set="shipOrder"
                          pattern="request" />
                    </correlations>
                 </invoke>
                 ...
                </sequence>
              </while>
            </sequence>
         </else>
      </if>
   </sequence>
</process>
```

Figure 2.9. Part of a shipping service specification in BPEL

want to analyze or verify. An underspecification can lead to wrong verification results because the specification is incomplete, whereas an overspecification can lead to the *state explosion problem*, making the intended analysis of the system infeasible.

Formal methods can be applied in different stages of the computer system development process, providing different information about the system in each one of these stages. The application of these methods is usually complex, being impossible without some kind of tool support. For this purpose, the language syntax of the specification must be explicit and the language semantics of the specification must be restricted. In this way, formal specifications are amenable to automate analysis and verification.

The choice of using one formal method or another depends on many factors: the problem we want to solve, the type of system, the properties we want to check, and so on. In the case of complex systems, such as distributed concurrent systems, it is needed a combination of several techniques for the analysis.

2.9.1 Specification Formalisms

2.10 Summary

This chapter has described the state of the art of Service-Oriented Computing (SOC), the formalization of Web Service compositions, and the specification of electronic contracts (e-contracts).

The development of systems based on Web Services allows the creation of fast, low-cost, flexible and scalable applications, where the integration is possible thanks to the definition of multiple standard protocols (WSDL, SOAP, UDDI). However, the correct composition of Web Services is still an open problem, where different approaches can be followed, such as orchestration (BPEL) and choreography (WS-CDL, WSCI, OWL-S). The formal analysis of Web Service compositions by means of formal methods is therefore necessary to guarantee the correct composition.

Several formal techniques can be used for the analysis of Web Service compositions, being model checking one of the most popular. This is an automated

technique consisting of the construction of a finite-state model of the system to check if some properties are satisfied. Different specification formalisms can be used in model checking (process algebras, Petri nets, automata) and there are several tools supporting each one of these formalisms (CWB-NC, CPN Tools, UPPAAL).

Chapter 3

BPELRF

Chapter 4

Timed-arc workflow nets

Chapter 5

Conclusions, Contributions and Future Works

This chapter presents the conclusions of this Thesis, reviews the contributions of this work, and suggests some possible future lines of research. It also includes a list of the publications obtained as a result of this work.

5.1 Conclusions

The work that has been carried out in this Thesis can be divided in two different topics: on the one hand the design and verification of Web Service compositions with time constraints by means of a top-down methodology called Correct-WS and the implementation of the WST tool supporting several phases of this methodology, and on the other hand the specification, validation and verification of electronic contracts with time constraints by means of the definition of a visual model with a formal background called *C-O Diagrams*.

Concerning the first topic, the work carried out in [**?**] has been continued and extended, adding things such as the treatment of exceptions. The WST tool has been improved in several aspects, completing the implementation of the translation from WS-CDL to timed automata and including the treatment of exceptions in all the different phases supported by the tool.

The advantage of having the Correct-WS methodology and the WST tool is that the correctness of the Web Service compositions, with respect to some properties of interest, can be formally verified in the early phases of the development process. In this way, any problem detected can be solved before implementation, saving the cost of correcting these errors after implementation. Another important advantage is that we can obtain automatically XML code from a graphical model of the composition, so non-XML experts do not have to care about specifying the composition in languages such as WS-CDL.

Regarding the second topic, a novel approach for the specification and verification of deontic e-contracts has been defined. It consists of a visual model called *C-O Diagrams* representing the obligations, permissions and prohibitions of the contract, including also any condition, reparation and time constraint. The syntax of the model has been formally defined, and the semantics is provided by means of a transformation into a network of timed automata, allowing the verification of some properties of interest on it by using the model-checker of the UPPAAL tool.

Apart from the timed automata semantics, some other applications of *C-O Diagrams* are envisaged. They can be used as a user-friendly way of specifying a contract in $\mathcal{CL}$ language, as a translation function from *C-O Diagrams* to $\mathcal{CL}$ has been defined. The diagrams can also be used to check if the behaviours of systems, modelled as timed automata, comply with the contracts represented by *C-O Diagrams*. This can be done by applying the set of satisfaction rules that has been defined.

The main advantage of *C-O Diagrams* is that they provide a graphical representation of e-contracts but amenable to formal verification. In this way, non-experts in formal methods can easily specify the contracts in a graphical manner and, as a formal semantics of the model has been defined, the specified contract can be formally verified with respect to some properties of interest. The formal verification of the e-contracts is important to find out any contradiction or unexpected behaviours allowed by the contract, so the contracts can be corrected in a suitable manner.

5.2 Contributions

In this section the contributions produced as a result of the development of this Thesis are summarized in the following list:

- The continuation and extension of the development of the Correct-WS methodology for the design and verification of Web Service compositions.
- The improvement of the implementation of the WST tool that supports several phases of the Correct-WS methodology.
- The definition of a visual model called *C-O Diagrams* for the specification of deontic electronic contracts including time constraints.
- The definition of formal syntax and semantics for *C-O Diagrams*, allowing the formal verification of the contract modelled.
- The definition of a set of satisfaction rules to check the compliance of the contracts represented by *C-O Diagrams*.
- The definition of a translation function from *C-O Diagrams* to the contract language $\mathcal{CL}$.

5.3 Future Works

The work that has been presented in this Thesis has led to several ideas for future work. Some of them are the following:

- Concerning the WST tool, it is currently under development a new tab supporting the analysis phase of the Correct-WS methodology. This tab will allow the users to develop the KAOS model for the specification of the properties of interest, generating automatically the queries corresponding to this model that have to be check in the verifier of the UPPAAL tool. The inclusion of the translation of WS-CDL documents into Coloured Petri nets in the WST tool is also under development.
- A study of the complexity and scalability of *C-O Diagrams* to see how complex can be the electronic contracts that we can tackle with these diagrams.

- The application of *C-O Diagrams* to different contexts, such as requirements engineering (to formalize requirements), product software families (as an extension of feature diagrams), ...

- The development of a tool to model *C-O Diagrams* and automatizing the transformation of the diagrams into timed automata supported by the UPPAAL tool, in the same way that the WST tool automatizes several transformations of the Correct-WS methodology.

- The development of a query language for contracts making possible the specification at a high level of abstraction of the properties of the contracts we want to check, instead of specifying the properties directly in the query language used by the UPPAAL tool. Therefore, a correspondence between the contract query language and the UPPAAL query language would be necessary.

- We also envisage the possibility of specifying a different diagram for each one of the parties involved instead of having global *C-O Diagrams* with multiple agents. This compositional approach can be useful if a composition operator is defined, specifying when two of these new *C-O Diagrams* can be composed and the result of the composition.

5.4 Publications and Collaborations

The work done during this Thesis has given rise to several publications. They include international journal papers, international conference papers, and a national conference paper. Two technical reports have also been published and works submitted for publication are mentioned too.

5.4.1 Journal Papers

- Cambronero, M.E., Díaz, G., Martínez, E., Valero, V. and Tobarra, L. (2010). WST: A Tool Supporting Timed Composite Web Services Model Transformation, *SIMULATION: Transactions of the Society for Modeling and Simulation International*.

- Cambronero, M.E., Díaz, G., Valero, V. and Martínez, E. (2011). Validation and Verification of Web Services Choreographies by Using Timed Automata, *Journal of Logic and Algebraic Programming* **80** pp. 25-49.

- Cambronero, M.E., Valero, V. and Martínez, E. Design and Generation of Web Services Choreographies with Time Constraints, *Journal of Universal Computer Science*, to appear in 2011.

5.4.2 International Conference Papers

- Cambronero, M.E., Díaz, G., Valero, V. and Martínez, E. (2008). A Tool for the Design and Verification of Composite Web Services, in *Second International Workshop on Formal Languages and Analysis of Contract-Oriented Software*, pp. 9-16.

- Martínez, E., Díaz, G., Cambronero, M.E. and Valero, V. (2009). Design and Verification of Web Services Compositions, in *The Fourth International Conference on Internet and Web Applications and Services*, pp. 395-400.

- Valero, V., Macia, H. and Martínez, E.(2009). Transforming WS-CDL specifications into Coloured Petri Nets, in *International Workshop on Petri Nets and Software Engineering (PNSE09)*.

- Martínez, E., Díaz, G., Martínez, C.R., Cambronero, M.E. and Valero, V. (2009). Time Ordering Architecture in SCA, in *TAMoCo 2009 Techniques and Applications for Mobile Commerce*, pp. 117-125.

- Cambronero, M.E., Díaz, G., Martínez, E. and Valero, V. (2009). A comparative study between WSCI, WS-CDL, and OWL-S, in *The 2009 IEEE International Conference on e-Business Engineering (ICEBE 2009)*, pp. 377-382.

- Andrés, C., Díaz, G., Martínez, E. and Zhang, Y (2009). Formal Study of Prioritized Service Compositions, in *International conference on signal-image technology & internet based systems, 5th (SITIS 2009)*, pp. 355-362.

- Mateo, J.A., Díaz, G., Martínez, E. and Cambronero, M.E. (2010). Modeling Conference Contribution Management Using Web Services, in *The Fifth International Conference on Internet and Web Applications and Services*, pp. 463-468.

- Martínez, E., Díaz, G., Cambronero, M.E. and Schneider, G. (2010). A Model for Visual Specification of e-Contracts, in *The 7th IEEE International Conference on Services Computing (SCC 2010)*, pp. 1-8.

- Martínez, E. and Schneider, G. (2010). Automated Analysis of Conflicts in Software Product Lines, in *1st International Workshop on Formal Methods in Software Product Lines Engineering*, pp. 75-82.

- Martínez, E., Díaz G. and Cambronero, M.E. (2010). Visual Specification of Formal e-Contracts, in *Fourth Workshop on Formal Languages and Analysis of Contract-Oriented Software*, pp. 55-62.

- Mateo, J.A., Valero, V., Martínez, E. and Díaz, G. (2011). Analysis and Verification of Web Services Resource Framework (WSRF) specifications Using Timed Automata Modeling Conference Contribution Management Using Web Services, in *The Sixth International Conference on Internet and Web Applications and Services*, pp. 222-227.

- Martínez, E., Cambronero, M.E., Díaz, G. and Schneider, G. (2011). Timed Automata Semantics for Visual e-Contracts, in *Fifth Workshop on Formal Languages and Analysis of Contract-Oriented Software*, pp. 7-22.

- Martínez, E., Díaz, G. and Cambronero, M.E. Contractually Compliant Service Compositions, *The Ninth International Conference on Service Oriented Computing*, to appear in 2011.

5.4.3 National Conference Papers

- Valero, V., Macia, H. and Martínez, E.(2009). A Petri Net Semantics for WS-CDL, in *XVII Jornadas de Concurrencia y Sistemas Distribuidos*, pp. 35-50.

5.4.4 Technical Reports

- Cambronero, M.E., Valero, V., Díaz, G. and Martínez, E. (2009). Web Services Choreographies Verification, *Technical Report DIAB-09-04-3*, Computing Systems Department, University of Castilla-La Mancha.

- Cambronero, M.E., Díaz, G., Martínez, E. and Valero, V. (2009). A comparative study between WSCI, WS-CDL, and OWL-S, *Technical Report DIAB-09-04-3*, Computing Systems Department, University of Castilla-La Mancha.

5.4.5 Collaborations

International Collaborations and stays at international universities and research groups:

- Department of Applied IT at Chalmers — University of Gothenburg - Sweden for 3 months in 2009.

5.5 Funds and Grants

The present Thesis has been carried out thanks to the funds received from a number of projects and grants:

- Research project: Modelling and Analysis of Composed Web Services Using Formal Techniques (TIN2009-14312-C02-02)
  Research project funded by Spanish Ministry of Education & Science.
  Participating Organizations: University de Castilla-La Mancha (Spain).
- The author of this thesis has been supported by a predoctoral grant from the *Junta de Comunidades de Castilla-La Mancha*, as a part of the program "*Plan Regional de Investigación Científica, Desarrollo Tecnológico e Innovación 2005-2010 (PRINCET)*".

Bibliography

[1] S. Askary C. Barreto B. Bloch F. Curbera M. Ford Y. Goland A. Guzar N. Kartha C. K. Liu R. Khalaf D. Knig M. Marin V. Mehta S. Thatte D. van der Rijn P. Yendluri A. Yiu A. Alves, A. Arkin. Business Process Execution Language for Web Services (version 2.0). 2007.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] T. Banks. Web Services Resource Framework (WSRF) - Primer. *OASIS*, 2006.

[4] E. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 1996.

[5] O. Grumberg E. Clarke and D. Peled. *Model Checking*. MITPress, Cambridge, MA, 1999.

[6] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[7] S. Graham S. Tuecke K. Czajkowski D. Ferguson F. Leymann M. Nally T. Storey W. Vambenepe I. Foster, J. Frey and S. Weerawarana. Modeling stateful resources with web services. *http://www-106.ibm.com/developerworks/library/wsresource/ ws-modelingresources.pdf, Tech. Rep., 2004.*

[8] S. Schuman J. R. Abrial and B. Meyer. A Specification Language, in On the Construction of Programs. *Cambridge University Press, eds. A. M. Macnaghten and R. M. McKeag*, 1980.

[9] P. Pettersson K.G. Larsen and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1997.

[10] A. Pnueli. The temporal logic of programs. *Proc. 18th IEEE Symp. on Foundations of Computer Science*, 1977.

[11] I. Foster J. Frey S. Graham C. Kesselman T. Maguire T. Sandholm P. Vanderbilt S. Tuecke, K. Czajkowsk and D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Recommendation. , *Global Grid Forum*, 2003.

[12] J. Sifakis T. Henzinger, X. Nicollin and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

Bibliography

[1] S. Askary C. Barreto B. Bloch F. Curbera M. Ford Y. Goland A. Guzar N. Kartha C. K. Liu R. Khalaf D. Knig M. Marin V. Mehta S. Thatte D. van der Rijn P. Yendluri A. Yiu A. Alves, A. Arkin. Business Process Execution Language for Web Services (version 2.0). 2007.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] T. Banks. Web Services Resource Framework (WSRF) - Primer. *OASIS*, 2006.

[4] E. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 1996.

[5] O. Grumberg E. Clarke and D. Peled. *Model Checking*. MITPress, Cambridge, MA, 1999.

[6] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[7] S. Graham S. Tuecke K. Czajkowski D. Ferguson F. Leymann M. Nally T. Storey W. Vambenepe I. Foster, J. Frey and S. Weerawarana. Modeling stateful resources with web services. *http://www-106.ibm.com/developerworks/library/wsresource/ ws-modelingresources.pdf, Tech. Rep., 2004.*

[8] S. Schuman J. R. Abrial and B. Meyer. A Specification Language, in On the Construction of Programs. *Cambridge University Press, eds. A. M. Macnaghten and R. M. McKeag*, 1980.

[9] P. Pettersson K.G. Larsen and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1997.

[10] A. Pnueli. The temporal logic of programs. *Proc. 18th IEEE Symp. on Foundations of Computer Science*, 1977.

[11] I. Foster J. Frey S. Graham C. Kesselman T. Maguire T. Sandholm P. Vanderbilt S. Tuecke, K. Czajkowsk and D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Recommendation. , *Global Grid Forum*, 2003.

[12] J. Sifakis T. Henzinger, X. Nicollin and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.