UNIVERSIDAD DE SALAMANCA

DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA

FACULTAD DE CIENCIAS

VNiVERSiDAD D SALAMANCA

Intelligent business Processes Composition based on mAs, Semantic and Cloud Integration

(IPCASCI)

Author:

José Alberto García Coria

Director:

Juan Manuel Corchado Rodríguez

September 2013

The research memory “Intelligent business Processes Composition based on mAs, Semantic and Cloud Integration: IPCASCI” presented by D. Jose Alberto García Coria to obtain a Ph.D. Degree by the Universidad de Salamanca has been carried out under the supervision of Dr. D. Juan Manuel Corchado Rodríguez, Full Professor – (Catedrático de Universidad) of the Departamento de Informática y Automática of the Universidad de Salamanca,

Salamanca, September 2013

The Director

Fdo: Dr. D. Juan M. Corchado Rodríguez
Catedrático de Universidad
Departamento de Informática y Automática
Universidad de Salamanca

The Graduate:

Fdo. D. José Alberto García Coria

José Alberto García Coria: Intelligent business Processes
Composition based on mAs, Semantic and Cloud
Integration: IPCASCI, Ph.D. in Computer Science, © September 2013
Supervisor:
Dr. D. Juan Manuel Corchado Rodríguez

Location:
Salamanca
Spain

ACKNOWLEDGEMENT

To Juan Manuel Corchado, for his guidance, advices, expertise, vision and inmense knowledge.

To Sonia, Laura and David, for their constant support and motivation.

In memory of Gabriela …

ABSTRACT

Component reuse is one of the techniques that most clearly contributes to the evolution of the software industry by providing efficient mechanisms to create quality software. Reuse increases both software reliability, due to the fact that it uses previously tested software components, and development productivity, and leads to a clear reduction in cost.

Web services have become are an standard for application development on cloud computing environments and are essential in business process development. These services facilitate a software construction that is relatively fast and efficient, two aspects which can be improved by defining suitable models of reuse. This research work is intended to define a model which contains the construction requirements of new services from service composition. To this end, the composition is based on tested Web services and artificial intelligent tools at our disposal.

It is believed that a multi-agent architecture based on virtual organizations is a suitable tool to facilitate the construction of cloud computing environments for business processes from other existing environments, and with help from ontological models as well as tools providing the standard BPEL (Business Process Execution Language). In the context of this proposal, we must generate a new business process from the available services in the platform, starting with the requirement specifications that the process should meet. These specifications will be composed of a semi-free description of requirements to describe the new service.

The virtual organizations based on a multi-agent system will manage the tasks requiring intelligent behaviour. This system will analyse the input (textual description of the proposal) in order to deconstruct it into computable functionalities, which will be subsequently treated. Web services (or business processes) stored to be reused have been created from the perspective of SOA architectures and associated with an ontological component, which allows the multi-agent system (based on virtual organizations) to identify the services to complete the reuse process.

The proposed model develops a service composition by applying a standard BPEL once the services that will compose the solution business process have been identified. This standard allows us to compose Web services in an easy way and provides the advantage of a direct mapping from Business Process Management Notation diagrams.

RESUMEN

La reutilización de componentes es sin duda uno de los actores que de forma más clara pueden contribuir en el desarrollo de la industria de la informática, facilitando mecanismos eficientes para crear software de calidad. La reutilización aumenta la fiabilidad de las aplicaciones, ya que se utilizan módulos que han sido probados, mejora la productividad en el desarrollo e implica una clara reducción de costes.

Los servicios Web son un estándar en la industria de la informática en la actualidad y forman parte de muchas de las aplicaciones que utilizamos todos nosotros hoy en día. También es un estándar en el desarrollo de aplicaciones para entornos Cloud o en nube. Los servicios Web son también elementos esenciales en el desarrollo de procesos de negocio. Estos servicios facilitan la construcción de software relativamente rápido y de forma eficiente. Estos dos aspectos pueden mejorarse definiendo modelos adecuados de reutilización. Con este trabajo se pretende definir un modelo que satisfaga los requisitos de creación de nuevos servicios a partir de la composición de servicios ya probados de forma eficiente y utilizando herramientas que la inteligencia artificial pone a nuestra disposición.

Se presenta una arquitectura multi-agente basada en organizaciones virtuales que facilita la construcción en entornos Cloud de procesos de negocio a partir de otros ya existentes, con la ayuda de un modelo ontológico y las herramientas que proporciona el estándar BPEL (Business Process Execution Language).

En el contexto de este propuesta, para generar un proceso de negocio nuevo a partir de los servicios de los que se dispone se parte de la especificación de los requisitos que debe cumplir el proceso. Esta especificación está formada por una descripción semi-libre de los requisitos que describen el servicio.

Una organización virtual, basada en un sistema multi-agente gestiona las tareas que requieren de un comportamiento inteligente. Este sistema analizará la entrada (descripción textual de la propuesta) para descomponerla en funcionalidad computable para su posterior tratamiento. Además, el sistema multi-agente sirve como soporte para el descubrimiento y agrupación de los servicios Web presentes en la definición del nuevo proceso de negocio.

Los servicios Web o procesos de negocio almacenados que podrán ser reutilizados se habrán concebido desde el punto de vista de las arquitecturas SOA, y tendrán asociado un componente ontológico que permitirá que el sistema multi-agente basado en organizaciones virtuales sea capaz de identificarlos para su reutilización.

El modelo propuesto implica que una vez se hayan identificado los servicios que compondrán el proceso de negocio solución se lleve a cabo su composición usando el estándar BPEL. Este estándar permite componer servicios Web de forma sencilla con la ventaja de tener un mapeo directo desde diagramas "*Business Process Management Notation*".

INDEX

1 INTRODUCTION

The software development industry is constantly evolving and looking for new technologies, languages and tools that are more powerful, efficient and safe. In this process it is essential to build models, architectures and agile technologies capable of introducing new tools as simply and economically as possible. Component reuse is one of the techniques that most clearly contributes to such a development by providing efficient mechanisms to create quality software (Schmid, 2011a) (Poulin 2006) (Poulin, 1997) (Lemley and O'Brien, 1997) (Shang *et al*., 2012) (Rehesaar, 2011). Reuse increases both software reliability (due to the fact that it uses previously tested software components) and development productivity, leading to a clear reduction in cost (Schmid, 2011b) (Xu *et al*., 2011) (Rehesaar, 2011) (Garcia et al., 2007). Due to the recent increases in software product volume and complexity, reuse is becoming a highly regarded field, serving as a fundamental stage for design and quality models such as CMMI (*Capability Maturity Model Integration*) (Osiecki et al., 2011).

In this context, Web service reuse is emerging as an interesting alternative with respect to classical code reuse. A Web service is a software component representing a service deployed on a Web platform and supporting automatic interactions between machines in the network (Walsh, 2002). SOA (Service-oriented Architecture) has emerged from this framework as a new architecture leading to conventional software development (Erl, 2009) (Ralyté, *et al*., 2011) (Alizadeh, *et al*., 2012). SOA introduces a new method of creating distributed applications where their basic services can be published, discovered and linked in order to achieve more complex services. Applications interact through existing services from entry points in interfaces rather than at a level of implementation (Papazouglu, 2008).

Software development can be analyzed from different perspectives offering a wide variety of alternatives to a methodological, functional and instrumental level, among others. One of the paradigms revolutionizing this industry in recent years has been *Cloud Computing* (Antonopoulos *et al*., 2012) (Stage and Setzer, 2009) (Duy *et al*., 2011) (Cloud Computing, 2011) (EuroCloud *et al*., 2011). The cloud represents a novel concept of service and information distribution, providing many possibilities of scaling solutions, facilitating the use of relatively simple and economic terminals (interesting models of pay per use, and multi-platform accessibility, etc). However, this model has certain drawbacks related to maintenance complexity and application development (Jaeger *et al*., 2009) (Dölitzscher *et al*., 2010). The number of engineers experienced in this field is relatively low and the development time is significantly high (Massonet *et al*., 2011) (Schaaf *et al*., 2010) (Sulistio *et al*., 2009).

Focusing on this specific area, this research proposes a methodology that facilitates the business process construction on cloud computing environments in an agile and efficient way from developed components. The research aims to present a proposal that will facilitate the creation of such processes in the form of Web services from other semi-automatic functional services; and will do so within the framework of a process guided by relatively inexpert programmers. The process is guided by a multi-agent architecture based on virtual organizations (Rodríguez et al., 2011) (Dignum et al., 2005) (Escriva et al., 2006) able to implement the intelligent behavior needed for

process management by using an *ontology* (Maedche and Staab, 2001) (Chandrasekaran, 1999) (Noy and McGuiness, 2001) (Guarino, 1998) (López, 1999). The goal of this research work is to provide a model, which allows us the construction of a business process from specifications in text format (with some constraints) of the process concerning us. The multi-agent system based on virtual organizations will facilitate the process by using standard BPEL (*Business Process Execution Language*). This standard makes it possible an easy composition of Web services, with the additional advantage of a direct projection to diagrams BPMN (*Business Process Management Notation*) (Pedrinaci *et al*., 2008).

1.1 Hypothesis and goals

Web services have become a standard for the informatics industry, being part of many of the applications currently used (Erl, 2009). Furthermore, they are also a standard for application development on cloud computing environments (Massonet *et al*., 2011) (Schaaf *et al*., 2010) (Sulistio *et al*., 2009). Web services are essential in business process development. These services facilitate a software construction that is relatively fast and efficient, two aspects which can be improved by defining suitable models of reuse. This research work is intended to define a model which contains the construction requirements of new Web services from service composition. To this end, the composition is based on tested Web services and artificial intelligent tools at our disposal. Taking this into account, the hypothesis of this research can be defined as follows:

> *"A multi-agent architecture based on virtual organizations is a suitable tool to facilitate the construction of cloud computing environments for business processes from other existing environments, and with help from ontological models as well as tools providing the standard for Business Process Execution"*

In the context of this proposal, we must generate a new business process from the available services in the platform, starting from the requirement specifications that the process must meet. These specifications will be composed of a semi-free description of requirements to describe the new service. The virtual organizations based on a multi-agent system will manage the tasks requiring intelligent behavior. This system will analyze the input (textual description of the proposal) in order to convert it into computable functionalities, which will be subsequently treated. Web services (or business processes) stored to be reused have been created from the perspective of SOA architectures and associated with an ontological component, which allows the multi-agent system (based on virtual organizations) to indentify the services to complete the reuse process. The proposed model develops a service composition by applying a standard BPEL once the services that will compose the solution business process have been identified. This standard allows us to compose Web services in an easy way and provides the advantage of a direct mapping from BPMN diagrams. The goals we intend to achieve in this research work are:

- Developing a model that allows us to build business processes on cloud computing environments in an efficient, agile and economically profitable way, while also facilitating the reuse of components and enabling the development of custom computer systems by inexperienced personnel.
- The model should be capable of semantically analyzing the textual definition of the business process to be implemented. The textual definition will be semi-free in such a way that the vocabulary defined on the ontology for each case can be used. This definition will be analyzed before carrying out a Web service search.
- A multi-agent system based on virtual organizations will also be included in our model to carry out a Web service search. Therefore, it deconstructs the description of the business process into modules, determines the connections between such modules, and performs a search for the Web services by implementing the required functionality. To this end, every Web service stored in the platform along with its specification WSDL (*Web Service Description Language*) couples an ontological content in order to recognize the functionality that it offers.
- The model will build a BPD (B*usiness Process Diagram*) adapted to the requirements of the problem (Owen and Raj, 2003) by: Using the analyzed input and the services that have been found, the design process of a BPD diagram (BPMN standard) is automated, enabling the projection of its content to a BPEL file. Each activity in the diagram is associated with one of the Web services. This diagram can be modified or adapted by the programmer to fit the need of the required software. By using a diagram that reflects the needs of the project, the proposed model creates a BPEL archive, which carries out a faithful and complete mapping of the diagram, taking into account the services performed by each activity.

1.2 Thesis structure

This Thesis has been divided into the following 6 chapters: In the introductory chapter (that is, this chapter) presents the motivation, the starting hypothesis and the goals of this research work. It also provides a brief summary of the items included in the proposed model.

The second chapter reviews the state of the art of cloud computing systems. This chapter also describes the fundamental components and concepts that make up a cloud system as well as the advantages supported by a global technologic environment. Moreover, it provides a description of the main cloud suppliers (*Windows Azure, Amazon y Google App Engine*) and a technological study on which the cloud computing systems are based. Finally, the reuse advantages of this type of environment are analyzed.

The third chapter presents BPM (*Business Process Management*) diagram and all concepts related to such models. Web services and their semantics are also presented. BPM is composed of a set of software systems, tools and methodologies focused on the way that the enterprises identify, model, develop, distribute and manage their

business processes. Within BPM, we can underscore BPNM *(Business Process Management Notation*), a graphic standard supported by BPM which allows us to define Web services from composition of existing Web services. Web services essentially become a technology that performs a software deployment via Web while offering on-line services. The two most relevant types of Web services (SOAP y REST) are presented along with their respective characteristics. The last section of this chapter presents the SOA architecture, an architecture where the basic services can be published, discovered and linked in order to build more complex services. At the end of the chapter, the state of the art of semantic Web services is presented focusing on OWL-S, being the most developed proposal to date.

The fourth chapter presents the state of the art on multi-agent systems and ontologies. Agent technology has grown extensively by passing from an academic study to real and successful implementations in various areas. Different agent concepts and classifications as well as several architectures oriented to building agents are also explained. Additionally, multi-agent systems are explained as systems that incorporate different agents to obtain a common goal. The virtual agent organizations and their advantages are analyzed. At the end of this chapter has been presented a section on ontologies, their meaning, guides for their development and languages for their construction.

The fifth chapter introduces our proposal, the service composition model called IPCASCI (*Intelligent business Processes Composition based on mAs, Semantic and Cloud Integration*) which facilities the business process composition on a cloud computing environment in an intelligent way, and provides support from a multi-agent system based on virtual organizations (described by an ontology). The model is presented and analyzed according to the processes it includes. The model is introduced as an alternative with respect to OWL-S models, which are oriented to the definition of semantic Web services, including search processes, composition and automatic invocation. To conclude, the sixth and final chapter of this research work presents the results and evaluation of our proposal on a real case study, conclusions and future work. The bibliography used in this research has been provided at the end of this document.

2 CLOUD COMPUTING

2.1 Introduction

Cloud computing facilitates the development of distributed computer systems, data management and computing resources through a scalable network node, data process centers and Web services (Massonet *et al.*, 2011) (Schaaf *et al*., 2010) (Sulistio *et al*., 2009). The definition of computing in the ninth proposal by the United States National Institute of Standards and Technology (NIST) specifies five "essential" characteristics for cloud computing: free-for-all; desktop access; laptop and mobile phone; resources shared by various users and applications; flexible resources that can be re-distributed fast according to need; and measured services. These characteristics are combined to turn cloud computing into a kind of infrastructure or public service. The main idea behind cloud computing is not new. In the 60´s, John McCarthy had already foreseen that computer facilities would be offered to the general public for their shared use (Padala *et al*., 2009). Since then, cloud computing has emerged as a paradigm to offer support and service delivery through the Internet. From a technological point of view, it constitutes a new way of creating Internet oriented applications, which also focus on a large number of users. The technological approach of cloud computing is to create services (whether applications, hosting or storage services) which can be offered to a large number of users using a minimal amount of hardware resources, and to easily deliver increased service capacity by increasing available resources. This characteristic is called service scalability (INSA, 2012).

With respect to the marketing model, cloud computing introduces a change in the way of exploiting and marketing a company´s products (Chang *et al*., 2010). The Hardware or software goods acquisition model becomes a subscription model (Nesse *et al*., 2011) (Jaeger *et al.*, 2009) or a service consumption model. This means that in lieu of acquiring the relevant computer resources, providers are hired instead. It is an attractive approach for companies as it eliminates requirements for future planning for resources and allows them to start out small and gradually increase resources as needed (Dölitzscher *et al*., 2010) (Zhang *et al*., 2003). The potential of the cloud computing technology reside in the following characteristics: (i) Resources in a cloud environment can be allocated and unallocated according to the needs at any given time, (ii) Resources can be shared or/and allocated to each customer depending on their needs, (iii) Services stored in the cloud are generally based on Web technology, which makes them accessible to a broad range of devices with Internet connection, (iv) Once cloud service infrastructure is outsourced, the service provider transfers its business risks (for example, hardware failures) to the infrastructure providers that are usually more experienced and better equipped to handle such risks.

The novelty if this technology and its options are enormous; its development has expanded exponentially in recent years. One of the biggest problems of its development is related to the definition of the pay-as-you-go model; another problem

has to do with the flexible and efficient building of applications for this infrastructure (Cloud Computing, 2011) (EuroCloud *et al*., 2011). One option when developing efficient software in this context is the use of methodologies which facilitate the design and programming of guided business processes. If this methodology or management model could facilitate the reuse of Web services/ business processes in a simple way, it would overcome one of the greatest challenges of cloud computing. In this chapter we present such a technology, its main concepts, service providers, its tools, limitations, its taxonomy and its structure at the layer level. Finally, we analyze its potential and establish the benefits of building business processes through reuse.

2.2 Main aspects

The term cloud computing has been used in different contexts to represent different ideas (Massonet *et al*., 2011) (Schaaf *et al*., 2010) (Sulistio *et al*., 2009). Undoubtedly, the absence of a standard definition for cloud computing has not only generated uncertainty, but also skepticism and confusion (Zhang *et al*., 2010). There are works which already intend to deal with this lack of a standard definition, such as the one provided by (Vaquero *et al*., 2009) which compares more than 20 definitions in an attempt to offer a standard definition of this concept. There are also proposals by NIST (*National Institute of Standards and Technology*), which, in September 2011, published a document of definitions for the fundamental concepts of cloud computing (INSA *et al*., 2012).

Next, we present various alternatives to define *cloud computing*:

> *"Cloud computing is a model for enabling convenient and a la carte access through the Internet, to a group of configurable computing resources (networks servers, storage applications and services), which can easily be used and published with minimum management effort, as well as minimum interaction with the service provider (Mell and Grance, 2011)".*

Other definitions of cloud computing which are generally accepted in the academic and professional field are those given in (Buyya *et al*., 2008) (Foster, 2008) (Armbrust *et al*., 2010) (Wang and Lazewsky, 2008).

> *"A Cloud is a type of a parallel and distributed system consisting of a collection of interconnected and virtualized computers. They are dynamically provisioned and presented as one or more unified computing resources, based on agreements to a service level and established through the negotiation between the service provider and the consumers" (Buyya et al., 2008).*
>
> *"Cloud computing is a set of services in scalable platforms, with guaranteed quality service, personalized and inexpensive, network enabled which can be easily accessed (Wang and Lazewski, 2008)."*

Cloud computing differs from other computing paradigms, such as *grid computing* (Foster and Kesselman, 1998), *global computing* (Fedak *et al*., 2001) and *Internet computing* (Milenkovic *et al*., 2003) in the following aspects (Wang and Lazewski, 2008). Grid computing is an innovative technology which allows the coordinated use of all kinds of resources that are not subject to centralized control. On the other hand, *global computing* refers to the use of a great range of heterogeneous computers that

are physically scattered, even in different continents and sharing resources (Babak *et al*., 2006).

- User centered interfaces: There has to be access to cloud services through simple methods. In fact, cloud computing adopts the concept of *Utility computing.* In other words, users obtain and use cloud platforms as easily as they access traditional public utilities (such as electricity, water, natural gas, telephone network).
- Contribution of services on demand: Cloud systems provide resources and services to users on demand. Users can later personalize their computing environment, for instance, software installation or network configuration.
- Guaranteed quality of service offer: The computing environments provided by clouds guarantee the quality of service for users. For example, hardware performance such as processing speed, input/output bandwidth, and memory size.
- Autonomous system: Clouds are autonomous systems and are transparently controlled by users. Cloud hardware, software and data can be reconfigured automatically to present a unique platform image.
- Scalability and flexibility: Scalability and flexibility are the most important capacities which cause the increase of cloud computing systems. Cloud services and computing platforms offered by clouds, can be escalated through various points, such as geographic location, hardware performance, software configuration etc. Computing platforms have to be flexible to adapt to different requirements and a potentially large number of users.

2.2.1 Structure and taxonomy

Cloud offers Services that are commonly summarized in three categories: *Software as a Service* (Saas), *Infrastructure as a Service* (IaaS) and *Platform as a Service* (Paas) (Zhang *et al*., 2010), Figure 2.1:

- Infrastructure as a Service: Also known as "Iaas", is in charge of distributing infrastructure resources on demand, generally in terms of virtual machines. The owner of a cloud that offers IaaS is called the IaaS provider. Some IaaS providers include Amazon EC2, GoGrid y Flexiscale.
- Platform as a Service: Also known as "Paas", is in charge of providing resources on the platform layer, including operating systems support and frameworks (standardized concept set, practices and criteria to approach) of software development. Examples of Paas providers: Google App Engine Microsoft Windows Azure (Windows Azure) and Force (SalesForce).
- Sofware as a Service: Also known as "Saas", is in charge of providing Applications on demand on the Internet. Saas providers include Force3 (SalesForce), Racspace (Rackspace) and SAP Business ByDesign.

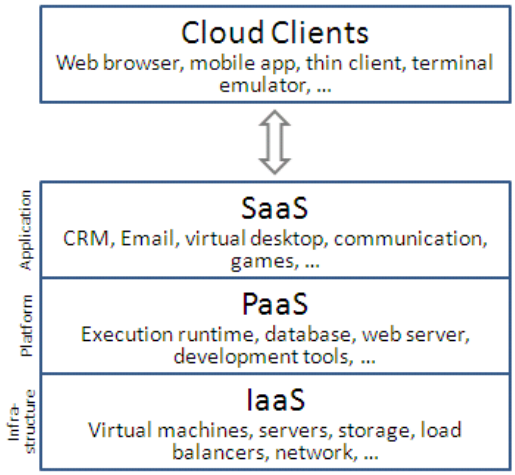

Figure 2.1: Cloud division in layers. (Obtained in Answers)

In (Wang and Lazewski, 2008) a different layer division is discussed: *Hardware as a Service* (HaaS), *Software as a Service* (SaaS) and *Data as a Service* (DaaS). The PaaS layer can be obtained from the three previous layers, as shown in Figure 2.1.

- Hardware as a Service (HaaS): As a result of the advances in hardware virtualization, the pay-per-use service model has become a very good option for consumers. HaaS is flexible, scalable and manageable to satisfy specific needs. Examples can be found in Amazon EC2 (Amazon), IBM's Blue Cloud Project (IBM) and Nimbus.
- Software as a Service (SaaS): Applications are organized as a service and are provided to customers via Internet. This eliminates the need to install and execute the application in customers´ local computers.
- Data as a Service (DaaS): There is access to data in multiple formats and sources via services. Users can, for example, manipulate remote data as if they were operating a local disc or accessing semantically through the Internet. Some examples can be found in Amazon Simple Storage Service (S3), Dropbox or Elastic Drive.

We can obtain a classification of cloud systems by virtue of the deployment infrastructure model, that is, a model where the organization provides, manages and exploits infrastructure (Huth and Cebula, 2011):

- Public cloud: A public cloud can be accessed by any subscriber with internet connection and access to cloud.
- Private cloud: A private cloud is established for a specific group or organization, but with limited access.
- Community cloud: A community cloud is shared by two or three companies with similar cloud requirements.
- Hybrid cloud: A hybrid cloud is essentially a combination of at least two clouds, and may include a mixture of public, private or community clouds.

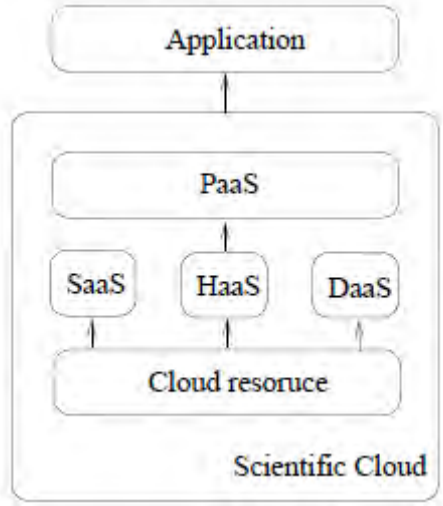

Figure 2.2- Layer division according to Wang

2.3 Cloud environments

SearchCloudCoumputing (2012) presents a list of the 10 main cloud computing providers for 2012, among which are included: VMWare (VMWare, 2012), Microsoft (Windows Azure, 2012), Bluelock (Bluelock, 2012), Citrix (Citrix, 2012), Joyent (Joyent, 2012), Terremark (Terremark, 2012) y Amazon (Amazon, 2012). We will now provide a detailed description of the services offered by three of the current main providers of Cloud Computing: Amazon, Google y Microsoft.

2.3.1 Amazon

Together with Google, Amazon is one of the pioneers in the distribution of services based on cloud computing. Nevertheless, unlike Google, it specializes in services of infrastructure layers and has recently incorporated platform services to offer. Amazon Web Services (AWS) offers a cloud computer platform which is scalable, highly available and reliable, and provides the flexibility needed to allow its customers to create a broad range of applications. In order to provide global security and privacy, AWS creates services according to the recommended security methods; it offers the appropriate security functions for such services and explains how to use these characteristics. AWS customers have to use these characteristics and recommended methods to design a secure application environment (Amazon Security) (Tao et *al.*, 2008).

Some of the services AWS has to offer are (AWS, 2010):

- Amazon Elastic Compute Cloud (Amazon EC2): Web service that provides cloud computing capacity. It provides the consumer with complete control over the computing resources that are at its disposal and allows their execution in a reliable and proven environment.
- Amazon Simple Storage Service (Amazon S3): Amazon S3 provides some simple Web service interfaces that allow storage and recovery of any amount of data, anytime, anywhere around the Web.

- Amazon Virtual Private Cloud (Amazon VPC): Amazon VPC provides a safe bridge between a company´s infrastructure and cloud AWS. It allows companies to connect their current infrastructure to an isolated set of computing resources through a Virtual Private Network connection(VPN).
- Amazon Cloud Front: Web Service for content delivery. It integrates with other Amazon Web services to offer developers and companies an easy way to distribute content to users, with low-latency and high-speed transference.
- Amazon Relational Database Service (Amazon RDS): Amazon RDS is a Web service that allows the creation and management of a cloud database. Amazon RDS provides access to the complete capacities of a MySQL database, so that all tools, code and applications used MySQL work in the same way with Amazon RDS.
- Amazon Simple Notification Service (Amazon SNS): Amazon SNS is a Web service that facilitates the creation, management and sending of notifications from the cloud. It gives developers the ability to publish messages from an application so that they are immediately sent to subscribers or other applications.

2.3.2 Google APP ENGINE

Google Inc. started specializing in high-performance, distributed computing through their star product, the Search Engine. Along with the introduction of Gmail and Google Docs, it became one of the first SaaS providers based on cloud computing. In 2008, both PaaS and its Google App Engine (GAE) product, a platform for the development of scalable applications which are deployed in the Google cloud (INSA, 2012), were launched in the market.

Currently, the only supported programming languages are Python, Java and Go. Web frameworks that run Google App Engine include Django (Django, 2012), CherryPy (CherryPy, 2012), Pylons (Pylons, 2012) y Web2py (Web2py, 2012), as well as written Web application frameworks according to Google. Google manages the deployed code in a cluster (computer sets or clusterings built through the use of hardware), monitoring and launching application instances when necessary. Current APIs (a set of functions and processes that a certain library offers to be used by other software as an abstraction layer) support characteristics such as information storage and recovery of a non-relational BigTable database (Chang *et al*., 2006). BigTable is a database engine created by Google and characterized by being distributed, highly-efficient and proprietary.

In particular, Google App Engine is designed to offer support to applications with many simultaneous users. When an application can provide a service to a large number of customers at the same time without degrading its performance, it is referred to as scalable. Applications written for App Engine escalate automatically. As more people use the application, App Engine contains more resources and controls the use of such resources (Sanderson, 2009).

According to Google Developers (2012), the main advantages of Google AppEngine with respect to their direct competitors are:

- Easy to use: In App Engine, the code of an application can be created, tried out in local equipment and uploaded on Google just by clicking a button or introducing a sequence on the symbol of the system. Once the application is uploaded, Google is in charge of placing and escalating it.
- Automatic escalation: Applications can make use of the same technologies with which Google applications such as BigTable and GFS have been created. App Engine contains an automatic scalability function.
- Liability, performance and security of Google infrastructure: The same security, privacy and data protection policy is applied on all App Engine applications, and all Google applications.
- Cost-effective Hosting: Registration to App Engine is always free. More computing resources can be obtained and only those used are paid for.
- Risk-free Trial period: An account can be created and an application can be used immediately, free of cost and obligation. An application of a free account provides up to 1GB of space and supports up to five million views per month.

2.3.3 Windows Azure

In February 2010, Microsoft launched its cloud platform, Windows Azure Platform (Azure Platform, 2011) to the market. Windows Azure allows compiling, implementing and administrating applications in a global network of data centers administered by Microsoft. It can compile applications in any operating system, language or tool (Windows Azure, 2012). The main object of this platform is to provide an integrated environment of development and control, so that developers can create, support, control and escalate Web applications as well as non-Web applications through Microsoft data centers (Buyya *et al*., 2009). Windows Azure Platform includes three main components (Azure Platform, 2011):

- Windows Azure: Provides a Microsoft Windows Server based environment (WindowsServer, 2012) for applications and persistent storage, for both structured and non-structured data and un-synchronized messaging.
- Windows Azure AppFabric: Provides a range of services that help with user connection, authentication management, implementing data management and similar characteristics.
- SQL Azure: Essentially an SQL server provided as cloud service.

Some of the services provided are:

- Computing services: Windows Azure Compute service can execute many different types of applications. The platform aims to support applications that have a large number of simultaneous users. Windows Azure is designed to support applications which execute multiple copies of the same code through various servers. In order to achieve that, Windows Azure applications can have different instances, each one being executed in its own virtual machine (Chapell, 2009).
- Storage services: Windows Azure, SQL Azure, and associated services, cause situations in which it is necessary to store and control data in many different ways. The following data management services are offered (Azure Platform, 2011):

- Azure Table Service: Provides a table storage system for structured data and supports searches to manage information.
- Binary Large Object Service (BLOB): BLOBs are elements used in data bases to store high volume data that change dynamically. It provides a series of containers to store text and data. It provides *Block* BLOB containers (BLOBs made of blocks, each one with a unique identifier. They are used to manage large files through the network, and are able to manage various in parallel, for data transmission and containers Page BLOB (Collection of pages of 512 bytes. They serve to carry out random reading and writing operations) for input/output operations.
- Queue Service: Provides a trustworthy and persistent mechanism for messaging between instances.
- Windows Azure Drive: Provides a mechanism enabling applications to set up a NTFS VHD volume (file system that virtually represents a hard disc) as a Page BLOB, so that it can upload and download VHDs through BLOB.

  The storage system can be accessed by a Windows Azure application, by an application which is executed under the premises of some organization, or by an application being executed in a platform which provides support (Chapell, 2009).

- Network services: Windows Azure provides different network services that allow improving performance, implementing authentication and improving management capacity of the applications:
  - Content Delivery Network (CDN): CDN can introduce into a cache the static data available to users for applications in proximal strategic points (in a network delivery system).
  - Virtual Network Connect**:** This service allows configuring roles in an application executed in Windows Azure and network computers in a way that makes it appear as if they were on the same network.

    Virtual Network Traffic Manager**:** Service that allows establishing redirections of requests and load balancing based on three different methods. *Traffic Manager* is commonly used to maximize performance when user requests are redirected to the closest data center using *Performance* method. Available load balancing methods are *Failover* and *Round Robin*.
  - Access Control: Service based on identification and access control standards that use a range of identity providers which can authenticate users.

    Service Bus: Provides secure messaging capacity and data flow for distributed and hybrid applications, such as communication between Windows Azure applications and internal applications and services.

2.4 Cloud environment

Cloud computing is fundamentally characterized by its scalability, that is, the ability of the system to assist a large number of customers by increasing hardware and

software resources without changing its implementation; and by its flexibility, which makes reference to the ability to dynamically escalate with service load. Consequently, traditional paradigms of communication between applications, operating systems, storage of files or managing database systems are not directly applicable in this new environment. Applications have to be able to escalate to all their levels, without having points that make it impossible to benefit from the flexibility of other layers in its architecture. The technological pillars on which cloud computing is based are: (i) The use of virtualization to facilitate the administration of large networks of computers (ii) The use of distributed database. (iii) The use of distributed files systems. (iv) The load balancing of requests to services between service providers, providing a unified access point and the capacity to escalate a service by adding providers.

2.4.1 Virtualization

Virtualization abstracts the physical structure of various technologies. In Computer Science, the term virtualization refers to the creation of a virtual version of ¨something¨, such as hardware, operating systems, network resources or storage device (Eisen, 2011). Modern virtualization systems are grouped into two categories: systems of complete virtualization and those of *paravirtualization* (Crosby and Brown, 2007). Complete virtualization provides a total abstraction of the underlying physical system and creates new virtual systems in which the guest operating system can be executed. There is no need to carry out any modification in the operating systems (the guest operating system does not recognize the virtualized environment and is executed as normal) (RedHat, 2007). Paving requires the modification on behalf of the user of the guest operating system which is executed in virtual machines (these operating systems are aware of the fact that they are being used in a virtual machine), and provides similar performance as if it were native (RedHat, 2007). The *paravirtualization* approach introduces some restrictions: compatibility between the guest and host system; and the need for the core of the guest operating systems to be modifiable. Nevertheless, it turns out to be more convenient, with regard to the general performance of the system.

Virtualization is one of the common denominators of solutions based on cloud computing found in the market, as it allows abstracting the applications from the software where they are located , facilitating their administration and relocation to other physical machines (to redistribute the load or increase fault tolerance in the hardware). Virtualization also increases security, confining each service provider to a virtual environment, so that a compromise in the security of a service will not directly affect the other system. Its contribution to security and the simplification of data center management largely outweigh the expected loss of performance with respect to the direct execution over the hardware. In the range of services based on Cloud Computing, the contribution of virtualization is especially important in three aspects: the simplification of server maintenance, flexibility support, and security support. A virtualization environment will not add the majority of these characteristics on its own, but it will facilitate its implementation (INSA, 2012). (Hawley, 2009) offers a

virtualization comparative in cloud systems with respect to the traditional concept of virtualization as shown in Table 2.1.

Traditional virtualization infrastructure	Virtualization designed for cloud systems
Storage and network: Consolidated servers with static and inflexible connections.	Shared storage, flexible networks to create sources of services.
Resource management: There is no dynamic automation of resource management.	Automation policy to dynamically the load between virtual machines.
Operation and administration model: Traditional.	Automation supports self-provisioning of virtual machines with administrators controlling the escalation of the package of services.
Provisioning of a Virtual Machine: Creation and manual deployment.	*Virtual appliances* libraries (Image of de virtual machine designed to be executed in a virtualization platform) for the quick and low-risk creation and deployment of VM

Table 2.1: Comparison between a traditional virtualization infrastructure and one designed for clouds.

2.4.2 Information storage in cloud systems.

Data storage scalability plays a key role in cloud computing. The increase of information process capacity cannot be useful if information access speed does not increase as well. The problem of creating faster, distributed and fault-tolerant databases has provided research with new algorithms and communication protocols between different nodes of a distributed database.

However, it has also revealed more profound proposals which question the database model that is being used in Internet-oriented applications. A distributed database (DDB: *Distributed database*) is a collection of multiple databases related to a computer network. A distributed system of database management (*distributed* DBMS) is the software system that allows a distributed database management to render transparent distribution to its users (Zsu and Valduriez, 1991). There are two main ways of using a database in the cloud: creating an image of a virtual machine, or using the database concept as service. As far as images of virtual machines are concerned, cloud platforms allow users to acquire instances of virtual machines for a limited period time. A database can be executed in such virtual machines. Users can upload their own machine images using an installed base, or use the existing ones that include the optimized installation of a database. Some cloud platforms offer the option of using a database as a service, without physically launching a virtual machine instance for the database. In this configuration, application owners do not need to install and maintain the database. Instead, the database service provider assumes the responsibility of its installation and maintenance. Users will pay for the use they make of them. Two types of database can be used in a cloud: (i) SQL databases: Oracle Database, Microsoft SQL Server and MySQL, are one of the types of databases which can be used in a cloud (as a virtual machine image or service). SQL databases are hard to escalate, meaning that they are not natively adequate to adjust to a cloud

environment, even though this problem is being tackled with (Rosenberg, 2011). (ii) NonSQL databases: Apache Cassandra, CouchDB y MongoDB, are other kinds of databases that can be used in a cloud. NonSQL databases are built in a way that allows them to offer services to large loads of reading/writing and be able to escalate easily (Agrawal *et al.*, 2008). However, most more contemporaneous applications are built on an SQL data model, which means that working with the NonSQL model requires the complete rewriting of the application code (North, 2011).

The document-oriented model is the most extended one among the alternatives to the relational model. In this case, the main unit of information is the document. A document is built from a collection of key-value pairs, in the same way in which a software object is defined through a set of attributes (keys) and its values. Compared to the relational model, the documents would correspond to the tuples, and the keys to the columns of a table. Values assigned to keys would correspond to values of the fields of the tuple. Documents are nest-able, that is, a value for a key of a document can be another document, and they are grouped in collections. Collections are semantic groups of documents, as in the relational model, and tuples are grouped into tables. Nevertheless, there are no other similarities: documents of a collection do not have to share a set of attributes, although they usually do so in real applications. A document with an attribute which others do not have can be inserted and integrated into a collection at any given time, without jeopardizing the performance of the system, or adding new attributes according to those already created. Relations can be represented by references or through nesting.

2.4.3 Filesystems

Traditional filesystems cannot fulfill the needs of large systems which containing a great amount of data.

This kind of system requires:

- Fault tolerance and high availability: information cannot be lost if a node failure is produced, and the service should not be interrupted if possible.
- Spatial scalability: it is necessary to be able to add new space to the file system without jeopardizing performance, and beyond the capacity of a unique node of computing.
- Scalability in speed: it is necessary to be able to copy information to increase access speed to it.

To achieve these characteristics it is necessary to use a distributed filesystem. A distributed filesystem allows access to files from multiple hosts through a network (Silberschatz, 1994). This enables various users in different machines to share files and storage resources. Customer nodes have no access to the underlying storage but interact through the network through a protocol. This allows restricting access to the filesystem according to access lists in both servers, as well as customers, depending on the way the protocol is designed. As in the case of distributed databases, the characteristics of distributed filesystems include scalability, fault tolerance and high availability. These characteristics are achieved through two mechanisms: replication and data partition. Data replication in different hardware nodes and different discs allows faster access to information, uninterrupted service because of node failure, and

will not lose information in case a disc stops functioning. Data partition increases information reading and writing speed and allows increasing system capacity through the addition of new hardware nodes.

2.4.4 Load balance

One of the main problems in cloud systems is managing requests of a great number of users. It is a scalability problem that arises from the continual increase in the number of active users in the system. As data have a way of producing picks with little or no prior notice, even the most advanced cloud environment will be little useful without a load balance automated component (Cole, 2011). Load balancers can be considered as a special type of proxy. Apart from modifying customer's requests, they can distribute requests that reach the server from a set of service providers, additionally providing an access point unified to the server sets. This way, customers do not have to know either the number or the location of service providers, but rather just the name of the network of the proxy team. The load balancer proxy, who does have this piece of information, will be in charge of redirecting the request back to one of the real servers (INSA, 2012).

Internally, the load balancer can use different algorithms to choose which server to forward the request to. One of the most common ones is *Round-Robin* algorithm, so that requests are delivered to servers alternatively (following a circular list order). Load balance can also be controlled so that it depends less on the exhaustive monitoring of the domain and its associated deliberation. On the other hand, there are methods that globally promote load balance through actions and interactions at the component level (individual server). There are three important solutions to this problem (Randles *et al.,* 2010): algorithm Honeybee Foraging (Nakrani *et al*., 2004), Biased Random Sampling (Abu-Rahmed, 2008), and Active Clustering (Safre *et al*., 2008).

2.5 Reuse

Software reuse is currently approached as a complementary way of improving system development processes, aiming at lightening all tasks, typical of these processes, and increasing quality of the obtained systems. Researchers in this field (Schmid, 2011a) (Poulin, 2006) (Poulin, 1997) coincide in assuring that a systematic, automated and formal reuse program would achieve all these goals, even though, to date, efforts to incorporate this kind of reuse plans have been slowed down by various factors, among which the lack of methodologies and appropriate technological environments could be the most significant. (Lemley and O'Brien, 1997) (Shang *et al*., 2012). It is in this field where, in the recent years, different studies have emerged with the aim of providing typical tasks of a reuse environment with technological and technical support, and allowing it to be used systematically (Sherif and Vinze, 2003) (IEEE1517-2009 D2, 2009) (Ravichandran, 1999). Nevertheless, to date, it has not been possible to obtain a wide and open methodology, one that is common for all development processes of systems in cloud computing environments. This is precisely the direction this PhD thesis follows, providing the methodological basis which

facilitates reuse in the field of marketing processes executed on cloud environments and implemented by Web services.

The distributed and adequate management of a large number of information is the main goal of a cloud environment; and to successfully reach that goal, in an agile, efficient and cost effective way, we suggest using a methodology that would facilitate reusing. As made clear in the state of the art review, within the context of cloud computing there are no methodologies and/or processes that would standardize the reuse of Web services and/or marketing processes. In the framework of this report, we have considered and shown that a viable alternative to development processes traditionally used in this field is the reuse of components. It is essential that access to information stored in this kind of environment, information related to previous developments, is adequately accessible and listed through an ontology. Without all this, the redundant work we are trying to minimize is turned into an equal or larger amount when it comes to recovering necessary Web services. This is precisely the reason why a large part of this work is dedicated to obtaining classification criteria that allow adequate management of the library of marketing processes. After examining the different alternatives, we have opted for making use of the formalization which is present in the typical processes or services as a basis for extracting an internal representation of each one, a representation that facilitates the classification, storage and subsequent recovery of each service, thus avoiding ambiguity problems which are typically inherent in any type of textual description.

We are still far from the day when automatic programming, starting from a set of normally vague, barely coherent or maybe incomplete requirements, will generate the code appropriate to customer needs is generated (if it can be reached at all one day), especially in a field as recent as that of cloud computing. Prior to this situation, the reuse of marketing processes and Web services was divided as a realistic and technically feasible alternative. We can say that, among specialists in the field, it is widely accepted to describe the reuse of software as (Krueger, 1992):

> *"... the process of creating computer systems starting from existing ones rather than creating new ones"*

It is precisely because of the flexibility of the definition that it does not correspond to the type of element to be reused, its grade of abstraction or granularity, and whether or not some kind of modification is carried out on it. Nevertheless, it is also true that other authors nuance this description. For example, (Sametinger, 1997) considers that it is only correct to speak of reuse when the software is applied to the new systems without undergoing any kind of modification. Others are even more restricting, also limiting the type of software element to be used. Thus (Stroustrup, 1996) talks about reusing when a code fragment is literally used in at least two computer program. Within the scope of this work we welcome the definition given by (Krueger, 1992) and we refer to both software reuse and the application of already existing software products (code, designs, documentation, specification, etc.) for the creation of a new system; that is, the use of "any kind of already existing information the developer needs in order to elaborate a new software system" (Prieto, 1993). Specifically, we reuse Web services that implement marketing processes.

In the following section we analyze the reasons that suggest reuse in general and those that have historically impeded its expansion. There will also be an introduction to existing methodologies and an analysis of what the impact of developing one in the cloud computing context would be.

2.5.1 Reuse potential

The use of good reuse practices in any organization that aims at software production has to be channeled to short or long-term economic benefits. These benefits are a direct consequence of an improvement in the quality of the generated software and the time and production cost saved. The fact that, until now reuse is not applied systematically, automatically and formally has to be attributed to a variety of factors, not just technological, but also cultural and economic in nature.

Software reuse directly affects the improvement of the majority of the factors that affect quality in a system:

- Funcionality: Functionality of a software system can be defined as the set of services it offers to users, as this is directly related to the initial requirements. It is very common that the establishment of these initial requirements requires user feedback , as they usually have a vague idea of the system in the early stages of its design. Reuse offers a basis for the establishment of fast prototypes (Sametinger, 1997) in such a way that it allows the user to estimate the system´s functionality and correct it in the early stages of its circle of life.
- Liability: System liability measures the capacity at which it maintains its services, under a series of application conditions, throughout a period of time. When software is reused, it has been verified and validated, vouched for, in many cases, by previous uses, thus improving global system liability.
- Efficiency: Software system efficiency relates its performance to the amount of resources it consumes during its durability, apart from the ones invested in its creation. The creation of reusable software does not prove to be efficient if the basis of reusable components is inadequate or does not contain homogeneous elements: software created for a particular environment is usually little efficient when applied to a different environment, as this additionally requires a larger investment in time and cost in order for it to be reusable. However, in this particular case efficiency would increase, as, in general, cloud computing environments hold very homogeneous applications. This research specifically focuses on creating a methodology that facilitates the composition of marketing processes involving previously defined ontology and semantics.
- Maintenance capacity: It is easier to maintain applications built from reusable software. Modifications that need to be carried out may have already been planned in many cases, documentation is much more carefully done and, what is more, cost will be shared, as maintenance will be centralized and not specific to each particular application.
- Portability: The more portable a software system is, the easier its transference is from one environment to another. As this is one of the most

significant characteristics when generating reusable software, portability is increased in systems where reuse has been applied, as a quality standard ISO 9126 specifies (ISO, 1991)

From all these characteristics we can deduce that, in general, reuse has a beneficial incidence in the increase of the quality of the generated software (Basili *et al*., 1996). On the other hand, as expected, the fact that a decrease in efficiency is nothing more than the price to pay, reasonable in the majority of cases —except in situations of serious temporary or resource constraints—, to obtain reusable software elements. Improving software quality has a clear economic repercussion as maintenance and development process costs drop (Karlsson, 1996). The software ought to:

- Work faster;
- Work more efficiently in order to keep the development process under more control (more precise estimations as for cost, improvement in the process itself, in the assessment and system improvement, ...); and
- Work better, when it comes to avoiding work duplication, with software created by specialists in each field, thus reducing the number of work teams and facilitating interoperability.
- In spite of all the benefits which reuse provides to software system development, organizations usually remain reticent towards the possibility of applying it in the elaboration of their products. There are various reasons for that which can be summarized as:
- Economic reasons: Implementing a reuse project requires high up-front investment from which benefits could only be obtained in the medium to long term. Finance is needed to: obtain reusable software, reuse the obtained software and define and implement reuse process. That is, we must at least invest in infrastructure, methodology and technical support. Moreover, generated software has to meet some more constraining quality requirements, rendering the whole process more expensive.
- Technical reasons: to date there are no tools that facilitate or automate the typical tasks of a reuse program, such as: software element classification and search, and adaptation or integration of such elements in the new systems.
- Organization reasons: Large scale software organization affects its whole life cycle and requires a re-structure of the traditional organization. It could be more useful, for instance, to create teams that are only in charge of generating and maintaining reusable software. In this situation, global management is crucial. Conservative attitudes on behalf of administrators towards these changes and the "NIH syndrome" (Not Invented Here), contribute to the delay in the application of a reuse policy in companies and software producing organizations.

Nevertheless, in different engineering fields reuse is an inherent part of the process of creating a system. For the design of an electronic system, for example, a group of engineers start from a series of requirements and specifications. Usually, they will reach a compromise between the requested requirements and the characteristics of the available electronic components, so that ad-hoc development is only rarely carried out. However, work dynamics in the software industry is radically different: it starts

from a set of specifications, but once this is fixed, it only has to satisfy and complete them. This approach supports containing particular solutions and hampers reuse. Thus, it seems that one of the main characteristics of the software, its adaptability and flexibility, is also its main obstacle when it comes to generalizing in this industry an effective and efficient reuse policy.

2.6 Conclusion

Cloud computing has great potential and is an expanding area within software development. Reuse is not a common practice in this context for the reasons mentioned in the previous section, which affect all types of reuse. The definition of a methodology that would render the reuse system efficient while maintaining high quality standards is one of the challenges of this review. For this, we need to analyze the grade of applicability of the proposal and the perspectives this methodology can have.

The amount of potentially reusable software depends on the level of functionality which is common between the systems that share it (Sametinger, 1997). Defining domain as area of application or field of system development, we understand that the level of reuse is high and efficient between systems belonging to the same domain. System reuse between systems of the same sphere of application is named vertical or domain dependant reuse. Horizontal, or general, reuse is established between systems that do not belong to the same domain, rendering it, in general, more complex and, consequently, reducing reuse level. In this context this review is presented in vertical reuse of marketing processes implemented as Web services and executed in a cloud computing environment. Traditionally two methodologies have been differentiated to focus on software reuse; the first one is based on obtaining a new system from the composition of an already existing element; the second methodology is based in generating a new system using a structure or model as a basis. Reuse by composition is rather intuitive; it is about combining elements and components to build a new system. It is where this research work is based. Although it is easy as a concept, it requires complex technical support as well as optimized classification and selection methods according to its components, and support for its adaptation and integration in the new system. For this, we propose the use of a multi-agent architecture based on virtual organizations. Reuse by generation is conceptually more complex as it is impossible to define the components as self-contained and concrete entities. In this case, we reuse generation processes obtained as a result of a structure codification. There are usually three different subtypes of methodologies: application generators, generators based on transformation language, and systems.

System generation is oriented towards the reuse of elements belonging to the first stages of the life cycle of software such as designs, architecture or requirements, rendering it more attractive, from a financial perspective, yet more difficult to apply. However, reuse by composition was the first one to be used, focusing on elements belonging to the last stages of the software process, as in the case of the code. In the case of reuse by composition, we need to specify how we intend to integrate components in the new system, which is why we need to talk about: component visibility and component modification.

We call *black boxes* those components whose interface and functionality is the only thing we know; as there is no access to its interior, it is impossible to carry out any kind of modification, so in this case we can only talk about integral reuse. Those components that allow access to their interior are called *white boxes*, in them modifications can be carried out to adapt their functionality as required. An intermediate type are *transparent boxes* (Sametinger, 1997), this type of component allows getting to know the interior, in the case of white boxes. However, its modification is not allowed, as in the case of white boxes. The reuse of black boxes, though much more difficult, presents many more advantages, such as quality and liability of the generated system increase, and the fact that components of this kind have been verified and are usually certified (Dunn y Knight, 1993; Knight y Dunn, 1998). Nevertheless, when white boxes are reused, modifications that cause verification tasks to be repeated are likely to be carried out, reducing its liability. Reuse by generation can be seen as a type of reuse of black boxes, as, although it does not contain the component itself, it does contain a generating programme which is reused as if it were a black box. Throughout this research we will identify a more adequate type to meet the objectives proposed.

According to the definition adopted here for software reuse, all the necessary information to design and develop a system will be considered potentially reusable. In the case at stake, it is even more important to have a clear ontology and well-defined semantics. Thus, there will be code as well as documents, designs, specifications, texts, etc. The differences between the reuse of some software from the reuse of others mainly lies in the level of abstraction and component granularity. Both characteristics are a source of controversy at the time of deciding what to reuse (Prieto, 1996). It is necessary to adopt a position of compromise between the benefits the reuse of a determined element offers, directly proportional to the level of abstraction and its size, and how easy this is reused, inversely proportional to the same factors. In this report, we present a methodology to render the efficient reuse of white boxes and to superimpose them to economic, technical and organization motives which, generally, involve carrying out software reuse. In the next chapter, we will define the organization of agents that will be in charge of managing that process, and in the fourth chapter we will define the field of marketing processes and related elements used in the proposed model.

3 MAS AND ONTOLOGIES

3.1 Introduction

The agent concept has widely been studied today. However, there are small gaps with regard to consider an agent as an entity performing within a society. This is an especially important issue for an agent, because one of its basic characteristics is its interaction with other agents in a cooperative environment. This chapter studies the agent concept focused on its use in cooperative environments. It starts with a review of the notion of intelligent agents and exposes then a description of *multi-agent systems* (MAS) and the definition of *agent society*. The purpose is to highlight current trends of MAS from an organizational point of view. The agent theory has been studied from such diverse fields as Psychology, Computer Science, Sociology, Medicine and Economy, having a different behavior on each of them. With respect to Computer Sciences, the term *agent* is becoming more known and is being used in environments as diverse as the Internet, Distributed Systems, Artificial Intelligence, or Human-Computer Interaction (Corchado, 2000). Both the diversity and the power of this technology are notably extensive. Although this chapter presents a generic description of the agent theory and MAS, it also outlines the concept of organization, since in order to develop the proposal model we have used a multi-agent virtual organization.

The agents can be described from different points of view. This is because they are a research field where scientists from very different areas such as Psychology, Sociology, Software Engineering and Artificial Intelligence approach the field from the perspective of their research areas (Forner, 1993). Hence, the definition of agent or agency is complex, due to the diversity of opinions existing in the scientific community (Franklin and Graesser, 1996). Even in the Computer Science field, there is not a well-determined definition of agent. The initial research in the agent field has been used in environments as varied as: Distributed Artificial Intelligent, Robotics, Artificial Life, Object Distributed Computing, Human-Computer Interaction, intelligent and adaptive interfaces, information search and filtering, Knowledge Discovery, etc. Since the proliferation of different types of agents, there has been an explosion in the use of the term without common agreement on its meaning (Bradshaw, 1997). Within all these definitions, there exists a formal and accepted definition in the scientific community, which according to (Labidi and Lejouad, 1993) is:

"*An agent is a physical or abstract entity that can perceive its environment through sensors, it is able to evaluate such perceptions and make decisions by means of simple or complex reasoning mechanisms. It is also able to communicate with other agents to achieve information and act upon the environment in which performs by using effectors.*"

Another fairly widespread way of introducing the agent concept is based on the characterization of the agent by a series of attributes that it should include. This has

resulted in the four attributed identified and defined by (Wooldridge and Jennings, 1995), to characterize an agent:

- Autonomy
- Social Skills
- Reactivity
- Pro-activity

The development of intelligent computer systems is guided by the construction of entities simulating human behavior (Russel and Norvig, 1995). As with humans, agents must have social skills, and be able to perform jobs or solve problems in a distributed way (D'Inverno and Luck, 2004). In fact, an agent can be seen as an evolution of the object concept that couples proper characteristics of human behavior. From these characteristics, we can highlight intelligence and learning capacity. The development of information processing and computing technology has allowed the construction and use of artificial agents (Burgin and Dodig, 2009). The emergence of the agent concept has made possible the convergence of diverse areas to a common space. Composite Systems by multiple agents are initially developed in an environment of distributed artificial intelligent (DAI) (O'hare *et al*., 1996). In the beginning, DAI raised the issue of distributed problem solving where a particular problem can be solved by a certain number of elements, which cooperate and share knowledge of the problem and its solution. This way, such systems are parts of one of the three basic categories in DAI, the other two being: Distributed Problem Solving and Parallel Artificial Intelligence. As a result, the systems composed of multiple agents inherit much of the motivations, aims, and powerful benefits of DAI (Nwana, 1995). The goal consists of building systems composed of multiple entities able to solve problems, in such a way that these entities interact to improve their performances (Jennings, 1993). Such systems of multiple entities are known as multi-agent systems (MAS), and they are suitable for solving problems where there are multiple troubleshooting methods and/or multiple entities able to work collaboratively to solve problems (Chu-Carroll *et al*., 1995). MAS can be used to approach problems that would otherwise be difficult or impossible to solve by means of a single agent.

Intelligent agents are a very important object for multiple fields of Computer Science and Artificial Intelligence (AI), because they represent a paradigm to develop applications (Jennings and Wooldridge, 1998). The challenges faced today by the computer system developers are becoming increasingly complex. Globalization and changes in technology have caused the current market to be in a constant state of fluctuation. Companies that do not adapt quickly to this environment will be left behind. As a reply, many companies are building agent-based systems. These systems use software-agents to distribute functionalities on the computer network. Furthermore, in addition to adapting to their environment, agents also evolve by learning from their environment, and using a variety of computational approaches from Expert Systems, Artificial Neuronal Network, and Genetic Algorithms, among others. Since the Internet is one of the most important areas of software development, MAS applications are not limited to the business and academic environment. Internet agents arose from problems of Web information searches and filtering. These agents

are useful because of the vast amount of information that can retrieve from the Internet (Burghoff et al., 1996). Since time is a determining factor for people, and the possibilities of accessing information are improving; the Internet users should develop skills and look for tools able to access the requested data in an opportune and acceptable way. Virtual agents are a good alternative for saving search runtime, analyzing and handling requests. Some MAS applications are:

- *Information filtering agents:* Find the content a user is interested in by using different information sources.
- *Off-line delivery agents:* Filter the customized information delivery without necessarily requiring an on-line Internet connection.
- *Search agents:* Use robots moving in Web hyperspace to provide a service to support the users.

Cloud computing is another field where agents and MAS are being used. Cloud computing can offer a powerful, predictable and scalable computing infrastructure to run MAS, and is able to implement complex agent-based applications when the model and the simulation of complex systems should be provided. On the other hand, software-agents can be used to implement intelligent behavior in cloud computing, thus making it more adaptive, flexible and autonomous with regards to resource management, service distribution and the execution of large-scale applications (Talia, 2012). One area where MAS are being used both properly and successfully is in the world of videogames. The increasing popularity of videogames has required a more natural and sophisticated behavior of the characters in the videogame. The more complex the interaction between the characters in a videogame, the more difficult the design of such characters without using tools aimed at implementing intelligent agents (Dignum, 2011). One of the first attempts to connect agents to videogames was carried out with the *Gamebots* (Adobbati, 2001). The *Gamebots* provide infrastructure allowing a connection for any agent platform to videogame *Unreal Tournament.* The Agents are currently used to emulate human behavior to reflect more realistic videogames.

Another area of MAS application is Data Mining. There are two known approaches: the agents perform the data mining process or data mining is used to improve the intelligent characteristics of the agents (Moemeng *et al*., 2011).

3.2 Basic concepts

The agent concept emerged on the 90s and from here; several scientific areas have been fused from AI to Psychology, and everything in between, including Software Engineering, Database and Distributed Systems, Sociology, etc. (Corchado, 2000). Since the concept of agent is in a multidisciplinary environment, there is no uniform definition (Rodríguez, 2010).

3.2.1 Agents

Next, we will provide several definitions that we have chosen according to their level of importance for the scientific area. One of the most widespread and accepted definitions of agent was given in (Wooldridge and Jennings, 1995):

"An agent is an encapsulated computational system and located in an environment, being able to act in an autonomous and flexible way in such an environment to reach its design goals".

Wooldridge also introduces a less formal and more contemporary definition with respect to the one provided above:

"An agent is hardware-system, or more usually, software holding the following properties (Wooldridge and Jennings, 1995)":

- *Autonomy:* The agents perform without direct intervention of humans, having a certain kind of control on their actions and internal states (Castelfranchi, 1995).
- *Social skills:* The agents interact with other agents, possibly human, by means of a language of agent communication (Genesereth and Ketchpel, 1994).
- *Reactivity:* The agents perceive their environment (it can be the physical world and user via graphic interface, a collection of other agents, Internet, or maybe a combination of them all) and respond in a certain time to changes that occur in it.
- *Pro-activity:* The agents do not only respond to their environment, they are able to take the initiative across to behaviors leading to aims.

More recently, *Wooldridge* redefined the concept of agent (Wooldridge 2002) as follows:

"An agent is an encapsulated computational system located in an environment, being able to act in an autonomous way in such an environment to reach its design goals".

The concept of *encapsulated computational system* refers to the clear distinction between the agent and its environment according to a well-defined interface for both. The first characteristic of this definition is the autonomy of the agent; that is, the agent is able to perform on its own without human assistance. From its internal state and perceptions, it can make decisions on whether to carry out an action or not. Furthermore, it is designed to hold specific goals. Note that the above definitions may give way to a more formal and general agent definition (see Figure 3.1):

> *"An agent is physical or abstract entity able to perceive its environment through sensors. It is capable of evaluating such perceptions and making decisions by means of simple or complex reasoning mechanisms, and communicating with other agents to achieve information in order to act in its environment through effectors (Labidi and Lejouad, 1993)."*

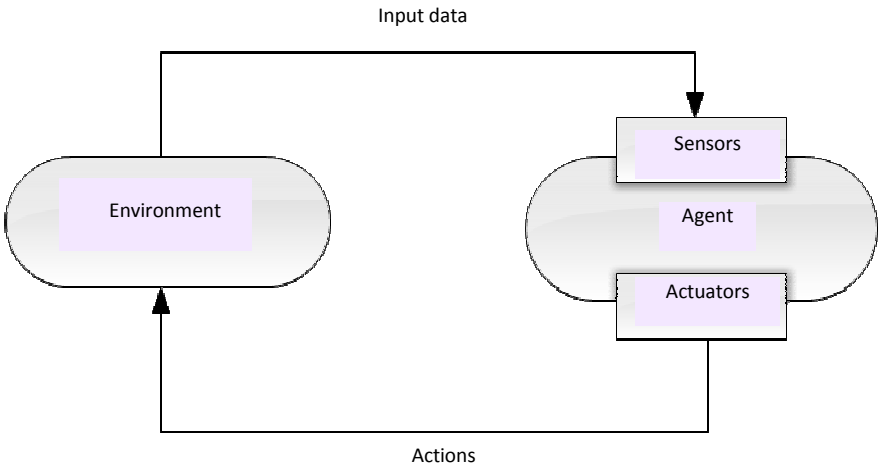

Figure 3.1: Agent model

From the most recent scientific publications on agents, we can highlight the one given in (Rahman, 2012):

> *An agent can be defined as a software and/or hardware component of a system, carrying out tasks on behalf of its user. The agents are reactive, autonomous and cooperative in nature. They have the ability of knowledge-based reasoning.*

Because of disagreement existing between different authors and the ambiguous nature of the definition of agent, this concept is used to differentiate itself from a series of desirable properties (starting from the properties defined in *(Wooldridge, 2002)*) to be considered by the agents:

- *Autonomy:* An agent should be act without the direct intervention of any other entity, having control on its actions and internal state.
- *Location:* An agent is located within an environment that can be real or virtual.
- *Reactivity:* An agent is able to detect events of its environment and can adapt to its needs.
- *Pro-activity:* An agent has the ability to define aims allowing it to reach its goals and a set of actions related to those aims.
- *Social skill:* An agent is able to interact with other agents, including human beings.
- *Intelligence:* An agent is able to incorporate knowledge (beliefs, desires, goals and intentions)
- *Organization:* The agents are able to organize themselves into societies by following a human-like or biological structure.
- *Learning:* An agent is able to adapt to changes in dynamic environments by means of machine learning techniques.

3.2.2 Taxonomy

Because of the great number of definitions and characteristics identifying the concept of agent, the task of classifying different types of agents is not less complex than the definition of agent (Gil, 2011). Even so, different agent classifications are given according to their attributes or environments.

Thus, (Russel and Norvig, 1995) propose an agent classification based on the kind of program used in the implementation of the agent functionalities by defining the intermediate state between perceptions and actions:

- *Simple reflex agent.* Agents connecting their perceptions and actions through rules of type condition-action. If the condition (evaluated by the perceptions) has been met, then the action corresponding to such a condition is raised. These agents have no memory.
- *Reflex agent with internal state.* Compared to the simple reflex agent, it adds a memory state that stores past experiences to improve the responses given in the future (Carrascosa *et al*., 2008). The state is updated with the obtained perceptions and the actions that have been carried out.
- *Goal-based agents.* The agents are entities or processes aimed at reaching goals in an organization or a system based on the implicit responsibility assumed by them to reach certain aims (Anton, 1996).
- *Utility-based agents.* The aims alone are not enough to generate high quality behaviors (Russel and Norvig, 1995). The agent must show a *degree of satisfaction* when it is in a any type of state. This is made by a utility function that associates a utility value with each state. Hence, the agent can make relational decisions when there are several goals without having the certainty of any of them, or when reaching a goal provides a conflict.

One of the most accepted agent classification is the one given in (Nwana, 1995), which is based on different dimensions of agents:

- *Mobility*: the agents can be classified according to their ability of moving (or not) to different nodes in the network.
  - *Static*: it is only able to perform in the computer where it was created.
  - *Mobiles:* The mobile agents are computer programs migrating through different hosts in a network, at such times and places as it deems appropriate. The state of an agent can be saved and then translated to a new host to be retrieved, which allows the program to run from the point where it left off (Kotz and Gray, 1999).
- *Reasoning model:* The agents can be classified as:
  - *Deliberative:* They are based on the paradigm of deliberative reasoning. The agents present an internal and symbolic reasoning model; they are involved in scheduling and negotiation processes leading to coordination with other agents (Nwana, 1995).

- *Reactive:* They have no a symbolic model of the environment and perform according to the type of stimulus/response by taking into account the current state of their environment (Ferber, 1994).

The agent classification can be carried out according to a list of three attributes: *(i)-Autonomy:* ability of acting without direct intervention of another entity. *(ii)-Cooperation:* when a task is too big or complex to be developed by a single agent, it can cooperate with other agents to solve such a task. *(iii)-Learning:* the skill of an agent to modify its behavior based on past experiences. From these three attributes it is possible to achieve four categories belonging to the typology of *Nwana.* Note that even though the categories have been built as an interception of two attributes, it does not mean that they do not have anything at all from the third. This implies that all attributes used in the classification are considered:

- *Collaborative learning agents:* Agents having the ability to learn by working in a coordinate way.
- *Interface agents:* Autonomous agents with the ability to learn.
- *Collaborative agents:* Autonomous agents cooperating with other agents.
- *Intelligent agents:* Autonomous agents with the ability of learning and coopering with other agents.

The agents can also be classified according to the performed roles. For example, we have the Internet agents or the information they are looking for and processing from wide networks such as the Internet.

- *Hybrid:* Combines two or more of the above philosophies in a single agent. Another classification can be done from a different point of view of previous ones as provided in (Franklin and Graesser, 1996). It is based on the represented biological model by following a natural hierarchy, Figure 3.2. The first level corresponds to the *kingdom* level and classifies the agents as: Biological, Robotic or Computational (Keil, 1989). At the next level, the computational agents are divided into: *(i)-Artificial life agents:* Computational model based on the analysis of the specific individuals of an environment. This model is aimed at studying complex systems that show a similar behavior as natural living systems. *(ii)-Software agents:* Software performing as an agent for a user or another computer program by working in a continuous and autonomous way within an environment. The last level corresponds to classifying the software agents into different categories according to their purpose.

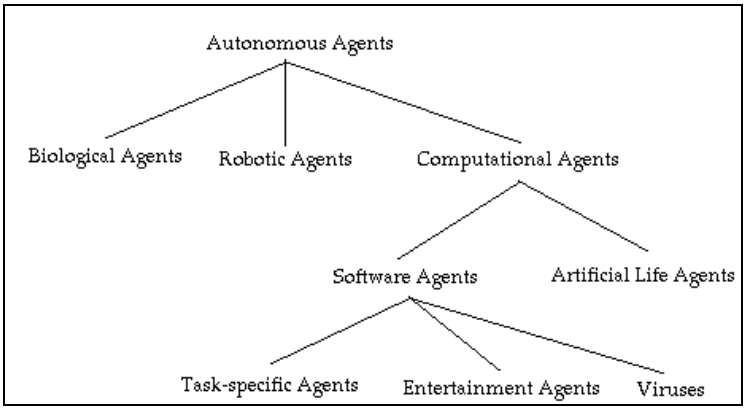

Figure 3.2: Agent Taxonomy (Franklin and Graesser, 1996).

3.2.3 Agent architectures

According to (Maes, 1991), an agent architecture is a particular methodology aimed at the construction of agents. It describes the form in which the agent is decomposed into a set of modules and the form in which such modules interact. The set of modules and interactions must provide a response by describing the actions to carry out from the data of sensors and the current internal state of the agent. An architecture covers techniques and algorithms supported by the methodology. Moreover, in (Kaelbing, 1991) the concept of architecture has been defined as:

> *A collection of hardware or software modules, which are usually represented by boxes and arrows, indicating the flow control between the different modules. A more abstract vision is: A general methodology to design the modular decomposition for specific tasks.*

In general terms, an architecture describes the main components of the system by defining the way in which they are related and interact to reach the proposed aim (Gil, 2010). Taking into consideration that an agent is a more or less complex system, its architecture describes the internal structure, the way in which it is deconstructed into a set of modules, and the way in which these modules interact with each other to determine the agent architecture (Mas, 2005). There are several proposed architectures which classify the agents defined within them (Wooldridge, 1999):

- *Logic-based agent:* The reasoning and decision making are carried out from logic and deduction (Genesereth and Nilsson, 1987) (Lesperance *et al*., 1996) (Fischer, 1994).
- *Reactive agent:* Decision making is carried out as a direct mapping from a situation to an action (Brooks, 1986) (Maes, 1990).

- *BDI Agents (Belief-Desire-Intention):* Decision making depends on the management of the belief, desire and intention representation by the agent (Bratman *et al*., 1988), (Rao, Georgeff, 1992).
- *Layer-based agents:* Decision making is carried out from different software layers, each one making reasoning on the environment for different abstraction levels (Ferguson, 1995).

One of the basic aspects to differentiate one architecture from another is the method of deconstructing the work into particular tasks. In this sense, it should be noted that scheduling is an area strongly linked to the agency. This area is focused on the study of mechanisms that make it possible to organize the running of actions, where an agent is nothing more than a system running actions in a determined environment (Corchado, 2000). The running of action plans is aimed at reaching the proposed goal. This way, the scheduling systems use symbolic models of knowledge representation and reasoning, designed for a search on a state space or plan space. The way in which these systems act is defined by the need of holding the basic aims to develop a plan for a complexity of space and time. The following classification provides three types of architectures, which are differentiated according to the reasoning model:

3.3 Multi-agent systems (MAS)

3.3.1 Basic concepts

In recent years there has been a growing interest in the decentralized approaches solving complex real-world problems. Many of these are within the area of distributed systems, where a number of entities work together to solve a problem in cooperative way. The combination of distributed systems and artificial intelligent is known as *distributed artificial intelligent (DAI).* DAI has been divided into two main areas: the first is the distributed resolution of problems, and is usually associated with the deconstruction and distribution of the resolution of a problem among multiple slave nodes and the collective construction of a solution to the problem. The second is *multi-agent systems (MAS),* which states the joint behavior of agents with a degree of autonomy and the complications resulting from their interactions (Panait, 2005). The main characteristic of these systems is that there is no system stated for global control and the data are arranged in a distributed way in favor of their asynchronous computation. This way, every agent can freely and dynamically decide which tasks it should carry out and who allocates them (Wooldridge, 2002). So we talk about a multi-agent system when two or more agents are able to jointly work in order to solve a problem (Mas, 2005). An important point of this is that for an agent association be considered a MAS, at least one of the agents should be autonomous, and at there should be a relationship between the two agents in which one of them meets the goals of the other.

A MAS extends the idea of a single agent and complements it with an infrastructure for interaction and communication. The problem can be stated as a goal that cannot be reached by a single subsystem, needing therefore, collaboration with other subsystems to obtain the solution (Cammarata *et al*., 1988). MAS are suitable to solving problems in which there are multiple resolution methods and/or entities able to work together to reach a solution (Chu-Carroll *et al*., 1995). Ideally, a MAS has the following characteristics (Huhns and Stephens, 1999):

- It is an open system with a non-centralized design.
- It contains autonomous, heterogeneous and distributed agents with different *personalities*.
- It provides an infrastructure to specify communication and interaction protocols.

Moreover, apart from the local goals of each agent, global goals are also stated by identifying a way in which all or some subgroups of agents compromise to reach the solution. Some advantages of this approach are (Abdelkader *et al.,* 2012):

- It is a natural way of controlling the complexity of highly distributed systems.
- It allows a construction of a scalable system since the addition of agents is simple task.
- A MAS builds a more robust and tolerant system with respect to failures than a centralized system.

3.3.2 Agent interaction

A MAS performs tasks of communication, coordination and negotiation. For the agents to be able to interact in a coherent way, they must share information on their goals and tasks. Thanks to the exchange of information, the agents coordinate the running of activities, being able to negotiate in case of conflict, and plan their actions to meet a goal (Rodríguez, 2010). In this context, four key and related concepts emerge by expressing different characteristics, namely: communication, coordination, cooperation, negotiation and adaptation.

1) Communication

An act of communication is defined as the exchanging of information between a sender and a receiver. The information is encoded into a language known by both the sender and receiver, and it is sent through a means of communication and for a determined context. (Finin *et al*., 1997) state:

> *The main block for an intelligent interaction is the knowledge sharing, including the mutual understanding of such knowledge and its communication. The importance of the communication is stated by (Genesereth and Ketchpel 1994), emphasizing that an entity is a software-agent if and only if it is able to properly communicate by using a communication language of agents. After all, it is difficult to represent cyberspace with entities that only exist in isolation; this would be contrary to our perception of an interconnected and decentralized electronic universe.*

When the interaction between agents is wide and there is a need to efficiently communicate with agents in other systems or organizations, the agents must have a

standard language with a set of conventions which allow them to communicate, connect and exchange information with other agents. The communication languages of agents, AC's, allow the agents to communicate in a clear and non-ambiguous way (Odell, 2010).

2) Coordination

(Malone, 1988) and (Malone and Crowston, 1994) describe the coordination of actions as a set of supplementary actions that can be carried out in a multi-agent environment to reach a goal that a single agent with the same goals could not otherwise achieve on its own.

From a practical point of view, coordination can be defined as the effort of managing the *interaction space* of a multi-agent system (Wegner, 1997) (Bussi *et al*., 2001). Coordination is strongly associated with planning since the plans allow predicting the behaviour of other agents and exchanging intermediate results leading to reach the final goal.

3) Cooperation

Cooperation is the mechanism whereby the agents working together to reach a common goal define a strategy to achieve this end (Rodríguez, 2010). The cooperative MAS are systems in which several agents attempt, through their interaction, to jointly solve tasks to maximize their usefulness (Panait and Luke, 2005). Ferber (1999) classified the cooperation methods into different categories where it is possible to use one or several methods at the same time to reach the best performance:

- *Grouping and multiplication.* Grouping of distinct agents in a single entity that performs in a coordinate way.
- *Communication.* In agent systems with representation of knowledge, the communication is stated by means of an exchange of messages between them.
- *Specialization*. Consists of the agent's adaptation to a very specific task.
- *Collaboration for the sharing of tasks and resources.* It should determine the way in which the tasks can be distributed among the agents.
- *Action coordination.* A set of tasks are managed to reach a common goal.
- *Resolution of conflicts by means of arbitrage and negotiation.* The agents should not be in conflict as to avoid damaging the performance of the whole system.

4) Negotiation

The negotiation process provides a model of the coordination between agents able to reach binding agreements. Negotiation allows reaching coordinated decisions by means of an explicit communication (Muller, 1996). A negotiation conflict is yielded when multiple agents attempt to reach an agreement. It is assumed that an agent has certain preferences on the possible agreements. The agents attempt to reach an agreement according to all parts (Vidal, 2010). Negotiation can be intended as the set of social rules imposing a standard behavior that must be met by the agents attempting to avoid the conflict (Castelfranchi *et al*., 1992).

Another important concept of MAS is its adaptive ability. Adaptation in MAS allows developing systems with the ability to reorganize, adapting to the changes of the environment and being able to evolve in runtime (Gil, 2011). MAS endow their agent-based dynamic organizations with mechanisms of self-adaptation by enabling them to operate in a changing environment (Weyns and Georgeff, 2010). (Rodríguez et al., 2009) propose a topology with four types of adaptive MAS in function of the interaction mechanisms used: mechanisms based on direct and indirect interaction, mechanisms based on the effort or cooperation.

3.4 Agent organizations

In the previous sections, the basic concepts necessary to understand the agent paradigm, agent interactions, their characteristics of planning and the concept of *agent society* have been given. This section focuses on the social characteristics of MAS. That is, essential concepts such as *virtual organization, social model, social structure,* standards, rules, etc., which help to understand the goal of this research. As adaptive coordination takes place within a society, it is necessary to review the possibilities, current studies and research leading to this end. To this end, a study of the agent-based social models was made, including how each of them carries out the coordination and adaptation. Is there a real analogy between human communities and agent-based organizations? While the answer to this question is being searched, the development of this kind of systems increases. There is a large number of research focused on the construction of virtual organizations, agent-based social simulation, the study of the behavior, etc. (Iglesias, 2010) (Sansores and Pavón, 2005). It is clear that we are heading towards a guild computational model. Due to this analogy, we can define *organization* from two points of view, namely, from the human and agent-based technology point of view. Human organizations can be defined as:

1. *Association of persons regulated by a set of rules according to specific purposes (Royal Academy of the Spanish language).*
2. *Social entity with a number of members that can be specified and an internal differentiation of the functions performed by such members* (Peiro, 1990).

Agent-based organizations, in turn, can be defined as:

3. *An organization that provides a suitable framework for the activity and interaction of agents through the definition of functions, expected behavior and authority relationships as the control (Gasser and Ishida, 1991).*
4. *The organization is a collection of functions keeping certain relationships between them, and taking place on interaction patterns with other functions in an institutionalized and systematic way (Zambonelli et al., 2003).*

The social models defend the MAS design inspired in social concepts and theories as rules, social conventions (or customs) or organizations. For this reason, the organizational structure is suitable to design mechanisms of coordination for MAS (Gasser *et al*., 1987) (Pattison *et al*., 1987). According to the computational paradigm,

the systems can be seen in natural way, in terms of entities (usually, agents) providing and consuming resources (Luck *et al*., 2008), that have likely been designed by different development teams, and can enter and leave an organization by different moments and due to different reasons. Moreover, the entities can form coalitions or organizations having the same goals. Current trends clearly lead to the paradigm of *virtual organizations* (VO) (Ferber *et al*., 2004).

A Virtual Organization is a set of individuals and entities needing to coordinate their resources and services within the institutional limits (Foster et al., 2001) (Boella et al., 2005). Therefore, a VO is an open system (Spencer, 1896) that has been built by grouping and collaboration of heterogenic entities, where there is a separation between the way and the role defined by its behavior. MAS technology, which allows the dynamic creation of agent-based organizations is particularly suitable for the development of this type of systems. The model of organizations based on open MAS not only makes the description of the structural composition of the system (for example, functions, agents, groups, tasks, plans or services), it also creates the rules to control the behavior of the agents, dynamic input/output of the components and dynamic creation of agent groups.

3.4.1 Organizational concepts

Continuing with the organizational perspective, a system is described by a social structure and a set of rules that state the interaction between agents. Such a description indentifies the functional components of the system (agents), its liabilities (the tasks to perform), resources (knowledge, software, hardware, tools, etc.) and the relationship between them (communication, allocation, etc.). All these organizational concepts are essential to understanding agent societies.

1. *Social entity*:

The organizations are formed by components or social entities which in turn can be composed of a specific number of members or agents. According to (Pattison et al., 1987), these entities:

- have liabilities; that is, a set of sub-tasks to perform since they have been included within the goals of the organization;
- have and consume resources. The components have certain resources on which their tasks are run. The resources required by a component depend of the function performed in that moment, within the organization;
- are structured according to determined patterns of communication;
- attempt to reach the overall goals of the organization;
- are regulated by rules and constraints.

2. *Structure*:

The entities in an organization are not independent of each other. They interact by passing information. These interactions are expressed as relationships between the components. In general terms, these relationships are not stated in an individual way in an organization, rather they require a conjunction of relationships between groups of entities. Such conjunctions define different

aspects to consider: functions, topology, and authority relationships, all of which determine the structure of an organization. The structure can be defined as the distribution, order and inter-relationship of the parts composing the organization. In such a structure, the agents will be ordered and communicate among themselves according to the topology defined by the structure. There are different topologies according to the type of organization: *hierarchies, holarchies* and *coalitions* (see Figure 3.3)*; groups* and *congregations* (see Figure 3.4)*; federations* and *matrix organizations* (see Figure 3.5).

- Hierarchies: The agents are ordered in a tree structure in which the lower levels have basic functionalities and the upper ones make decisions.
- Holarchies: They are hierarchical and nested structures of holons.
- Coalitions: They are temporal groupings to reach a concrete goal that is used to obtain certain benefits and reduce costs. The coalitions are removed when the goal is reached, when there is no longer a need for grouping, or when a critical number of agents leave the organization.
- Groups: Groupings of cooperative agents working together to reach a common goal. This way, the usefullness of the computer is maximized.
- Congregations: Agent groupings with complementary or similar characteristics. In this case, there is no specific goal, but they facilitate the search for suitable collaborators to reach global long term goals.
- Federations: Agent groupings with a representative. The remaining members of the organization interact only with the representative and give up part of their autonomy.
- Matrix organizations: In this topology an agent can be controlled by more than one supervisor. For this reason, it is necessary to have mechanisms of evaluation for compromising and resolving local conflicts.

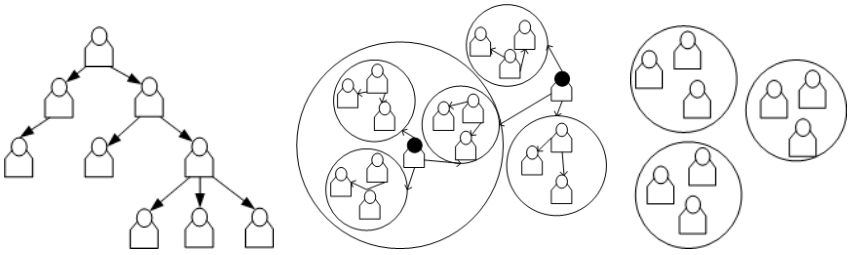

Figure 3.3: Organizational topology. From left to right, hierarchy, holarchy and coalition.

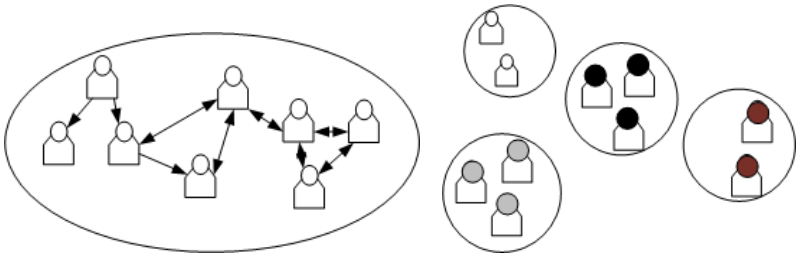

Figure 3.4: Organizational topology. From left to right, group and congregation.

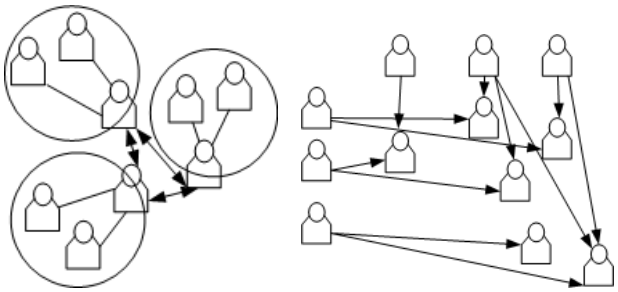

Figure 3.5: Organizational topology. From left to right, federation and matrix organization.

3. *Functionality*:

The functionality in an organization is determined by its mission; that is, the global goals describing its own existence. The mission defines the strategy, the functional requirements (what does the organization do?) and the interaction (how does it do it?). The goals can be classified as:

- Functional: for each group or organizational unit;
- Operative: for the agents, their plans (tasks to carry out)

4. *Norms*:

The norms define the consequences of the actions of the agents:

- Constraints on the organization.
- Liabilities and penalties to apply.
- External access control.
- Deconstruction: Actions to activate the norm, set of liabilities acquired by the agent, and actions performed to remove the liabilities.

5. *Environment*:

The environment defines what exists around the system: resources, applications, objects, constraint, stakeholders (supplier, clients, and beneficiaries). By defining the environment, the relationship of the roles is stated with respect to the elements of such an environment: access mode (reading, interaction, and information extraction), access permission, etc.

6. Dynamicity:

The organization dynamics are related to the input/output of agents, with the roles they adopt, the creation of groups and the control of their behavior. When defining the dynamics of an organization, the following must be specified (Esteva, 2003): how the agents enter the systems, the adoption of roles, the dynamic creation of agents, and behavior control.

7. *Social adaptation:*
Adaption in a society is an ability to interact with the environment by creating a symbiotic state. Adaption is not only an ability, it is also a need of becoming involved in the environment to maximize the learning needs of each individual so that the system can obtain meaningful learning.

8. *Social learning:*
Social learning is a process in which, as a result of a common environment (provided by an artificial society), different entities can interact and evaluate their experiences and information (Duong and Grefenstette, 2005). Every member of the artificial society considers the other as a simple data source, where the relevance of the found data is defined on the utility function or the goal proposed by the entity that is learning. In this way, the entity has the ability to decide what to learn (Conte and Paolucci, 2001). The two most relevant types of learning in the society are: *social facilitation* (Mataric, 1997) and *imitative learning* (Conte and Paolucci, 2001).

Learning is intrinsically related to adaptation. Both social facilitation and imitation are techniques that allow us to learn from new situations to have an action plan for the future. Ultimately, that can be considered an adaptation process to new situations. Furthermore, there are different approaches of society adaptation and in them; adaptation is one of the most used.

3.4.2 Architectures

Virtual organizations are considered open systems formed by the grouping and collaboration of heterogeneous entities, where there is a clear separation between structure and functionality (Foster *et al*., 2001) (Boella *et al*., 2005). This way, we can find works focused on the development of new methodologies and procedures of design on the organization aspects of MAS as *Gaia (Zambonelli et al., 2003), AGR (*Ferber *et al.,* 2004*),* MOISE (Hubner, 2004), OperA (Dignum, 2004) (based on ISLANDER (Esteva, 2003) framework), Tropos (Bresciani *et al*., 2004), PASSI (Cossentino, 2005), SODA (Molesini *et al*., 2006) MenSA (Ali *et al*., 2008) , O-MASE (DeLoach, 2009), INGENIAS (Pavón *et al*., 2005) and VOM (Criado *et al*., 2009). Many recent studies are not only focused on the use of organizational structures during the design process, but are also interested in the regulation and adaptation of open MAS.

Although some platforms face the organizational concepts by means of design patterns and similar techniques, most of them cannot be directly applied to the development of open MAS where the organizational structures can emerge

dynamically and change at runtime. The main problem in implementing a virtual organization is the lack of a platform giving support to these systems. The primary function of an agent-based platform is to offer a running environment for the agents. In recent years, there have been research approaches attempting to provide an environment for these systems. Some examples of agent-based platforms are JADE (JADE, 2012) , FIPA-OS (FIPA, 2012), RETSINA (Giampapa and Sycara, 2002) Grasshopper (Baumer *et al*., 2000), Jack (Howden, 2001), ZEUS (Hyancinth *et al*., 1999), MadKit (Gutknecht and Ferber, 1997), EIDE (Esteva, 2003), RICA-J (Serrano and Ossowski, 2004), S-Moise+ (Hubner *et al*., 2006), Jack Teams (una extensión de JACK) (Agent-Oriented-Software, 2004), SIMBA (Carrascosa *et al*., 2003) y SPADE (Escriva *et al*., 2006). Additionally, a comparative study can be found in (Argente *et al*., 2004).

The above characteristics along with an environment specially designed for running virtual environments are presented in the THOMAS architecture *MeTHods, Techniques and Tools for Open Multi- Agent Systems* (Carrascosa *et al*., 2009) (Giret *et al.*, 2009). THOMAS is the architecture used in this research and on which the adaptive model of the proposed virtual organization was developed.

3.4.3 THOMAS

THOMAS (Thomas, 2010) emerged from the need to support the development of an architecture with the characteristics previously exposed in order to develop open MAS from an organizational point of view (GTI-IA, 2009). This architecture presents the necessary infrastructure to use the concepts of agent-based technology in the development process by applying deconstruction techniques, abstraction and organization. THOMAS proposes a model of this research that can evaluate the behavior of a virtual organization in such a way that the agents can dynamically adapt and reorganize.

This architecture is basically formed by a set of modularly structured services. THOMAS is based on FIPA (FIPA, 2012a) architecture. It expands its abilities with respect to the organizational design and improves the ability of the services. In THOMAS there is a service with the singular goal of managing the organizations introduced in the architecture. The FIPA Directory Facilitator is redefined to be able to deal with the services in a more elaborate way.

The agents take part in the infrastructure offered in THOMAS by means of a series of services included in what is referred to as OMS (Organization Manager Service). We can see the main components of THOMAS in Figure 3.6 (GTI-IA, 2009).

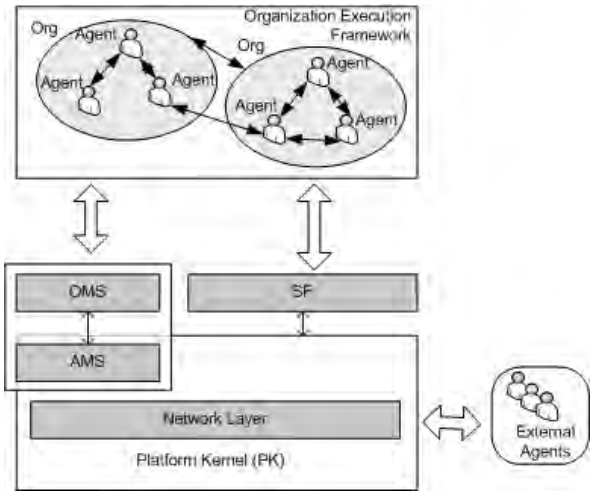

Figure 3.6: Representation of the THOMAS architecture.

As shown in this figure, there are three main components of THOMAS:

1. *Service Facilitator (SF):* This component offers both simple and complex services for the active agents and organizations. Basically, its functionality is summarized as the ability to provide a directory of yellow and green pages for the available services.

2. *Organization Manager Service (OMS):* It is primarily responsible for the management of organizations and entities that it includes. Therefore, it allows the creation and management of the life cycle of an organization.

3. *Platform Kernel (PK):* It allows keeping the basic services in the management of an agent-based platform. It is responsible for managing the life cycle of the agents present in the organizations and also works as a communication channel (by implementing mechanisms of message transport), thus facilitating the interaction between the various entities. The necessary PK services in the THOMAS architecture can be classified into four groups. *Registration:* necessary services to add, modify, remove the native agents in the platform; *discovery:* service providing the functionality of obtaining information; *management:* services to control the stage of activation of the native agents in the platform and *communication:* services for the communication between agents, inside and outside of the platform.

From a global point of view, the THOMAS architecture states a full integration allowing agents to offer and invoke services in a transparent way towards other agents or entities. This allows the external entities to interact with the agents in the architecture by using the provided services. Table 3.1 shows a summary of the services offered by THOMAS.

SF Services		
Type	**SF Service**	**Description**
Registration	Register Profile	Creates a new service description
	RegisterProcess	Creates a particular implementation (process)
	ModifyProfile	for a service
	ModifyProcess	Modifies an existing service profile
	DeregisterProfile	Modifies an existing service process
	DeregisterProcess	Removes a service description
		Removes a service process
Affordability	AddProvider	Adds a new provider to an existing service process
	RemoveProvider	Removes a provider from a service process
Discovery	SearchService	Searches for a service that satisfies the user
	GetProfile	requirements
	GetProcess	Gets the description (profile) of a specific a service
		Gets the implementation (process) of a specific
		a service
OMS Services		
Type	**OMS Service**	**Description**
Structural	RegisterRole	Creates a new role within a unit
	RegisterNorm	Includes a new norm within a unit
	RegisterUnit	RegistrationCreates a new unit within a specific
	DeregisterRole	organization
	DeregisterNorm	Removes a specific role description from a unit
	DeregisterUnit	Removes a specific norm description
		Removes a unit from an organization
Information	InformAgentRole	Indicates roles adopted by an agent
	InformMembers	Indicates entities that are members of a specific unit
	QuantityMembers	Provides the number of current members of a
	InformUnit	specific unit
	InformUnitRoles	Provides unit description
	InformRoleProfiles	
	Infor, RoleNorms	Indicates which roles are the ones defined within a specific unit
		Indicates all profiles associated to a specific role
		Provides all norms addressed to a specific role
Dynamic	AcquireRole	Requests the adoption of a specific role within a unit
	LeaveRole	Requests to leave a role
	Expulse	
		Forces an agent to leave a specific role

Table 3.1: Summary of the services given in THOMAS.

As for the virtual organizations, all agents included in the framework should belong to an organization. The THOMAS framework provides a virtual organization where any entity can automatically be included, and a general function allowing an entity to request descriptions of the services in order to meet its needs. By the service description, the client is notified on the roles needed to request a specific service, or the roles needed to provide a specific service to the organization.

3.4.4 Coordination

Every organization needs support of coordination by explicitly determining how the organization should be structured and how its actions and tasks are carried out. In general, in every MAS, the agents represent the subjects whose activities need to be coordinated; the entities, where dependencies emerge, are goals, actions and plans. A mechanism of coordination determines the way in which one or more agents carry out a task (Ossowski, 1998).

In this research the organizations are composed by *organizational units* that provide communication and visibility constraints on the agents (based on THOMAS). These organizational units can be of three types: *hierarchy*, *team* or *flat*. In a hierarchy, a supervisor-agent has the control over remaining members, coordinates tasks, and centralizes decision making. In the teams, all members collaborate to reach a common goal by sharing their information. The coordination emerges through plans and coordinated decision made by the members. Finally, in the flat units no single member has control over another. This way, the members can know the existence of the remaining members in the structure. This last unit is mainly used to model more complex structures. By using the concept of organizational unit, it is possible to build more elaborate and complex organizational structures such as a as matrix, federation, coalition or congregation structures previously presented.

From a practical perspective, it is better to consider coordination as the effort of managing the space of interaction in a MAS (Wegner, 1997) (Busi *et al*., 2001). This coordination is related to the action planning for task resolution, since these plans allow:

- Predicting the behavior of other agents in the system.
- Exchanging intermediate results to develop the progress of the global task resolution.
- Avoiding redundant actions, if they are not desirable.

Examples of coordination models used as sets of intentions (Cohen and Levesque, 1991), (Dunin-Keplicz and Verbrugge, 2002), shared plans (Grosz and Kraus, 1996), and the models of independent teams of the domain (Tambe, 1997). All approaches are based on observations of human work teams. The approach proposed by the theory of shared plans is the most similar to the model that we propose in this document. The formalization of the shared plans states the need of a common team model to a high level, allowing the agents to understand all plan requirements carried out by a system, such as the group goal. This allows the members of the team to activate their abilities to carry out the plan and reach the global goal.

The current trend is to implement *multi-centric* mediated coordination (Ossowski, 1998) (Ossowski, 2001), partly centralized and partly decentralized, based on multiple and heterogeneous intermediate agents. This way, both the efficiency engineering needs and the restrictions imposed by increasingly open environments can be reached. We shall use the model proposed in this PhD Thesis as an example:

- *Problem-oriented coordination:* In this type of coordination, the agents should coordinate the plans to carry out the actions to prevent deadlocks, repetition of actions and creation of inconsistencies.

- *Cooperation-oriented coordination:* The agents are not coordinated at a plan level; rather at an action level. This means that the agents are coordinated at the time of running the action.

Choosing the coordination model depends of the scope of the organization itself, but there should always be an attempt to cover all performance possibilities of the agents. To this end, the coordination model for organizations proposed in this research covers all coordination types, since the organization globally determines the actions of the agents (global coordination), but is able to decide how to solve their problems (individual coordination).

This research presents an approximation to carry out a global coordination within an organization developed by THOMAS, which is an architecture based on organizational concepts, as previously seen. The coordination consists of the distributed planning of tasks on the agent members of the organization. Moreover, such coordination can adapt to changes in the plans of the organization, which provides the property of *adaptation*. The description of such a property in the organizations is provided in the next section.

3.4.5 Adaptation

There are several problems that must be taken into account when agents are coordinated in an organization, including how to keep a global coherence of the system without explicit global control (Huhns and Stephens, 1999). For an organization to be able to adapt quickly to changes in its environment; the agents should coordinate when it is necessary to carry out changes related to their goals or assigned roles. In short, it is necessary for an organization to be able to adapt.

Therefore, a virtual organization can be seen as a cooperative system, in which coordination is based on a planning and distribution of tasks. The coordination of shared tasks or tasks that are combined to solve a common problem, requires a centralized planning or a distributed planning carried out by the agents themselves in the system. All this defines an open problem, since the classic planning systems (McAllester and Rosenblitt, 1991) are not suitable for several reasons:

- They assume that the agents have full knowledge of their environment;
- They assume that the actions will not fail; and
- They assume that the environment will only change by running the actions of the agents.

None of these are realistic assumptions in a MAS. Thus, it is necessary to have a plan capable of adapting to these circumstances. The proposed model attempts to reach an adaptive planning state within the agent organization without taking into account the previous assumptions, and it can be applied to real environment. There are different paradigms of planning, but the most suitable for reaching the above goal is case-based planning, since it provides the flexibility needed to carry out a re-planning of tasks and actions related to the goals of the organization. This flexibility comes from the possibility of using plans (or sub-plans) based on past experiences. In addition, if this type of planning is combined with a satisfactory plan, then it is possible to introduce temporal and resource constraints to the problem.

Case-based planning allows the system to adapt from the dynamic planning point of view. That is, it allows re-planning concepts to be formalized by means of techniques

based on the fields of Jacobi (Lee, 1997). Other important questions in the adaptation are: what should be adapted, how should be adapted and who is responsible for the adaptation. Such questions are determined by the goal that the organization should or wants to fulfill, a factor widely influenced by life in a society. This way, the utility or efficiency functions determine the decision making process that can be applied by an individual or collective group.

The adaptation model of the organization proposed in this research uses mechanisms based on direct interactions (Zambonelli *et al*. 2004): local interactions and computations to re-organize the agents and obtain a global and coherent state of the system. Furthermore, the use of the THOMAS platform (Carrascosa *et al*., 2009) brings us new methods of adaptation based on architectures (Razavi *et al*., 2005) in which it is possible to modify the structure of the organization to obtain an adaptation to the changes.

Taking advantage of the development of MAS from an organizational point of view, and considering the currently existing gaps regarding adaptive planning on a social model, we propose a coordination model for the dynamic and adaptive planning in an agent-based organization. By means of MAS technology and the planning techniques based on operational research, we are interested in an optimum distribution of tasks for the agents of the organization. It deals with a single model which can provide an organization with self-adaptive abilities at runtime on high-dynamic environments. This will allow the behavior of an agent to be determined by the goals it wants to reach, but also takes into account the goals of the remaining agents and the changes in the environment.

3.5 Ontologies

Communication implicitly presents a series of problems such as language inconsistency, different contexts, ambiguities, etc. The idea, then, is to establish a shared understanding to solve the problems of communication based on (Uschold and Gruninger, 1996):

- interpersonal communication on different contexts and points of view;
- interoperability between systems;
- knowledge interoperability ;
- facilitating of the requirement specification process;
- allowing communication and understanding among software agents.

From this context emerges the concept of ontology, immersed in multiple disciplines and with disparate meanings. The term ontology in philosophy refers to the study of the things which exist (Chandrasekaran and Josephson, 1999). The ontology concept in *artificial intelligence* is attributed to the specification of a conceptualization (it includes the definition of terms and the relationships between them), which should preferably be formal and computable (Hendler, 2001). Ontologies represent essential technologies that enable and facilitate semantic interoperability by providing a formal conceptualization of the information being shared and reused (d'Aquin and Noy, 2011). They provide a way of representing and sharing knowledge by using a common

vocabulary. Furthermore, they allow using a format of knowledge exchange and reusing such knowledge (Cantera *et al*., 2007). An ontology typically provides the vocabulary describing an application domain and the specific meaning of the terms used by the vocabulary (Euzenat and Shvaiko, 2007).

Ontologies are successfully being used in areas as *web semantics,* where a suitable design of the ontology can improve the performance of the semantic web services. Another area in which ontologies are widely used is multi-agent systems. Since ontologies describe a set of concepts and relationships, they can be used to build the hierarchical architecture of business knowledge and the logic of the negotiations and activities regulation between agents (Wang et al., 2012).

3.5.1 Basic aspects

Due to the multidisciplinary environment where the ontology concept is found (including computer sciences), there is no agreement as to its definition. However, different studies on ontologies (including taxonomy, conceptual maps, conceptual models and formal ontologies given in several logical languages) have allowed ontology to develop without a common understanding of its definition, implementation and applications (Gruninger *et al*., 2007). One of the most accepted definitions was given in (Borst, 1997):

An ontology is a formal and explicit specification of a shared conceptualization.

This definition introduces the term *conceptualization*, which refers to an abstract model of a constructed phenomenon by indicating the important concepts of such a phenomenon. *Formal* indicates that it is computable, which excludes natural languages. Furthermore, the goal is to represent knowledge in such a way that it can be used by various individuals. Other definitions of ontology are given in (Gruber, 1995) (Cantera *et al*., 2007) (Swartout *et al*., 1997) (Sowa, 2000) (Noy and McGuinness, 2001) (Farquhar, 1997).

An ontology must include at least a concept hierarchy (see example in Figure 3.7) organized by the semantic relationships which specify the relationship between one concept and another. Ontologies are mainly used to carry out a search and retrieve of conceptual/semantic information in a more efficient, adapted and intelligent way. Indeed, an ontology allows us to work with complete concepts rather than keywords (White, 2004).

Ontologies represent the essential technology that enables and facilitates semantic interoperability by providing a formal conceptualization of the shared and reused information (d'Aquin and Noy, 2011). Despite the differences existing within the different areas covering ontologies; there is a general agreement on certain questions (Chandrasekaran and Josephson, 1999), for example:

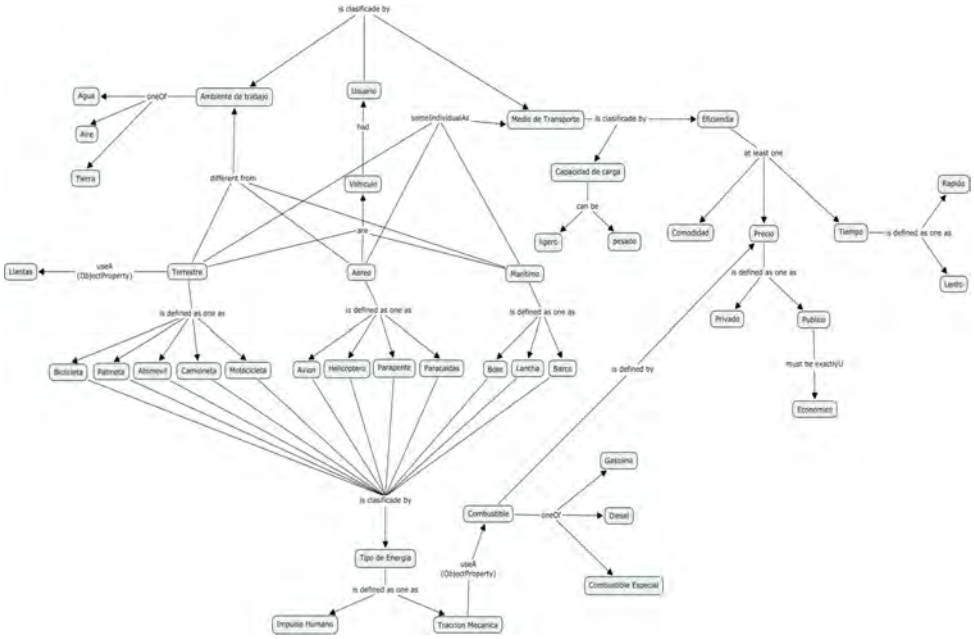

Figure 3.7: Example of knowledge hierarchy in an ontology.

*There are **objects** in the World. The objects have **properties** or **attributes** having certain values. Objects may have several **relationships** with other objects. The relationships and properties can change over **time**. There are events occurring in a specific **instant of time**. The World and its objects can be in different **states**. Events can **cause** other events or states, which are known as the **effects**. The objects may be composed of **parts**.*

Several authors and scholars have agreed on the different components of an ontology (Sowa, 2000) (Noy and McGuinness, 2001) (Farquhar, 1997). These are *axioms, class, instances, relations, properties or slots, frame, conceptualization, taxonomy* and a *vocabulary*. A distinction between different ontologies can be given depending on the complexity of its construction.

- *Reference ontologies:* Rich and axiomatic theories that focus on clarifying the intended meaning of the terms used in specific domains (Borgo *et al*., 2002).
- *Application ontologies:* Provide a minimal terminological structure that fits the needs of a specific community (Menzel, 2003).

Considering the amount and type of conceptualization the following classification (van Heijst *et al*., 1997) is obtained:

1. *Terminological ontologies.* They specify the terms that are used to represent knowledge in the universe of discourse. They are often used to unify vocabulary in a given field.
2. *Information ontologies:* Specify the storage structure of databases. They provide a framework for storing standardized information.

3. *Knowledge modelling ontologies:* Specify conceptualizations of knowledge. They contain a rich internal structure and often fit the particular use of the knowledge they describe.

3.5.2 Descriptive logic-based representation paradigm

As explained in the above point, ontologies are used in search systems to retrieve conceptual/semantic information as efficiently and intelligently as possible. The required information will be obtained by means of a question-answer process, replacing the search system based on keywords. Such a process would need a procedure of computable reasoning. The best candidate to add the reasoning is *descriptive logic,* which offers a high level language to express knowledge with a high expressive power, allowing inference tasks.

By definition, descriptive logic is a logical formalism designed to represent knowledge. This logic defines concepts by providing a formal syntax and semantic. These characteristics allow avoiding ambiguities in the language. The kinds of components allowing descriptive logic to perform ontologies are *concepts*, *roles* and *individuals.*

- *Concepts:* Represent entities in which certain information will be stored.
- *Roles:* They are relationships between concepts of the domain, and allow their properties to be described.
- *Individuals*: They are instances within the ontology as well as the specific values of the roles represented by the properties of the individuals.

Finally, according to (Uschold and Gruninger, 1996), the application area of ontologies can be divided into three sections: Communication, Interoperability and Systems Engineering.

3.5.3 Defining ontologies

There are several methodologies to build ontologies. To define an ontology an in-depth and consistent analysis should be made of how to avoid inconsistencies or incoherencies (Chandrasekaran *et al*., 1999). Regardless of how an ontology is created, there is a series of general guidelines of design (Gruninger and Fox, 1995): *clarity, consistency, extensibility, minimum ontological commitment, minimum dependence of the encoding.* One of the most accepted methodologies for creating ontologies is given in (Noy and McGuinnes, 2001). This methodology is based on the following steps:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of the classes.
6. Define the characteristics of the properties.
7. Create the instances.

Several methods and methodologies f developing ontologies, as defined in method Ontology 101 (Noy and McGuinness, 2001) and Methontology (Gomez and Rojas, 1999), include the reuse step in the life cycle of developing ontologies. This allows ontological engineers to integrate existing ontologies by avoiding the design and implementation of an ontology that has already been created. Reuse of ontologies usually takes place in the design and implementation stages.

3.5.4 Languages for building ontologies

Although ontologies have been used for many years in different fields of research, ever since the emergence of the semantic web, ontologies have become the de facto standard for knowledge representation (García, 2007). The first languages for representing ontologies with certain relevance were KIF (Knowledge Interchange Format), OCML (Operational Conceptual Modellig Language) and F-Logic (Frame Logic). However, with the emergence of the semantic web, the most widely accepted ontological languages have been those with which the semantic web is associated. We present two of them, which are the most significant, RDF and OWL.

Lenguaje RDF

Language RDF (Resource Description Framework) is not based on any logical language; it only takes the syntax of the ontologies (classes, instances and properties that are related to each other). Various authors do not consider RDF as an ontology language due to its lack of inferences, but it is the basis for the vast majority of languages and for representing ontologies graphically (Romero, 2007). RDF is the model recommended by W3C (World Wide Web Consortium) to represent metadata by allowing web information to be defined on any domain and represent any type of arbitrary data. This language, along with other technologies, will add meaning to the content of web pages and other web-based technologies. Furthermore, it will be extremely useful when the information is processed by applications instead of persons. The data model of RDF is based on the known triplets that are commonly expressed as A (O, V). The elements in a triplet (García, 2012) are:

- Object: Resource or concept to represent. Each having a single and universal Identifier (URI).
- Attribute: Aspect or property of the object to represent. The relationships of an object with other objects or values can be defined.
- Value: Value or attribute of the property.

The triplets allow nesting and chaining between annotations, since the object of one triplet can take the role of the value in another. For the RDF data model, an XML-based syntax has been proposed, where a description is a set of assertions about a resource. Ultimately, RDF provides extensibility and the ability to link resources based on its content. However, it has problems of ambiguity in the definition of its elements because the recommendations to represent the semantic content or the vocabulary to be used are not defined. This can lead to two documents using different definitions of the same concept but unable to decide if they are equivalent (Ruckhaus, 2006).

OWL

The web ontology language (Web Ontology Language, OWL) appeared in 2004, addressing the need of a language that meets certain characteristics:

- Extend existing web standards such as XML, RDF or RDFS.
- Easy to use; that is, based on common terms of knowledge representation.
- Sufficient expressiveness to store the semantic content necessary to enable the development of the semantic web.
- The language should be formally specified, allowing machine learning.

OWL can be considered the most complete language for the semantic web, as it provides a common vocabulary and semantic content representation. OWL is presented in three levels according to its expressiveness (Romero, 2007). This allows us to choose the language that best fits the needs of each system: the simplest systems limit their expressiveness by looking for a more effective calculation.

- *OWL Lite* is the most basic sublanguage and therefore the simplest, providing support to build hierarchies and the use of simple constructs (García, 2012).
- *OWL DL* is the sub-language that allows maximum expressiveness by ensuring computational completeness, that is, all conclusions are computable and implemented in a finite time (García, 2012).
- *OWL Full* is the sub-language that implements maximum functionality without restrictions of any type, being used in very specific situations (Romero, 2007).

The following Table 3.1 shows the equivalence between statements in the OWL language and the syntax of descriptive logic (DL).

Statement OWL	Syntax DL
intersectionOf	C_1⊓…⊓C_n
unionOf	C_1⊔…⊔C_n
complementOf	¬C

oneOf	⟦{x}⟧_1⊔…⊔⟦{x}⟧_n
allValuesFrom	∀P.C
someValuesFrom	∃P.C
maxCardinality	≤nP
minCardinality	≥nP
subClassOf	C_1⊑C_2
equivalentClass	C_1≡C_2
disjointWith	C_1≡⟦¬C⟧_2
sameIndividualAs	⟦{x}⟧_1≡⟦{x}⟧_2
differentFrom	⟦{x}⟧_1⊑¬⟦{x}⟧_2
subPropertyOf	P_1⊑P_2
equivalentProperty	P_1≡P_2

inverseOf	P_1≡P_2^-
transitiveProperty	P^+⊑P
functionalProperty	T⊑≤1P
inverseFunctionalProperty	T⊑≤1P^{-}

Table 3.1: Equivalence between language OWL and descriptive logic.

3.6 Conclusions

The goal of this research is to design a methodology that facilitates the semiautomatic reuse of business processes in a cloud computing environment. Automation in this process requires the use of efficient and intelligent mechanisms, and a standardization process. Intelligent management will take place in a distributed architecture of agent-based organizations such as the one presented in this chapter. The management of the creation and reuse of a business processes is generally more efficient if applied to similar processes and/or if they have a common semantic (that is, the use of a common ontology). The proposed process has been guided by a virtual organization managed by a multi-agent architecture that implements intelligent behavior to manage the process by using an ontology.

The goal of this research is to present a model allowing the construction of a business process from specifications given in text format (with certain constraints) of the process to be created. The multi-agent system based on virtual organizations will have the necessary level of knowledge and intelligence to establish the composition of processes by using standard BPEL (Business Process Execution Language). This standard can in turn compose Web services in a simple way and with the advantage of being able to project directly to diagrams BPMN (Business Process Management Notation). The next topic will review the BPEL standard and the related items as well as the web services to be used in the composition of business processes.

4 BUSINESS PROCESSES

4.1 Introduction

The goal of this research is to develop a methodology facilitating the agile and efficient construction of business processes on cloud environments from developed components. This goal involves the construction of business processes from other ones implemented in form of Web services. This process is guided by an agent virtual organization coupled in a multi-agent architecture that implements the intelligent behavior necessary to manage such a process by basing on an ontology. In the previous chapter, the multi-agent systems (MAS) that will be applied to our model have been analyzed and an ontological model has also been described. Such a MAS based on virtual organizations will facilitate the process composition by using standard BPEL, which in turn facilitates the process of composing Web services with the advantage of having a direct projection from BPMN diagrams. This chapter presents Web service technology, the SEO architecture, the concept of business process management and finally, semantic Web services will be analyzed.

4.2 WEB services

The Web has gone from being a collection of pages to a collection of services (Paolucci et al., 2002). For many years, companies have interacted using ad hoc approaches that take advantage of the basic infrastructure of the Internet. However, in recent years Web services have increased in importance, providing a systematic and extensible framework for the interaction between application-application. This framework was built from existing Web protocols and based on XML standards (Curbera *et al.*, 2002). This working style has changed the way software systems are conceiving distributed software systems (Newcomer, 2002).

A Web service is a software component that represents a service deployed on the Web platform, and supports automatic interactions between machines in the network (Walsh, 2002). In addition, they must be described so that they can be discovered, associated or composed (Le *et al*., 2009). To support the Web services approach, many new languages, most XML-based languages have been designed as business coordination languages (WS-BPEL, (OASIS, 2007)), description languages (WSDL (Curbera *et al*., 2002)) and query languages (Xpath, (Clark and DeRose, 1999)) (Lapadula *et al*., 2010). When most software and processes are supported by Web services, new types of business paradigms, discussion groups, interactive forums, and models of publishing will emerge to take advantage of this technology (Newcomer, 2002). Furthermore, the composition of Web services has emerged as a promising approach to integrating business applications within organizational boundaries (Zeng et al., 2003).

Service-oriented computing is being considered as the next generation of distributed computing, being widely adopted. SOA (Service Oriented Architecture) aims at implementing service-oriented computing by using Web services as the main block of the applications (Besson *et al*., 2011). The three key elements associated with SOA are: WSDL (Web Service Description Language), which describes the Web

service; SOAP (Simple Object Access Protocol), which is a transport protocol for the exchange of information; and UDDI (Universal Description, Discovery and Integration), which is a register used to store the service and its discovery (Parimala and Saini, 2011).

SOAP offers the basic communication for Web services, but does not tell us what messages need to be exchanged to interact successfully with a service. This will be completed by WSDL, an XML format developed by IBM and Microsoft to describe Web services as collections of communication points allowing the exchange of certain messages (Curbera *et al*., 2002). WSDL describes a Web service interface, consisting of messages exchanged between the client and the server. Such messages are abstractly described, and are linked to a specific network protocol and message format. Web service definitions can be mapped to any implementation language, platform, object model, or messaging system (Tere and Jadhav, 2012). Web service discovery systems were developed to search for a suitable Web service from a large number of published Web services (Le *et al*., 2009). UDDI appears as an acceptable means of listing and publishing Web services.

Another advantage of SOA is its close relationship with cloud systems. Applications in cloud systems need to be flexible; the adoption of SOA can provide *cloud computing* developments based on a design to access services through low coupling, and easy movements that would otherwise be very complex (Arévalo, 2011). In addition, cloud architectures based on Web services have been proposed (SOCCA, Service-Oriented Cloud Computing Architecture), so that cloud systems can interact with each other (Wei-tek *et al*., 2010).

4.2.1 Basic concepts

There are several complementary definitions of the concept of Web service, depending on the author or organization. One such definition is provided below:

> *Web services are self-contained, modular and dynamic applications, that can be described, published, localized or invoked through the network to create products, processes and supply chains. These applications can be local, distributed or Web based. Web services are built on standards such as TCP/IP, HTTP, Java, HTML and XML (IBM, 2009).*

An important benefit of Web services technology is the ability to integrate existing light applications over public or private networks such as the Internet. New applications can be developed only using the composition of several Web services (Lee and Hwang, 2009). Web services have three basic features (Shooting and Foxvog, 2006):

- Functional: indicates what the service does.
- Behavior: details how the Web service works and how it can be integrated.
- Non-functional: restricts the functional properties, which are given by the user for service discovery.

The Web service architecture varies considerably from one organization to another. Figures 4.1 and 4.2 show the different components of a Web service-based architecture:

- UDDI (Universal, Description, Discovery and Integration)
- WSDL (Web Service Description Language)
- SOAP (Simple Object Access Protocol)

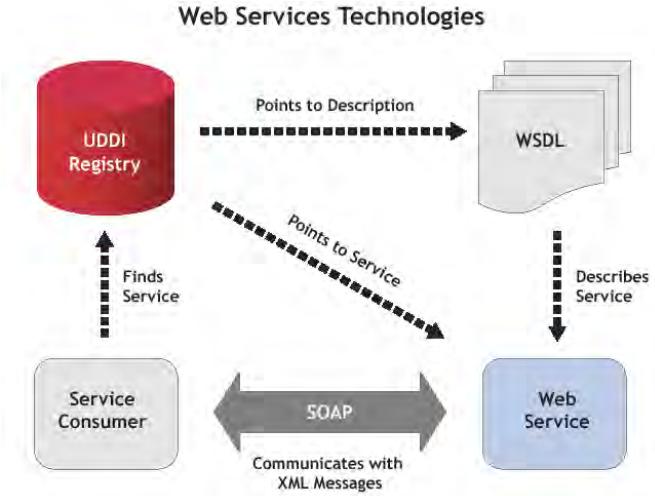

Figure 4.1: List of related Web service components.

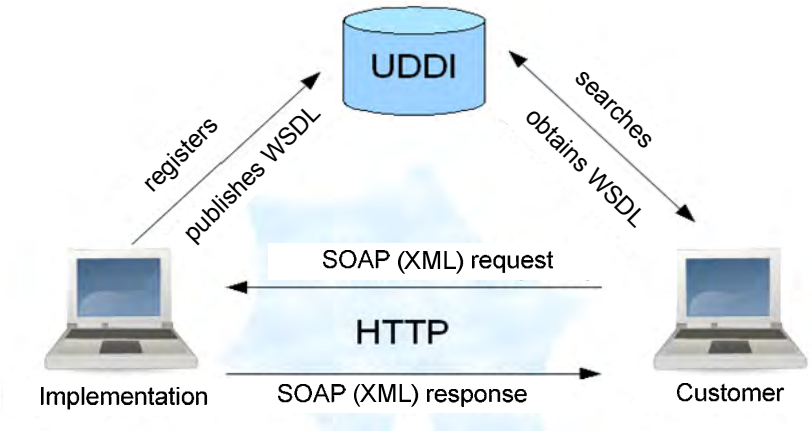

Figure: 4.2: Web service architecture (GramaticasFormales, 2011).

The programmer builds a Web service by using a specific programming language. This service is published by using a WSDL interface and can be invoked by a client using this interface. Web services are presented to customers as a set of operations that provide business logic on behalf of the provider, enabling customers to invoke operations on the server's side.

A different approach of Web services is one that includes semantic content. A semantic *Web* service, which is an extension of the traditional concept of Web service, overcomes the limitations of Web services by using the knowledge representation that provides the semantic Web. In particular, it uses ontology to describe Web services (Le *et al*., 2009), (Kennedy *et al*., 2012).

Another area improving the efficiency of Web services is multi-agent systems (MAS). A MAS can be used to assist in the task of Web service composition (Rao,

2012). During service composition, software agents engage in conversations with their peers to agree on the *Web* services taking place in the process (Maamar *et al*., 2005). Several proposals for handling this problem can be found, such as those given in (Maamar *et al.*, 2005) and (Greenwood and Calisti, 2004).

(i) SOAP (Simple Object Access Protocol)

SOAP is an XML-based protocol for messaging and invoking remote procedures (RPC). Instead of defining a new protocol, SOAP performs on existing protocols such as HTTP, SMTP and MQSeries (Curbera *et al*., 2002). SOAP enables interoperability between a wide range of programs and platforms. In this sense, the existing applications can be accessed by a wider range of users (Papazouglu, 2008). The structure of a SOAP message is composed of the following tags: *Envelope, Header, Body* and *Fault.*

In addition to the basic structure of the messages, the SOAP specification defines a model that indicates how recipients should process the SOAP messages. The message model also includes actors indicating who should process the message. A message can identify the actors that indicate a series of intermediaries who in turn process the message parts specified for them and pass the rest to others (Papazouglu, 2008). The *Web* service communication model describes how to invoke *Web* services and is based on SOAP, which is defined from its communication and coding style. SOAP supports two communication styles (Albreshne et al., 2009): *Web services styles RPC* and *document style.* The codes for both styles have been given below, Codes 4.1 and 4.2.

```
<Envelope xmlns="http://www.w3.org/2001/12/soap-envelope"
        <Header>
        ...
        </Header>
        <Body>
                <GetProductPrice>
                        <product-id>4562</product-id>
                </GetProductPrice>
        </Body>
</Envelope>
```

Code 4.1: SOAP styles RPC, (Albreshne, 2009).

```
<soap: Envelope xmlns:SOAP=http://www.w3.org/2001/12/soap-envelope>
  <soap:Body>
    <pourchaseOrder orderDate=”2009-05-
20”xmlnso=http://www.amzon.com/POs>
      <po:accountName>Ricard</po:accountName>
      <po:accountNumber>1234</po:accountNumber>
       <po:book>
          <po:title>J2EE Web services</po:title>
        <po:quantity>300</po:quantity>
        <po:price>24.5</po:price>
```

```
10          </p:book>
11       </pourchaseOrder>
12    </soap:Body>
13 </soap:Envelope>
```

Code 4.2: SOAP styles document, (Albreshne, 2009).

Web service invocation through SOAP requires a SOAP engine (that is, a Web service engine). The basis of some popular engines such as *Apache Axis 2* (http://ws.apache.org/axis2/) is an Internet domain where time requirements are not very important (Mathes *et al.*, 2009).

(2) WSDL (Web Service Description Language)

WSDL is an XML document describing the access to a Web service and the offered operations (Walsh, 2002). It defines an abstract description with respect to the messages exchanged on a service interaction. During the development, the developers use WSDL as input to a proxy generator that produces the code of the client according to the requirements of the service (Curbera *et al.*, 2002). Informally, we can understand that WSDL describes a *Web* service similarly to an interface (Li *et al.*, 2006). With WSDL, a client can locate a *Web* service and invoke any of its public functions (Tere and Jadhav, 2012).

A WSDL document defines the services as a collection of endpoints[1] on the network or ports. The abstract definition of endpoints and messages in WSDL is separated from concrete network deployment or data link format. This allows reusing abstract definitions: *Messages*, which are abstract descriptions of the exchanged data; and *Prototypes,* which are abstract collections of operators. The concrete protocol and the specification of data format for a particular *Port Type* is the reusable link. A port is defined by associating a network address with a reusable link (W3C, 2001). According to (Erl, 2006), the WSDL description of a service can be separated into *Description Abstract* and *Concrete Description.* A WSDL document uses the following items (W3C, 2001): *types, message, port type, operation, binding, service* and *port.* See Code 4.3 and Figure 4.3.

```
1 <definition .. >
2       <types>
3             <xsd:schema .... />
4       </types>
5       <import namespace"http://www.xml.com/tls/schema"
6             Location=http://www.xml.com/tls/schema/car.xsd/>
7       <message name="getID">
8             <part type="xsd:integer"/>
```

[1] endpoint: Association between a link and a network address, specified by a URI, which can be used to communicate with an instance of a service. An endpoint indicates a specific location for accessing a service using a specific protocol and a data format (Haas and Brown, 2004).

```
9       </message>
10      <portType name="CarInterface">
11              <documentation>
12                      Get Car Details operation.
13                      </documentation>
14              <operation name="getCarDetails">
15                      <input message="tns:rentCar"/>
16                      <output message="tns: rentCarResponse"/>
17              </operation>
18              <operation name="UpdateCarDetails">
20              </operation>
21      </portType>
22      <binding name="CarBinding" type="tns:CarInterface">
23              <soap:binding style="document"
24              Transport=http://schemas:xmlsoap.org/soap/http/>
25              <operation name="GetCarDetails">
27              </operation>
28      </binding>
29      <service name="CarService">
30              <port binding="tns:CarBinding" name="CarPort">
31                      <soap:address   location=http://www.localhost:8080/car/>
32              </port>
33      </service>
34 </definitions>
```

Code 4.3: WSDL structure (Albreshne, 2009).

(3) UDDI (Universal Description, discovery and Integration)

After defining the data in the message (XML), describing the services that will receive and process the message (WSDL), and identifying the means of sending and receiving messages (SOAP), it is necessary to publish the offered services and find the published services to be used. This is the function performed by UDDI (UDDI, 2002) (Newcomer, 2002). According to (Sing *et al*., 2005), UDDI can be defined as:

> *an XML-based platform independent register for all companies around the world that can be listed on the Internet. UDDI is an open, industry initiative sponsored by OASIS, allowing the publication and discovery of services, (Sing et al., 2005).*

The UDDI specifications (Universal Description, Discovery, and Integration) provide users with a systematic and unified way of finding service providers through a service register similar to a *phone book* of Web services (Curbera *et al*., 2002). The functionalities offered by UDDI are facilitated by WSDL and SOAP standards. UDDI also provides a set of categories such as NAICS[2] and UNSPSC[3] to organize the services offered by the companies in the directory, and enable fast

[2] North American Industry Classification System (NAICS) published by *Census*.

[3] United Nations Standard Product and Services Classification (UNSPSC), System jointly developed by UNDP (United Nations Development Program) y D&B (Dun & Bradstreet Corporation) in 1998.

searches at company and service level (Akkiraju *et al*., 2003). UDDI supports three types of service descriptions (Albreshne *et al*., 2009):

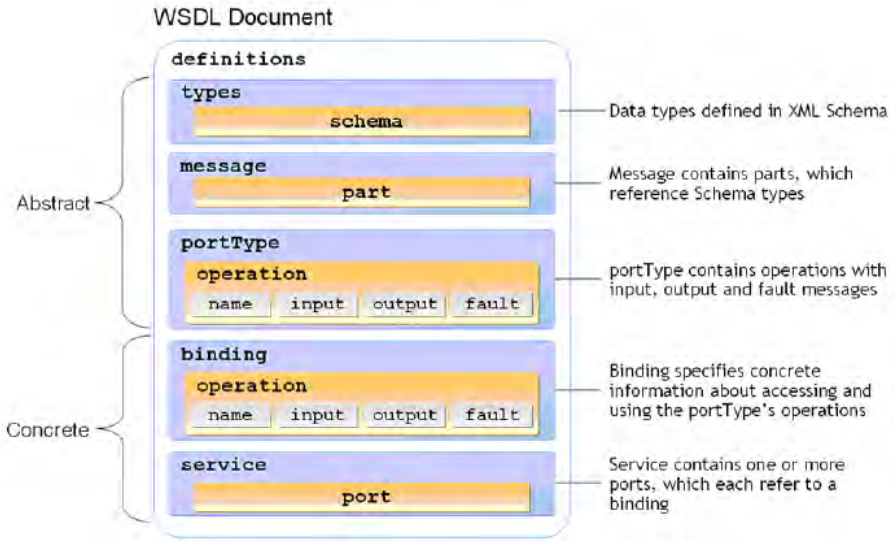

Figure 4.3: Conceptual WSDL, (active endpoints, 2009).

- *White Pages.* Contain the following fields of information;
- *Yellow Pages:* Provide business categories organized as taxonomies;
- *Green Pages:* Contain business information used to describe the way in which other businesses can conduct electronic commerce with them.

The representation of Web services in UDDI (Paolucci *et al*., 2002) is shown in Figure 4.4. A business is represented as an object *Business Entity* that stores information such as company name, contact information, business URL, and others. A Business Entity is associated with one or more *Business Services*, which are descriptions of the specific services provided by the company. In turn, a Business Service is associated with one or more *Business Templates*, and specifies the access point to the service. Moreover, UDDI provides a data structure called *TModel,* which allows specifying additional attributes of the entities described in the UDDI repository (Paolucci *et al*., 2002). A TModel can be understood as metadata containing information on the artifacts that are being modeled. A TModel in UDDI can refer to a technical and standard specification (as WSDL) to describe *Web* services, or abstract specifications of taxonomic schemes as NAICS or UNSPSC (Akkiraju *et al*., 2003).

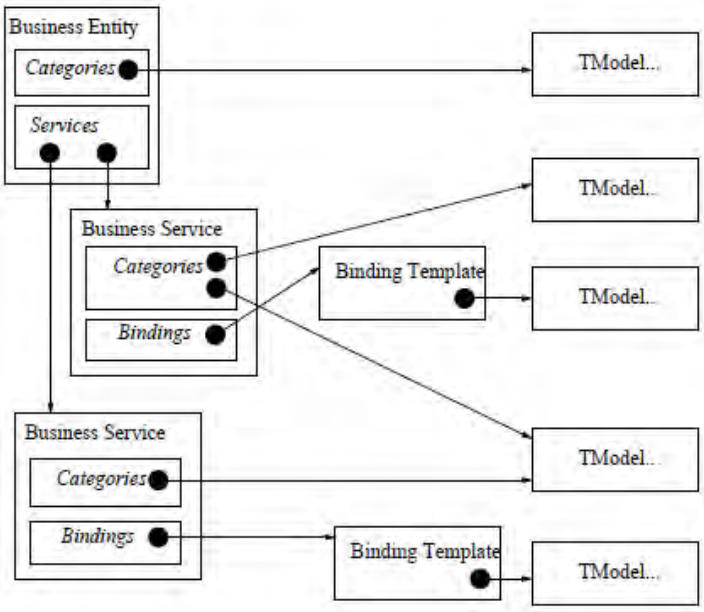

Figure 4.4: UDDI register.

UDDI allows us a wide range of searches: the services can be searched by name, location, links, or TModels. Unfortunately, the search mechanism supported by UDDI is limited to matching keywords, and does not support any inference based on taxonomies referring to a TModel (Srinivasan *et al.,* 2004). Several authors have proposed solutions to this problem by associating UDDI to a mechanism that provides semantic information (Srinivasan *et al*., 2004) (Akkiraju *et al*., 2003).

4.2.2 WEB REST services

REST is a term coined by *Roy Fielding* in his doctoral thesis (Fielding, 2000) to describe an architectural style of systems in the network. REST is an acronym for *Representational State Transfer*. It is convenient to emphasize that REST is not a standard, but an architectural style. Such a style cannot be encapsulated; we can only understand and design Web services by following it. Although REST is not a standard, it uses the following standards (Costello, 2002): HTTP, URL and XML/HTML/GIF/JPEG (resource representation).

We can summarize the REST architectural style with four principles (Pautasso *et al.*, 2008):

- *Resource identification by URI.* The resources are identified by URI, which provides a service discovery mechanism.
- *Uniform interface:* To manage resources, there is a set of four operations: create, read, update, delete.
- *Self-descriptive messages*: The resource content can be in various formats (HTML, XML, plain text, PDF, JPEG, etc.). The metadata on the resource can be used to detect transmission errors or perform access control and identification.

- *Stateful interactions through hyperlinks*: interactions without states can be reached using different techniques, such as URL rewriting, cookies, or hidden form fields.

4.2.3 SOA (Service Oriented Architecture)

SOA provides a new method to create distributed applications where basic services can be published, discovered and linked to build more complex services. The applications interact with services through interfaces and entry points at the implementation level. Furthermore, the applications become more flexible as a result of their ability to interact with any implementation of a contract (Papazoglou, 2008). Web services have enabled companies to cross boundaries and have facilitated better integration because their purpose is to deal directly with interoperability challenges. However, they do not help businesses to dynamically accommodate to the changes since a company's goals do not include facing challenges related to their agility in dealing with changes. Fortunately, when they are properly implemented, SOA can address this problem by coordinating distributed IT resources into a cohesive system that maximizes organizational agility. This agility is translated to a reduction in development time for new business solutions (Microsoft, 2010). OASIS defines SOA as:

> *Paradigm for organizing and using distributed capabilities that may be under the control of several owners (domain). It provides a uniform means to discover, interact and use capabilities to yield the desired effects by being consistent with measurable preconditions and expectations (OASIS, 2006).*

SOA allows reusing existing code in the way that can create new services in an infrastructure or an existing system. In other words, it allows companies to benefit from the existing investments, enabling the reuse of existing applications, and promising interoperability between technologies and heterogeneous applications. SOA provides a level of flexibility that was not possible before its existence (Mahmoud, 2005). A common misunderstanding is that SOA is a new version of *Web* services. The distinction between services SOA and Web services resides in their respective designs. While SOA defines a model to run a data process, Web services provide a tactical implementation of the SOA model. Ultimately, Web services are one of the many ways that we can implement SOA (IBM, 2008). As shown in Figure 4.5, there are three basic roles in a Web service-based architecture (Albreshne, *et al*., 2009): *Service Provider, Service Consumer* and *Service Register.* Consequently, the fundamental concepts in SOA (Hashimi, 2003) can be described as follows:

- *Service*: for SOA, a service is an exposed functionality with the following properties:
    - The contract of the interface of the service is platform independent.
    - The service can be dynamically invoked and located.
    - The service is self-contained. That is, the service maintains its own state.
- The remaining components, *message*, *dynamic discovery* and *Web services,* can be studied in (Hashimi, 2003).

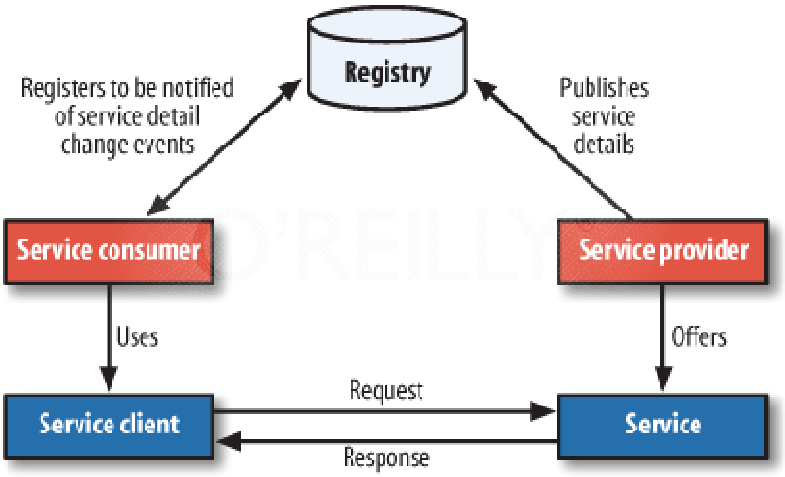

Figure 4.5: SOA architecture, (Governor *et al.*, 2009).

There are many approaches on how to implement and define SOA. (Arsanjani, 2004) proposes an architecture based on seven layers, Figure 4.6:

- *Operational System*: This layer consists of applications specifically built and already existing. SOA-based architectures take advantage of existing systems to integrate them by using techniques of service-oriented integration.
- *Enterprise components layer*: This layer is the component of the company responsible for making the functionality and maintaining the quality of service of the exposed services.
- *Services*: The services that the company chooses to fund and expose reside in this layer. They may be discovered or statically linked and then invoked, or possibly choreographed into a composite service.
- *Business process choreography*: The compositions or choreography exposed in the third layer are defined in this layer. Services are grouped into a flow through an orchestration or choreography able to act together as a single application.
- *Access or presentation layer*: This layer has the services that allow access or presentation. Although this layer is usually beyond the scope of the discussions on SOA, it is gradually gaining importance.
- *Integration*: This layer allows the integration of services through the introduction of a reliable set of capabilities such as intelligent routing, mediation protocol and other transformation mechanisms.
- *QoS*: This layer provides the capabilities required to monitor, manage and maintain the service quality as security, performance and availability.

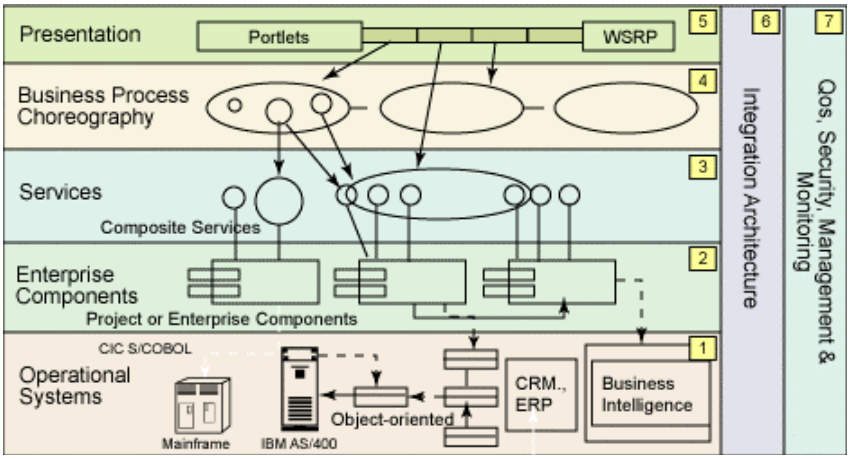

Figure 4.6: SOA layers.

4.3 Semantic Web services

The key element in the *Semantic Web* is the development of a language for encoding and describing Web content. Such a language should have well-defined semantics, be sufficiently expressive to describe the complex inter-relationships and the constraints between Web objects, and be responsible for automatic manipulation and reasoning according to certain acceptable limits with respect to resources and time (McIlraith and Martin, 2003). Web services technology is a distributed computing framework, which provides information and services in demand, in a machine-processable way; probably including a software component to integrate the results provided by different services (Sycara *et al*., 2003).

A semantic Web service is a Web service enriched with metadata for an easy automatic search and composition, Figure 4.7. Technology used in semantic Web services employs formal descriptions of machine reasoning to provide the possibilities described in the previous section. Semantic Web services involve the integration of the semantic Web and Web services. A semantic Web service extends the concept of Web service, providing semantic aspects that can be used in an autonomous way by an information system with access to the Web (Garcia, 2011).

4.3.1 OWL-S

OWL-S (OWL Web Ontology Language for Services) is a language that describes an ontology specifying semantic Web services. It is a continuation and evolution of DAML-S (DARPA Agent Markup Language) (OWL, 2012). Based on OWL, OWL-S provides an ontology that allows us to perform desired tasks in a Web services architecture: the discovery, invocation, composition and monitoring of Web services. The ambitious expected scope is that these tasks can automatically and dynamically be performed through intelligent systems without human intervention.

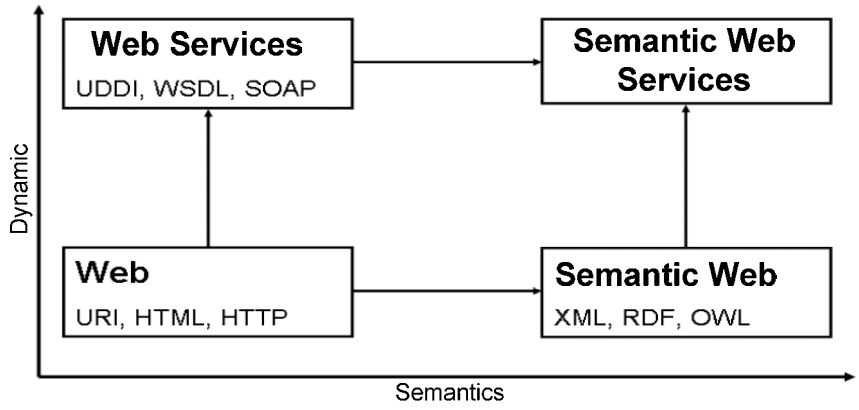

Figure 4.7: Semantics extension of Web services.

OWL-S is a set of high-level ontologies written on OWL specifically for Web service descriptions. It is designed to allow the automation of Web service discovery, service invocation and composition (Luo *et al*., 2006). The structure of the service ontology is motivated by the need to provide three essential types of knowledge of the service, each characterized by questions (OWL, 2012) such as W*hat does it do? How is it accessed? How does it work?* The answers to these questions are in *Profile, Grounding* and *Model* as shown in Figure 4.8.

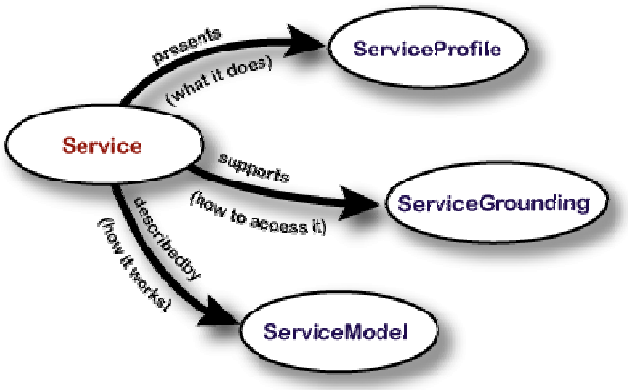

Figure 4.8: OWL-S structure.

(1) Service Profile

The *profile* section provides a set of concepts to specify the functionality of the services (Booth et al., 2004) with the goal of supporting functionality-based discovery, Figure 4.9. Specifically, it enables providers to indicate what makes their services, and allows the applications to specify the functionality expected from the services. Basically, *ServiceProfile* provides an explicit description of this functionality so that it does not need to be extracted from incidental properties as the name of the service or the company offering it (Martin *et al*., 2006).

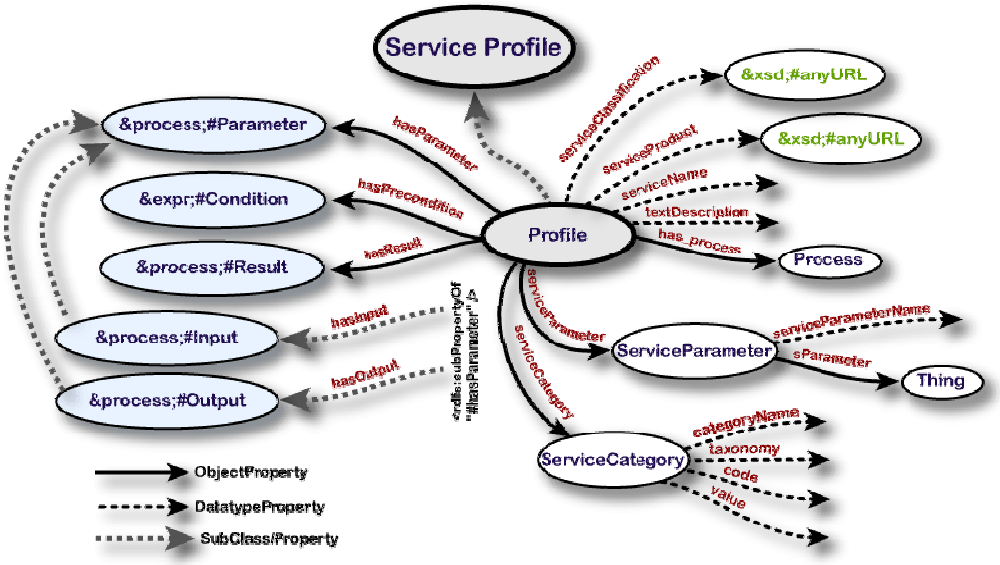

Figure 4.9: Service Profile.

The *ServiceProfile* class provides a superclass of every type of high-level description of the service. ServiceProfile does not force any representation of the services, but it indicates the basic information to link any instance of the profile with a service instance. There is a bidirectional relationship between a service and a profile, so that a service can be linked to a profile and a profile can be linked to a service. These relationships are expressed by the *presents* and *presentedBy* properties (Garcia, 2012):

- *presents*: Describes a relationship between a service instance and a profile instance, which basically says that the service is described by the profile.
- *presentedBy*: The inverse of *presents*; specifies that a determined profile describes a service.

Name of the service, contact and description

Some properties of the profile provide human readable information that is unlikely to be automatically processed. These properties include *serviceName* (name of the service), *textDescription* (description of the text) and *ContactInformation* (contact information). A profile can have a single service name and description text, but it can have as many elements of contact information as the provider wants to offer.

Description of the functionality

An essential component of the profile is the specification of the functionality provided by the service and the conditions that must be met to produce a successful result. In addition, the profile specifies what conditions are required by the service, including the expected and unexpected results of the service activity. The OWL-S Profile represents two aspects of the functionality of the service: information transformation (represented by inputs and outputs) and the state change yielded by the running of the service (represented by preconditions and effects), (Garcia, 2012).

Specifications of the type of service and product

The two properties, *serviceProduct* and *serviceClassification* are used to specify the type of service provided and the products that are handled by the service. The values of the two properties are instances of the classes specified in the OWL ontologies of services and products. The *ServiceClassification* and *serviceProduct* properties are similar to *serviceCategory* previously described, but differ in that the values of the properties are OWL instances instead of strings related to a business taxonomy that is not OWL.

(2) Service model

Once a service has been identified to accomplish its goals, a detailed service model is necessary to determine if it can meet the required needs. If the above holds true, then it would be necessary to determine what constraints must be met and what pattern of interactions is required to use the service (Martin *et al*., 2006).

OWL-S defines a class derived of *ServiceModel* to describe the modeling of the processes. This class is called *Process*. The specific processes are classes derived from the *Process* class. Each described process is a specification of the interaction between the client and the service. Therefore, it is not the specification of a program to be executed as a style BPEL. Since the preconditions and effects are represented by logical rules; a process must not be executed unless their preconditions are met. If the process is executed without meeting the preconditions, then the result can be undefined.

These rules will be expressed in any language that has a textual representation. The logic expressions will have a type *literal* or *literal XML*. To the first type, *literal,* corresponds to languages such as PDDL (The Planning Domain Definition Language) or KIF (Knowledge Interchange Format), whereas *literal XML* corresponds to SWRL (Semantic We Rule Language).

The processes can be atomic, simple or composed, Figure 4.10. The atomic processes correspond to actions that the service can make in a single interaction; they have only two participants: the client and the server (service). The composed processes are carried out on several steps and can be classified into composed and not composed processes by means of control structures. The simple processes provide an abstract mechanism to carry out multiple views of the process.

(3) Service grounding

The *ServiceProfile* and *ServiceModel* classes are considered abstract specifications, in the sense that they do not give details of the message format, used protocol, URL for access to the service, etc. The ServiceGrounding class mission is to provide all those details (Garcia, 2012).

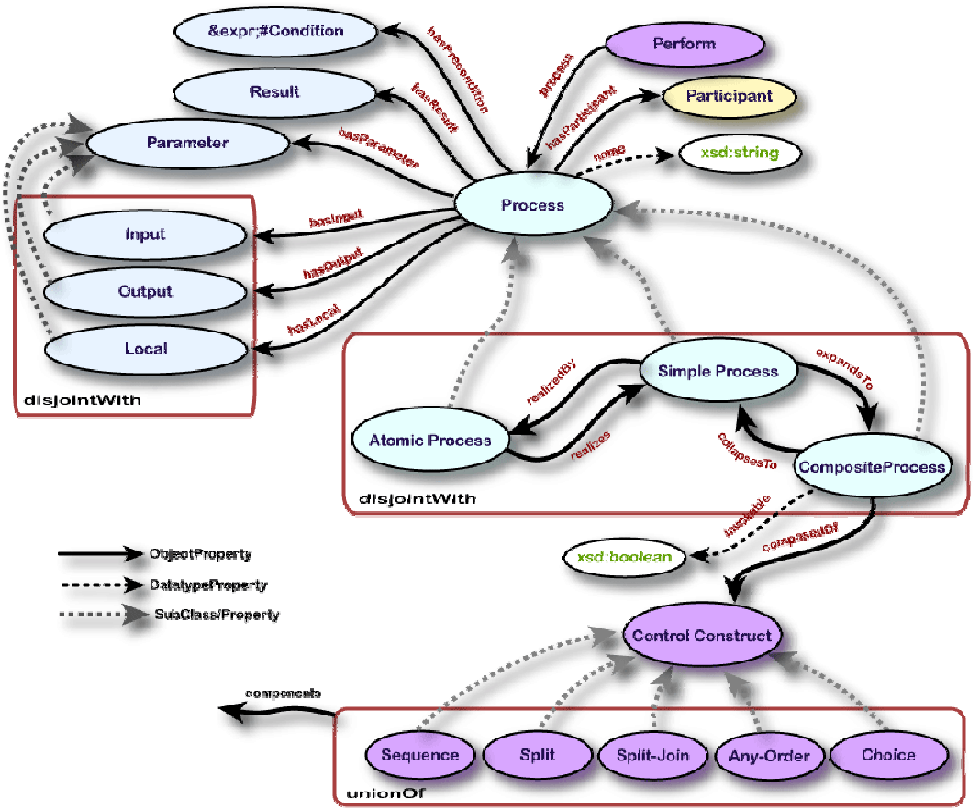

Figure 4.10: ModelService for OWL-S.

The task of specifying abstractly defined messages and operations was already done by bindings. Having a particularly stable and extendible specification, widely used by the industry, OWL-S takes advantage of WSDL and defines new extension points. It also takes advantage of the relationship of WSDL with SOAP and UDDI, allowing a theoretically simple extension to add semantic content to traditional Web services. To convert types of data OWL-S (OWL classes) to the types of WSDL data (XML Schema types), XSLT transformations can be used (only from OWL-S 0.9). See Table 4.1 and Figure 4.11.

For OWL-S	For WSDL
Atomic processes	Operation
Data of input an output for an atomic process	Messages of input and output of *operation.*
Types OWS-L (Classes OWL)	Types XML Schema

Table 4.1: OWLS-S/WSDL

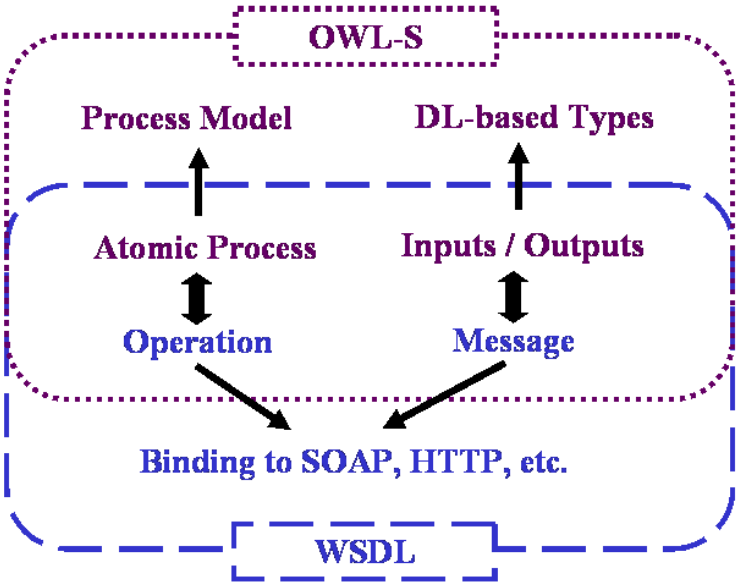

Figure 4.11: WSDL and OWL-S for Service Grounding

(4) Automatic composition of services

The automatic composition of Web services is the process of automatically performing the selection procedure, combining, integrating and running the service to achieve the goal of the user. There are industry standards that provide tools to automatically run services, which have been previously specified in a manually generated composition. Such is the case of WS-BPEL (Web Service Business Process Execution Language) (Arkin *et al*., 2005). But none of these industry standards of Web services have information explicitly interpretable by computers to automate the composition.

Some research has been carried out in this line. The first was based on Golog (McIlraith and Son, 2002), in which the system combines the execution of information-providing *Web* services, performed online, with the simulation of world-altering Web services, performed offline, adapted to the preferences and needs of the user. Other research uses the planning paradigm HTN (Hierarchical Task Network) (Nau *et al*., 2003).

Finally, a mixed approach is to use OWL-S to increase the capabilities of running a WS-BPEL engine with ontologies. This way, it allows the discovery and coupling of services at runtime, as in an SDS (Semantic Discovery Service) proposal.

4.3.2 WSMO (Web Services Modeling Ontology)

This proposal, which had to do with the semantic description of Web services, chronologically followed OWL-S. Holding a similar vision as the authors of OWL-S, WSMO provides a conceptual framework and a formal language to semantically describe all relevant aspects related to Web services so that it facilitates automation

tasks such as the discovery, combination, and invocation of electronic services on the *Web*. In WSMO, a *Web* service is defined as a computational entity that, once invoked, is able to satisfy the goal of the user.

WSMO is based on the conceptual model proposed in the Web Service Modeling Framework (WSMF), which identifies four key elements to describe semantic *Web* services:

1. *Ontologies*: provide the terminology that will be used by the remaining elements.
2. *Goals*: represent the user desires or intentions that must be held by a *Web* service.
3. *Web service descriptions*: define the functional and behavior aspects of a Web service.
4. *Mediators*: aimed at automatically managing the interoperability problems arising between the remaining elements.

Taking into account the concepts identified in WSMF as a basis, WSMO provides ontological specifications for the elements integrating the kernel of semantic *Web* services. Unlike OWL-S, the ontology language used for this purpose in WSML is WSMO. Figure 4.12 shows the high-level concepts coupling WSWO ontology .

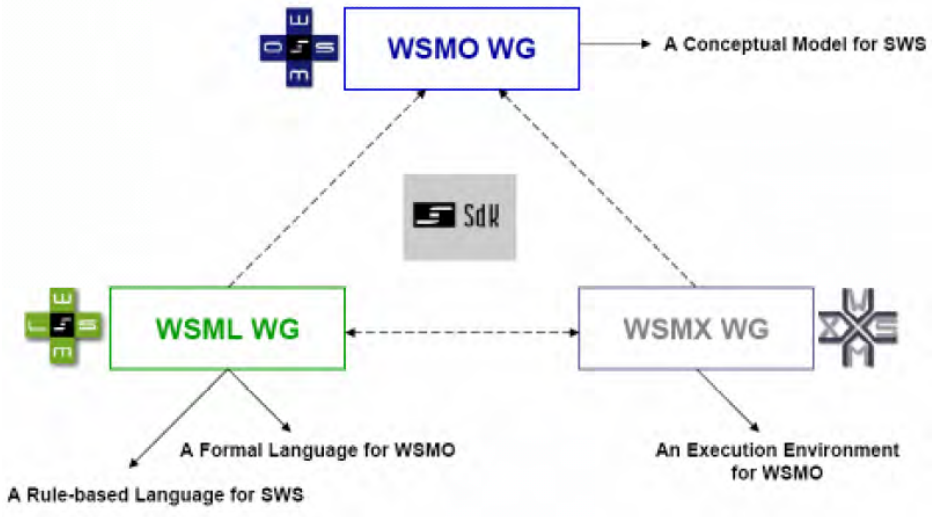

Figure 4.12: General vision of the WMSO performance.

4.3.3 SWSF (Semantic Web Services Framework)

SWSF is a new initiative to define a richer semantic specification on the current *Web* service technology that allows greater automation and flexibility in the provision and use of services. The framework is designed to support the construction of more powerful tools and methodologies in the *Web* services environment, and promote the use of reasoning processes in semantic-based services. In this sense, and as with previous approaches, SWSF attempts to incorporate richer semantics by supporting increased automation of tasks such as selection and invocation of services, translation of the message content between heterogeneous services operating with each other, service composition, etc. Moreover, it allows integral approximations for service monitoring and error recovery.

This proposal consists of two main components: Semantic Web Services Language (SWSL) and Semantic Web Services Ontology (SWSO). SWSL is a general purpose logical language that includes certain features making it more appropriate for the needs on the *Web* and semantic *Web* services. Among these features are the use of URIs, integration of the types that are part of XML, the use of import mechanisms and namespaces supporting XML. This language is used to specify the formal characterizations of concepts related to Web services and their descriptions. It includes two sublanguages: SWSL-FOL, based on first-order logic with extensions of HiLog and F-Logic frame syntax. SWSL-Rules, based on the logic programming paradigm, and used to support the use of the service ontology for reasoning processes and execution environments based on this paradigm.

SWSO defines a conceptual model on which Web services can be described, and a formal representation or axiomatization of this model. The full axiomatization is implemented on first-order logic by using SWSL-FOL, with a Model Theory semantics that specifies a precise meaning of the ontology concepts. This form of ontology using first-order logic is called "First-Order Logic Ontology for Web Services" (FLOWS).

4.3.4 WSDL-S (Web Service Semantics)

(WSDL-S, 2005) radically changed with respect to the "traditional" perspective for the incorporation of semantics to Web services. WSDL-S defines a mechanism to associate semantic annotations with *Web* services by using WSDL (Figure 4.13). Unlike the languages previously described, WSDL-S assumes the existence of semantic models of the relevant domain for each service. The services are maintained out of the scope of documents WSDL, but can be referenced from WDSL documents through extensibility elements conceived as part of the WSDL-S proposal.

As in the previous cases, the conceptual basis of this approach is based on the fact of that the current WSDL standard operates at a syntactic level by lacking the semantic expressiveness necessary to represent the requirements and capabilities of the services. This way, the incorporation of semantics can improve software reuse and discovery, facilitate Web service composition, and enable the integration of applications inherited as part of the business integration processes. The semantic information considered by WSDL-S includes the definitions of preconditions, inputs, outputs and effects of the Web service operations.

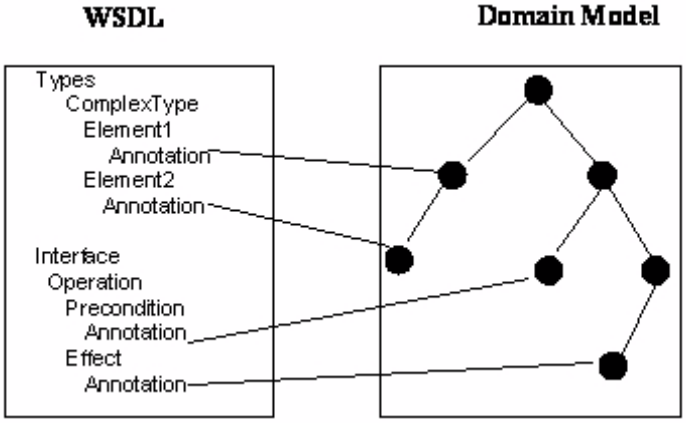

Figure 4.13: Semantics annotations for elements in WSDL-S.

Among the advantages of this novel approach over others, such as OWL-S and WSMO, we can highlight the following:

1. Users can incrementally describe all details, both semantic and at the operational level, for WSDL, which is a language that is familiar to the developer community.
2. By outsourcing semantic models of domain, WSDL-S remains independent of the ontology representation language to use. This is an additional advantage, because reusing models of the existing domain that are expressed in modeling languages as UML can accelerate the incorporation of semantic annotations.
3. It is relatively easy to modify the existing tools around the WSDL specification to incorporate the elements proposed by this approach. In any case, it is faster and reliable than the development of tools implemented from zero.

4.3.5 SAWSDL (Semantic Annotations for Web Services Description Language)

SAWSDL is an extension of the language of *Web* service description (WSDL) developed by a working group of W3C (in 2007). It is comprised of two basic types of annotations: the reference to the model; and the mapping of the scheme.

Annotations to reference the model are the same as those used in model WSDL-S and are used to associate interfaces, ports, operations, inputs, outputs, XML elements and attributes with semantic concepts. With regards to mapping the scheme, these elements are attributes added to the XML schema declaration of the elements in order to specify mappings between XML and semantic information. They are used during the invocation of services to format the information of the XML customer so that it can be understood by the Web service. This solves the structure problem of inputs and outputs of the services. It used to be associated with a XSLT transformation sheet.

Unlike WSDL-S, SAWSDL specifies the behavior of semantic Web services by using ontology description languages such as OWL, and is useful for carrying out service choreographies. It also allows the development of the following features:

- Service classification
- Discovery
- Matching
- Composition
- Dynamic invocation.

4.4 BPM and BPEL

A business process is a real-world activity consisting of a set of logically related tasks that, when they are run in the suitable sequence and follow the business rules, generate a valid output for the company (make a payment, perform a cash extraction from a bank account, etc.) (Bazan, 2010). In this context, the flexibility of information systems has become a major concern for business analysts. In fact, the constant evolution of the requirements of a company needs to implement a flexible and

adaptable information system able to face the modifications of business processes (Radgui *et al*., 2012).

The development and analysis of complex business processes requires advanced tools and methods (Ligeza *et al*., 2012). BPM (Business Process Management) emerged as a way to manage business processes. It represents methods, techniques and software tools to design, enact, control and analyze the operational processes related to people, organizations, applications, documents and other information sources (Van Aalst, 2003).

An area intrinsically linked to BPM is one corresponding to Web services and service-oriented architecture. Web services represent a new generation of Web applications. They are self-contained, self-describing and modular software components; that can be accessed, located and invoked from anywhere on the Internet. BPEL (Business Process Execution Language) appears within this framework as a de facto standard to run business processes by enabling the composition and integration of various Web services by itself (Viroli *et al*., 2007). As the use of Web services increases, so does the choice of "Business Process Execution Language" (BPEL) by parts of companies. BPEL is aimed at modeling business processes within the *Web* service architecture. By developing Web services from BPEL, companies can implement aspects of service-oriented architecture that have previously been difficult to achieve (Pasley, 2005).

When we compare BPEL with similar standards (as XPDL and WSCI), we conclude that it has a good expressiveness (Wohed, 2003) and is currently the only standard that has running engines such as (Oracle BPEL Manager, 2012) (IBM BPEL4WS, 2012) and (ActiveBPEL, 2012) (Morrison and Nugrahanto, 2007). As a result, BPEL has successfully been implemented in very different areas: software for diagnostic decision support (Morrison and Nugrahanto, 2007), modeling of clinical applications (Morrison *et al*., 2006), multi-agent systems (Viroli *et al*, 2007) and so on. Moreover, its implementation in the business environment keeps growing.

4.4.1 BPM (business process management)

BPM (Business Process Management) is the name of a set of software systems, tools and methodologies, focused on how companies identify, model, develop, distribute and manage their business processes (Bazan, 2010); it deals with change management of requirements for improvement and unifies previous disciplines of process modeling, simulation, work dynamic, enterprise application integration (EAI[4]), and integration of "Business-to-Business (B2B[5]), in a single standard (Owen and Raj, 2003). All this allows us to identify whether business processes are optimal or beneficial for the company, focusing on continual improvement of such processes. According to (Van Aalst, 2003), BPM can be defined as:

[4] EAI: Uses software and system architecture principles to integrate a set of applications within any company.

[5] BSB: Transmission of information concerning to electronic commercial transactions, usually using technology as Electronic Data Interchange (EDI).

Discipline supporting business processes by using methods, techniques and software tools to design, enact, control and analyze operational processes related to people, organizations, applications, documents and other information sources.

The main benefits of BPM are according to (Bazan, 2010):

1. Reduces the errors of obstruction between business requirements and IT systems, as business users model the processes and then the IT department provides the infrastructure to run them.
2. Increases productivity of employees.
3. Increases corporate flexibility and agility by separating the logic of the process from other business rules. This better absorbs changes in the requirements.
4. Reduces the cost of development using high-level graphic programming languages.

A key concept regarding BPM is the definition of its life cycle. There are many approaches on the life cycle of generic BPM (Havey, 2005), (Hill *et al.*, 2006). Due to its relevance in the field, a widely accepted definition of the BPM life cycle is given in (Van Aalst, 2003), which considers the following processes: *process design, system settings, process enactment* and *diagnosis.* Despite the different specifications of the BPM life cycle, four main processes can be abstracted, Figure 4.14. Thus, the BPM life cycle would be based on the feedback of the cycle:

1. *Modeling*: Definition and specification of business processes.
2. *Execution*: Execution of the processes specified in the previous step.
3. *Monitoring*: Review of the results of the executed processes.
4. *Optimization*: From the results, to improve the business processes for future executions.

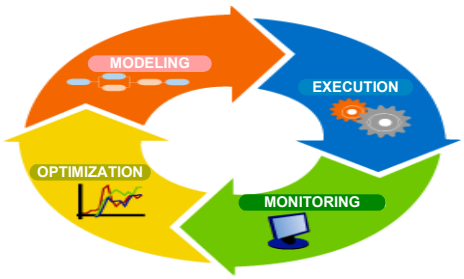

Figure 4.14: Life cycle of BPM.

A categorization of the standards belonging to BPM can be made by considering four groups according to their functionality and similar characteristics (Ko *et al.*, 2009):

1. *Graphic standards*: allow users to express business processes, flows and transitions in a schematic way (BPMN, UML).
2. *Execution standards*: the deployment and automation of business processes (BPEL, BPML) is computed.
3. *Interchange standards*: facilitate the portability of data, for example, the portability of business process designs through BPMS, different standards of execution on different BPMS, translation from graphic standards to execution standards, and vice versa (XPDL , BPDM).

4. *Diagnostic standards*: provide management and monitoring capabilities. These standards identify bottlenecks and carry out real-time queries of the business processes of the company.

An important concept of BPM is its joint application with a service-oriented architecture. SOA enables the design and construction of a set of services, but this set will not be of much use unless there is something that makes use of such services. BPM appears as an alternative to provide maximum performance to SOA, where SOA will greatly help to build faster processes in the BPM system. All of this means that both technologies complement each other.

(1) BPMN (Business process management notation)

BPMN (Business Process Modelling Notation) is a graphical notation that describes the logic of the steps in a business process. This notation has been specially designed to coordinate the sequence of processes and messages that flow between participants of different activities. A recent standard is to model the flow of business processes and *Web* services. It was created by BPMI (Business Process Management Initiative), so the main goal of BPMN is to provide a notation readable by all users of the business. This includes business analysts that create the initial drafts of the processes, and the technical developers responsible for implementing the technology that will perform those processes (Owen and Raj, 2003). Despite being a relatively recent proposal, BPMN is supported by a considerable number of tools (Ouyong *et al*., 2006).

BPMN specifies a diagram of business processes called *Business Process Diagram* (BPD).This diagram has several main points:

- Modeling the business processes in a simple way.
- To be used by non-technical users.
- To offer the expressiveness to model complex business processes.
- Simple mapping to business execution languages (such as BPEL).

Some of the elements that are in the BPD diagram are conceptually presented below. For the modeling of processes or activities are the following entities (Figure 4.15):

- Task: A task is used when the work in the process does not break down in more details. It is performed by a person and/or an application. There are specialized types of tasks for sending and receiving, user-based tasks, etc. Icons or markers can be added to tasks to help identify the type of task (White, 2006).
- Subprocess: A subprocess is a composed activity included within a process. This activity in turn includes a set of activities and a logical sequence (process) indicating that the activity can be analyzed in more detail. It may visually appear as collapsed or expanded.

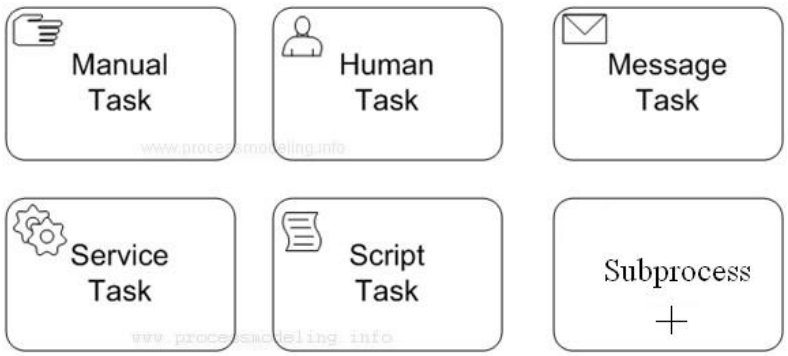

Figure 4.15: Activities in BPMN.

The execution flow can be modeled using two elements, that is:

- Gateways, (Figure 4.16): Modeling elements used to control the divergence and convergence of flow. They are represented by a diamond.
- Connectors (Figure 4.17): They connect different elements of the diagram (Mancarella, 2011).

Figure 4.16: Gateway in BPMN.

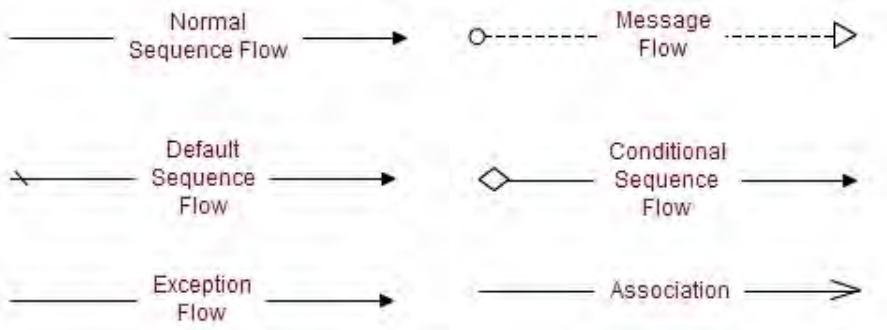

Figure 4.17: Connectors in BPMN

Another important component in the BPD diagram modeling is the events, Figure 4.18. An event occurs during the execution of a process, and begins or ends the process. They are modeled as circles with different lines of thickness or double lines.

Figure 4.18: Events in BPMN

In addition, different triggers indicate specific circumstances of the event (such as an email message or an error). They are represented by an icon within the circle of the event, Figure 4.19.

Figure 4.19: Trigger events in BPMN.

BPMN is in version 2.0, released in 2010. With Regard to version 1.0, it includes improvements such as direct mapping to BPEL and serialization to XML among others. It also presents execution semantics, which accurately describes the way in which the BPMN models should behave when they are executed in a tool (Van Gorp and Dijkman, 2012).

4.4.2 BPEL (Business Process Execution Language)

BPEL is the de facto standard to specify business processes in a Web services environment. It allows composing Web services and specifying the composition as a Web service by itself (Weerawarana, 2005). The Web service composition can be specified as a flow of Web service operations. Therefore, BPEL provides *certain structured activities* that prescribe the control flow between *interactive activities*, which are the activities modeling the interactions with other Web services (Nitzsche *et al*., 2007). Moreover, BPEL is an open standard, making it interoperable and portable across different environments (Pasley, 2005).

An important advantage of BPEL over other languages is its association with BPMN. BPMN can specify business processes in a graphical way and is easily compressible. Furthermore, BPMN supports the graphical properties of the objects that they will generate an executable BPEL. An example of how to map a BPMN diagram on booking travel to a BPEL process is given in (White, 2005). (Ouyang *et al*, 2006) proposes a method for mapping BPMN to BPEL that solves the underlying problems of mapping graph-based languages with parallelism. In 2010, with the release of BPMN 2.0, the standard itself includes a mapping to BPEL.

BPEL is an XML-based standard to define process flows (Pasley, 2005) and is a fully executable specification language (Bazan, 2010). There are different uses and approaches of BPEL in the scientific community. One of them is given in (Ferber *et al.*,

2010), and presents an approach for modeling of BPEL processes (BPELROs) to facilitate the integration of BPELROs to object-oriented programming languages.

In addition to facilitating synchronous (client-server) and asynchronous (*p2p*[6]) *Web* service orchestration, BPEL provides specific support for long-running processes that maintain the state. Some of the strongest points of BPEL (Ko *et al*., 2010) are:

- Popular without serious competitors in the industry (Havel, 2005) (Koskela and Haajanen, 2007). This implies that the BPEL-compatible products are stable and the risk of obsolescence is minimal. Having been adopted by major software vendors, portability is not a problem with small BPMS providers.
- Focuses on process constructions instead of low-level programming. Compared with conventional programming languages such as Java; BPEL is able to model the interactions of typical business processes as long-term transactions, asynchronous messaging and parallel activities. It would take much more effort and lines of code to express the same process in a conventional programming language (Van der Aalst *et al*, 2005 a, b).
- It is subscribed to the *Web* service paradigm. This means that BPEL takes advantage of the highly adaptive and dynamic nature of *Web* services. BPEL incorporates a number of specialized features for the development of *Web* services, including direct support to define and handle XML data, a dynamic mechanism based on explicit handling of *endpoint* references, a declarative mechanism to correlate incoming messages to process instances, which is essential for asynchronous communication. As such, BPEL can be seen as an attractive alternative with respect to conventional programming languages according to the development of *Web* services (Van der Aalst *et al*, 2005 a, b).

From the point of view of the creators of BPEL (Andrews *et al*. 2003), business processes can be described in two ways (Shapiro, 2002):

- Executable business processes: The model describes the behavior of a participant in a business interaction.
- Business Protocols: In contrast, the protocols include the use of process descriptions that specify the message exchange behavior of each party involved, without revealing details or internal behavior. The process descriptions for protocols are called *abstract processes* (Havey, 2005).

On the other hand, BPEL is based on two types of files (Andrews et al., 2003). The BPEL file, encoded as XML is the definition of a process, includes its main activities such as *partner* links, variables, and event handlers. The accompanying WSDL file specifies the *Web* service interfaces that are of interest for the process defined in the BPEL file (that is, implemented services and called by the process).

The BPEL programming language also provides:

- A message correlation mechanism based on properties.
- Variables of type XML and WSDL.

[6] p2p: A peer-to-peer network is a computer network in which some or all aspects work without fixed clients or servers, but a series of nodes that behave as equals.

- An extensible language model of components to write expressions and queries in multiple languages: BPEL supports by default, XPath 1.0.
- Structured programming constructs including "if-then-else if-else", "while", "sequence" (enables execution of command in order) and "flow" (enables execution of command in parallel).
- A scope system (scoping) allowing the encapsulation of logic with local variables, fault handlers, compensation handlers and event handlers.
- Serialized scopes to control access to variables.

(1) WS-BPEL

WS-BPEL (Web Services Business Process Execution Language) is an OASIS standard that implements the features of BPEL. The basic concepts of WS-BPEL can be applied in two different ways, as a process Abstract or Executable (OASIS, 2007).

- An abstract WS-BPEL process is a partially specified process that is not made to be executed and must be explicitly declared as abstract. Unlike the executable processes, the abstract processes can hide part of the concrete operational details.
- An executable process is designed to run processes. Moreover, these processes can interact with other processes in a consistent way regardless of the support platform or the programming model used for the implementation of the underlying environment.

WS-BPEL is extensible and supports XML implementations; allowing attributes defined in a namespace to appear in any WS-BPEL element (OASIS, 2007). Furthermore, a WS-BPEL process is a container which includes the declaration of relationships with external *partners*, the statement of the process data, handlers for different purposes and most importantly, the activities to be executed. The basic structure of a WS-BEPL process in Code 4.3 is as follows:

```
<process name="ncname" targetNamespace="uri"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         enableInstanceCompensation="yes|no"?
         abstractProcess="yes|no"?
         xmlns="http://schemas.org/ws/2003/03/businessprocess/">

         <partnerLinks>?
         <!-- Note: At least one role must be specified. -->
                  <partnerLink name="ncname"
                   partnerLinkType="qname"
                   myRole="ncname"?
                   partnerRole="ncname"?>+
                  </partnerLink>
         </partnerLinks>

         <partners>?
                  <partner name="ncname">+
                           <partnerLink name="ncname"/>+
                  </partner>
```

```
        </partners>

        <variables>?
                <variable name="ncname"
          messageType="qname"?
          type="qname"? element="qname"?/>+
        </variables>

        <correlationSets>?
                <correlationSet name="ncname"
              properties="qname-list"/>+
        </correlationSets>

        <faultHandlers>?
        <!--There must be one fault handler or default.-->
                <catch faultName="qname"?
               faultVariable="ncname"?>*
                            activity
                </catch>

                <catchAll>?
                        activity
                </catchAll>

        </faultHandlers>

        <compensationHandler>?
                activity
        </compensationHandler>

        <eventHandlers>?
        <!--There must beone onMessage or onAlarm handler. -->
                <onMessage partnerLink="ncname"
                portType="qname"
                operation="ncname"
                variable="ncname"?>
                        <correlations>?
                                <correlation set="ncname"
                        initiate="yes|no"?>+
                        <correlations>
                        activity
                </onMessage>
                <onAlarm for="duration-expr"?
               until="deadline-expr"?>*
                        activity
                </onAlarm>
        </eventHandlers>

        activity

</process>
```

Code 4.3: WS-BPEL structure.

The definition of a WS-BPEL process is always included within the label <process> and has the following attributes (OASIS, 2007):

- *queryLanguage:* Specifics the query language used in the process to select nodes in allocation.
- *expressionLanguage*. Specifies the expression language used in <process>.
- *suppressJoinFailure*. Determines if error *suppressJoinFailure* will be annulled for all process activities.
- *exitOnStandardFault*. Indicates that the process must finish its execution before the occurrence of activity <exit>.

An important use for WS-BPEL is the description of business interactions in which the business processes of each company interact through *Web* service interfaces. WSDL already describes the functionality of a service provided by a *partner*, on both levels: abstract and executable. Usually the relationship of a business process to a partner is peer-to-peer, requiring a two-way dependency at the service level. In other words, a *partner* represents both the consumer of a service provided by the business process, and the service provider of a business process (OASIS, 2007).

WS-BPEL defines several structures to identify roles and relationships in the interactions. These structures are *Partner, Partner Link* and *Partner Link Type*. Basically, a *Partner* (also called *Business Partner*) is a collection of *Partner Links,* grouping several services in one *Partner*, for example: a *Partner* of "customer service" could group "price inquiry", "sale" and "shipment" services. At the same time, a *Partner Link* describes the roles processes, services, or data manipulated by that role. A *Link Partner* is defined by its *Partner Link Type* as shown in Figure 4.20.

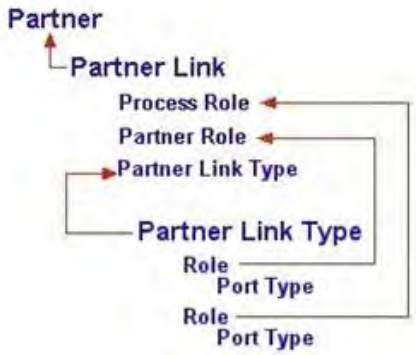

Figure 4.2: *Partners* in WS-BPEL.

It is important to note that the definition of a WS-BPEL process is based on *XML Schema and WSDL 1.1* for the definition of data types and service interfaces. The process definition is also based on concepts such as variable properties, property alias, types of links etc., which are defined within documents WSDL 1.1 by using the extensibility features of language WSDL-1.1. The <import> element is used in a WS-BPEL process to declare a dependency towards an XML document, or WSDL definitions (OASIS, 2007).

WS-BPEL needs WSDL by assuming that all external interactions of the business process occur through Web service types of operations. However, WS-BPEL business processes represent interactions stored by the preceding events in long-running

interactions, where each interaction has a beginning, a behavior defined during its execution cycle, and an end (OASIS, 2007).

4.5 Conclusions

Effective and proper management of business processes can guarantee the successful development of a company. The analysis, based on the goals, scope and limitations, made by applying the standards, methodologies and models to carry out the process approach and the management of these, proves that in many cases different business processes include similar or identical elements or components. In particular, if these elements or components are created on the basis of Web services in the environment of an ontology and similar semantics, it becomes relatively simple to identify the extent to which this happens without making an exhaustive analysis. For the development and integration of enterprise applications as well as business processes to be efficient, it is necessary to have a well-structured methodology and model. In this chapter we have shown business processes as useful, efficient and methodologically proven tools to use when implementing computer applications. These processes can be built with a technology as solid as the technology provided by Web services. In this context, it is necessary to apply an ontology and maintained semantics, so that the process of reuse and composition of new processes can be performed quickly and efficiently. The next chapter presents a systematic and disciplined process to create new semi-automatically business processes by basing them on process that already exist in the cloud environment.

5 IPCASCI

5.1 Introduction

In this chapter we present a business process construction model called IPCASCI (*Intelligent business Processes Composition based on MAS, Semantic and Cloud Integration*). The software development industry requires the fast construction of new products that will adapt to the emerging needs of an ever-changing market. In this context and as a method for reusing software components, we present a new model or methodology that facilitates the reuse of web services in cloud environments to compose business processes. Here, we present an architecture proposal that, based on web service technology, allows: (i) Automatically discovering Web services. (ii) Providing Web services a semantic description. (iii) Automatically composing web services to generate new services and (iv) Automatically invoking Web services.

All this is done in a way that allows the process of automating the building of new services to be efficient, and for the services to be associated with intelligent behavior. By intelligent behavior we mean that staring from an input of requirements carried out informally (textually), the system is able to: (i) Analyze such an input (ii) Find the services that allow meeting requirements (iii) carry out the automatic composition of web services and the corresponding business processes that they define. As a result, we will obtain new web services which implement the requirements given by the client, in an automatic process of discovery and composition. There are different approximations for the implementation of platforms based on semantic Web services. A solution based on diffuse logic for the discovery of semantic Web services was proposed in (Su *et al.,* 2012), a solution based on agents and the DAML-S ontological language is proposed in (Sycara *et al*., 2003), and in (García *et al.*, 2012) a proposal based on queries SPARQL and the DAML-S ontological language is used. In general, the greatest part of the proposals of semantic Web services architecture is based on OWL-S language. The proposal carried out in this document differs from proposals carried out until the moment when:

- A global platform is designed, embedded in a cloud environment, whose structure is specifically thought to offer fast and efficient execution.
- The semantic information of the Web services is independent of the internal construction of the ontologies, allowing the reuse of any existing ontology, regardless of the format in which they are built.
- A multi-agent system based on virtual organizations facilitates the process of discovery and requirement analysis.
- A Web service solution is built from an informal definition carried out by the client.

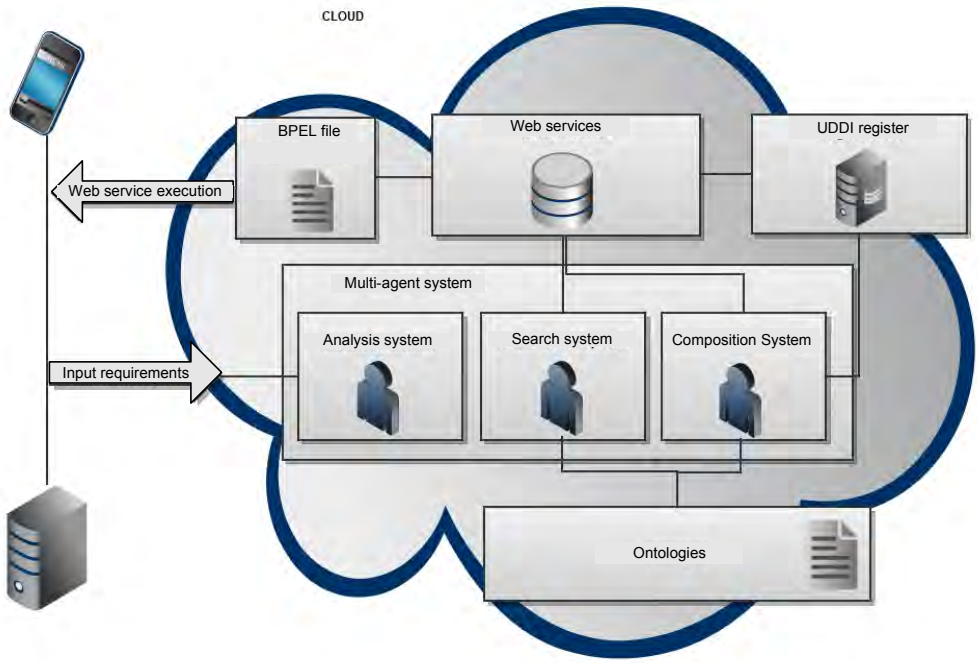

Figure 5.1: Platform description

In Figure 5.1 the different elements composing the platform are shown through a conceptual diagram:

- Cloud System: A system that follows the paradigm of cloud computing in which the different elements of the platform are included, offering an execution and storage environment.
- Web services: Different Web services registered in the platform that can be used to compose new Web services.
- UDDI registry: A registry system where the different Web services of the platform are registered.
- Multi-Agent system based on virtual organizations: It offers the functionality that will allow carrying out the discovery and composition of Web services.
    - Analysis System: Analyzes the semantic content introduced by the user and structures it in such a way that it is computationally analyzable.
    - Search System: In charge of finding out Web services that meet the semantic as well as format restrictions introduced by the user.
    - Composition System: Once Web services that meet the semantic requirements of the user and their relations are determined, they will be composed so as to obtain a new Web service.
- Ontologies: Different ontologies that model semantic knowledge which can be included in Web services.
- BPEL file: Composition of Web service that meets the requirements marked by the user.

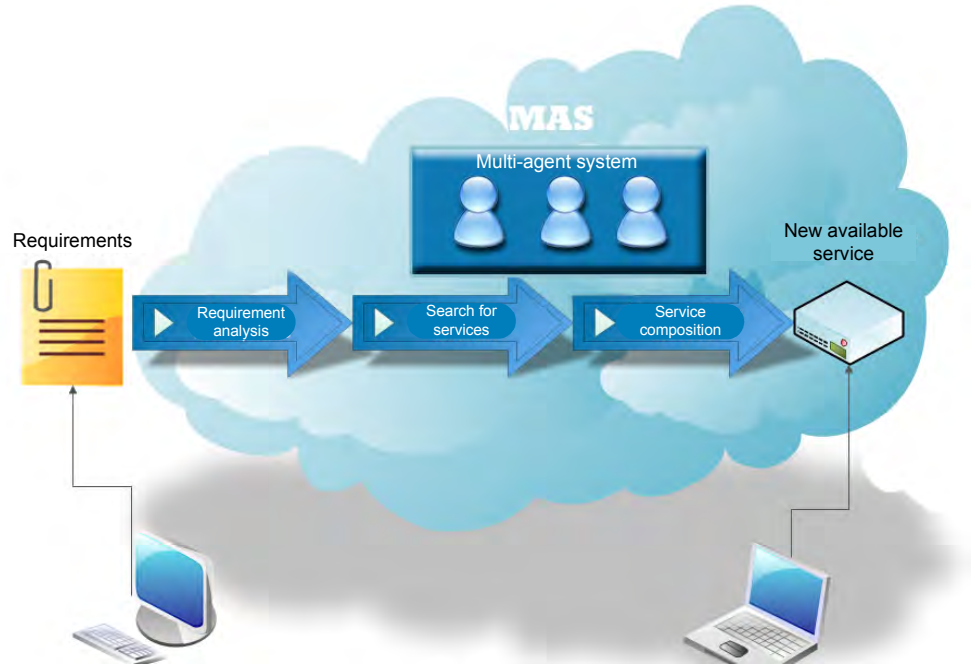

Figure 5.2: Process of obtaining Solution Service.

The process to generate a Web service is as follows:

- Users introduce the requirements of the service they wants to obtain through an assisted system (the system will offer the user the possibility of defining a series of representative terms of the system requirements – the term list will be limited by the web service repository of the database). The result will be a set of related modules, so that each module carries the following information:
  - Module input.
  - Output it produces.
  - Process carried out in the module. (semantics).
  - Domain of the semantics concept (*ontological domain)* that represents the module.
  - Precondition for its execution.
  - Relation with previous modules.
- For each of the modules of the previous stage, a search of the Web services that fulfill the module requirements will be carried out (the search can also be done by form of invocation and semantic content).
- We will build a reduced BPD diagram (it will not express all the modeling capabilities of BPMN standard) in which each Web service is represented through an activity, and all interactions produced between the various web services are reflected.
- We present a BPD diagram to the user. It may be the case that for one activity there are various services that implement its functionality. In that case, one will be chosen by default; however, the user will be allowed to choose the service they want among those available.

- Starting from the BPD diagram we will carry out a BPEL composition so that the service requested by the user can be directly invoked.

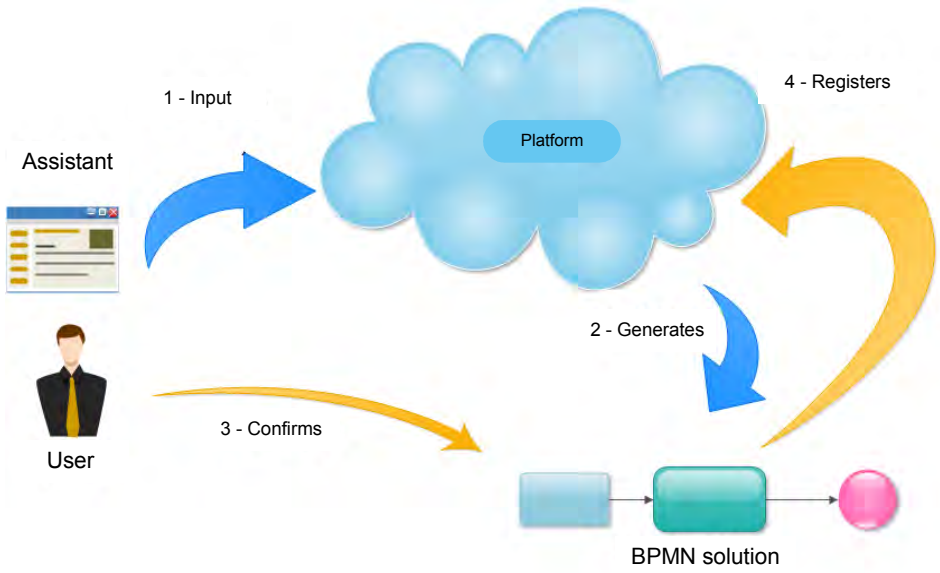

Figure 5.3: Process of obtaining the Web service.

5.2 Cloud system

When analyzing the proposed methodology it is necessary to bear in mind what has been addressed for the composition of web services in a cloud environment. In this sense, as the generated software will be for shared and distributed use, its management has to be efficient and reliable, and it must be accessible from any platform and place. Cloud systems have advantages over conventional distributed systems for reasons such as those outlined below:

- Elastic peak load management: If many concurrent requests are made to the server, its performance might be negatively affected. Moreover, we have to take into account that it is impossible to anticipate when and how much those peak loads will be. We could carry out a statistical analysis on the time when more requests are made (to increase the processing capacity of server load or temporarily add more servers); however, this option would not ensure that the platform would be able to manage peak loads at any given time. What is more, the option of hiring more servers without any kind of control proves to be extremely costly.
- Reliable architecture: Cloud environments are very reliable and robust, as well as always being active regardless of hardware failure. This is harder to manage in conventional distributed environments.
- "Unlimited" storage capacity: Applications or business processes developed for cloud can grow at any moment without requiring important cost in the software.

The main objective of cloud computing is to offer software, services and computing infrastructures which are carried out independently by the network. This concept is based on the development of scalable, dynamic and distributed software (Rodríguez *et al*., 2010). In this sense, multi-agent systems and SOA integrate well together by mutually increasing their capacities. There are projects that successfully integrate SOA and multi-agent systems, such as CISM@ (Bajo *et al*., 2010), used in the analysis of microarray, or the project proposed in (Cao *et al*., 2009), where a cloud architecture using a multi-agent system and SOA is used to ensure Cloud QoS. One of the greatest inconveniences and one of the obstacles of this technology is the lack of models or methodologies that facilitate the agile development of software. By agile, we refer to the idea that the development process of the new software component is automatic. In this context, the model proposed here would try to eliminate this barrier and offer a mechanism to facilitate the composition of new business processes in a significantly efficient way. From the point of view of the client, apart from the advantages previously mentioned, cloud systems offer the advantage of a limited economic cost and pay-per-use. Clients will not have to spend large amounts of capital to acquire storage systems nor to hire staff to offer support for such systems. Instead, they will only pay for the use they make of the platform, allowing small and medium-size companies to benefit from the services offered. All this while ensuring fast and reliable access, from any place, and at any time.

5.2.1 Cloud services

Cloud systems offer services that can be grouped into three categories:

- Infrastructure as service (IaaS): Distribution of infrastructure resources upon request, generally in terms of virtual machines.
- Platform as service (PaaS): Support of operating systems and framework of software development.
- Software as service (SaaS): Contribution of applications upon request on the Internet.

In the framework of this classification, the proposed model is focused on the layers of *Infrastructure as Service* and *Software*. The layer of infrastructure supplies the services in charge of storing files, disclosing to the client the responsibility of acquiring the corresponding storage systems and their maintenance. With respect to the SaaS layer, the model facilitates the management of a set of REST services (IBM, 2008) that allow managing the various necessary operations for the discovery and automatic composition of the services.

analyze&Descovery

The functionality analyze&Discover of the proposed model receives as input an XML document that stores the information about the requirements introduced by the user, following the structure described in point 5.4.1. Starting from the input, a diagram will be generated according to the BPMN standard, to represent such an input. For each activity of the diagram a set of web services will be given back, which will adjust the semantic, as well as the format requirements of the module corresponding to the input. The input document will follow the structure described in Code 5.1.

```
<Descripcion>
        <Modulo nombre=”modulo1>
                <ListaEntradas>
                        <Entrada id="idEnt1"   tipo= "tipoEnt1">
                                Concepto1
                        </Entrada>
                        ----------------------------------------
                        <Entrada id="idEntN" tipo="tipoEntN">
                                ConceptoN
                        </Entrada>
                </ListaEntradas>
                <Salida tipo="tipoSalida">Concepto1</Salida>
                <Funcionalidad>Funcionalidad1</Funcionalidad>
                <Dominio>Dominio1</Dominio>
                <Precondicion>Precondicion1</Precondicion>
        </Modulo>
        ----------------------------------------
        <Modulo nombre=”moduloN”>----------</Modulo>

        <ListaRelaciones>
                <RelacionesSi>
                        <RelacionSi condición=”Condicion1”>
                                <Origen>ModuloOrigen</Origen>
                                <Si>ModuloSi</Si>
                                <Sino>ModuloSino</Sino>
                        </RelacionSi>
                        ---------------------------------
                        <RelacionSi condición=”CondicionN”>
                                ----------------------------
                        </RelacionSi>
                </RelacionesSi>

                <RelacionesSiguiente>
                        <RelacionSiguiente>
                                <Origen>ModuloOrigen</Origen>
                                <Destino>ModuloDestino</Destino>
                        </RelacionSiguiente>
                        ---------------------------------------
                        <RelacionSiguiente>
                        ---------------------------------------
                        </RelacionSiguiente>
                </RelacionesSiguiente>

                <RelacionesParalelo>
                        <RelacionParalelo>
                                <Origen>ModuloOrigen</Origen>
                                <Destino>ModuloDestino1</Destino>
                                <Destino>ModuloDestino2</Destino>
                        </RelacionParalelo>
                        ---------------------------------------
                        <RelacionParalelo>
                        ---------------------------------------
                        </RelacionParalelo>
```

```
                    </RelacionesParalelo>
            </ListaRelaciones>
</Descripcion>
```

Code 5.1: Input of Analyze&Discover.

The input of the analyze&Discover functionality consists of an XML document that represents the mapping of the functionality, following the BPMN graphic, in Figure 5.4. Furthermore, a list of Web services fit the input specification of the corresponding module for each activity.

```
<BPMN name="bpmnName">
  <ListaActividades>
    <Actividad id="Actividad1">
              <ListaEntradas>
                <Entrada id="idEnt1"   tipo= "tipoEnt1"> Concepto1 </Entrada>
                                        ----------------------------------------
                    <Entrada id="idEntN" tipo="tipoEntN"> ConceptoN
          </Entrada>
              </ListaEntradas>
              <ListaSalidas>
                    <Salida tipo="tipoSalida">Concepto</Salida>
       </ListaSalidas>
             <Precondicion>Concepto</Precondicion>
             <Efecto>Concepto</Efecto>
             <ListaWS>
        <WS nombre=”ws1”></WS>
        ----------------------------------------
        <WS nombre=”wsN”></WS>
      </ListaWS>
    </Actividad>
 ----------------------------------------------------
    <Actividad id=”ActividadN”>
    ----------------------------------------
    </Actividad>
  </ListaActividades>

  <ListaCompuertas>
    <Compuerta tipo=["XOR"/"AND"] id="idComp1"></Compuerta>
              ----------------------------------------------------
              <Compuerta tipo=["XOR"/"AND"] id="idCompN"></Compuerta>
  </ListaCompuertas>

  <ListaEventos>
    <Evento tipo=[“Inicial”/”Final”]  id="idEvento1"> </Evento>
              ----------------------------------------------------
              <Evento tipo[“Inicial”/”Final”]   id="idEventoN"> </Evento>
  </ListaEventos>

  <ListaConectores>
    <Conector origen="Nombre" destino="Nombre"> Condicion
```

```
    </Conector>
        ----------------------------------------------------
        <Conector origen="Nombre" destino="Nombre"></Conector>
  </ListaConectores>
</BPMN>
```

Code 5.2: Output of BPMN.

- BPMN : Root node
  - *name*: File name
  - **ListaActividades**: Set of diagram activities
    - **Actividad:** Concrete diagram activity
      - *id*: Activity name
      - **ListaEntradas:** Set of inputs the representing model receives.
        - **Entrada:** Input of the module representing a semantic concept it has to receive.
          - *id:* Input identifier.
          - *tipo:* Data type.
        - **Salida:** Output of the module representing a semantic concept that it generates. Output generated by the module representing a semantic concept.
        - *tipo:* Data type.
      - **Precondition:** Precondition that has to be met for the correct function of the module.
      - **Efecto:** Effect the module produces.
      - **ListaWS:** Set of services that adjust to the activity and its domain
        - **WS:** Web service that adjusts to the activity requirements
          - *nombre:* Web service name that could later be used to carry out a search by name in UDDI.
  - **ListaCompuertas:** Set of diagram gate.
    - **Compuerta:** Concrete diagram gate.
      - *tipo:* gatetype. Possible values are "XOR" and "AND".
      - *Id:* gate identifier.
  - **ListaEventos:** Set of diagram events.
    - **Evento:** Concrete diagram event.
      - *tipo:* Event type. Possible values are "Inicial" and "Final".
      - *Id:*Event identifier.

- **ListaConectores:** Set of diagram connectors.
  - **Conector**: Concrete connector of a diagram, it can optionally include a condition in the label content.
    - *origen:* Origin element identifier.
    - *destino:* Outcome element identifier.

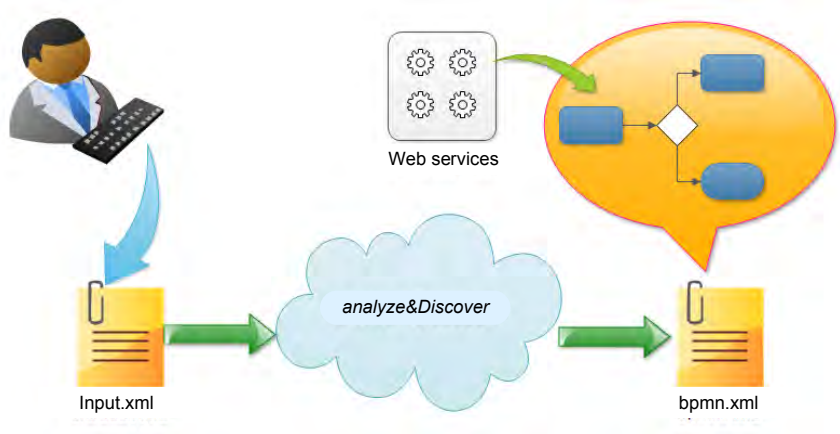

Figure 5.4: analyze&Discover functionality.

composeService

Functionality of our model that receives as input an XML document with a reduced BPMN diagram such that each activity is associated with a Web service. The input format is in the same as Code 5.4, except that instead of an activity it will be associated with a list of services; it is associated with a service of the list. The expression in Code 5.5 must be replaced by the one in Code 5.6 on Code 5.4.

```
<BPMN name="bpmnName">
   <ListaActividades>
      <Actividad id="Actividad1">
        <ListaEntradas>
           <Entrada  id="idEnt1"      tipo=  "tipoEnt1">  Concepto1
           </Entrada>
                      ----------------------------------------
           <Entrada id="idEntN" tipo="tipoEntN"> ConceptoN
             </Entrada>
        </ListaEntradas>
        <ListaSalidas>
           <Salida tipo="tipoSalida">Concepto</Salida>
         </ListaSalidas>
       <Precondicion>Concepto</Precondicion>
       <Efecto>Concepto</Efecto>
       <ListaWS>
           <WS nombre=”ws1”></WS>
           ----------------------------------------
           <WS nombre=”wsN”></WS>
        </ListaWS>
```

```
      </Actividad>
            ----------------------------------------------------
      <Actividad id=”ActividadN”>
      ----------------------------------------
      </Actividad>
   </ListaActividades>

   <ListaCompuertas>
      <Compuerta tipo=["XOR"/"AND"] id="idComp1"></Compuerta>
     ----------------------------------------------------
     <Compuerta tipo=["XOR"/"AND"] id="idCompN"></Compuerta>
   </ListaCompuertas>

   <ListaEventos>
      <Evento tipo=[“Inicial”/”Final”]  id="idEvento1"> </Evento>
     ----------------------------------------------------
     <Evento tipo[“Inicial”/”Final”]   id="idEventoN"> </Evento>
   </ListaEventos>

   <ListaConectores>
      <Conector origen="Nombre" destino="Nombre"> Condicion
      </Conector>
     ----------------------------------------------------
     <Conector origen="Nombre" destino="Nombre"></Conector>
   </ListaConectores>
</BPMN>
<BPMN name="bpmnName">
   <ListaActividades>
      <Actividad id="Actividad1">
        <ListaEntradas>
           <Entrada  id="idEnt1"      tipo=  "tipoEnt1">  Concepto1
           </Entrada>
                      ----------------------------------------
           <Entrada id="idEntN" tipo="tipoEntN"> ConceptoN
             </Entrada>
        </ListaEntradas>
        <ListaSalidas>
            <Salida tipo="tipoSalida">Concepto</Salida>
          </ListaSalidas>
       <Precondicion>Concepto</Precondicion>
       <Efecto>Concepto</Efecto>
       <ListaWS>
            <WS nombre=”ws1”></WS>
            ----------------------------------------
            <WS nombre=”wsN”></WS>
         </ListaWS>
      </Actividad>
            ----------------------------------------------------
      <Actividad id=”ActividadN”>
      ----------------------------------------
      </Actividad>
   </ListaActividades>

   <ListaCompuertas>
      <Compuerta tipo=["XOR"/"AND"] id="idComp1"></Compuerta>
```

```
    ------------------------------------------------------
    <Compuerta tipo=["XOR"/"AND"] id="idCompN"></Compuerta>
  </ListaCompuertas>

  <ListaEventos>
     <Evento tipo=[“Inicial”/”Final”]  id="idEvento1"> </Evento>
    ------------------------------------------------------
    <Evento tipo[“Inicial”/”Final”]   id="idEventoN"> </Evento>
  </ListaEventos>

  <ListaConectores>
     <Conector origen="Nombre" destino="Nombre"> Condicion
     </Conector>
    ------------------------------------------------------
    <Conector origen="Nombre" destino="Nombre"></Conector>
  </ListaConectores>
</BPMN>
```

Code 5.4: composeService.

```
<ListaWS><WS nombre=”ws1”></WS>-------</ListaWS>
```

Code 5.5: ListaWS.

```
<WS nombre=”ws1”></WS>
```

Code 5.6: WS.

Starting from the input document we will carry out a composition of the Web services in a unique Web service through BPEL (for a detailed description see Section 5.4.3).

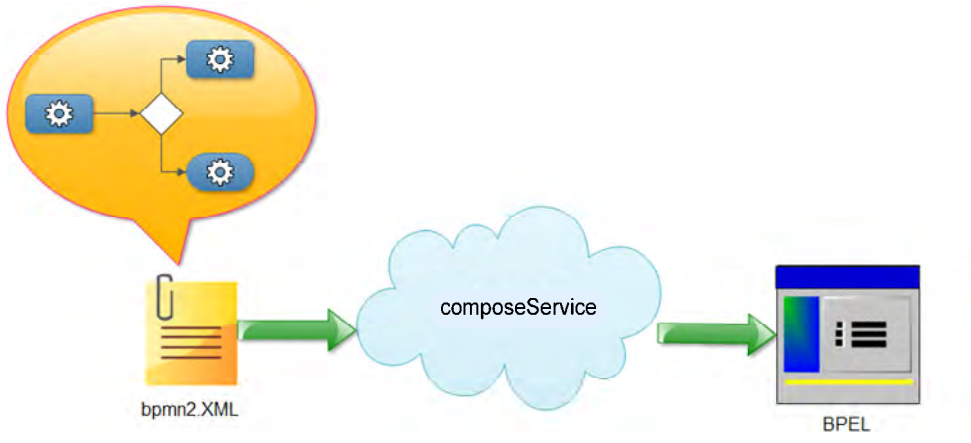

Figure 5.5: composeService functionality.

5.3 Web services

The main aim of the proposed model consists of re-using web services to compose new services (automatically). For the representation of services we have opted for the SOA paradigm (*Service Oriented Architecture)* with the most common description, register and messaging protocols:

- SOAP messaging: SOAP is a protocol based on XML for messaging and invocating remote procedures (RPC). Instead of defining a new protocol, SOAP works on existing protocols such as HTTP y SMTP, among others.
- WSDL description: WSDL is an XML document that describes how to access a web service and what operations it offers. It defines an abstract description with respect to the exchanged messages in a service interaction.
- UDDI registry: UDDI specifications (Universal Description, Discovery, and Integration) offer users a systematic and unified form to find service supplies through a service registry that looks like a "telephone directory" of web services.

The choice of Web services as a basis for the reuse of software components is due to the modular nature of (Web services) in which the service interface of their implementation and their ability to dynamically carry out service links, decouple. Moreover, we have opted for classic protocol alternatives (SOAP, WSDL, UDDI) as they offer a basic service register mechanism, a description of interactions among the different services and a complete and robust messaging protocol.

Nevertheless, SOAP is considered a slower technology compared to other *middleware* technologies (Olson and Odbuiji, 2002) and it may lead to security problems due to the introduction of malicious code. Having taken these cases into account and, given that the model is included in a cloud environment with good load balance and multiple virtual machines management, the problem of velocity is considerably reduced.Moreover, studies were carried out on methods to increase their performance (Tekli *et al*., 2012). With respect to SOAP security, there are different alternatives to protect the system when facing malicious attacks (Pinzón *et al.,* 2011) (Wei *et al*., 2012).

The discovery engine of service and requirement analysis is developed through a multiagent system based on virtual organizations, the interaction between multi-agent system technologies and SOA has been implemented in various projects with good results, such as: FUSION@ (Tapia *et al.,* 2008) or the work addressed in (Huhns, 2012).

Figure 5.6 shows a diagram with the main components used in building Web services in the platform.

- Ontologies: Set of ontologies present in a platform. They represent the semantic knowledge that will be associated with the Web services. Moreover, when using an annotation WSDL-S system, these ontologies can be written in any language.
- UDDI registry: Consists of a UDDI registry where the different Web services that are present in the platform are published. The WSDL

descriptions of the corresponding Web services will be linked to the different registry inputs.

- MAS: For each ontological concept present in the ontologies there is an associated agent that will assist in the discovery process of the Web services for semantics content. Every agent will be associated to a set of UDDI inputs corresponding to the Web services that make use of the ontological concept represented by the agent. In Section 5.3.2 we offer details on how these associations are carried out.
- WSDL: WSDL documents that describe the Web services. These documents will have semantic annotations following the WSDL-S specification. These specifications will allow referencing semantic content in the Web services (Section 5.3.1).

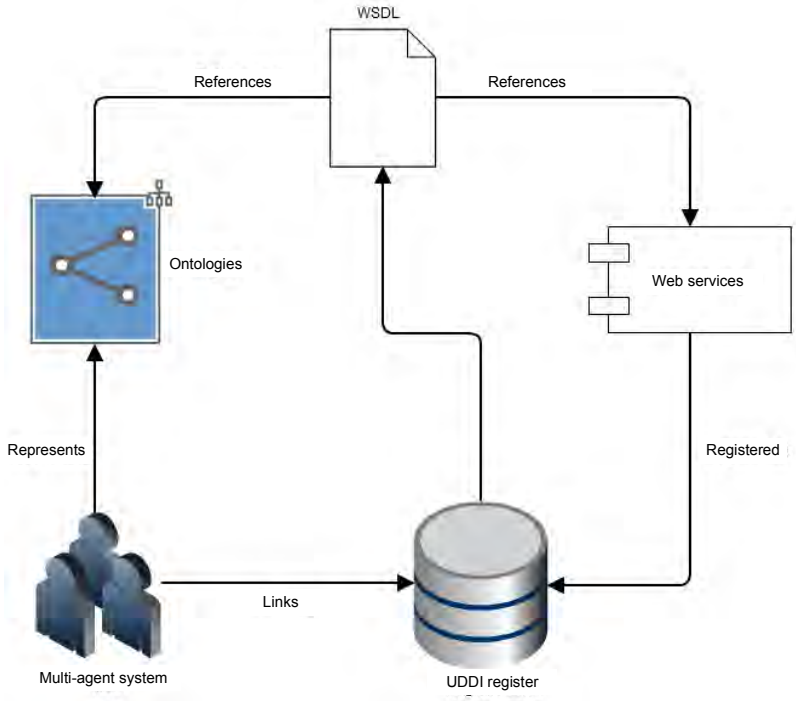

Figure 5.6: Web service structure.

5.3.1 Semantics for Web services

The next point to consider is the integration of semantic knowledge into *web* services. The project OWL-S (W3C, 2003), previously DAML-S (DAML-S, 2012) (the most used to provide semantics to web services), defines a domain ontology for Web services. This ontology provides tags that can be used to describe Web services. The ontology consists of three sub-ontologies: *service profile, service grounding* and *process model*. This option can be excessively complex depending on the application area (Miller *et al*., 2004).

By considering different options to provide semantic knowledge to *web* services, WSDL-S (Akkiraju *et al*., 2005) appears as the most suitable for the proposed platform (Garcia, 2011) due to the fact that:

- Users can incrementally describe all details, both semantic and at the level of operations in WSDL, a language that is familiar to the developer community.
- By outsourcing the domain semantic models, WSDL-S remains independent of the ontology representation language to be used. This allows *web* service developers to annotate their services with the ontological language of their choice (that is, UML, OWL, etc.). This is an additional advantage, because reusing models of the existing domain, which are expressed in modeling languages as UML, can accelerate the incorporation of semantic annotations.

The WSDL document is the anchor point to describe Web services. Based on the descriptive power of WSDL, a mechanism to annotate the capabilities and requirements of Web services with semantic concepts was provided by referencing a semantic model, Figure 5.7. Mechanisms are used to specify and annotate preconditions and effects in the Web service WSDL files. These preconditions and effects, along with the annotations of inputs and outputs allow the service discovery process to be automated (Akkiraju *et al*., 2005). This makes it possible to link the WSDL description to Web service semantic knowledge in such a way that this link is independent of the language of constructing Web service ontologies.

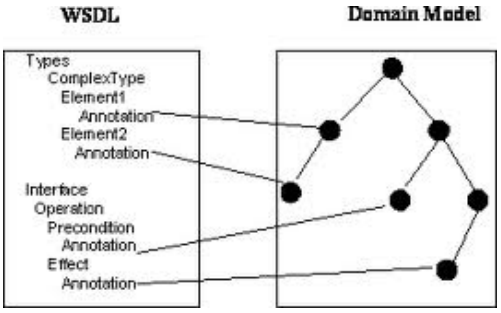

Figure 5.7: Annotations WSDL to add WSDL-S.

(Akkiraju *et al*., 2005) describes the attributes and extensible elements allowing the WSDL description to be associated with the semantic content of the Web service: *modelReference* (extension attribute), *schemaMapping* (extension attribute), *precondition* (extension attribute), *effect* (elements) and *category* (extension attribute). These annotations are designed for WSDL 2.0, however, in (Akkiraju *et al*., 2005) some minor changes to fit WSDL 1.1 have been indicated. The above attributes and elements can be adapted to WSDL 1.1 without modification. However, in certain cases they are applied on different elements.

Input and Output: Attributes *schemaMapping* and *modelReference* can be added to element <part> (behind the *message* element) to specify an annotation for *input* or *output* that applies to the whole message. These elements are part of structure <portType> in WSDL 1.1, which corresponds to the structure *interface* in WSDL 2.0.

Preconditions and Effects: Attributes *precondition* and *effect* are descendants of the element *operation* in <portType>. *Operation* is an extensible element in http://schemas.xmlsoap.org/wsdl/. However, if you use an old version of *XML schema*

for WSDL, item *operation* may not be extensible. It is therefore recommended to use element <documentation> to capture the semantics of preconditions and effects.

```
<portType name="PurchaseOrder">
<operation name="processPurchaseOrder"parameterOrder="billingInfoOrderItem">
  <documentation>
    <wssem:precondition name="PreExistingAcctPrecond"
     wssem:modelReference="POOntology#AccountExists">
    <wssem:effect name="ItemReservedEffect"
     wssem:modelReference="POOntology#ItemReserved"/>
  </documentation>
  <input message="tns:processPurchaseOrderRequest"
       name="processPurchaseOrderRequest"/>
  <output message="tns:processPurchaseOrderResponse"
       name="processPurchaseOrderResponse"/>
</operation>
</portType>
```

Code 5.1: Annotations in WSDS 1.1.

Code 5.7 obtained from (Akkiraju *et al*., 2005) is an example of annotations on WSDL- 1.1. The *operation* element has been annotated with a precondition and an effect.

To model the semantic content necessary for the appropriate performance of the platform (and considering annotations to be carried out on WSDL-1.1), the following notations of WSDL-S will be used:

- *Preconditions and effects*: We associate semantic content on an operation of a Web service to express what this operation makes and the preconditions that must be taken for its invocation. The annotations will be carried out on tags <documentation> belonging to <operation>. The *precondition* annotation will be used for the preconditions and *effect* for the performance of the operation. In both cases *modelReference* should be included to link the corresponding semantic content.
- *Data types*: Both inputs and outputs of an operation may be associated with an element of the ontological domain. The annotations are carried out on tags <part> corresponding to the elements *modelReference* to link the semantic concept and *schemaMapping* for complex types.

Code 5.8 shows an example on how to perform annotations necessary for the WSDL-S description. A Web service called airline has an operation *getPlazas* which, using a flight identifier, returns an integer number representing the number of seats for that flight. The annotations(in bold) are the following:

- Precondition of operation *GetPlazas*: To run getPlazas, it would be passed as an argument; a flight identifier, which must belong to an existing flight. To annotate this precondition annotation <wssem:precondition name="ExisteVueloCond" ssem:modelReference="AeroOnt#ExisteVuelo"> is used in tag <documentation>.

- Effect of operation *GetPlazas*: The effect occurred in the operation is annotated as follows <wssem:modelReference name="SeObtienePlazas" wssem:modelReference= "AeroOnt#PlazasObtenidas"/>
- Annotation of types: The operation *getPlazas* receives a flight identifier as an argument, and returns a number that represents the number of available seats.
    - Flight identifier: On the tag <types> where element *idVuelo* is defined, **wssem:modelReference= "AeroOnt#Vuelo"** is added.
    - Free seats: Tag <types> where element *return* of *getPlazasResponse* message is defined, **wssem: modelReference = "AeroOnt # PlazasLibres"** is added.

```
<types>
 <xsd:schema>
  <xs:element name="getPlazas" type="tns:getPlazas"/>
  <xs:element name="getPlazasResponse"
         type="tns:getPlazasResponse"/>
  <xs:complexType name="getPlazas">
           <xs:sequence>
            <xs:element name="idVuelo"
             type="xs:string"minOccurs="0"
             wssem:modelReference="AeroOnt#Vuelo"/>

           </xs:sequence>
  </xs:complexType>

  <xs:complexType name="getPlazasResponse">
   <xs:sequence>
            <xs:element name="return" type="xs:int" minOccurs="0"
             wssem:modelReference="AeroOnt#PlazasLibres"/>
   </xs:sequence>
  </xs:complexType>
 </xsd:schema>
</types>

<message name="getPlazas">
 <part name="parameters" element="tns:getPlazas"/>
</message>

<message name="getPlazasResponse">
 <part name="parameters" element="tns:getPlazasResponse"/>
</message>

<portType name="Aerolinea">
 <operation name="getPlazas">
  <documentation>
   <wssem:precondition name="ExisteVueloCond"
    wssem:modelReference="AeroOnt#ExisteVuelo">
         <wssem:effect name="SeObtienePlazas"
    wssem:modelReference="AeroOnt#PlazasObtenidas"/>
  </documentation>
```

```
  <input wsam:Action=
              "http://Aerolinea.com/Aerolinea/getPlazasRequest"
       message="tns:getPlazas"/>
  <output wsam:Action=
        "http://Aerolinea.com/Aerolinea/getPlazasResponse"
         message="tns:getPlazasResponse"/>
 </operation>
</portType>
```

Code 5.8: Example of WSDL-S airline.

5.3.2 Register system

UDDI is an initiative aimed at creating a network of Web service registries in the Internet by allowing the discovery of Web services in a simple, fast and dynamic way (Snirinivasan *et al*., 2004), Figure 5.8. UDDI is specified by (IBM, 2001):

- *businessEntity*: Contains information about the company that publishes the service.
- *businessService*: Description of the Web service.
- *bindingTemplate*: Contains technical information to determine the entry point and specifications for Web service invocation.
- *tModel*: Provides a reference system to assist in the process of Web service discovery and acts as a technical specification.

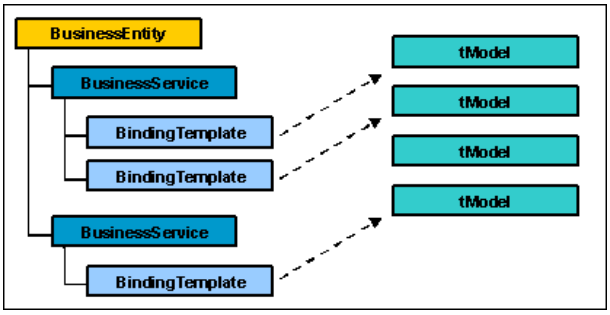

Figure 5.8: UDDI content.

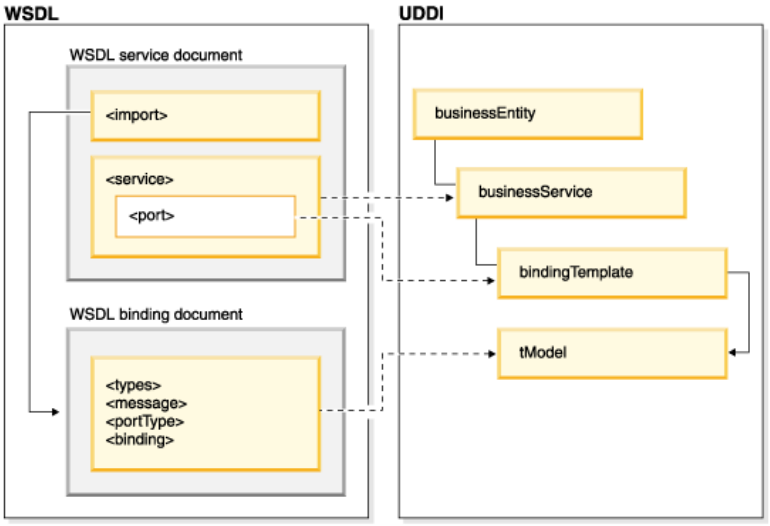

Figure 5.9: Link UDDI/WSDL

Figure 5.9 shows how to make the association between lines of register UDDI and the WSDL specification of different Web services. But one problem with UDDI is that it only supports keyword-based searches of companies, services and *tModels*. For example, it is possible to find all services that contain a specific value associated with *tModel*. Because the search in UDDI is restricted to keywords, neither inference methods nor flexible association can be performed. The UDDI limitation is the lack of an explicit representation of Web service capabilities. The result is that UDDI supports the location of the Web service key information once the existence of such a service is known, but it is impossible to locate a service based solely on its functionality (Sycara *et al*., 2003).

Several authors have made proposals to address this deficiency of UDDI. In (Luo *et al.*, 2006) the authors propose a model in which the whole ontology is imported in the UDDI register, where every ontological concept and every property are represented in a separated *tModel,* which can be referenced individually. Another similar proposal has been given in (Srinivasan *et al*., 2004).

In order to extend the search of Web services to semantic content in the proposed model, we adopt a posture similar to those described above, in the sense of mapping the ontological information. However, instead of importing the whole ontology to the UDDI register as in (Luo *et al*., 2006), our model uses a multi-agent system based on virtual organizations able to represent this ontology from the user's initial specifications. In this representation, each agent models an ontological concept and will be connected (lines of communication) to the other agents that model equivalent ontological concepts. In addition, each agent will have a list of links to UDDI entries, thus identifying Web services that contain the concept that the agent represents (Figure 5.10). This way, the multi-agent system represents the whole ontology of our proposal as a graph. This process is specified in detail in Section 5.4.1.

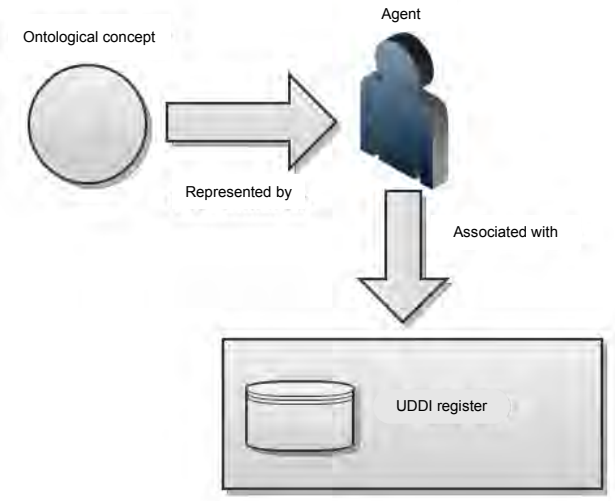

Figure 5.10: Semantic register

Summarizing:

- Agents were created from the ontological information in order to represent the expressed concepts. In addition to representing concepts, these agents also include the properties and attributes of the ontological equivalent from which the mapping is performed.
- Each agent will be associated with different entries of the UDDI register such that:
    - The Web service has an associated ontological content.
    - The Web service is registered by UDDI.
    - For each ontological concept used in the service, the agent representing such a concept will be associated with the line of the UDDI register to locate the service.
- Agents can be associated with other agents by representing different ontological relationships such as:
    - Equivalence between concepts.
    - Subclass relative to other concepts.

Example: Let us suppose there are two Web services "Serv1" and "Serv2". "Serv1" uses ontological concepts "Ont1, Ont2" and "Ont3" and is registered in "Line1" of UDDI. "Serv2" uses ontological concepts "Ont2" and "Ont4" and is registered in "Line2". The representation of this situation is shown in Figure 5.11, where the agent represents "Ont2" is associated with UDDI lines "Line1" and "Line2" (for services “Serv1” and “Serv2”).

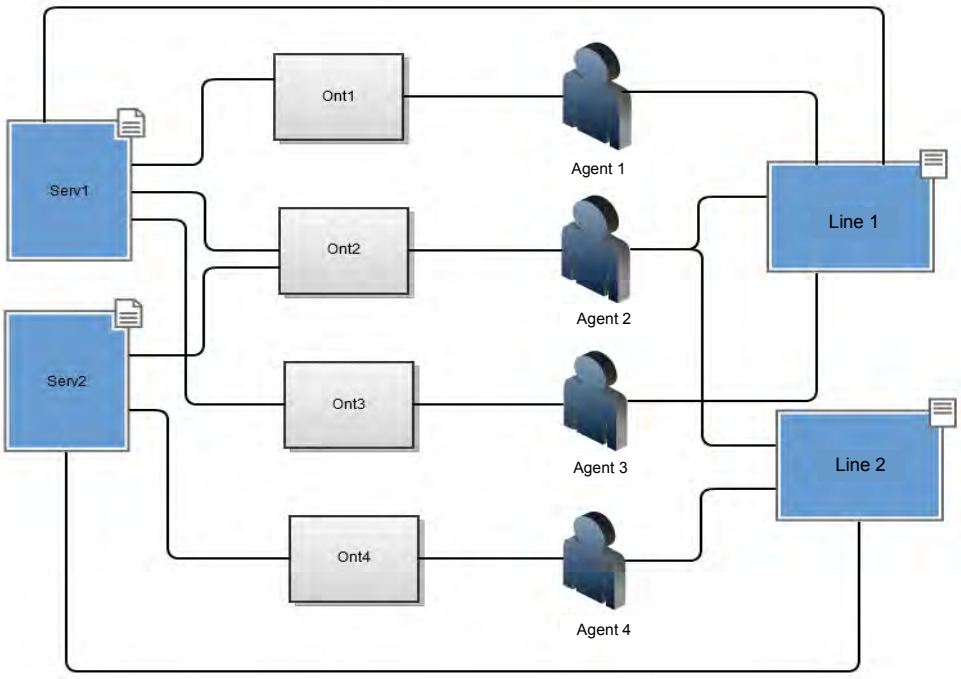

Figure 5.11: Example of ontological mapping.

Note that we are proposing an architecture where the Web services to be used in the service composition have been implemented in the repository of the same architecture. This means that we can only keep an internal register of Web services without using the UDDI register. However, we have chosen to also maintain the UDDI register since it provides a more universal solution to our proposal. This allows the final system an increased scalability and maintainability.

5.4 Multi-agent system (MAS)

This section presents the structure and performance of the multi-agent system deployed in the platform, which consists of three subsystems:

- **Analysis System:** This system is responsible for analyzing the requirements introduced by the user in computationally processable information.
- **Search System:** This system is responsible for finding the services that meet a description received from the input.
- **Composition System:** This system will be responsible for performing the BPEL composition of the specified services.

The *manager* agent acts as the link between the subsystems of search and analysis and it is responsible for coordinating the operations between them. The search and analysis systems are designed to have a black-box behavior; that is, their operations do not depend on other systems and there are no direct communication lines between

them. All relationships necessary to reach a complex goal will be organized by the *manager* agent. Thus, we can assume each subsystem is a separate module, so they can be modified and adapted separately without interfering with the performance of the remaining systems.

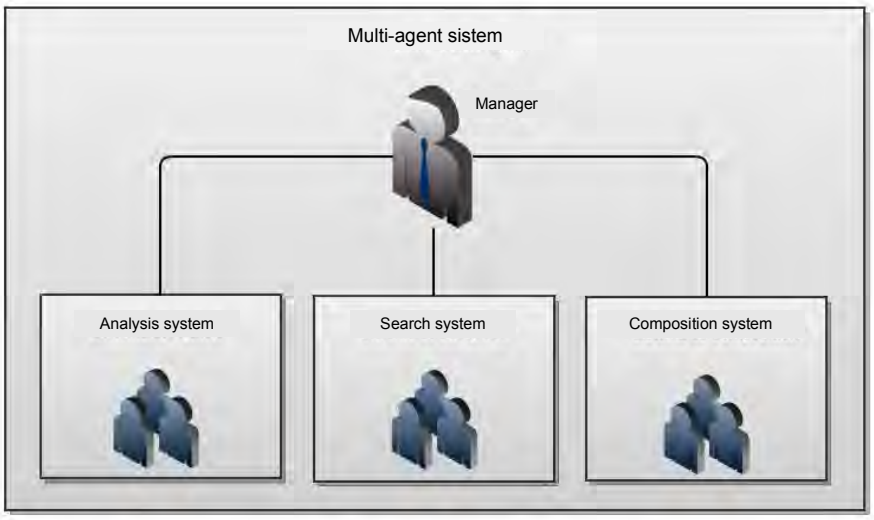

Figure 5.12: Architecture of the multi-agent system.

The next subsection describes the operation of each system by indicating their motivations and presenting the advantages they bring over other design options.

5.4.1 Analysis system

This system is responsible for managing the requirement input in such a way that it is computable and able to retrieve enough semantic information to perform a search and subsequent composition. It consists of a coordinator agent (*analysis coordinator agent*).

The requirements by the user are input via a graphical assistant, so that an assisted implementation for each module created by the user will be made. For each module, the assistant will guide the user in:

- *Functionality definition*: The functionality of the module is introduced in text format by using a list of semantic concepts that should be associated with the module.
- *Ontological domain definition*: In the definition of each module, the assistant allows choosing a domain name from a dropdown list with the available domains (by default, at least one domain will be available). The list of domain names is determined by the diversity of available Web services. This means that each Web service in the architecture has annotated the domain to which it belongs.
- *Inputs*: The user will determine the inputs that must receive the module indicated as representing the semantic concept (the concept can be associated with a basic data type or composed).

- *Outputs*: Similar to the input but specifying the output produced by the module.
- *Interconnection*: The assistant will guide the user on how the module is interconnected with respect to the previous modules. In this sense we have the following options for flow control between modules:
    - [ModuleX] If [Condition] Then [Module]
    - [ModuleX] If [Condition] Then [Module1] Else [Module2]
    - [ModuleX] Parallel Output [Module1], [Module2]
    - [Module1] Follows [Module2]

Once the assistant has represented the information as a diagram flow through interrelated modules, the information is structured on an XML document. The assistant then invokes the functionality (or option) analyze&Discover that takes the XML document as input. analyze&Discover starts agent-based analysis subsystem through the *manager* agent, which invokes the agent that will be responsible for interpreting the information and relationships of each module stored in the XML document, Figure 5.13. Therefore the *analysis coordinator* agent is responsible for interpreting the information in the XML document and creates an agent for each module, representing its concept. Moreover, the same agent (*analysis coordinator*) creates lines of communication (conditional flow control) between the built agents by basing them on the existing relationships between the modules of the XML document. This way, the diagram of flow representing the ontology can be modeled as a graph.

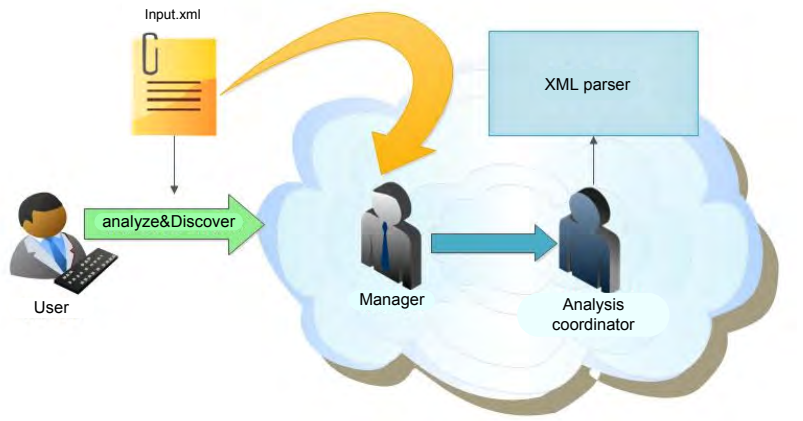

Figure 5.13: Analysis system as agent-based virtual organizations.

Each agent created by the *analysis coordinator* agent will manage the following information (see Figure 5.14):

1. *Inputs*: Each input will have an associated type and semantic concept.
2. *Output*: The output is specified by the type and the concept it represents.
3. *Description*: Concept associated with the operation that the module carries out; it also includes its domain.
    a. *Preconditions*: List of preconditions that must be met to ensure appropriate performance of the module.
    b. *Effects*: actions running in the module.

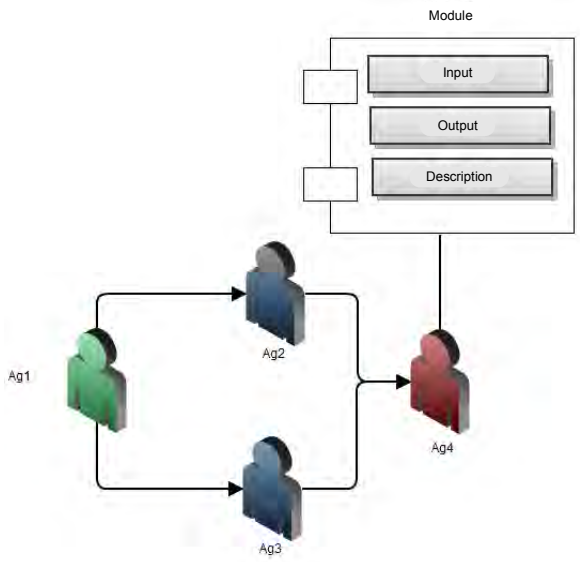

Figure 5.14: Relationship Module/Agent.

The relationships between the modules must meet the following properties:

1. There must be an *initial* module
2. There must be one or more *end* modules

After the construction of the agent-based subsystem to represent the ontology, a BPD (Business Process Diagram) of graphic standard BPMN (Business Process Management Notation) will be built. In this diagram each activity is associated with an agent of the ontology. The flow control of the BPD diagram will be associated with the existing lines of communication between agents. The aim is for the final content the diagram to be projected to a BPEL (Business Process Execution Language) file by making a faithful and complete mapping of the diagram. Both the features that the diagram may contain and the process of mapping done from the *analysis system* are presented below.

1. *Activities*: An activity is the mapping of an agent.
2. *Gates*:
    a. *XOR gate* (Figure 5.15): It has been obtained from the relationship "[ModuleX] If [Condition] Then [Module1] Else [Modulo2]" specified by the user. If the relationship is "[ModuleX] If [Condition] Then [module]", it will then be represented in the same way but adding only a flow of execution. The mapping will be carried out by adding this gate and the normal connectors connecting the activities to the gate, according to the agent who initiates the relationship.
    b. *AND gate* (Figure 5.16): Corresponds to the relationship "[ModuleX] Parallel Output to [Module1], [Module2]". The agent representing "ModuleX" will connect to an AND gate with the parallel execution flows towards "Module1" and "Modulo2".
3. *Events*:

a. *Start* and *End*: Agents associated with brands "Start" and "End", will create the corresponding *start* and *end* events and connect with these agents.

4. *Connectors*:
    a. Connectors that are not specified above will be obtained from the relationship "[Module1] Follows [Modulo2]".
    b. If two relationships "Follows" have been obtained from the same predecessor module and the successor is distinct, then an AND gate is added by connecting the successors and go to the predecessor.
    c. Example (Figure 5.16):
        i. Module3 Follows Module1
        ii. Module3 Follows Module2

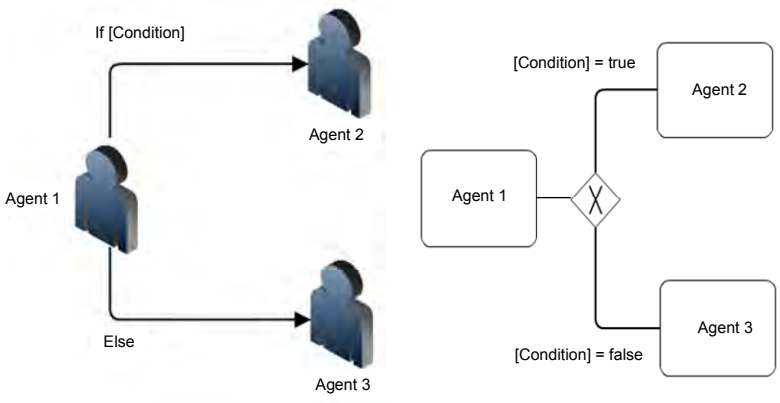

Figure 5.15: Mapping of a gate XOR.

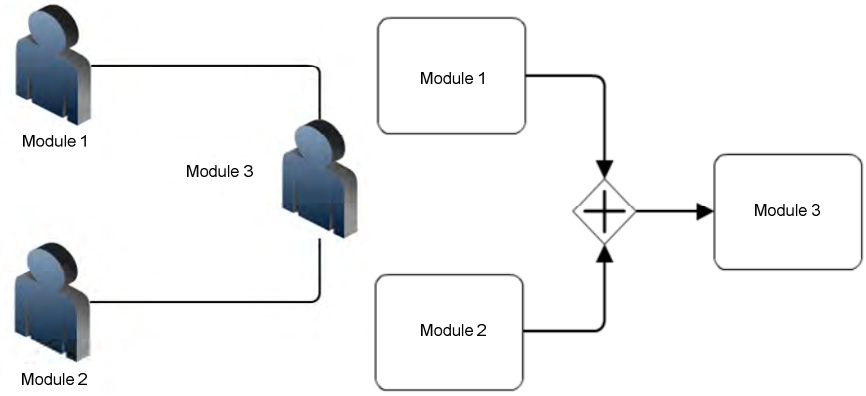

Figure 5.16: AND gate as a join of flow.

There may be cases where the proposed relationship system does not express some cases of flow control allowed by BPMN. As a solution, the definition of virtual modules does not represent a behavior or concept which will only be allowed to introduce relationships that otherwise would not be made.

For example, we want connects the output of a parallel flow to a XOR gate. To represent this situation, we need a virtual module *ModuleV* that does not represent

any behavior. The relationships required to express this situation are described below and in Figures 5.17 – 5.18:

1. ModuleV Follows Module1
2. ModuleV Follows Module2
3. ModuleV If [Condition] Then ModuleX Else ModuleY

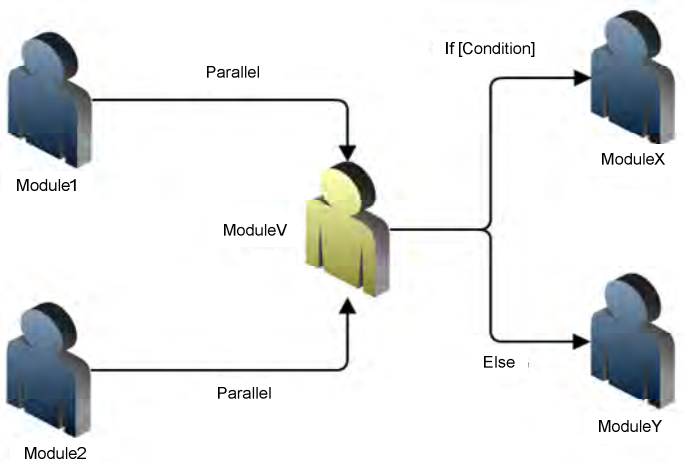

Figure 5.17: Agent-based virtual organization.

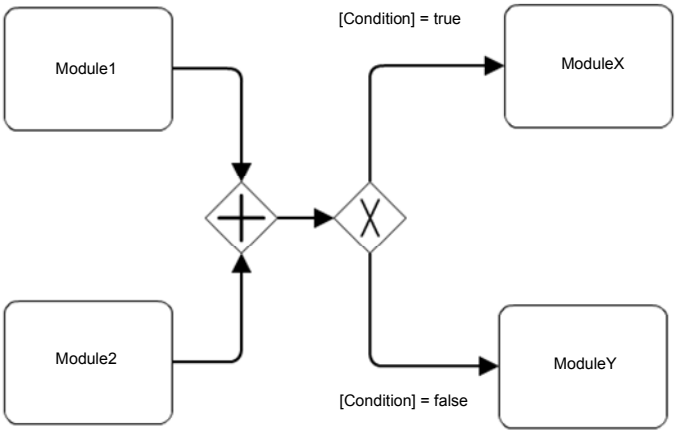

Figure 5.18: Mapping of a virtual module.

We see a complete example where the following modules *Module1, Modulo2, Module3, Module4, Module5, Module6 and Module7* have been defined. The initial and final modules are Module1 and Module7. The following relationships of interconnection between modules have been defined:

1. Module1 Parallel Output to Module2, Module3
2. Module2 If [Condition] Then Module4 Else Module5
3. Module6 Follows Module4
4. Module6 Follows Module5
5. Module7 Follows Module6
6. Module7 Follows Module3

The multi-agent system built by the *system analysis* for this input is shown in Figure 5.19. From this multi-agent system we are able to obtain the mapping to BPMN as displayed in Figure 5.20. The output of this system will be an XML file (BPEL) specifying both the structure of the diagram obtained in the previous step, and the semantic information collected from the user. Code 5.9 shows the different activities and attributes that the file contains:

```
<BPMN name="bpmnName">
   <ListaActividades>
      <Actividad id="Actividad1">
       <ListaEntradas>
          <Entrada  id="idEnt1"      tipo=  "tipoEnt1">  Concepto1
          </Entrada>
                    ----------------------------------------
          <Entrada id="idEntN" tipo="tipoEntN"> ConceptoN
            </Entrada>
       </ListaEntradas>
       <ListaSalidas>
          <Salida tipo="tipoSalida">Concepto</Salida>
        </ListaSalidas>
      <Precondicion>Concepto</Precondicion>
      <Efecto>Concepto</Efecto>
      <ListaWS>
          <WS nombre=”ws1”></WS>
          ----------------------------------------
          <WS nombre=”wsN”></WS>
       </ListaWS>
    </Actividad>
          ------------------------------------------------------
    <Actividad id=”ActividadN”>
    ----------------------------------------
    </Actividad>
   </ListaActividades>

   <ListaCompuertas>
      <Compuerta tipo=["XOR"/"AND"] id="idComp1"></Compuerta>
    ----------------------------------------------------
    <Compuerta tipo=["XOR"/"AND"] id="idCompN"></Compuerta>
   </ListaCompuertas>

   <ListaEventos>
      <Evento tipo=[“Inicial”/”Final”]  id="idEvento1"> </Evento>
    ----------------------------------------------------
    <Evento tipo[“Inicial”/”Final”]   id="idEventoN"> </Evento>
   </ListaEventos>

   <ListaConectores>
      <Conector origen="Nombre" destino="Nombre"> Condicion
      </Conector>
    ----------------------------------------------------
    <Conector origen="Nombre" destino="Nombre"></Conector>
   </ListaConectores>
</BPMN>
```

Code 5.9: Activities and attributes included in the BPEL file (XML).

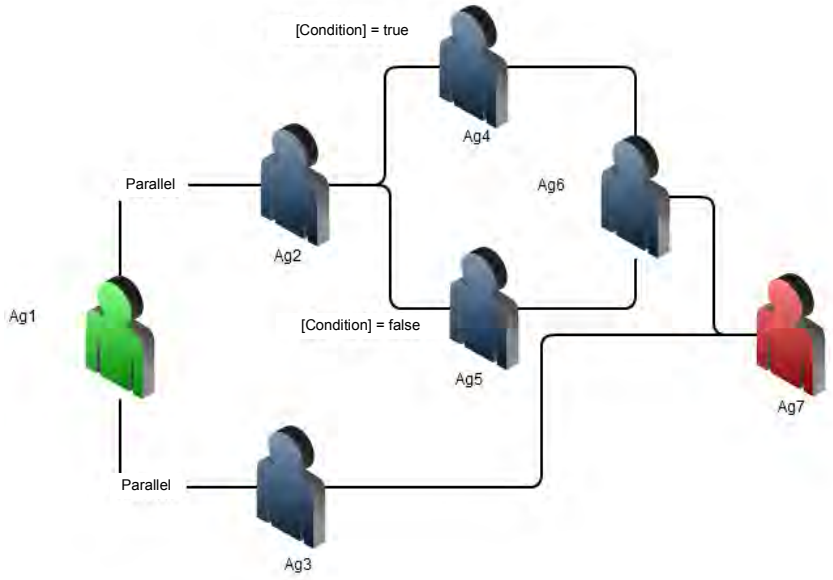

Figure 5.19: A multi-agent system description.

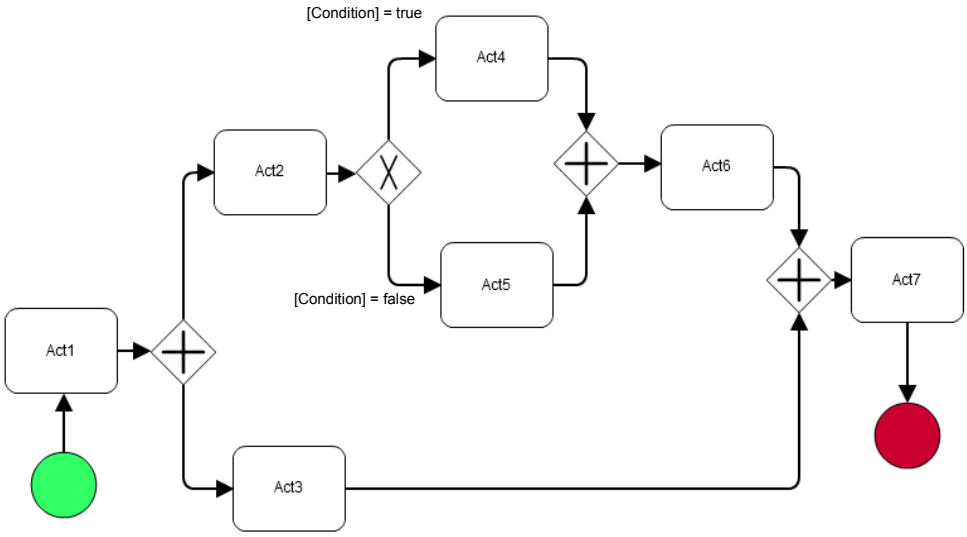

Figure 5.20: Mapping to standard BPMN from the multi-agent system given in Figure 5.19.

5.4.2 Search system

The *search system* is responsible for finding a list of Web services from the semantics and format specifications received from the *manager* agent (which in turn comes from *analysis system*). The discovery process is performed once for each received input module, and returns a list of Web services fitting the provided description. Figure 5.21 shows the agents of the *search subsystem* and its main connections.

1. *Search coordinator*: Receives as input the description of a module (via the agent representing the concept associated with the module), which has been provided by the *manager* agent. Therefore, it is responsible for coordinating the whole process for finding Web services. It returns Web services that fit the description of the input.
2. *Semantic coordinator*: Receives the semantic concepts that the Web service must have. For each concept, it will send a message to the *localizer* agent (that belongs to *register subsystem*), which will return the list of UDDI registers associated with the services using that concept. It should be noted that the web services to be used in this proposal are implemented in the Web service repository of our platform. This means that before publishing them in the UDDI register, the WSDL-S (in the WSDL files) annotations have been made to provide them with semantics. Thus, we already know that these services exist (we know their names and characteristics) and once published in the UDDI register, we can easily locate them. This implies that it would be enough to request the UDDI register each file WSDL of registered Web services by our model.
3. *Checking coordinator*: Receives as input the WSDL file of a Web service, and the conditions it must meet. It communicates with the *checking system* to determine whether the Web service meets the specified requirements. It will return the checking result to the *search coordinator* agent.

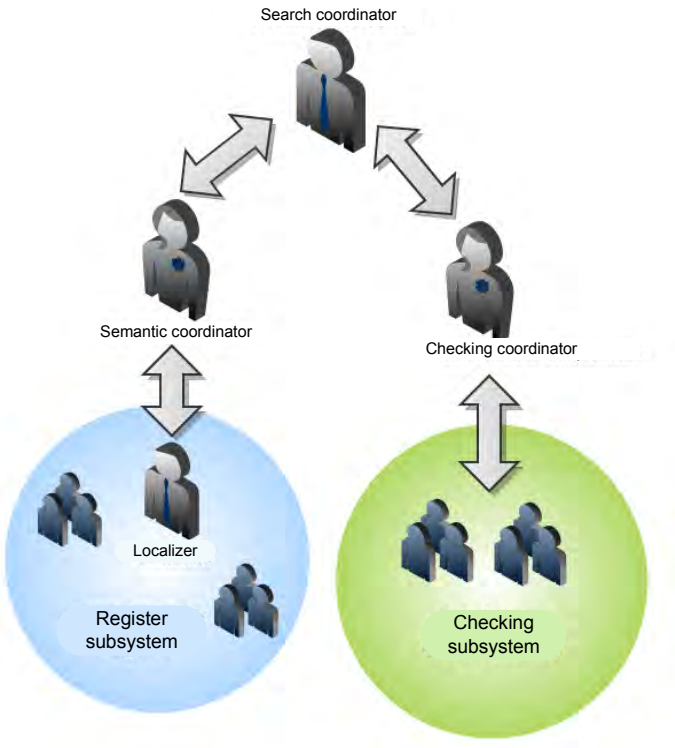

Figure 5.21: virtual organization of the *search system*

The first part of the search process consists of finding Web services associated with a semantic content, Figure 5.22. To do this, the *search coordinator* agent communicates with the *semantic coordinator* agent, which will ask the *localizer* agent for the list of entries to the UDDI register with Web services implementing the specified concept. This process is repeated once for each semantic concept. When the *semantic coordinator* agent has found the services associated with each concept, it will carry out a comparison to eliminate the services associated with all concepts and will, finally, return them as a result to the *search coordinator* agent. Note that the fact of having a *domain* attribute in the ontological concepts allows us to classify Web services into groups, so that the search for them by agent *localizer* will be much more efficient because it is focused on a domain (on a subset of services) and not the whole Web service repository. The next step consists of using the list of registers UDDI of the above step to verify that the services associated with each line UDDI meet the format constraints.

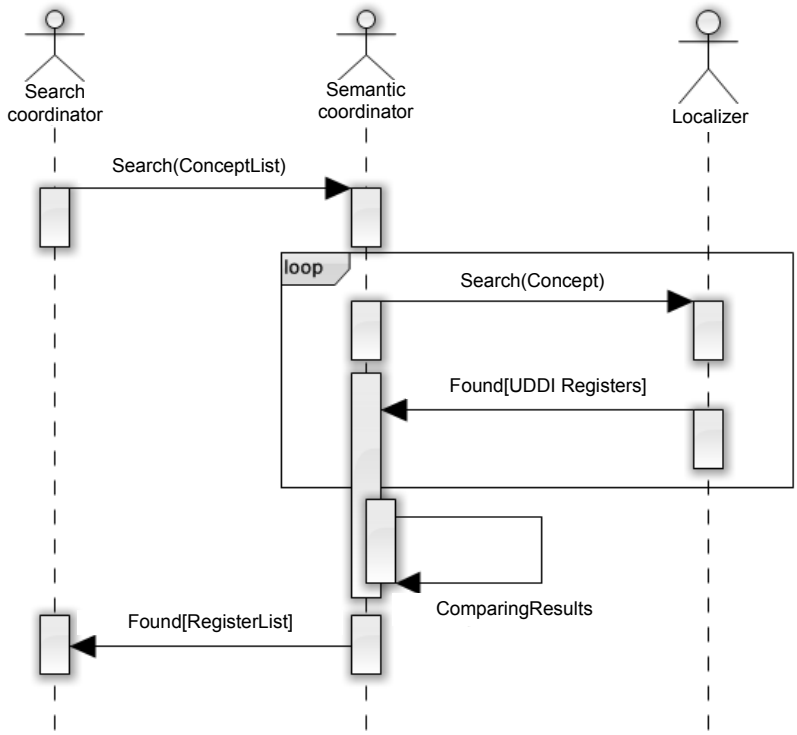

Figure 5.22: Semantics search.

Checking subsystem

The *search system* consists of a subsystem (*checking system*) that will be responsible for checking that the previously obtained Web services meet the implementation constraints imposed by the user (Figure 5.23). It has three stages:

1. *Input checking*: Determines whether the input format of the Web service holds the description given by the user.
2. *Output checking*: Determines whether the output format of a Web service holds the description given by the user.

3. *Checking of the process*: Determines whether the exchange of SOAP message of the Web service is coherent with the description given by the user.

The checking process carried out by the *checking subsystem* is shown in Figure 5.24 and consists of the following steps:

1. The *search coordinator* agent receives the list of entries to the UDDI register from the *semantic coordinator* agent.
2. It sends a message to the *coordinator comparison* agent to check the validity of the services along with the descriptions given by the *analysis coordinator* agent.
3. For each input in the list, the *comparison coordinator* agent does the following:
    a. Obtain document WSDL of the Web service from the UDDI register.
    b. Send a message to the input, output and process *checking systems*, with the WSDL document and the constraints for each system.
    c. Once the responses from the three systems have been received, it verifies that all of them have been approved by these systems. If so, it marks the Web service as valid.
4. The *comparison coordinator* agent sends a message to the *search coordinator* agent with the services that have overcome the checking process.

Although the processing load can be considered high since the multi-agent system is on a *cloud system*, this type of processing will be performed in a distributed way, which allows for a quick response.

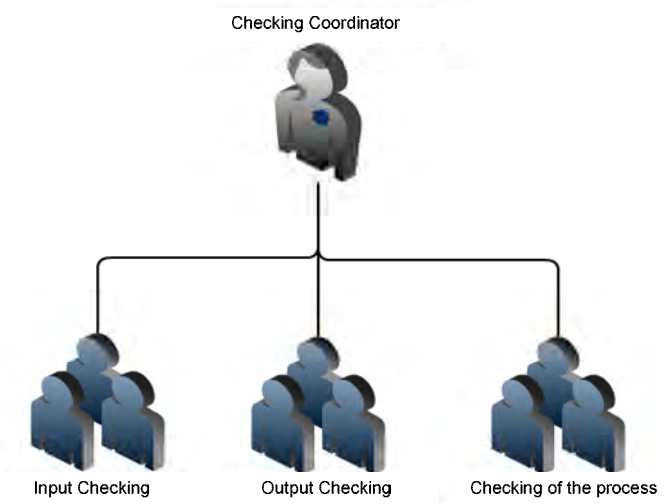

Figure 5.23: Checking system.

Register subsystem

As mentioned in Section 5.3.2, the ontological information will be mapped by a subsystem of agents in a multi-agent system platform. Consecuently, ontologies are usually represented by graphs where nodes represent concepts and edges are the ontological relationships between these concepts.

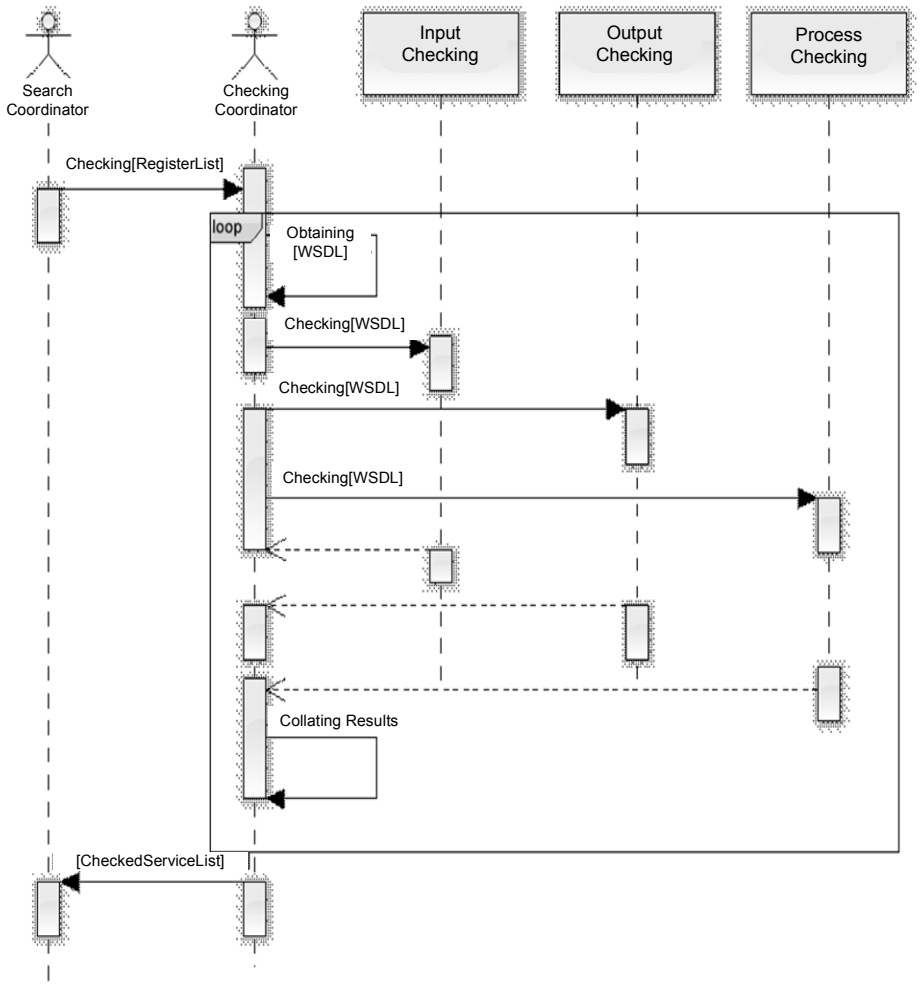

Figure 5.24: Checking process.

To map the ontological knowledge, an agent will be created for each concept, representing in this way, the relationships between concepts through the use of communication lines between the different agents. By using this mapping we can perform the same operations of inference that can be made on the ontology. Each agent has the following structure:

1. The ontological concept it represents.
2. The domain of the ontological concept.
3. Communication lines:
    a. Destination agent.
    b. Line description.
4. List of links to lines of the UDDI register.

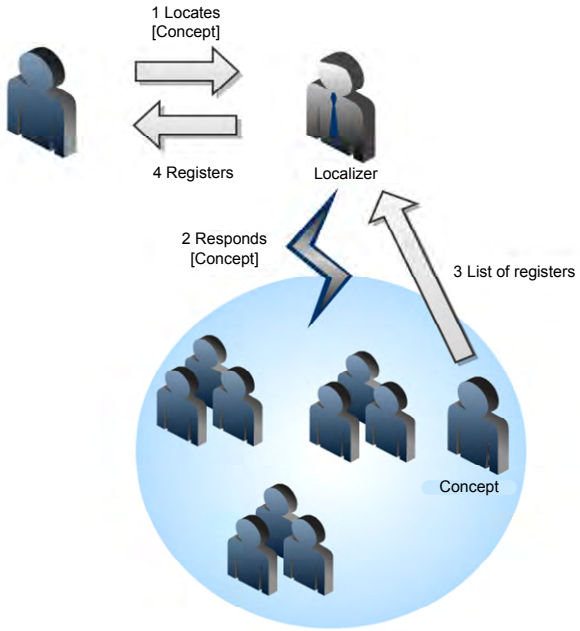

Figure 5.25: Search process for an ontological concept.

In addition to the agents developing the ontological mapping, there is a *localizer* agent responsible for establishing the link between this subsystem and the other multi-agent systems, Figure 5.25. The *localizer* agent receives as an ontological concept as input and sends a broadcast message to the whole subsystem by asking for services related to the input concept. The agents representing the concept will respond to the *localizer* agent by returning it a list of UDDI-registers representing the services related to the ontological concept. The *localizer* agent concatenates all the answers and sends this to the agent who requested the service.

In order to meet the requirement of ontological equality, if an agent responds to the request message of the *localizer* agent, it will also send the name of the concepts associated with equivalent lines (if any). The *localizer* agent will again launch a broadcast message asking for those equivalent concepts. This process will be repeated until no equivalent concepts exist. Thereby, all concepts equivalent to the initial concept are obtained.

5.4.3 Composition system

Once the reduced BPMN diagram has been obtained, and the Web services have been associated with each activity, the system is ready to make the composition of such services by using standard WS-BPEL 2.0 (Arkin *et al*., 2004). The *composition system* will act in response to the invocation of the composeService functionality specified in Section 5.2.1 (Cloud Services).

There are several alternatives for this process such as those in (Ouyang *et al*., 2006) (Jan and Jan, 2006) (Ouvans *et al*., 2006) (White, 2005). The method provided in (White, 2005) for the BPEL composition from the BPMN diagram was performed by obtaining information from the WSDL files of participating Web services and the input

the service receives. Preliminary information from the WS-BPEL composition has been obtained from the input document, which specifies certain general properties of the final process. These attributes are automatically generated by the system or obtained from the input:

1. *name*: Name of the service (obtained from the *name* attribute of tag BPMN in the input).
2. *expressionLanguage*: Language of the expression of the solution (obtained by default).
3. *suppressJoinFailure*: Indicates whether it can launch *fault joinFailure*. The value will always be "false".
4. *abstractProcess*: Indicates whether it is an abstract process. The value will always be "false".

To the *PartnerLink* section (which shows the different Web services participating in the composition *PartnerLinkType*), the values of these elements are obtained from the WSDL files corresponding to the involved Web services.

- *partnerLinkType*: expresses the conversational relationships between two services by defining the roles of the different participants and specifying the *portType* provided by each Web service for message reception (Arkin *et al.*, 2004).
- *name*: Name of the *partner*.

The *variable* section provides the means for storing the messages constituting the state of the process. In (GuideToBPEL, 2012) the authors show how to link the different variables of WS-BPEL in the WSDL files of the services.

```
<types>
  <xsd:schema>
    <xs:element name="getPlazas" type="tns:getPlazas"/>
    <xs:element name="getPlazasResponse"
                type="tns:getPlazasResponse"/>
    <xs:complexType name="getPlazas">
       <xs:sequence>
         <xs:element name="idVuelo"
                     type="xs:string"minOccurs="0"
                     wssem:modelReference="AeroOnt#Vuelo"/>

       </xs:sequence>
    </xs:complexType>

    <xs:complexType name="getPlazasResponse">
      <xs:sequence>
         <xs:element name="return" type="xs:int" minOccurs="0"
                     wssem:modelReference="AeroOnt#PlazasLibres"/>
      </xs:sequence>
     </xs:complexType>
  </xsd:schema>
</types>

<message name="getPlazas">
  <part name="parameters" element="tns:getPlazas"/>
</message>
```

```
<message name="getPlazasResponse">
  <part name="parameters" element="tns:getPlazasResponse"/>
</message>

<portType name="Aerolinea">
  <operation name="getPlazas">
    <documentation>
      <wssem:precondition name="ExisteVueloCond"
       wssem:modelReference="AeroOnt#ExisteVuelo">
       <wssem:effect name="SeObtienePlazas"
       wssem:modelReference="AeroOnt#PlazasObtenidas"/>
    </documentation>
    <input wsam:Action=
          "http://Aerolinea.com/Aerolinea/getPlazasRequest"
           message="tns:getPlazas"/>
    <output wsam:Action=
            "http://Aerolinea.com/Aerolinea/getPlazasResponse"
             message="tns:getPlazasResponse"/>
  </operation>
</portType>
```

Code
```
<message name="creditInfo">
   <part name="firstname" type="xsd:string"/>
   <part name="surname" type="xsd:string"/>
   <part name="credit" type="xsd:string"/>
</message>
```

5.26: Example of a WSDL *message.*

```
<variable name="creditInformation" messageType="creditInfo"/>
```

Figure 5.27: Link of a *variable* WS-BPEL with WSDL *message.*

For the definition of other properties of composition WS-BPEL, the methods shown in (White, 2005) are followed. These include:

1. Definition of the start of the process.
2. Definition of the end of the process.
3. Mapping of the parallel flow.
4. Synchronization of the parallel flow.
5. Mapping of loops.

The process described to compose different Web service compositions is performed by the *composition system*. After receiving the initiation request from the *manager* agent, the *composition coordinator* agent will create an agent for each activity/service, Figure 5.28. Each agent achieves the required data from the WSDL file and will send them to the *composition coordinator* agent who, by using the method specified in (White, 2005), will generate the WS-BPEL file. The performance of this process has been benefitted since it is within a distributed environment of *cloud computing*.

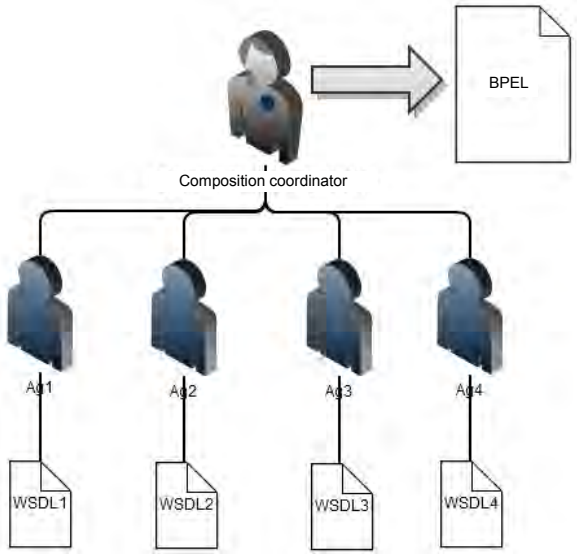

Figure 5.28: Composition subsystem.

5.5 Insertion of new services into the platform

The platform is scalable with respect to Web services that have registered; that is, it is possible to insert and register new Web services, making them available for subsequent executions. Since the system is immersed in a *cloud* environment, there are no problems with respect to the size that the service repository can reach, allowing it to grow indefinitely. To insert a new Web service into the platform and allow it to be localized, the following conditions must be met:

1. To have a description file WSDL.
2. To be registered by UDDI.
3. The WSDL file must be endowed with semantic annotations corresponding to its semantic meaning.
4. If the references to the semantic content are not among the ontologies of the platform, then that ontology should be added and should go through a process in which its content is converted into one that is computable. This process will create an agent for each semantic concept as explained in Section 5.4.2.

When a new service has been inserted into the platform, the *register subsystem,* belonging to the *search system* is responsible for conducting operations of locating such a service by carrying out semantic searches. The WSDL file of the Web service will be analyzed in the search of semantics annotations. For each annotation, the agent representing that ontological concept will be generated. The line of input to the UDDI register of the Web service is being introduced will be added to the references of the agent.

5.6 Conclusions

In this chapter we have presented the process of building a model (IPCASCI) in a *cloud* environment which focuses on creating new business services in a semi-automatic way starting from Web services that exist in the platform of the model. The user introduces a set of specifications (converted in XML format) into the platform. These specifications are later converted, following the BPMN standard, into a set of modules with computable information. This information is used to build the agents associated with each module that will be in charge of representing the introduced information by forming a multi-agent system based on virtual organizations. For this, the agents will be in charge of finding Web services that respond to each module and through a process of Web services composition, a new business service is created as a solution to the specifications introduced by the user.

One of the main contributions of this model that we can underline is its implementation on a *cloud* environment, which brings with it all the advantages and benefits this paradigm offers (see Chapter 2). This allows users better access. the ability to use services in the platform from heterogeneous sources, and no infrastructure costs. Likewise, the automatic composition of Web services to generate new services (due to the reuse of software components) according to user specifications, guarantees autonomy in the platform, which is desirable in any system implemented by a company. Consequently, the two previous contributions are the basis of our architecture, revealing great advantage over other approaches

Finally, starting from the exhaustive description of the components of the proposed architecture, we have reached the goals proposed in Chapter 1 of this PhD Thesis. The next chapter will, therefore, assess our proposal through a case study and analyze its results, concluding in this way, our global proposal of the IPCASCI architecture.

6 RESULTS AND CONCLUSIONS

6.1 Case study

In this section, we introduce an implementation of our proposal (IPCASCI model) on a practical case study. The aim of this case study is to bring our proposal to fruition in a real environment and thus develop a tool to represent the case study. In this context, we will evaluate different parameters of the tool, both internal and external, to attain some results that will give us a general vision of the strength of our proposal. We begin by introducing the proposal of the case study as follows:

This case study develops a tool to automatically build a Web service to reserve a book through the book lending system in a library. As (the minimum required) input, the Web service must have the identifier of the user that carries out the request, and the required book. The task of the Web service is then to check whether the user is registered in the system and their subscription has not expired. After successfully checking the above, the Web service must then verify whether the book is available to loan. Moreover, it would also check whether the user has exceeded the maximum number of books borrowed from the library. If all the above conditions are held, then the loan is carried out. Otherwise, the user will be notified (for example, by e-mail) that the request has been denied along with the reason for it.

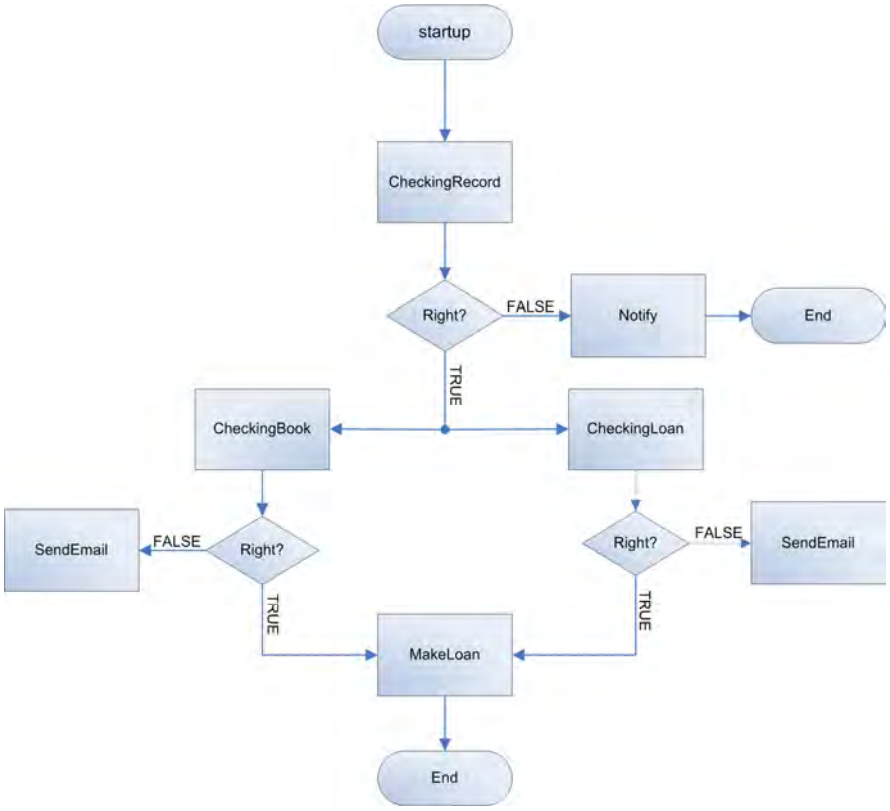

Figure 6.1: Flow diagram with the specifications (in form of modules) of the case study to be introduced by the user.

A representation of the scene proposed in the case study can be implemented from the flow diagram in Figure 6.1 (the components will be explained later). Notice that this diagram represents the user specifications (through the interrelated units as indicated by the arrows) to build the new Web service. Therefore, from now on, we will explain the characteristics of the sub-processes that are part of the tool to convert the user specifications into processes and connections, which are addressed to create the new Web service based on the composition from existing services.

6.1.1 Specifications of the users

The tool (named *LibraryBookReserve* or *LBR* for short) implements an assistant to introduce the user specifications, which will be used in the construction of the Web service that will respond to such effects. Hence, the assistant allows introducing the specifications though a graphical interface, where we can define each module of the Web service that we want to generate. Likewise, for each defined module, the assistant provides a graphical interface to introduce its functionality and its relationship with other modules. A module is represented by a set of parameters, which define its behaviour, namely: *name, type, action, domain, input, output and interconnection of the module with other modules.* In this context and for our case study, the modules listed in Table 6.1 have been generated. Notice that the modules defined in this table will define a flow diagram as the one shown in Figure 6.1.

MODULE	DESCRIPTION
CheckingRecord	Determine whether the user is registered in the system.
NotifyState	Notify the user that the connection has not been established and the reason.
CheckingBook	Verify the existence of the requested book.
ChekingLoan	Verify the maximum number of loans stated for the user.
NotifynoStock	Notify to the user that there is no existence of the requested book.
NotifyLoanExceed	Notify the user that the maximum number of loans has been exceeded.
MakeLoan	Carry out and register the book loan.
CheckedRecord	Auxiliary module useful for the later construction of diagram BPD of graphic standard BPMN. This module is defined by the system.

Table 6.1: Definition of the modules and their functions for the composition of the Web service.

As mentioned before, each module is described by an ontology that defines its behaviour and its action. Therefore, the semantics for each module of the previous table have been described as shown in Tables 6.2, 6.3, 6.4 and 6.5.

Name:	*CheckingRecord*	*Name:*	*NotifyState*
Input:	Person	Input:	person
	Organization		
Output:	recordState	Output:	-

Precondition:	organizationExists	Precondition:	personExists
Action:	verifyRecord	Action:	sendEmail
Domain:	authentication	Domain:	message
Type:	startup	Type:	end

Table 6.2: Semantics description of the attributes for modules CheckingRecord and NotifyState.

Name:	*CheckingBook*	*Name:*	*ChekingLoan*
Input:	book	Input:	person
	Organization		organization
Output:	bookState	Output:	loanState
Precondition:	bookExists	Precondition:	personExists
	organizationExists		
Action:	checkingAvailableBook	Action:	checkingLoan
Domain:	Library	Domain:	library
Type:	Intermediate	Type:	intermediate

Table 6.3: Semantics description of the attributes for modules CheckingBook and ChekingLoan.

Name:	*NotifynoStock*	*Name:*	*NotifyLoanExceed*
Input:	person	Input:	person
	book		
Output:	-	Output:	-
Precondition:	bookExists	Precondition:	personExists
	personExists		
Action:	notifyNoBooks	Action:	notifyNoLoan
Domain:	library	Domain:	library
Type:	end	Type:	end

Table 6.4: Semantics description of the attributes for modules NotifynoStock and NotifyLoanExceed.

Name:	*MakeLoan*
Input:	person
	book
Output:	-
Precondition:	personExists
	bookExists
Action:	makeLoan
Domain:	library

Type:	End

Table 6.5: Semantics description of the attributes for module MakeLoan.

Note that the modules in these tables have been represented in the flow diagram in Figure 6.1, therefore, once their connections have been established, the tool will generate a diagram in XML format, which automatically models the requirements that have been introduced to initiate the process of analysis. The connections defined for these modules are explained in Tables 6.6 and 6.7. When modelling the introduced relationships and modules, we obtain a flow diagram as the one in Figure 6.1.

Name:	*Relation1*	*Name:*	*Relation2*
Type:	If/Else	Type:	Parallel
Startup:	CheckingRecord	Startup:	CheckedRecord
Destination1:	NotifyState	Destination1:	CheckingBook
Destination2:	CheckedRecord	Destination2:	ChekingLoan
Condition:	registerState	Condition:	-

Table 6.6: Semantics description of relationships Relation1 and Relation2 defined on the modules introduced by the user.

Name:	*Relation3*	*Name:*	*Relation4*
Type:	If/Else	Type:	If/Else
Startup:	CheckingBook	Startup:	ChekingLoan
Destination1:	MakeLoan	Destination1:	MakeLoan
Destination2:	NotifynoStock	Destination2:	NotifyLoanExceed
Condition:	bookState	Condition:	loanState

Table 6.7: Semantics description of relationships Relation3 and Relation4 defined on the modules introduced by the user.

6.1.2 The process of analysis

As mentioned before, the specifications given by the user are taken as an input (XML format) for the *analysis system*. This system is the content for a virtual organization of agents in the multi-agent system of the tool (Figure 5.1 of previous chapter). The tool calls on the functionality *analyze&Discover* (and therefore, the *analysis system*, see section 5.2.1), which will create an agent for each module indicated in the input specifications. Subsequently, it will analyze the connections between the modules to reflect them as lines of unidirectional communication between the agents that have been created. As the result of this process, we will obtain a diagram of agents (Mapping of the system of agents). In the next step, a diagram of agents is converted into a BPD diagram (from the standard BPMN) and following the algorithm explained in section 5.4.1. In this last process, we obtain an XML file (BPEL file) that stores the information of the BPD diagram and which will be

subsequently completed with the information of the *search system*. Therefore, this XML file is the output of functionality *analyzes&Discover.*

6.1.3 Discovery process

The discovery process of the Web services associated with each module is carried out by the *Search system* (subsystem of agents) and has two stages:

- Find the Web services associated to the semantic concepts of the modules.
- From the obtained modules, filter those which fulfill the format specifications of the modules (input, output, preconditions).

To find the set of Web services that are associated with a semantic concept, the *search coordinator* agent will send a message to the *semantic coordinator* agent asking for the set of services associated with a concept. The above agent will communicate with the *localizer* agent for each concept and it will thus obtain the semantically associated Web services (the *localizer* agent will carry out a semantic search). Once the Web services associated with the ontological concepts of the module have been obtained, the *semantic coordinator* agent will filter the results obtained from the *localizer* agent, thus avoiding duplicated results. Furthermore, it carries out the intersection of the found Web services, so it will offer as a result those Web services associated with the total set of concepts. Table 6.8 lists those Web services existing in the platform and which will be used for the Web service composition.

WEB SERVICES	FUNCTION
checkPersonRegister checkOrgRegister	Check whether a user is registered in the system.
SendMail sendEMail emailManager	Send and manage the electronic mail (e-mail).
libraryManager checkBook compHLib	Identify books and manage their availability in the library.
stockManager loanManager checkLoans loanController	Check and manage the (maximum) number of books loaned to a user.
sendNotAvaliableBook notifyBookError	Notify the unavailability of a book, or any other problem preventing the loan.

loanPassed notifyLoans libraryLoanError	Notify that the maximum number of book loans has been reached.
doLoan libraryLoanEffect bookStoreDoLoan	They are used to register a book loan by the library.

Table 6.8: Web services available in the platform for the case study of architecture IPCASCI.

Once the Web services associated with the ontological concept of each module have been obtained, those services that fit to the format of each module will be filtered as a second part of the process. This includes the following filtering process:

- Checking the service input.
- Checking the service output.
- Checking the service process.

These checks must be done because in the previous step we have obtained the Web services associated with a set of semantic concepts. However, it is not checked if those Web services have the associated concepts in the place where the requisites of the corresponding modules determine. For this reason, the *checking coordinator* agent will communicate with the subsystems' *input checking, output checking and process checking*. The input for each subsystem will include the WSDL file of the Web service and the corresponding description of the module. When receiving the results of each subsystem, the *checking coordinator* agent will compare the results to obtain the services that meet all the specifications. In particular, the process of checking if a Web service holds the ontological specifications in the input section is carried out on the basis of:

- The Web service WSDL file.
- The list of the semantic concepts of the input.
- The list of the semantic concepts of the output.

On the WSDL file of a Web service, a search for the ontological annotations is performed (through a XML parser) **wssem:modelReference** on the tags **input**, **output** or on the *XML Schema* corresponding to those parameters. The annotations can be done on simple or compound types. Once all the annotations corresponding to the inputs and outputs have been localized, the ontological concepts of those annotations, the input and the output concepts of the module being analyzed will be checked.

After checking that the inputs and outputs of the Web service have coincided with the analyzed module, the process checking stage is carried out to ensure that the

actions carried out by the Web service correspond with the module specifications. In this sense, two *check*s have been carried out:

- *Checking preconditions:* the content of the WSDL file of the Web service is analyzed to look for the extensions **wssem:precondition** on the tags *operation* or *documentation*. In any case the ontological concept is obtained from the attribute **wssem:modelReference.**
- *Checking actions:* the content of the WSDL file is analyzed to look for the extensions **wssem**: **effect** on the tags *operation* or *documentation*. In any case the ontological concept is obtained from the attribute **wssem:modelReference.**

At the final stage of the filtering process of the Web services that fit the specifications, the *checking coordinator* agent has received the results of the different subsystems and will compare those results in order to obtain the specific Web services that fulfil the specifications of the module that is being checked. When this process has been completed, the list of the Web services will be sent to the s*earch coordinator* agent that will complete the corresponding file. This way, the analysis and discovery process invoked by tool functionality *analyze&Discover* will return the file that represents the BPMN diagram and the list of the Web services associated with each activity. Table 6.9 shows the list of the Web services adjusted to the semantic specifications of each module/agent (activity of the BPMN diagram).

MODULES	FOUND *WEB* SERVICES
CheckingRecord	checkPersonRegister
	checkOrgRegister
NotifyState	sendMail
	sendEMail
CheckingBook	compHLib
ChekingLoan	checkLoans
NotifynoStock	sendNotAvaliableBook
	notifyBookError
NotifyLoanExceed	notifyLoans
	libraryLoanError

Table 6.9: Web services found and checked by the *search system* for each module defined by the user.

6.1.4 Validation of the solution

When functionality *analyze&Discover* ends its task, the tool has executed the search for the Web services that fit the modules introduced by the user, and elaborated the BMP diagram (standard BPMN) of the introduced requirement.

To build the Web service solution we need to:

- Select the Web service desired for each activity of the BPMN diagram for each module (it may be the case that different Web services fulfil the requirements marked by the platform).
- Connect the outputs and input of each activity of the diagram.

Consequently, the user, through the assistant that creates the BPMN diagram, would need to select the Web service to be used for the implementation of each activity of the BPMN diagram. The assistant has three sections to carry out this task:

- *BPMN diagram:* Representation of the BPMN diagram returned in XML format by the *analyze&Discover* functionality. The graphic interface of the application allows us to select and move the different elements in the diagram.
- *List of activities:* In this section we can see the different activities of the BPMN diagram (corresponding to the modules introduced in the section of introduction of requirements). For each activity in the diagram, the following information is shown:
    - Name of the activity.
    - Action performed.
    - Domain
    - List of inputs to the activity.
    - List of outputs of the activity.
    - List of preconditions needed for the proper working of the activity.
- *Found Web services:* List of Web services available for the selected activity. The first service will be selected by default, thus allowing the user to choose another one from the list.

The Web services chosen for each activity/module in our case study are shown in Table 6.10.

ACTIVITY/MODULE	CHOSEN WEB SERVICES
CheckingRecord	checkPersonRegister
NotifyState	sendEMail
CheckingBook	compHLib
ChekingLoan	checkLoans
NotifynoStock	notifyBookError
NotifyLoanExceed	libraryLoanError

Table 6.10: *Web* services selected for each defined module.

To complete the process of validating othe solution proposed by the tool and finally build the solution Web service, we need to associate the data flow with the different activities of the BPMN diagram.

Each activity of the BPMN diagram receives some inputs that could come from the previous activity, from other previous activities or from the input to the Web service we want to build. Therefore, we need a mechanism that allows us to relate the entries of the different services with the outputs of the preceding ones. The tool would not be able to automatically carry out this task if there is any repeated concept between the union of the set of outputs of the previous services. If this were the case, the user would have to select the desired output to be associated with the input to the activity. In this specific case, there is no duplicity of concepts between the outputs of the activities, so the links are 100% reliable.

6.1.5 Composition of the solution Web service

When ending the association process of the Web services between the different activities and the data connection between them, we can request the tool to execute the composition of the solution Web service. For that purpose, the *composeService* functionality of the tool will be invoked, after which the composition of the services will take place following the WS-BPEL standard and the specified algorithm (White, 2005). Basically, *composeService* will invoke the *manager* agent running the BPMN diagram. The *manager* agent will process the request by giving the information to the *composition coordinator* of the *composition system*. The *composition coordinator* creates an agent for each Web service (or activity). Each agent will obtain the necessary data for the composition process from WDL files (see Figure 5.28). Finally, the *composition coordinator* receives all the WSDL files and generates the WS-BPEL file from the data.

The *partnerLink* section corresponds to the links between the BPEL entities and the external Web services. In Code 6.1 we can see the *partnerLink* corresponding to the Web services *checkOrgRegister, senEMail, compHLib* and *checkLoans*.

```
<partnerLink name="checkOrgRegisterPartnerLink"
             xmlns:tns="http://checkOrgRegister.wsdl"
             partnerLinkType="tns:checkOrgRegister"
             myRole="checkOrgRegisterPortTypeRole"/>
 <partnerLink name="sendEMailPartnerLink"
             xmlns:tns="http://sendEMail.wsdl"
             partnerLinkType="tns:sendEMail"
             myRole="sendEMailPortTypeRole"/>
 <partnerLink name="compHLibPartnerLink"
             xmlns:tns="http://compHLibr.wsdl"
             partnerLinkType="tns:compHLib"
             myRole="compHLibPortTypeRole"/>
 <partnerLink name="checkLoansPartnerLink"
             xmlns:tns="http://checkLoans.wsdl"
             partnerLinkType="tns:checkLoans"
             myRole="checkLoansPortTypeRole"/>
```

Code 6.1: partnerLinks

In a BPEL process, the *variable* section is used for the storage of those messages that can serve as an input to a *partner*, output from a *partner,* or required data to store the state of the process. The data of input to the Web service will also be stored in this section. That way, the information will be available for the links between the Web services. In Code 6.2 we can see the variables for the Web services, *checkOrgRegister* and *compHLibr*.

```
<variable name="OrgRegisterRequest"
          messageType="tns:checkOrgRegistertRequestMessage"/>
<variable name="OrgRegisterResponse"
          messageType="tns:checkOrgRegistertResponseMessage"/>
<variable name="compHLibrRequest"
          messageType="tns:compHLibrRequestMessage"/>
<variable name="compHLibrResponse"
          messageType="tns:compHLibrResponseMessage"/>
```

Code 6.2: Variables

Following the steps indicated in (White, 2005), the different components of the BPEL composition are modeled for the new Web service, which includes:

- Flow control.
- Service start.
- Exchange of messages and
- Events.

6.2 Assessment of the case study

In this section we will evaluate tool *LibraryBookReserve* built from the approach of our case study. The evaluation will be carried out following the guideline of software engineering, taking both internal and external qualities into account. The accomplishment of an internal quality of a *software* will therefore allow an easy, tidy and efficient task for the developers of the system, while the external qualities evaluate the expected and useful (or usable) factors for the users. Proceeding then from the previous classification of software quality, we know that the internal quality is associated with the *white box testing* (Pressman, 2005), which is a test of the procedural details of the *software*. Therefore, all the logical paths of the *software* are checked, providing different tests that will execute sets of specific conditions or loops. Furthermore, the state of the application can be verified in different points with the goal of determining if the real state coincides with the expected one. The external quality of a *software* is associated to the *black box testing* (Pressman, 2005), which is a test on the interface of the application. In other words, testing cases try to demonstrate that the functions of the application are operating and with a correct result, in addition to the integrity of the external information. A black box test examines different aspects of the system without having to take into account the logical internal structure of the *software*.

In our particular case, we will use the *basis path testing* as the white box testing. This test is used to evaluate the logic of the flow diagram followed by the composition of the Web services in the tool. It will thus measure the correction of the new Web service which has been automatically created. Additionally, as a black box test, we will use a system of user survey which will cover the software qualities classified in the ISO 9126 quality metrics (Pressman, 2005). Both tests have been designed according to the results obtained from *12 users* that specified the requirements to create the Web service (with the tool) from the composition of the existing services, as explained in the definition of the case study. Each user also completed the survey about the tool quality.

6.2.1 Basis path testing of the services composition (*white box testing*)

The basis path testing is a white box technique suggested by (McCabe, 1976). The basis path method allows test case designers to create a measure for the logical complexity of a procedural design and use it as a guide to define a *set of basis paths in implementation*. The goal of this test is to determine the number of *independent paths* of a structured construction (in our case the flow diagram) to create test cases that force our Web service to execute each possible path of its logical structure. Thus, it guarantees that every sentence in its procedure is executed at least once, thus proving its correctness and integrity.

In our case, which involves the evaluation of the flow diagram that makes up Web services, a test case was created for each path of the diagram. In Figure 6.2 we can see the flow diagram with the Web services that take part in the building process our new Web service according to Figure 6.1 and Table 6.10. From this diagram the different paths of execution were evaluated. To do so, it is first necessary to build the flow graph of the diagram in Figure 6.2.

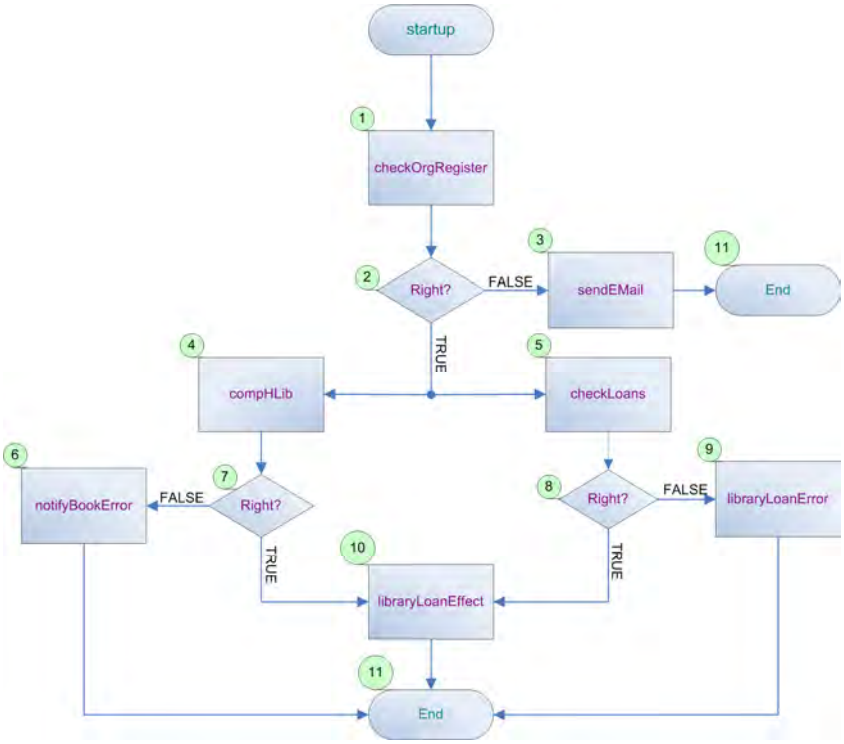

Figure 6.2: Flow diagram of the Web service composition to create the service specified by the user.

In Figure 6.3 we can see the flow network of the process of the Web service composition. The flow network represents the logical control structure through the notation given in this figure. The circles, named nodes, represent one or more procedural sentences (Pressman, 2005). The consecutive sentences can be in the same node, as the figure shows. The arch represents the flow controls, which have the same meaning as the ones in the flow diagram of Figure 6.2. The predicate nodes are not the nodes that represent a conditional sentence in the flow diagram and they are characterized by the two or more arches coming out from them. Once the flow network has been defined, we can calculate its *cyclomatic complexity* and determine the number of paths independently executing. The cyclomatic complexity is a measure of the software that provides a quantitative measure of the logical complexity of the program. It gives us the maximum limit of tests (independent paths) that must be done to ensure that each sentence is executed at least once. On the other hand, an independent path is any path in the flow diagram (or flow network) that introduces, at least, a new set of sentences.

According to (Pressman, 2005), cyclomatic complexity of a flow graph can be calculated from the expression $V(G)= E - N + 2$, where E is the number of arcs in the graph and N is the number of nodes. Therefore in our case:

$$V(G)= E - N + 2 = 11 \text{ arcs} - 8 \text{ nodes} + 2 = 5.$$

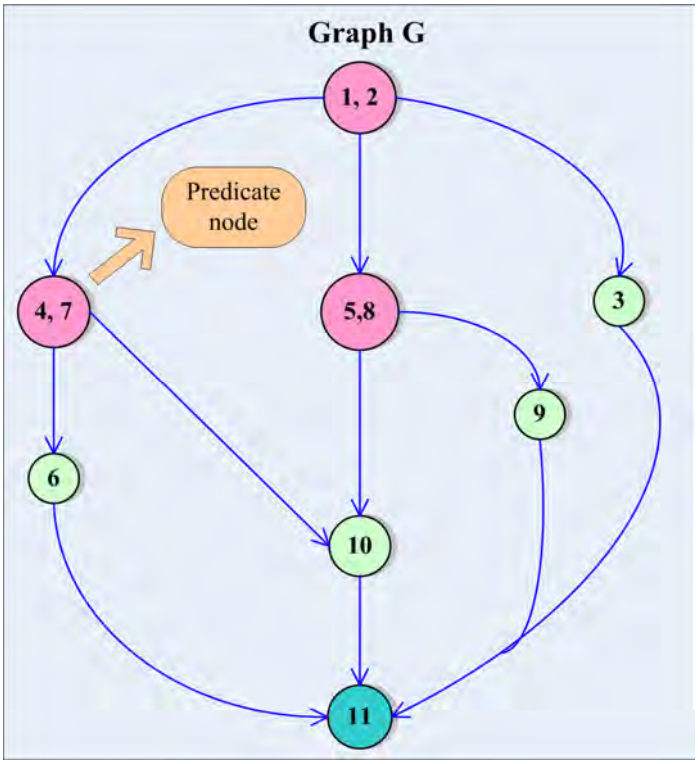

Figure 6.3: flow graph of the flow diagram in Figure 6.2.

Consequently, we can now determine the basic set of linearly independent paths. Value *V(G)* defines the number of independent paths to check in the control structure of the flow diagram. As previously calculated from flow graph, we have five paths that can be described in the following Table 6.11.

LOGICAL PATHS	
Path #1:	1 – 2 – 4 – 7 – 6 – 11
Path #2:	1 – 2 – 4 – 7 – 10 – 11
Path #3:	1 – 2 – 5 – 8 – 10 – 11
Path #4:	1 – 2 – 5 – 8 – 9 – 11
Path #5:	1 – 2 – 3 – 11

Table 6.11: Linearly independent paths of the flow graph in Figure 6.3.

For each path, we have developed a test case which forces the Web service (product of the service composition) to execute the corresponding path. The results showed that each path had been executed at least once and the results obtained coincided with the expected ones. Therefore, the logic implemented (based on the user specifications) in the flow diagram in Figure 6.2, along with the performance of the Web services used in the composition, worked properly. Additionally, those same results were obtained for each different flow diagram that had been built with the tool by the 12 selected users. Notice that in the service composition process (BPMN

diagram) the user can select a different Web service from the predetermined by the tool, which produces a different flow diagram from the one given in Figure 6.2.

6.2.2 Tool functionality test (*black box testing*)

In this section we will carry out a *black box test* (external qualities) on the functionality (interface) of the tool based on the user experience after having used the tool. As previously mentioned, a black box test (also called *behavior testing*) is focused on the functional requisites of the software. Hence, by the application of this test, we attempt to complement the white box test carried out in the previous section. This will help to detect errors which are not possible to detect with the white box testing.

To carry out this test, 12 users independently executed the tool and generated the specifications (modules and relationships) to build the Web service solution based on the existing Web services in our platform. After the users gained experience with the use and the generation of the new Web service, they were asked to fill in a survey on different desirable qualities in a tool framed in the domain of our proposal. The survey was divided into specific questions on the functionalities of the tool according to the case study and questions on the desirable functionalities in a tool in general from the area of our proposal. Therefore, the goal of this test is to validate the performance of the tool and thus, the case study is based on the experience of the final user.

Figure 6.4 shows the model of the survey applied to each user of the tool. This model is based on a system of questions that represent desirable attributes and assess the efficiency (from the point of view of the user) for the tool. For each question the user can give a score (evaluation) between 1 and 10, where a value less than 5 is considered a negative evaluation and a value greater than 5 is positive. Notice that in this case, our model does not include attributes such as *efficiency*, referring to how efficiently the software can process/store data; *scalability*, with regards to data volume growth; or *portability and security* of the tool, because these attributes are obtained from the *cloud computing* platform, on which our architecture is based.

#	QUESTIONS	EVALUATION (score between 1 and 10) Mark with an "X" the chosen score. 1 2 3 4 5 6 7 8 9 10
1	The wizard that guides the user through the introduction of the specifications (through modules and their relationships) to build the *web* service is intuitive and relatively easy to use.	
2	The time taken for the discovery process (multi-agent system) of the *web* services containing the concepts introduced by the user is reasonable/acceptable.	
3	The time taken by the process of *web* service composition to construct the new service is reasonable/acceptable.	
4	The new *web* service built from the service composition represents the user-entered specifications.	
5	The selection process (by the user) of a different *web* service from the default one shown in the diagram BPM (BPMN) by the tool for service composition is intuitive and relatively easy.	
6	The tool step-by-step guides and reports the process followed to build the solution *web* service.	
	OVERALL ASSESSMENT QUESTIONS	
7	*Usability*: The tool, in general, is simple to understand and operate from the point of view of the user.	
8	*Functionality*: The tool solves almost all the problems of operation and information management.	
9	*Availability/Recoverability*: The tool is usually not easily fall.	
10	*Availability/Recoverability*: The tool takes short time to boot up to reach its functional status.	

Figure 6.4: Table representing the attributes (in the form of questions) measured in the survey to the users of the tool *LibraryBookReserve.*

Figure 6.5 shows the table of average scores for each question in the model in Figure 6.4. The reached average score shows user satisfaction with our tool; the same for the requirements of the proposed case study. In this sense, and based on the case study, our tool fulfils the goals proposed by our architecture.

QUESTIONS	AVERAGE SCORE ACHIEVED FROM THE SURVEY
1	9,34
2	9,15
3	9,30
4	8,97
5	8,93
6	9,70
7	8,77
8	8,65
9	10,00
10	8,93

Figure 6.5: Table of average scores achieved from the 12 surveyed users for tool *LibraryBookReserve.*

Reinforcing Figure 6.5, Figure 6.6 shows a bar chart representing the scores in this table. In the *x-axis* we can see the numbers of the questions whereas in the *y-axis* we can see the average scores. As we can see in this figure, the three highest scores were with questions 1, 6 and 9, which prove that introducing the process of specifications to build the new Web service is intuitive, the tool is well designed for the end user, and its behavior is stable.

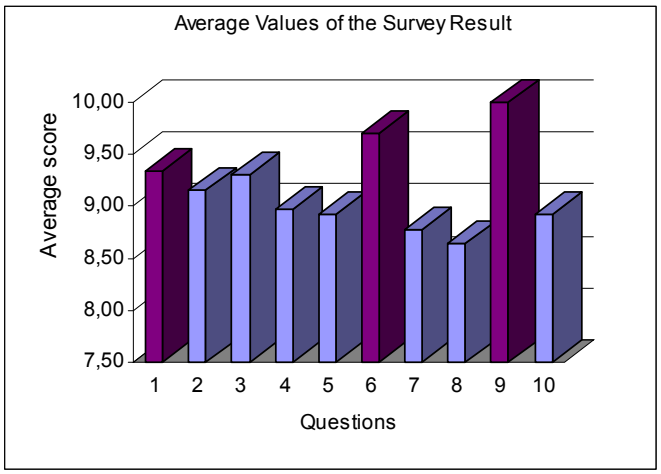

Figure 6.6: Bar chart with the questions vs. Score of the table in Figure 6.5.

An alternative view to Figure 6.6 is Figure 6.7, which shows the scores of the table in Figure 6.5 but in a curve.

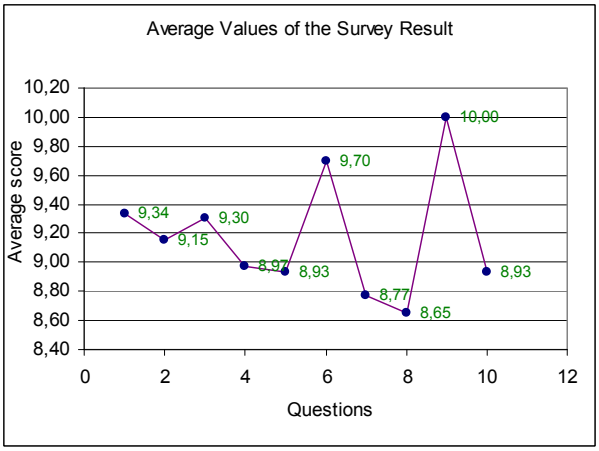

Figure 6.7: Dot plot (q*uestions vs. average score*) of the table values in Figure 6.5.

6.3 Analysis of the case study

In this chapter we have introduced a case study to show the implementation of our proposal, the *business building process model IPCASCI* (Chapter 5). The initial results from our case study was the construction of a tool (*LibraryBookReserve*) which creates a Web service (semi-automatically) to reserve books in a library, for which we based our work on the composition of the existing services in our architecture. This result can also be seen in our IPCASCI model. Once the tool has been built, we introduced the process of its validation, evaluating both internal (white box testing) and external (black box testing) characteristics. The white box testing, also known as *small-scale testing* is focused on the control structure of the application, in our case, the flow diagram that the tool executes in the process of composing/building Web services (BPMN diagram). The black box testing, also known as *large-scale testing* expands the

approach; it is designed to validate the functional requisites without taking into account the internal functionality of an application. In our case, the survey system given to 12 users of the tool aimed to evaluate the quality of the final product of our model, the *LibraryBookReserve* tool. The results of this study has proven that our tool, in a general sense, fulfills the requirements of the proposed case study and that the Web service built from the BPMN diagram (service composition) holds the specifications introduced by the user. This also proves the strength as well as the reliability of the design and the logic structure of the multi-agent virtual organizations (in our architecture), which are oriented to the discovery/refinement of the Web services that will take part in the service composition. Finally, the survey results have also proven that the tool meets most of the expected requirements in an application oriented to Web services.

6.4 General conclusions

Software development is traditionally associated with a series of problems, the most common being delay in the date of delivery and the uncontrolled increase in product cost. As a solution to these problems there are techniques and tools related to the area of Software Engineering that allow organizing, managing, planning and securing the quality of the development process. When using these techniques the risk of the cost of the project going out of control is considerably reduced. However, carrying out a process model where the project is completely planned (including analysis costs, temporal planning, development lifecycle and extensive technical documentation) is a task that takes up a lot of time and work, so it is necessary to assess if it is worth the cost.

In contrast to a development model that is very controlled, planned and documented, there are methodologies that speed up this process by making the development cycle shorter, although they do not have the advantages that a complete technical documentation offers.

On many occasions, the actual situation in the business world requires developing software quickly and with a reduced cost. Even with lively methodologies, the development cost of new software products is high for small and medium size companies. Apart from that, and due to the fast and constant change of the market situation, there is a definite need for new software products in a very short time.

In this document we propose a model that adapts itself to the technological environment of this moment, one that is accessible and useful for the business world. The main characteristics of this proposal are that:

- It allows creating software from components already implemented.
- It is possible to access anywhere and with any platform.
- It is fast and simple to use.
- The creation of the new software product is done automatically.
- There are reduced and controlled costs.

To fulfill these characteristics we have chosen a model where the reusable software components are the Web services. Additionally, the access and storage will be done in

a cloud system, making this option more affordable for small businesses (as they do not have to buy big storage systems) and with a good performance.

However, the main attraction of this proposal lies in the automation of the whole process of software building. The client just needs to introduce the specifications of the software to be built (easily and with little training), and the model will create the product as a Web service from the services already stored in the system. To carry out this task, the system will have a repository of Web services, an ontology that allows modeling the knowledge on its performance and a multi-agent system that, in an intelligent way, will be able to create the stored systems to satisfy the client needs.

Unlike the proposals based on OWL-S, the proposed model is based on the semantic association of the Web services with WSDL-S and a multi-agent system in charge of discovering and creating tasks.

All of this leads to a system that adapts to the present needs of software development, where it is possible to have new products in very little time, with a reduced cost and without going through a long and arduous process.

In short, we present a simple and intuitive model that allows building Web services from other services, that is usable by clients without technical knowledge, and that has a relatively low cost.

Some of the most relevant characteristics are:

- Pay-per-use: As the model is incorporated to a Cloud environment, clients will only pay for the actual use, lowering the usage cost and making it more accessible to small and medium size companies.
- Scalability regarding size: As it is incorporated in a Cloud, the system can grow with regard to the amount of reusable services stored without incurring in an significant cost in hardware components. It will also have a vast storage system, exempting the business from acquiring large storage systems.
- Scalability regarding sphere of application: On account of the ontology structuring and the multi-agent system, the client can add new areas of application without further work.
- Scalability regarding functionality: The multi-agent system will be designed in such way that it will make it possible to increase the given functionality. Furthermore, the Web services used to access the model are extendable.
- Ubiquity: It will be able to access the system anywhere as the execution will be done via Web services.
- Multiplatform execution: The use of the system and its access is done with a BPEL machine, which is an accepted standard by the majority of manufacturers.
- Business orientation: The execution is done with a BPEL machine, which can adapt itself to a specific business process (BPM) for each company.
- Reusable functionality: The necessary time to obtain a product that satisfies client needs is reduced as Web services that have already been implemented are used and reused for creating new ones.

- Quality control: The services given fit the specific process of the company's business within a BPM frame. This allows the process to embrace quality standards.
- Energy efficiency: The energy consumption is reduced as the system is incorporated in a cloud system, which in addition to reducing costs, is a better ecological solution than having local severs.
- Ease of use: The platform that implements the model will be designed for ease of use and learning. Thus, it will be possible to obtain results with little effort. The use of the platform will be done with intuitive graphic interfaces.
- Usable for non technical users. The specifications that the solution must fulfill will be introduced through an assistant that will guide the user through the entire process. Furthermore, the proposed solution will be modified with a BPD diagram (Business Process Diagram) belonging to the BPMN standard (Business Process Management Notation), which is relatively easy and does not require the user to have advanced computer knowledge. The result is that practically any person will be able to use the platform within a short learning period.

7 REFERENCES

- Akkiraju R., Farrell J., Miller J., Nagarajan M., Schmidt M., Sheth A., Verma K. (2005): Web Service Semantics - WSDL-S, W3C Member Submission. http://www.w3.org/Submission/WSDL-S/.
- Arkin A., Askary S., Bloch B., Curbera F., Goland Y., Kartha N., LiuK., Thatte S., Yendluri P., Yiu A. (2004): Web Services Business Process Execution Language. Version 2.0. OASIS.
- Bajo J., Zato C., de la Prieta F., de Luis A., Tapia D. (2010): Cloud Computing in Bioinformatics. Distrib. Computing & Artif. Intell. (DCAI'10), AISC, Springer-Verlag Berlin Heidelberg 79: 147-155.
- Blanco S. (2004): Anotaciones semánticas en WebQuest. PhD Thesis, Department of Informatics, University of Valladolid.
- Cao B., Li B., Xia Q. (2009): A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture. Cloud Computing, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg 5931: 644-649.
- DAML-S. DAML-S 0.7 Draft Release. (Accessed in 2012) http://www.daml.org/services/daml-s/0.7/
- García JA. (2011): IRIS & ME OWL-S Sistema de Recomendación en Dispositivo Móvil con OWL-S. Master's Thesis, University of Salamanca.
- García JM., Ruiz D. and Ruiz-Cortés (2012): Improving semantic web services discovery using SPARQL-based repository filtering. Web Semantics: Science, Services and Agents on the World Wide Web, 17: 12-24.
- GuideToBPEL. The BPEL Language. Obtained from http://www.radikalfx.com/bpel/language.html. (Accessed in 2012).
- Huhn M., Kowalczyk R., Maamar Z., Unland R. (2012): IOS Press. Special issue: development of service-based and agent-based computing systems.
- IBM (2008): Accessed in 2012. RESTful Web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/.
- IBM (2001): Understanding WSDL in a UDDI registry, Part 1. http://www.ibm.com/developerworks/webservices/library/ws-wsdl/.
- Jan R. and Jan M. (2006): On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In Latour, Thibaud & Petit, Michael (Eds.) The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, Namur University Press, 521-532.
- McCabe T. (1976): A Software Complexity Measure. IEEE Trans. Software Engineering, (SE-2): 308–320.
- Miller J., Verma K., Rajasekaran P., Sheth A., Aggarwal R., Sivashanmugam K. (2004): WSDL-S: Adding Semantics to WSDL - White Paper. LSDIS Lab, University of Georgia, http://lsdis.cs.uga.edu/projects/meteor-s/.
- Olson M. and Ogbuji U. (2002): The Python Web services developer: Messaging technologies compared. IBM developerWorks. Retrieved 2011-02-01.
- Ouvans C., Dumas M., Hofstede A., van der Aalst W. (2006): From BPMN Process Models to BPEL Web Services. Proceedings of the IEEE International Conference on Web Services (ICWS '06), IEEE Computer Society Washington, DC, USA, 285-292.
- Ouyang C., van der Aalst W. Marlon D., Hofstede A. (2006): Translating BPMN to BPEL. Digital Repository, Queensland University of Technology, Australia.
- Pinzon C., Bajo J., De Paz JF., Corchado JM. (2011): S-MAS: An adaptive hierarchical distributed multi-agent architecture for blocking malicious SOAP messages within Web Services environments. Expert Syst. Appl., Elsevier 38 (5): 5486-5499.
- Pressman R. (2005): Software Engineering: A Practitioner's Approach. Seventh Edition, Ph.D., McGraw-Hill Companies.
- Rodríguez S., Tapia D., Sanz E., Zato C., de la Prieta F. and Gil O. (2010): Cloud Computing Integrated into Service-Oriented Multi-Agent Architecture. Balanced Automation Systems for Future Manufacturing Networks, IFIP Advances in

Information and Communication Technology, Springer 322:251-259.
- Srinivasan N., Paolucci M., Sycara K. (2004): Adding OWL-S to UDDI, implementation and throughput. First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC).
- Su Z., Chen H., Zhu L., Zeng Y. (2012): Framework of Semantic Web Service Discovery Based on Fuzzy Logic and Multi-phase Matching. Journal of Information & Computational Science 9: 1 203–214.
- Sycara K., Paolucci M., Ankolekar A., Srinivasan N. (2003): Automated discovery, interaction and composition of Semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier 1:27-46.
- Tapia D., Rodríguez S., Bajo J., Corchado JM. (2008): FUSION@, A SOA-Based Multi-agent Architecture. International Symposium on Distributed Computing and Artificial Intelligence (DCAI), Advances in Soft Computing, Springer 50: 99-107.
- Tekli J.M., Damiani E., Chbeir R., Gianini, G. (2012): SOAP Processing Performance and Enhancement Services Computing, IEEE Transactions 5 (3): 387-403
- W3C OWL-S, (Accessed in 2012) http://www.w3.org/Submission/OWL-S/#5.
- Wei R., Qiao L., Yang Z. (2012): A message interaction security mechanism based on SOA. Systems and Informatics (ICSAI), International Conference on 1503-1506.
- White S. (2005): Using BPMN to Model a BPEL Process. BPTrends, http://www.bptrends.com.
- Abdelkader A., Bakhta N., Abdelkader O.M. (2012): Multi-Agents Model for Web-based Collaborative Decision Support Systems. In ICWIT, 294-299.
- Abu- Rahmeh O., Johnson P. and Taleb-Bendiab A. (2008): A Dynamic Biased Random Sampling Scheme for Scalable and Reliable Grid Networks. INFOCOMP - Journal of Computer Science, ISSN 1807-4545, N.4, December, 7: 01-10.
- Adobbati R., Marshall A. N., Scholer A., Tejada S., Kaminka G. A., Schaffer S., Sollitto C. (2001): Gamebots: A 3D virtual world test-bed for multi-agent research. In Proceedings of the second international workshop on Infrastructure for agents, MAS, and Scalable MAS, Montreal, Canada.
- Agrawal R. et al. (2008): The Claremont report on database research. SIGMOD Record, ACM 37 (3): 9–19. doi:10.1145/1462571.1462573. ISSN 0163-5808.
- Akkiraju R., Farrel J., Miller J., Nagarajan M., Schmidt M., Sheth A. and Verma K. (2005): Service Semantics - WSDL-S, A joint UGA-IBM Technical Note, version 1.0, April 18. http://lsdis.cs.uga.edu/projects/METEOR-S/WSDL-S
- Akkiraju R., Goodwin R., Doshi P., and Roeder S. (2003): A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. In IIWeb, 87-92.
- AlBreshne A., Fuhrer P., Pasquier J. (2009): Web Services Technologies: State of the Art Definitions, Standards, Case Study. Internal working paper / Department of Informatics, University of Fribourg.
- Ali R., Bryl V., Cabri G., Cossentino M., Dalpiaz F., Giorgini P., Molesini A., Omicini R., Puviani M., Seidita V. (2008): MEnSA Project - Methodologies for the Engineering of complex Software systms: Agent-based approach, Tech. Rep. 1.2, UniTn.
- Alizadeh K., Seyyedi M., Mohsenzadeh M. (2012): A Service Identification Method Based on Enterprise Ontology In Service Oriented Architecture. International Journal of Information Processing and Management (IJIPM), 3(2).
- Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S. (2003): Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, New York, NY.
- Anton A. (1996): Goal-Based Requirements Analysis. Requirements Engineering, Proceedings of the Second International Conference on, 136-144.
- Antonopoulos N., Anjum A., Gillam L. (2012): Intelligent techniques and architectures for autonomic clouds: introduction to the itaac special issue. Journal of Cloud Computing: Advances, Systems and Applications, 1:18.
- Arévalo J.M. (2010): Cloud Computing: fundamentos, diseño y arquitectura aplicados a un caso de estudio. Master's Thesis, University Rey Juan Carlos.
- Argente E., Giret A., Valero S., Julian V., Botti V. (2004): Survey of MAS Methods and Platforms focusing on organizational concepts. In: Vitria J., Radeva P. and Aguilo I. (ed).

Recent Advances in Artificial Intelligence Research and Development, Frontiers in Artificial Intelligence and Applications, 309-316.

- Argente E., Julian V., Botti V. (2006) Multi-agent system development based on organizations. Electronic Notes in Theoretical Computer Science, 150:55–71.
- Arkin A., Askary S., Bloch B., et al. (eds.) (2006): Web Services Business Process Execution Language, Version 2.0., Committee Draft. http://www.oasis-open.org/committees/download.php/14616/wsbpel-specification-draft.htm.
- Armbrust M. et al. (2010): A View of Cloud Computing. Communications of the ACM, 53 (4): 50-58.
- Arsanjani A. (2004): Service-oriented modeling and architecture. How to identify, specify, and realize services for your SOA. IBM.
- Artikis A., Kaponis D. and Pitt J. (2009): Multi- Agent Systems: Semantics and Dynamics of Organisational Models. Chapter in Dynamic Specifications of Norm-Governed Systems, IGI Globa.
- AWS (2010): Overview of Amazon Web Services. Amazon Web Services.
- Chandrasekaran B. and Josephson J.R. (1999): What Are Ontologies, and Why Do We Need Them?, Ohio State University V. Richard Benjamins, University of Amsterdam.
- Behsaz B., Jaferian P., Meybodi M.R. (2006): Comparison of Global Computing with Grid Computing. International Conference on Parallel and Distributed Computing, Applications and Technologies, 531-534.
- Bajo J., Julian V., Corchado J.M., Carrascosa C., de Paz, Y. Botti V., de Paz J. F. (2008): An Execution Time Planner for the ARTIS Agent Architecture. Engineering Applications of Artificial Intelligence, 21 (5): 769-784.
- Basili V. R., Briand L. C., Melo W. L. (1996): Assessing the Impact of Reuse on Quality and Productivity in Object-Oriented Systems. Communication of ACM.
- Baumer G., Breugst M., Choy S. and Magedanz T. (2000): Grasshopper: A Universal Agent Platform based on OMG MASIF and FIPA standards, in: Agents Technology in Europe.
- Bazán P. (2010): BPEL: una propuesta para el uso de Web Services. XIII Congreso Argentino de Ciencias de la Computación, IV Workshop de Ingeniería de Software y Bases de Datos, 306-315.
- Ben van Eyle (2001): Web Services – A Business Perspective on Platform Choice. www.theserverside.com.
- Berners-Lee T., Hendler J., Lassila O. (2002): The Semantic Web. Scientific American.
- Besson F., Leal P., Kon F., Goldman A., and Milojicic D. (2011): Supporting. Test-Driven Development of Web Service Choreographies. ACM.
- Bizagi. Process Modeler BPMN Business Process Modeling Notation.
- Boella G., Hulstijn J., Van Der Torre L. (2005): Virtual organizations as normative multiagent systems. In HICSS IEEE Computer Society.
- Boissier O. and Gateau B. (2007): Normative multi-agent organizations: Modeling, support and control. In Normative Multiagent Systems.
- Booth D., Champion M, Ferris C., McCabe F., Newcomer E., Orchard D. (2003): Web Services Architecture, Http://www.w3.org/TR/2003/WD-ws-arch-20030514/, W3C Working Draft.
- Booth, D., Haas, H., McCabe, F., et al. (eds.) (2004): Web services architecture. W3C Working Group Note. Available at: http://www.w3.org/TR/ws-arch/
- Borgo S., Gangemi A., Guarino N., Masolo C., Oltramari A. (2002): Ontology Roadmap.
- Borst, P. (1997): Construction of Engineering Ontologies for Knowledge Sharing and Reuse. Ph.D. Dissertation, Tweente University.
- Bou E., López-Sánchez M., Rodríguez-Aguilar J. A. (2007b): Towards self-configuration in autonomic electronic institutions. To appear In Coordination, Organization, Institutions and Norms in agent systems, Lecture Notes in Computer Science.Springer Verlag, 2007.
- Bou E., López-Sánchez M., and Rodríguez-Aguilar J. A. (2007a): Adaptation of autonomic electronic institutions through norms and institutional agents. To appear In Engineering Societies in an Agents World, Lecture Notes in Computer Science. Springer Verlag, 2007.
- Bou E., López-Sánchez M., Rodríguez-Aguilar J. A. and Sichman J. S. (2009): Adapting Autonomic Electronic Institutions in Heterogeneous Agent Societies. Organized

Adaption in Multi-Agent Systems, Lecture Notes in Computer Science, 5368: 18-35.
- Bradshaw J. (1997): An introduction to software agents. MIT Press Cambridge, MA, USA, 3-46, ISBN:0-262-52234-9.
- Bratman M.E., Israel D., Pollack M.E. (1988): Plans and resource-bounded practical reasoning. Computational Intelligence, 4: 349-355.
- Brescian P., Perini A., Giorgini P., Giunchiglia F. and Mylopoulos J. (2004): Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, 8 (3): 203-236. doi:http://dx.doi.org/10.1023/B:AGNT.0000018806.20944.ef.
- Brogi A., Corfini S., Popescu R. (2003): Composition-oriented service discovery. Proceedings of 5th International Conference on Autonomous Agents an Multi-Agent Systems (AAMAS).
- Brooks R. (1986): A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation 2(1).
- Burghoff U., Pareschi R., Karch H., Noehmeier M., Schlichter J. (1996): Constraint based Information Gathering for a Network Publication system. PAAM'96, First International Conference and Exhibition on the Practical Application of Internet Agents and Multi-Agent Technology, London 22-24.
- Burgin M. and Dodig G. (2009): A Systematic Approach to Artificial Agents. CoRR, abs/0902.3513, http://arxiv.org/abs/0902.3513.
- Burt R. (1987): Social contagion and innovation: Cohesion versus structural equivalence. American J. of Sociology, 92:1287-1335.
- Busi N., Ciancarini P., Gorrieri R. and Zavattaro G. (2001): Coordinations Models: A Guide Tour. Coordination of Internet Agents: Models, Technologies, and Applications (Omicini, Zambonelli, Klush y Tolksdorf, editors). Springer-Verlag, 6-24.
- Buyya R., Yeo C., Venugopala S., Broberg J., and Brandic I. (2009): Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, Elsevier 25: 599-616.
- Cammarata S., McArthur D., and Steeb R. (1988): Strategies of Cooperation in Distributed Problem Solving. In Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufmann.
- Cantera J.M., Hierro J.J., Romo P.A. (2007): La Web Semántica, la siguiente generación de Webs. Fundación Telefónica, Sociedad de la Información, http://sociedadinformacion.fundacion.telefonica.com/.
- Carrascosa C., Bajo J., Julián V., Corchado J.M., Botti V. (2008): Hybrid multi-agent architecture as a real-time problem-solving model. Expert Systems with Applications, 34 (1): 2-17. Pergamon-Elsevier Science LTD. doi:10.1016/j.eswa.2006.08.031.
- Carrascosa C., Giret A., Julian V., Rebollo M., Argente E., Botti V. (2009): Service Oriented MAS: An open architecture (Short Paper). Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Decker, Sichman, Sierra and Castelfranchi (eds.), May/10–15, Budapest, Hungary, 1291–1292.
- Carrascosa C., Rebollo M., Soler J., Julian V. and Botti V. (2003): SIMBA Architecture for Social Real-Time Domains EUMAS 2003: The First European Workshop on Multi-Agent Systems.
- Castelfranchi C., Miceli M. and Cesta A. (1992): Dependence relations among autonomous agent. Decentralized Artificial Intelligence, eds. Werner E. and Demazeau Y., SIGOIS Bull 13 (3), http://doi.acm.org/10.1145/152683.152697.
- Castelfrancis C. (1995): Guarantees for autonomy in cognitive agent architecture. In: M Wooldridge and NR Jennings eds. Intelligent Agents: Theories Architectures, and Languages (LNAI), Springer-Verlag 890: 56-70.
- Chandrasekaran B., Josephson J. R., Benjamins R. (1999): What are Ontologies, and Why do we need them? in IEEE INTELLIGENT SYSTEMS.
- Chang F., Dean J. et al (2006): Bigtable: a distributed storage system for structured data. In Proc of OSDI.
- Chang V., Bacigalupo D., Wills G., De-Roure D. (2010): A Categorisation of Cloud Computing Business Models. 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 509-512.
- Chapell D. (2009): INTRODUCING WINDOWS AZURE. Microsoft Corporation.
- Chu-Carroll C. and Carberry S. (1995): Conflict Detection and Resolution in

Collaborative Planning. INTELLIGENT AGENTS II: Agent Theories, Architectures and Languages, Springer Verlag, Berlin 111-127, Wooldridge, Michael J., Mueller P. and Tambe, Milind, (Ed.).

- Clark J., DeRose S. (1999): XML path language (XPath). Version 1.0. Tech rep, W3C. Available at http://www.w3.org/TR/xpath/
- Cloud Computing. Retrieved at 03. (June 2011): [http:/ / fclose.com/ b/ cloud-computing/ article/ mrcc-a-distributed-c-compiler-syste m-on-mapreduce/]
- Cohen P., Levesque H. (1990): Intention is Choice with Commitment. Artificial Intelligence 42(2-3): 213-261.
- Cole A. (2011): Load Balancing in the Cloud. ITBusiness Edge.
- Conte R. and Paolucci M. (2001): Intelligent social learning. Artificial Society and Social Simulation, 4(1):1–23.
- Corchado J.M., (2000): Inteligencia Artificial Distribuida. El concepto de agente. Department of Computers and Automation, University of Salamanca.
- Cossentino M. (2005): From requirements to code with the passi methodology. Agent Oriented Methodologies, IGI Global, 79-106, Web doi:10.4018/978-1-59140-581-8.ch004.
- Costello R. (2002): Building Web Services the REST Way. http://www.xfront.com/REST-Web-Services.html
- Criado N., Argente E., Julian V., Botti V. (2009): Designing Virtual Organizations. 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS2009), 55:440-449.
- Crosby S. and Brown D. (2007): The Virtualization Reality. Queue - Computer Architecture, 4 (10).
- Curbera F., DuftlerM., Khalaf R., Nagy W., Mukhi N., Weerawarana S. (March-April, 2002): Unraveling the Web Services Web : An Introduction to SOAP,WSDL and UDDI. IEEE Internet Computing, 6 (2): 86-93.
- d'Aquin M. and Noy N. (September, 2011): Where to publish and find ontologies? A survey of ontology libraries. Web Semantics: Science, Services and Agents on the World Wide Web.
- D'Inverno M. and Luck M. (2004): Understanding Agent Systems. Springer-Verlag. ISBN: 354040700.
- Deloach S. (2009): Multi-Agent Systems: Semantics and Dynamics of Organizational Models, IGI Global, Ch. Organizational Model for Adaptive Complex Systems, 1–26.
- Dignum F. (March 2011). Agents for games and simulation. Artificial Intelligence, Springer, 5920, ISBN 978-3-642-11198-3.
- Dignum V. (2004): A model for organizational interaction: based on agents, founded in logic, PhD. Thesis, NBC: 54.72: kunstmatige intelligentie, Proefschrift Universiteit Utrecht.
- Dignum V. and Dignum F. (2007): A logic for agent organization. In Proc. FAMAS@Agents'007.
- Dignum V., Vazquez-Salceda J., Dignum F. (2005): OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. Proceeding of 3rd International Workshop on Programming Multi-Agent Systems, Utretch, The Netherlands, 181-198.
- Dölitzscher F., Reich C., Sulistio A. (2010): Designing Cloud Services Adhering to Government Privacy Laws. In: Proceedings of 10th IEEE International Conference on Computer and Information Technology (CIT 2010). Furtwangen, Germany, 930-935.
- Doran J. E., Franklin N., Jennings N. R., Norman T. (1997): On cooperation in multi-agent systems. The Knowledge Engineering Review, 12(3):309-314.
- Drogul A. and Ferber J. (1992): Using Reactive Multi-Agent Systems in Simulation and Problem Solving. In Distributed Artificial Intelligence: Theory and Praxis (eds N.M. Avouris and L. Gasser).
- Dunin-Keplicz B. and Verbrugge R (2003): Calibrating Collective Commitments. In Marik, V., Müller, J., Pechoucek, M. (Eds.), Multi- Agent Systems and Applications, LNAI 2691, Springer 73-83.
- Dunn, M. F. y Knight, J. C. (1993): Certification of Reusable Software Parts. Technical Report CS-93-41, University of Virginia.
- Duong D. V. and Grefenstette J. (2005): The emulation of social institutions as a method of coevolution. In GECCO '05: Proceedings of the conference on Genetic and

evolutionary computation, ACM Press, 555-556.
- Durfee, E. H. (1999): Distributed problem solving and planning. In G. Weiß, editor, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.
- Duy T., Sato Y., Inoguchi Y. (2011): A prediction-based green scheduler for datacenters in clouds. IEICE Trans Inf Syst E94-D(9):1731-1741.
- Eisen M. (April, 2012): Introduction to Virtualization. The Long Island Chapter of the IEEE Circuits and Systems (CAS) Society.
- Erl T. (2006): Service-Oriented Architecture, Concepts, Technology, and Design. s.l. Prentice Hall Indiana. 0-13-185858-0.
- Erl T., (2009): SOA Design Patterns. The Prentice Hall Service-Oriented Computing Series.
- Escrivá M., Palanca J., Aranda G. and García F.A. (2006): A Jabberbased multiagent system platform. In Proc. of AAMAS06, 1282-1284.
- Escriva M., Palanca J., Aranda G., García A., Julian V. and Botti V. (2006): A Jabber-Based Multi-Agent System Platform. Proceeding of 5th International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 1282-1284.
- Esteva M. (2003): Electronic Institutions: from specification to development. Ph. D. Thesis, Technical University of Catalonia.
- Farquhar A. (1997): Ontolingua Tutorial. University of Stanford.
- Fedak G., Germain C., N´eri V., Cappello F. (2001): XtremWeb: a generic global computing system. In Proceedings of the 1st IEEE International Symposium on Cluster Computing and the Grid, 582-587.
- Ferber J., Gutknecht O., Michel F. (2004): From Agents to Organizations: an Organizational View of Multi-Agent Systems. In Giorgini P., Muller J., Odell J. (Eds.), Agent-Oriented Software Engineering VI, LNCS 2935 of Lecture Notes in Computer Science, Springer-Verlag 214-230.
- Ferber M., Rauber T., Hunold S. (2010): BPEL Remote Objects: Integrating BPEL Processes into Object-Oriented Applications. Services Computing (SCC), IEEE International Conference on, 33-40.
- Ferber J. (1994): Simulating with Reactive Agents. In Hillebrand E. and Stender J. (Eds.), Many Agent Simulation and Artificial Life, Amsterdam: IOS Press, 8-28.
- Fergusson I. (1995): Integrated Control and Coordinated Behavior: A case for agent models. Wooldridge M., Jennings N. (Eds.), Intelligent Agent: Theories, Architectures and Languages, LNAI 890, Springer 203-218.
- Fernandez M., Gomez-Perez A., Juristo N. (1997): Methodology: From Ontological Art Towards Ontological Engineering. From AAAI Technical Report ss-97-06.
- Fielding R. (2000): Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine.
- Finin L. and Mayfield (1997): KQML as an agent communication language. In Bradshaw, J., Software agents, The MIT Press.
- Fischer M. (1994): A Survey of METATEM, the Language and its Applications. In Gabbay D., Ohlbach H. (Eds.), Proceedings of the 1st. International Temporal Logic Conference, LNAI 827, Springer.
- Foner L. N. (1993): What´s an agent, anyway? A sociological case study. Agents Memo 93-01, Agents Group, MIT Media Lab.
- Foster I., Zhao Y., Raicu I., Lu S. (2008): Cloud Computing and Grid Computing 360-Degree Compared. Grid Computing Environments Workshop GCE '08: 1-10.
- Foster I. and Kesselman C. (1998): The Grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, ISBN:1-55860-475-8.
- Foster I., Kesselman C., Tuecke S. (2001): The anatomy of the grid: Enabling scalable virtual organizations. Int. J. High Perform, Comput. Appl, 15 (3): 200-222.
- Fox M. and Gruninger M. (1998): Enterprise Modeling. AI Magazine, AAAI Press, Fall, 109-121.
- Franklin S. and Graesser A. (1996): Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agent. ECAI '96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, 21-35.
- Dudek G., Jenkin M., Milios R., Wilkes D. (1993): A taxonomy for swarm robots. In Proceedings of IEEE/RSJ, Conference on Intelligent Robots and Systems.

- Galvez R. S. (2008): Fundamentos de WS-BPEL. Curso de Tecnologías Emergentes Multiplataformas. Department of Languages and Computer Science, University of Málaga.
- Garca-Magario I., Gómez-Sanz J. and Fuentes R. (2009) Ingenias development assisted with model transformation by-example: A practical case. In 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), 40-49.
- García F. (2007): Sistema basado en tecnologías del conocimiento para entornos de servicios web semánticos. University of Murcia.
- García J.A. (2012): IRIS & ME OWL-S - Sistema de Recomendación en Dispositivo Móvil con OWL-S . Master's Thesis. University of Salamanca.
- Garcia V.C., Lucrédio D., Alvaro A., de Almeida E.S., de Mattos R.P., de Lemos S.R. (2007): Towards a maturity model for a reuse incremental adoption. In: The 1st Brazilian Symposium on Software Components, Architecture and Reuse, Campinas, Sâo Paulo, Brazil, 61-74.
- Gasser L. and Ishida, T. (1991): A dynamic organizational architecture for adaptive problem solving. In: Proc. of AAAI-91, 185-190.
- Genesereth M., Nilsson N. (1987): Logical Foundations of Artificial Intelligence, Morgan Kaufmann Publishers In, ISBN-10: 0934613311.
- Genesereth M.R. and Ketchpel S.P. (1994): Software Agents. Communications of the ACM, 37(7) 48-53.
- Giampapa J.A. and Sycara K. (2002): Team-Oriented Agent Coordination in the RETSINA Multi-Agent System. Tech. Report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University. Presented at AAMAS 2002 Workshop on Teamwork and Coalition Formation.
- Gil A.B. (2011): Recuperación inteligente de contenidos digitales educativos. PhD Thesis, University of Salamanca.
- Giner P., Torres V., Pelechano V. (2010): Bridging the Gap between BPMN and WS-BPEL. M2M Transformations in Practice project DESTINO TIN2004-03534.
- Giret A., Julian V., Rebollo M., Argente E., Carrascosa C., Botti V. (2009): An Open Architecture for Service-Oriented Virtual Organizations. Seventh international Workshop on Programming Multi-Agent Systems (PROMAS), 23- 33.
- Goldner S. and Papproth A. (2011): Extending the BPMN Syntax for Requirements Management. Business Process Model and Notation, Lecture Notes in Business Information Processing, 95 (2): 142-147, DOI: 10.1007/978-3-642-25160-3_13.
- Gomez-Perez A. and Rojas-Amaya D. (1999): Ontological reengineering for reuse. In Knowledge Acquisition, Modeling and Management: 11th European Workshop, EKAW '99, Dagstuhl Castle, Germany. Proceedings, volume 1621 of Lecture Notes in Computer Science, page 139. Springer Berlin.
- Governor J., Nickull D., Hinchcliffe B. (2009): Web 2.0 Architectures Specific Patterns of Web 2.0. Chapter 7, O'Reilly Media / Adobe Dev Library.
- Greenwood D. and Calisti M. (2004): Engineering Web service - agent integration. Systems, Man and Cybernetics, IEEE International Conference on, 2: 1918-1925.
- Grosz B. y Sidner C. (1990). Plans for discourse. Intentions for Communication, pp 417-444, MIT Press.
- Grosz B., Kraus S. (1996): Collaborative Plans for Complex Group Actions. Artificial Intelligence 86: 269-358.
- Gruber T.R. (1995): Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human-Computer Studies, Special Issue on the Role o formal Ontology in the Information Technology, 43(5/6): 907-928.
- Gruninger M., Bodenreider O., Olken F., Obrst L., Yim P. (2007): Ontology Summit 2007 – Ontology, taxonomy, folksonomy: Understanding the distinctions. Applied Ontology, IOS Press, 191-200, DOI:10.3233/AO-2008-0052.
- Gruninger M. and Fox M.S. (1995): Methodology for the Design and Evaluation of Ontologies. In Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal.
- GTI-IA. (2009): An Abstract Architecture for Virtual Organizations: The THOMAS project. http://www.fipa.org/docs/THOMASarchitecture.pdf
- Guarino N. (1998): Formal Ontology in Information Systems. Proceedings of FOIS'98,

Trento, Italy, 6-8 June. Amsterdam, IOS Press, 3-15.
- Gutknecht O. and Ferber J. (2000): MadKit: a generic multiagent platform. In Proceedings of the Fourth international Conference on Autonomous Agents (Barcelona, Spain, June 03 - 07). AGENTS '00. ACM, New York, NY, 78-79.
- Muller H. J. (1996): Negotiation principles. In John Wiley & Sons, editor, Foundations of Distributed Artificial Intelligence, Norway, 211-230.
- Parunak H.V.D. (1996): Applications of distributed artificial intelligence in industry. In G. M. P. O'Hare and Jennings N. R., editors, Foundations of Distributed AI. JohnWiley & Sons.
- Hashimi S. (2003): Service-Oriented Architecture Explained. Published on ONDotNet.com (http://www.ondotnet.com/).
- Hass H. and Brown A. (2004): Web Services Glossary. W3C Working Group W3C, http://www.w3.org/TR/ws-gloss/.
- Havey M. (2005): Essential Business Process Modeling. 1st ed., O'Reilly Media, Sebastopol, CA.
- Hawley A. (2009): Virtual Infrastructure: What Is Required for the Cloud? ORACLE VM.
- Hendler J., Berners-Lee T., Miller E. (2002): Integrating Applications on the Semantic Web. Journal of the Institute of Electrical Engineers of Japan, 122(10): 676-680.
- Hernandez L., Botti V., Garcia-Fornes A. (2006): A deliberative scheduling technique for a real-time agent architecture. Engineering Applications of Artificial Intelligence 19 (5): 521-534.
- Hill J.B., Sinur J., Flint D., Melenovsky M.J. (2006): Gartner's position on business process management, Business Issues, Gartner, Stamford, CT.
- Hill J.B., Pezzini M., Natis, Y.V. (2008): Findings: confusion remains regarding BPM terminologies. Gartner Research, Stamford, CT. Vol. ID No. G00155817.
- Hoare C. A. R. (1978): Communicating sequential processes. Communications of the ACM, 21:666-677.
- Horling B. and Lesser V. (2005): Using ODML to Model Multi-Agent Organizations. In Proc. of the IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology.
- Howden N. (2001): JACK intelligent agents—summary of an agent infrastructure. In Proceedings of IEEE international conference on autonomous agents, Montreal.
- Hubner J.F., Sichman J.S., Boissier O. (2004): Using the Moise+ for a cooperative framework of mas reorganisation. In: LNAI Proc. of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04), Springer 3171: 506–515.
- Hubner J.F., Sichman J.S., Boissier O. (2006): S-Moise+: A middleware for developing organised multi-agent systems. In: Proc. Int. Workshop on Programming in MAS, LNCS, 3913: 64-78
- Huhns M. and Stephens L. (1999): Multiagent Systems and Societies of Agents. In: Weiss G. (Ed.), Multi-agent Systems: a Modern Approach to Distributed Artificial Intelligence, MIT Press.
- Huhns M. and Larry M. (1999): Multiagent Systems and Societies of Agents. In Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, ed. Gerhard Weiss, The MIT Press, Chapter 2, 121-164.
- Huth A. and Cebula J. (2011): The Basics of Cloud Computing. US-CERT.
- IEEE1517-2009 D2 (2009): IEEE1517 Standard for Information Technology - Software Life Cycle Processes - Reuse Processes: D2. Software Engineering Standards Committee of the IEEE Computer Society, USA.
- Martínez I. A. (2010): Modelado automático del comportamiento de agentes inteligentes. PhD Tesis, Department of Informatics. University Carlos III of Madrid.
- ISO (1991, 1994): Quality Management and Quality Assessment Standards, Part 3: Guidelines for the ISO 9001 to the Development, Supply and Maintenance of Software. ISO 9000-3, International Standards Organization.
- Euzenat J. and Shvaiko P. (2007): Ontology matching. Springer-Verlag, Berlin Heidelberg (DE), SBN (hardcover): 978-3-540-49611-3.
- Diggelen J.v. (2007): Achieving Semantic Interoperability in Multi-agent Systems: A Dialogue-based Approach, PhD Thesis, Utrecht University.
- Ferber J. (1999): Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Longman Publishing Co., Boston, MA, USA.
- Jaeger P. T., Lin J., Grimes J.M., Simmons S.N. (2009): Where is cloud? Geography,

Economics. Env. Jurisdiction Cloud Comput 14(5).
- Jennings N. (1993): Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems, The Knowledge Engineering Review, 8 (03): 223-250.
- Jennings N. and Wooldridge M. (1988). (eds.), Agent Technology: Foundations, Applications and Markets, Springer, ISBN-10: 3540635912.
- Jiao W.P. and Mei H. (2004): Automated adaptations to dynamic software architectures by using autonomous agents. Engineering Applications of Artificial Intelligence 17 (7): 749-770.
- Gottschalk K. et al. (2000): Web Services Architecture Overview: The Next Stage of Evolution for E-Business. http://www-106. ibm.com/developerworks/library/w-ovr.
- Kaelbling L.P. (1991): A situated automata approach to the design of embedded agents. SIGART Bulletin, 2(4):85-88.
- Karlsson E. A. (1996): Software Reuse: A Holistic Approach. Wiley Series in Software based Systems. John Wiley and sons.
- Keil F. C. (1989): Concepts, Kinds, and Cognitive Development. MIT Press.
- Kemsley S. (2007): Business Process Design. TIBCO.
- Kennedy S., Molloy O., Stewar R., Jacob P., Maleshkova M., Doheny F. (2012): A Semantically Automated Protocol Adapter for Mapping SOAP Web Services to RESTful HTTP Format to Enable the Web Infrastructure, Enhance Web Service Interoperability and Ease Web Service Migration. Future Internet, 4(2): 372-395.
- Khondoker M. R. and Mueller P. (2010): Comparing Ontology Development Tools Based on an Online Survey. In Proceedings of the World Congress on Engineering, Vol. I.
- Kittock J. E. (1993): Emergent conventions and the structure of multi-agent systems. In Lectures in Complex systems: the proceedings of the 1993 Complex systems summer school, Santa Fe Institute Studies in the Sciences of Complexity Lecture Volume VI, Santa Fe Institute, Addison-Wesley 507–521.
- Klusch M. and Sycara K. (2001): Brokering and matchmaking for coordination of agent societies: a survey. En Coordination of Internet Agents: Models, Technologies, and Applications (Omicini et al.), Springer 197-224
- Knight, J. C. y Dunn, M. F. (1998): Software Quality through Domain-Driven Certification. Annals of Software Engineering, 5: 293-315.
- Ko R., Lee S. and Lee E. (2009): Business process management (BPM) standards: a survey. Emerald Business Process Management Journal, 15 (5): 744-791.
- Kortz D. and Gray R. (1999): Mobile Agents and the future of the Internet. Department of Computer Science / Thayer School of Engineering, Dartmouth College.
- Koskela M. and Haajanen J. (2007): Business process modeling and execution: tools and technologies report for the SOAMeS project. VTT Research Notes No. 2407, VTT Technical Research Centre of Finland, Espoo.
- Krueger C. W. (1992): Software Reuse. ACM Computing Surveys, 24(2):131-183.
- Labidi S. and Lejouad W. (1993): Del'intelligence artificielle distribu'eeaux syst'emes multiagents.
- Lakkaraju K. and Gasser L. (2008): Norm Emergence in Complex Ambiguous Situations. In Proceedings of the AAAI Workshop on Coordination, Organizations, Institutions and Norms AAAI, Chicago, July.
- Lapadula A., Pugliese R., Tiezzi F. (2010): A WSDL-based type system for asynchronous WS-BPEL Processes. Form Methods Syst Des, 38: 119-157.
- Lee C.H. and Hwang S.Y. (2009): A Model for Web Services Data in Support of Web Service Composition and Optimization. ISBN: 978-0-7695-3708-5.
- Lee J.M. (1997): Riemannian Manifolds. An introduction to Curvature. Springer- Verlag, New Tork, Inc.
- Lemley M. and O'Brien D. (1997): Encouraging Software Reuse. Stanford Law Review, 49, 255.
- Lésperance Y., Levesque H., Lin F., Marcu D., Reiter R., Scherl R. (1996): Foundations of a Logical Approach to Agent Programming. In Wooldridge M., Müller J., Tambe M. (Eds.): Intelligent Agents II, LNAI 1037, Springer.
- Li K., Verma K., Mulye R., Rabbani R., Miller J., Sheth A. (2006): Designing Semantic Web Processes: The Wsdl-S Approach. Semantic Web Services, Processes and Applications Semantic Web and Beyond, Part II, 3: 161-193.

- Ligeza A., Kluza K., Potempa T. (2012): AI Approach to Formal Analysis of BPMN Models. Towards a Logical Model for BPMN Diagrams. Preprints of the Federated Conference on Computer Science and Information Systems, 959–962.
- López F. (1999): Overview of Methodologies for Building Ontologies. In Proceedings of IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5) in Stockholm Sweden, August 2.
- Lowe-Norris A.G. and Denn R. (2000): Windows 2000 Active Directory, O'Reilly & Associates, Sebastopol, CA.
- Lu J., Chen G., Yu X. (2011): Modelling, Analysis and Control of Multi-Agent Systems: A Brief Overview. Circuits and Systems (ISCAS), IEEE International Symposium on, 2103-2106, ISSN: 0271-4302.
- Luck M. and McBurney P. (2008): Computing as interaction: Agent and agreement technologies, in: IEEE SMC Conference on Distributed Human-Machine Systems: 1-6.
- Luo J., Montrose B., Kim A., Khashnobish A., Kang M. (2006): Adding OWL-S Support to the Existing UDDI Infrastructure. Web Services, ICWS '06, International Conference on, 153-162.
- Wooldridge M. and Jennings N. R. (1995): Intelligent Agents: Theory and Practice. In KNOWLEDGE ENGINEERING REVIEW 10(2).
- Maamar Z., Mostefaoui S.K., Yahyaoui H. (2005): Toward an agent-based and context-oriented approach for Web services composition. Knowledge and Data Engineering, IEEE Transactions on, 17: 686-697.
- Maedche A. and Staab S. (2001): Learning Ontologies for the Semantic Web. In Semantic Web Workshop, Hongkong, China
- Maes P. (1990)(Ed.): Designing Autonomous Agents. MIT Press.
- Maes P. (1991): The agent network architecture (ANA). SIGART Bulletin, 2(4): 115-120.
- Mahmoud Q. (2005): Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). ORACLE.
- Malone T. and Crowston K. (March, 1994): The Interdisciplinary Study of Coordination. ACM Computing Surveys 26(1).
- Mancarella S. (January 2011): Business Process Modelling Notation – A tutorial. OMG SOA Healthcare. SPARX Systems.
- Martin D., Burstein M., McDermott D., McIlraith S., Paolucci M., Sycara K., McGuiness D., Sirirn E., Srinivasan N. (2006): Bringing Semantics to Web Services with OWL-S. Springer Science + Business Media, LLC.
- Mas A. (2005): Agentes software y sistemas multiagente: conceptos, arquitecturas y aplicaciones. 29-64, ISBN 84-205-4367-5.
- Massonet P., Naqvi S., Ponsard C., Latanicki J., Rochwerger B., Villari M. (2011): A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW). 2011 IEEE International Symposium on, 1510-1517.
- Mataric M.J. (1997): Learning Social Behavior. In Robotics and Autonomous Systems, 20: 191-204.
- Mathes M., Gartner J., Dohmann H., Freisleben B. (2009): SOAP4IPC: A Real-Time SOAP Engine for Industrial Automation. Parallel, Distributed and Network-based Processing, 17th Euromicro International Conference on, 220-226.
- McAllester D. and Rosenblitt D. (1991): Systematic nonlinear planning. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), 2: 634-639, Anaheim, California, USA, AAA Press/MIT Press.
- McIlraith S., Son T. (2002): Adapting golog for composition of semantic web services. In Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR), 482-493. Toulouse, France, April.
- McIlraith S.A. and Martin D.L. (2003): Bringing semantics to Web services 18 (1): 90-93.
- Mell P. and Grance T. (2011): NIST Definition of Cloud Computing v15. csrc.nist.gov/groups/ SNS/cloud-computing/cloud-def-v15.doc .
- Mendling J. and Neumann G. (2005): A comparison of XML interchange formats for business process modeling, Workflow Handbook, Future Strategies, Lighthouse Point, FL.

- Menzel C. (2003): Reference Ontologies —Application Ontologies: Either/Or or Both/And? Texas A&M University.
- Milenkovic M., Robinson H., Knauerhase R., Barkai D., Garg S., Tewari V., Anderson T., Bowman M. (2003): Toward Internet distributed computing. IEEE Computer, 36 (5): 38-46.
- Moemeng C., Wang C., Longbing C. (2011): Obtaining an optimal MAS configuration for Agent-Enhanced Mining Using Constraint optimization. Agents and Data Mining Interaction, Part I, 46-57, DOI:10.1007/978-3-642-27609-5_5.
- Molesini A., Omicini A., Denti E. and Ricci A. (2006): SODA. A roadmap to artefacts. Engineering Societies in the Agents World VI LNAI 3963: 49–62.
- Morrison I. and Nugrahanto S. (2007): Decision Support With BPEL and Web Services. International Journal of Healthcare Information Systems and Informatics (IJHISI), 2 (2).
- Morrison I., Lewis B. and Nugrahanto S. (2006): Modelling in Clinical Practice with Web Services and BPEL. International Journal of E-Business Research (IJEBR), Chapter X, DOI: 10.4018/jebr.2006010103.
- Noy N. and McGuinness D. (2001): Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics (SMI), Department of Medicine, Stanford University School of Medicine.
- Nakrani S. and Tovey C. (2004): On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. Adaptive Behavior 12: 223-240.
- Nau D., Au T.C., Ilghami O., et al. (2003): SHOP2: An HTN planning system. J. Artif. Intell. Res. 20: 379-404.
- Nesse P.J., Undheim A., Solsvik F.H., Dao M., Salant E., Lopez J.M., Elicegui J.M. (2011): Exploiting cloud computing: A proposed methodology for generating new business. 15th International Conference on Intelligence in Next Generation Networks (ICIN), 241-246.
- Newcomer E. (2002). Understanding Web Services, XML, WSDL, SOAP and UDDI. Independent Technology Guides. David Chapell: Series Editos. ISBN: 0-201-75081-3
- Nitzsche J., Van Lessen T. Karastoyanova D., Leymann F. (2007): BPEL for Semantic Web Services (BPEL4SWS). Institute of Architecture of Application Systems, University of Stuttgart.
- Noy N. F. and McGuiness D. L. (2001): Ontology Development 101: A Guide to Creating Your First Ontology. Stanford University. SMI-2001-0880.
- Nwana H. S. (1996): Software Agents: An Overview. Knowledge Engineering Review, Cambridge University Press,11(3):1-40,
- Nwana H.S., Ndumu D.T., Lee L.C. and Collis J.C. (1999): ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems Agents, Applied Artifical Intelligence Journal, 13: 129-186.
- O'hare G.M. and Jennings N.R. (1996): Foundations of Distributed Artificial Intelligence. Willey-Interscience.
- OASIS (April, 2007): Web Services Business Process Execution Language. Version 2.0, OASIS Standard.
- Odell J. (October 2010): Agent Technology: An Overview. Ann Arbor, MI USA, http://www.jamesodell.com.
- Omicini A., Ricci A. and Viroli M. (2004): Coordination artifacts: Environment-based coordination for intelligent agents. In Proceedings of 3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 286-293.
- Osiecki L., Phillips M., and Scibilia J. (2011): Understanding and Leveraging a Supplir's CMMI Efforts: A Guidebook for Acquirers (Revised for V1.3). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2011-TR-023.
- Ossowski S. (2001): Agent Coordination by Constraint Optimisation. Electronic Notes in Theoretical Computer Science 48, Elsevier.
- Ossowski, S. (1998): Co-ordination in Artificial Agent Societies, LNAI, Springer 1535.
- Ouyang C., Dumas M., ter Hofsetede A., van derl Aalst W. (2006): From BPM Process Models to BPEL Web Services. In Proceedings of 4th International Conference on Web
- Services. IEEE Computer Society, 285-292.
- Owen M. and Raj J. (2003): BPMN and Business Process Management – An

Introduction to the New Business Process Modeling Standard. TechRepublic/US
- OWL-S (2012): Semantic Markup for Web Services. Available in http://www.w3.org/Submission/OWL-S/.
- Padala P., Hou K-Y et al. (2009): Automated control of multiple virtualized resources. In Proc of EuroSys.
- Panait L. and Luke S. (2005): Cooperative Multi-Agente Learning: The State of Art. George Mason University.
- Paolucci M., Kwamura T., Payne T., ycara K. (2002): Importing the semantic Web in UDDI. Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, 225-263.
- Papazoglou M. (2008): Web Services: Principles and Technology. s.l., Prentic Hall.
- Parimala N. and Saini A. (2011): X-WSDL: An Extension to WSDL and its mapping to X-UDDI. The International Journal of Information Studies, 3 (3): 115-127.
- Pasley J. (May/June 2005). How BPEL and SOA Are Changing Web Services Development. Published by the IEEE Computer Society, IEEE Internet Computing, 60-67.
- Pattison H. E., Corkill D. D. and Lesser V. R. (1987): Distributed Artificial Intelligence. Chapter Instantiating Descriptions of Organizational Structures, 59-96. Pitman Publishers.
- Pautasso C., Zimmerman O., and Leymann F. (2008): Restful web services vs. "big"' web services: making the right architectural decision. Published in WWW '08 Proceedings of the 17th international conference on World Wide Web, 805-814.
- Pavon J. and Gomez-Sanz J.J. (2003): Agent oriented software engineering with INGENIAS. In Procedings of CEECMAS, 2691: 394– 403.
- Pavon J., Gomez-Sanz J., Fuentes R. (2005): The INGENIAS Methodology and Tools, Idea Group Publishing, article IX: 236–276.
- Pedrinaci C., Brelage C., van Lessen T., Domingue J.; Karastoyanova d. and Leymann F. (2008): Semantic business process management: scaling up the management of business processes. In 2nd IEEE International Conference on Semantic Computing (ICSC), Santa Clara, CA, USA.
- Peiró J.M. (1990): <<Las Nuevas Tecnologías» en Organización. Nuevas perspectivas psicosociales, Barcelona, PPU S.A. Penberthy y Weld, Penberthy J.S. and Weld D., UCPOP: A Sound, Complete, Partial-Order Planner for ADL. Third International Conference on Knowledge
- Poulin J. S. (1997): Measuring software reuse - principles, practices, and economic models. Addison-Wesley-Longman 1997, ISBN: 978-0-201-63413-6, I-XIX, 1-195.
- Poulin J. S. (2006): The Business Case for Software Reuse: Reuse Metrics, Economic Models, Organizational Issues, and Case Studies. Lecture Notes in Computer Science 4039: 439.
- Prieto, R. (1993). Status Report: Software Reusability. IEEE Software, 10 (3): 61-66.
- Prieto, R. (1996). Reuse as a New Paradigm for Software Development. In Sars- har, M., editor, Systematic Reuse: Issues in Initiating and Improving a Reuse Program, London. Springer-Verlag.
- Radgui M., Saidi R., Moulina S. (September 2012): A Pattern for the Decomposition of Business Processes. Special Issue of International Journal of Computer Applications (0975 – 8887) on Software Engineering, Databases and Expert Systems – SEDEXS.
- Rahman S., Bhardwaj A., Pathak M., Rathore S. (2012): INTRODUCTION OF KNOWLEDGE MANAGEMENT ARCHITECTURE USING MULTI AGENT. International Journal of Computer Science & Information Technology (IJCSIT), 3 (6): 229-239.
- Ralyté J., Mirbel I., Deneckère r. (2011): Engineering Methods in the Service-Oriented Context. 4th IFIP WG 8. 1 Working conference on method engineering. Paris, France.
- Randles M., Lamb D., Taleb-Bendiab A. (2010): A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing. IEEE, 24th International Conference on Advanced Information Networking and Applications Workshops.
- Rao A. and Georgeff M. (June, 1995): Bdi agents: From theory to practice. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 312-319, Menlo Park, California. AAAI Press.
- Rao J. (2012): Semantic Web Service Composition via Logic-based Program Synthesis, ISBN 82-471-6464-7

- Rao A. and Georgeff M. (1992): An Abstract Architecture for Rational Agent. In Rich C., Swartout W., Nebel B. (Eds.): Proc. of KR'92, Morgan Kaufmann.
- Ravichandran T. (1999): Software reusability as synchronous innovation: a test of four theoretical models. European Journal of Information Systems 8: 83-199
- Razavi R., Perrot J., Guelfi N. (2005): Adaptive modeling: an approach and a method for implementing adaptive agents. Lecture Notes in Artificial Intelligence, 3446: 136-148.
- RedHat (2007): Virtualization Guide. Red Hat Virtualization.
- Rehesaar H. (2011): Capability Assessment for Introducing Component Reuse. Lecture Notes in Computer Science, Springer 6727: 87-101.
- Rehesaar H. (2011): Capability Assessment for Introducing Component Reuse, Top Productivity through Software Reuse. Lecture Notes in Computer Science, Springer 6727: 87-101
- Rodríguez S. (2010): Modelo Adaptativo para organizaciones virtuales de agentes. PhD Thesis. University of Salamanca.
- Rodríguez S., de Paz Y., Bajo J., Corchado J. M. (2011): Social-based Planning Model for Multiagent Systems. Expert Systems with Applications 38 (38): 13005-13023.
- Rodríguez S., Pérez-Lancho B., De Paz J.F., Bajo J., Corchado J.M. (July, 2009): Ovamah: Multiagent-based Adaptive Virtual Organizations. 12th International Conference on Information Fusion, Seattle, Washington, USA.
- Romero R. (2007): Especificación OWL de una ontología para teleeducación en la Web Semántica. PhD Thesis, Polytechnic University of Valencia.
- Rosenberg D. (November, 2011): Are databases in the cloud really all that different?, CNET.
- Rosenschein J. and Zlotkin G. (1994): Rules of Encounter - Designing Conventions for Automated Negotiation among Computers. MIT Press.
- Rosu M.C. (july, 2007): A-SOAP: Adaptive SOAP Message Processing and Compression. Web Services. ICWS, IEEE International Conference, 200-207
- Ruckhaus E. (2006): RDF and RDF-Schema. http://www.ldc.usb.ve/~ruckhaus/materias/ci7453/clase4.pdf
- Rueda S., García J., Simari G. R. (2002): Argument-based negotiation among bdi agents. Journal of Computer Science and Technology, 2 (7): 1-8.
- Russell S. and Norvig P. (1995): Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice-Hall.
- Saffre F., Tateson R., Halloy J., Shackleton M., Deneubourg J.L. (march, 2008): Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications. The Computer Journal, 52 (4): 397-412.
- Sametinger J. (1997): Software Engineering with Reusable Components. Springer Ver-lag, ISBN-10: 3540626956.
- Sanderson D. (November, 2009): Programming Google App Engine. O'Reilly Google Press.
- Sansores C., Pavón J. (2005): Simulación social basada en agentes. Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial, ISSN 1137- 3601, 9 (25): 71-78.
- Schaaf M., Koschel A., Grivas S .G. and Astrova I. (2010): An active DBMS style activity service for cloud environments. Cloud Computing: The First International Conference on Cloud Computing, GRIDs, and Virtualization in Cloud Computing, Computation World 2010 IARIA, 80-85.
- Schein E. H. (2004): Organizational culture and leadership. Jossey-Bass Edition: 3 - 437 pages.
- Schelling T. C. (1960): The Strategy of Conflict. Harvard University Press, Cambridge, MA.
- Schmid K. (2011a): 12th International Conference on Software Reuse, ICSR 2011, Pohang, South Korea. Series: Lecture Notes in Computer Science, 6727.
- Schmid K. (2011b): Top Productivity through Software Reuse. 12th International Conference on Software Reuse, ICSR, Pohang, South Korea, Proceedings, ISBN: 978-3-642-21346-5.
- Serrano J.M. and Ossowski S. (2004): RICA-J -- A Dialogue-Driven Software Framework for the Implementation of Multiagent Systems. JISBD Taller en Desarrollo de Sistemas Multiagente (DESMA-2004), Málaga, 48-61.

- Shang R. D., Mohan K., Lang K. R., Vragov R. (2012): A market mechanism for software component reuse: opportunities and barriers. Proceedings of the 14th Annual International Conference on Electronic Commerce (ICEC), 62-69.
- Shapiro, R. (2002): A technical comparison of XPDL, BPML and BPEL4WS. CAPE VISIONS, Software To Simplify Complexity. OASIS XML Cover Pages, Rotterdam.
- Sherif K. and Vinze A. S. (2003): Barriers to adoption of software reuse: a qualitative study. Information and Management 41 (2): 159-175.
- Shoham Y. and Tennenholtz M. (1992): Emergent conventions in multi-agent systems. In Proceedings of Knowledge Representation and Reasoning, 225-231.
- Shoham Y. and Tennenholtz M. (1997): On the emergence of social conventions: Modeling, analysis, and simulations. Artificial Intel ligence, 94(1-2):139-166.
- Silberschatz G. (1994): Operating System concepts. Chapter 17, Distributed file systems. Addison-Wesley Publishing Company, ISBN 0-201-59292-4.
- Simchi-Levi D., Kaminsky P., Simchi-Levi E. (2000): Designing and Managing the Supply Chain: Concepts, Strategies and Case Studies. McGraw-Hill/Irwin, New York, NY.
- Singh A. and Malhotra M. (2012): Agent Based Framework for Scalability in Cloud Computing. International Journal of Computer Science & Engineering Technology (IJCSET)
- Sirin E., Parsia B., Cuenca B., Kalyanpur A., Katz Y. (2003): Pellet: A Practical OWL-DL Reasoner at University of Maryland, MIND Lab.
- Sowa J.F. (2000): Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, CA: Brooks Cole Publishing Co.
- Spencer H. (1896): The Study of Sociology. New york, D. Appleton and company.
- Srinivasan N., Paolucci M., Sycara K. (July, 2004): Adding OWL-S to UDDI, implementation and throughput. Proceedings of the first international workshop on Semantic Web Services and Web Process Composition – SWSWPC.
- Stage A. and Setzer T. (2009): Network-aware migration control and scheduling of differentiated virtual machine workloads. In Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09). IEEE Computer Society, Washington, DC, USA, 9-14.
- Stone P. (1997): Reactive vs. Deliberative agents. Multiagent Systems: A Survey from a Machine Learning Perspective, Computer Science Department, Carnegie Mellon University.
- Stone P. and Veloso M. (2000): Multiagent systems: A survey from a machine learning perspective. Autonomous Robots, 8(3):345–383.
- Stroustrup B. (1996): Language-technical Aspects of Reuse. In Sitaraman M., editor, Fourth International Conference on Software Reuse, IEEE Computer Society Press, 11-19.
- Sulistio. A, Reich C., Dölitzscher F. (2009): Cloud Infrastructure & Applications - CloudIA. In Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09), Beijing, China, 583-588.
- Swartout B., Ramesh P., Knight K., Russ T. (1997): Toward Distributed Use of Large Scale Ontologies. In AAAI'97, Spring Symposium on Ontological Engineering. Stanford University, California.
  http://ksi.cpsc.ucalgary.ca/KAW/KAW96/swartout/Banff_96_final_2.html
- Swartout W., Tate A. (1999): Ontologies. IEEE Intelligent Systems, 14 (1): 18-19.
- Sycara K., Paolucci M., Ankolekar A., Srinivasan N. (2003): Automated discovery, interaction and composition of Semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier 1 (1): 27-46.
- Sycara K., Paolucci M., van Velsen M., Giampapa J. (2003): The RETSINA MAS Infrastructure. Journal of Autonomous Agents and Multi-Agent Systems 7(1 & 2), Kluwer Academic Publishers, 2003.
- T. W. Malone. (1988): What is coordantion theory? In National Science Foundation Coordination Theory Workshop, MIP, EE.UU.
- Talia D. (2012): Cloud Computing and Software Agents: Towards Cloud Intelligent Services. WOA, CEUR Workshop Proceedings, 741: 2-6. CEUR-WS.org.
- Tambe M. (1997): Towards Flexible Teamwork. Journal of Artificial Intelligence Research 7: 83-124.

- Tan M. (1993): Multiagents Reinforcement Learning: Independent vs. Cooperative Agents. In Proceedings of the Tenth International Conference on Machine Learning, Morgan Kaufmann, 330-337.
- Tao J., Kunze M., Castellanos A. C. , Kramer, D., Karl, W. (2008): Scientific Cloud Computing: Early Definition and Experience High Performance Computing and Communications. HPCC '08, 10th IEEE International Conference, 825-830.
- Tere G.M. and Jadhav B.T. (2012): Designing Application Framework using WSDL. International Conference & Workshop on Recent Trends in Technology (TCET).
- Toma I., Foxvog D. (2006): Non-functional properties of web services. WSMO Working draft. http://www.wsmo.org/TR/d28/d28.4/v0.1/20060616/.
- Uschold M. and Gruninger M. (1996): Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, 11: 93-136.
- van der Aalst W.M.P. de Beer, H.T. and van Dongen B.F. (2005). Process Mining and Verification of Properties: An Approach Based on Temporal Logic, Springer, New York, NY.
- van der Aalst W.M.P., Dumas M., ter Hofstede A.H.M., Russell N., Verbeek H.M.W., Wohed P. (2005): Life after BPEL?, Lecture Notes in Computer Science, Springer 3670: 35-50.
- van der Aalst, W.M.P. (2003): Patterns and XPDL: a critical evaluation of the XML process definition language. QUT Technical Report, FIT-TR-2003-06, Queensland University of Technology, Brisbane.
- van der Aalst W.M.P. (2005): Don't go with the flow: web services composition standards exposed. IEEE Intelligent Systems, DOI: 10.1041/x1072s-2003.
- van Gorp P. and Dijkman R. (September, 2012): A visual token-based formalization of BPMN 2.0 based on in-place transformations. Information and Software Technology, 55 (2): 365-394.
- van Heijst G., Schreibe A., and Wielinga B. J. (1997): Using Explicit Ontologies in KBS Development. International Journal of Human and Computer Studies, 46(2/3): 183-292.
- Vaquero L., Rodero-Merino L., Caceres J., Lindner M. (2009): A break in the clouds: towards a cloud definition. ACM SIGCOMM computer communications review.
- Vidal J.M. (March, 2010): Fundamentals of Multi-Agent Systems with Net Logo Examples. Copyrigh José M. Vidal. All rights reserved.
- Villatoro D. and Sabater-Mir J. (2008c): Towards the Group Formation through Social Norms. Proceedings of the Sixth European Workshop on Multi-Agent Systems (EUMAS08).
- Villatoro D. and Sabater-Mir J. (2008a): Categorizing Social Norms in a Simulated Resource Gathering Society. Proceedings of the AAAI Workshop on Coordination, Organizations, Institutions and Norms (COIN @ AAAI08).
- Villatoro D. and Sabater-Mir J. (2008b): Mechanisms for Social Norms Support in Virtual Societies. Proceedings of the Fifth Conference of the European Social Simulation Association (ESSA08).
- Viroli M., Denti E., Ricci A. (January, 2007): Engineering a BPEL orchestration engine as a multi-agent system. Science of Computer Programming, 66: 226–245.
- Von Martial F. (1992): Co-ordinating Plans of Autonomous Agents. LNAI 610, Springer.
- W3C OWL-S (2012): http://www.w3.org/Submission/OWL-S/#5.
- Walsh A.E. (2002): UDDI, SOAP and WSDL: The Web Services Specification. 1st edition ed. 2002, Pearson Education.
- Wang B. and Liu L. (2012): Ontology-Based Multi-Agent Diagnostic System of Enterprise Management. Computer Science and Automation Engineering (CSAE), IEEE International Conference on, 577-581.
- Wang L., and Lazewsky G. (2008): Scientific Cloud Computing: Early Definition and Experience. Service Oriented Cyberinfrastruture Lab, Rochester Institute of Technology.
- Wang G., Wong T.N., Wang X. (2012): An Ontology based Approach to Organize Multi-Agent Assisted Supply Chain Negotiations. Computers & Industrial Engineering, Elsevier 65 (1): 2-15.

- Weerawarana S., Curbera F., Leymann F., Storey T. and Ferguson D. (2005): Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice-Hall, Upper Saddle River.
- Wegner P. (1997): Why Interaction is more Powerful than Algorithms. communications of the ACM, 5(40): 80-91.
- Wei-TekTsai, Sun X., and Balasooriya, J. Service-Oriented Cloud Computing Architecture (April, 2010). Information Technology: New Generations (ITNG), Seventh International Conference, 684-689.
- Werner E. (1989): Cooperating agents: A unified theory of communication and social structure. Distributed Artificial, Vol. II: 3-36. Pitman Publishing: London andMorgan Kaufmann: San Mateo, CA.
- Weyns D. and Georgeff M. (2010): Self-Adaptation Using Multiagent Systems. Software, IEEE, 27 (1).
- White S. (2005): Using BPMN to Model a BPEL Process. IBM Corp., United States.
- White S. (October, 2006): Introduction to BPMN. IBM Software Group.
- Wiederhold G. (1992): Mediators in the Architecture of Future Information Systems. IEEE Computer Magazine.
- Wohed P., van der Aalst W. M. P., Dumas M., ter Hofstede A. H. M. (Octbober, 2003): Analysis of Web services composition languages: The case of BPEL4WS. In Proceedings of the 22nd International Conference on Conceptual Modeling (ER'2003), Chicago, 200-215.
- Wooldridge M. (2002): An Introduction to Multiagent Systems. Chichester, England, John Wiley & Sons, ISBN 047149691X.
- Xu Y., Singh N., Deshpande S. (2011): Reuse by Placement: A Paradigm for Cross-Domain Software Reuse with High Level of Granularity. Top Productivity through Software Reuse, Lecture Notes in Computer Science, Springer 6727: 69-77
- Yukio S., Silva H.m. and Barthès J.P. (April 2011): Agent and multi-agent applications to support distributed communities of practice: a short review. Autonomous Agents and Multi-Agent Systems, 25 (1): 87-129.
- Zambonelli F. and Parunak H. (2002): From design to intention: Signs of a revolution. In Proc. 1st Int. Joint Conference on AAMAS, 455-456.
- Zambonelli F., Gleizes M.P., Mamei M. and Tolksdorf R. (2004): Spray computers: frontiers of self-organisation for pervasive computing. Workshop on Enabling Techonologies: Infrastructure for Collaborative Enterprises - WETICE04, IEEE Computer Society, 397-402.
- Zambonelli F., Jennings N. R., Wooldridge M. (2003): Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology, 12(3): 317-370.
- Zeng L., Benatallah B., Dumas M., Kalagnanam J., Sheng Q. (2003): Quality Driven Web services Composition.
- Zhang Q., Cheng L., and Boutaba R. (April, 2010): Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, Springer 1 (1): 7-18.
- Zsu M.T. and Valduriez P. (1999): Principles of Distributed Database Systems. Third Edition, Springer, ISBN 978-1-4419-8833-1.

7.1 WEB REFERENCES

- Active endpoints. (2009). In Depth. http://www.activevos.com/
- ActiveBPEL. (2012). http://swik.net/ActiveBPEL
- Amazon Elastic Computing Cloud (2012). aws.amazon.com/ec2
- Amazon Security. (2012). Amazon Web Services Security. http://aws.amazon.com/es/security/
- Answers. (2012). Cloud Computing. http://www.answers.com/topic/cloud-computing
- Apache Cassandra. (2012). http://cassandra.apache.org/
- Azure Platform. (2011). Microsoft Azure. Introducing Microsoft Azure Platform.

- Bluelock. (2012). http://www.bluelock.com/
- CAR. (2010). Arquitectura Orientada a Servicios (SOA) Cómo reformular la Arquitectura Corporativa para alcanzar el alto rendimiento. Un estudio publicado por el Centro de Alto Rendimiento de Accenture (CAR)
- CherryPy. (2012). http://www.cherrypy.org/
- Citrix. (2012). http://www.citrix.es/lang/Es-es/home.asp
- CouchDB.(Accedido en 2012). http://couchdb.apache.org/
- Django. (2012). http://django.es/
- Dropbox. (2012). https://www.dropbox.com/
- Elasticdrive Project. (2012). http://www.elasticdrive.com/
- EuroCloud Deutschland_eco eV (2011). Eurocloud star audit saas certificate. http://www.saas-audit.de
- FIPA. (2012). http://fipa-os.sourceforge.net/
- FIPA. (2012a). http://www.fipa.org
- FlexiScale Cloud Comp and Hosting. (2012). www.flexiscale.com
- GoGridCloud Hosting (Accedido 2012). CLoud Computing and Hybrid Infrastructure from GoGrid, http://www.gogrid.com
- Google App Engine. (2012). URL http://code.google.com/appengine
- Google Developers. (2012).¿Por qué App Engine? https://developers.google.com/appengine/whyappengine?hl=es
- GramaticasFormales. (2011). Web Services. http://gramaticasformales.wordpress.com/2011/01/24/web-services/
- IBM BPEL4WS. (2012). http://www.ibm.com/developerworks/webservices/library/ws-bpelcol4/
- IBM. (2009). WebSphere Business Integration Adapters. http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/index.jsp?topic=/com.ibm.wbia_adapters.doc/doc/webservices/webservices17.htm
- IBM. (January, 2008). How service-oriented architecture (SOA) impacts your IT infrastructure Satisfying the demands of dynamic business processes. IBM Global Technology Services
- IBM. (2012). IBM Blue Cloud project http://www-03.ibm.com/press/us/en/pressrelease/22613.wss/
- INSA, NEBUSENS, DIVISAIT, JLLORENS and USAL (2012). Análisis del Estado del Arte y la Problemática.
- JADE (2012) http://jade.tilab.com/
- Joyent. (2012). http://joyent.com/
- Microsoft (2010). Service Oriented Architecture Infrastructure Business agility through service virtualization.
- Microsoft SQL Server (2012). http://www.microsoft.com/es-es/sqlserver/default.aspx
- MongoDB. (Accedido en 2012). http://www.mongodb.org/
- MySQL. (Accedido en 2012). http://www.mysql.com/
- Nimbus. (2012). Nimbus Project. http://workspace.globus.org/clouds/nimbus.html/
- North K. (2011). SQL, NoSQL or SomeSQL?, Dr. Dobb's,
- OASIS. (2006). Reference Model for Service Oriented Architecture 1.0 Committee Specification.
- OASIS WSBPEL TC. (2007). Web services business process execution language version 2.0. Tech rep, OASIS. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/
- Oracle BPEL Manager. (2012). http://www.oracle.com/technetwork/middleware/bpel/overview/index.html
- Oracle Database. (2012). http://www.oracle.com/es/products/database/index.html
- OWL (2012). http://www.w3.org/TR/owl-features/
- Point, Tutorials. (2009). What are Web services. http://www.tutorialspoint.com/webservices/what_are_web_services.htm
- Pylons. (2012). http://www.pylonsproject.org/
- RacksPace, Dedicated Server. (2012)., Managed Hosting, Web Hosting by Rackspace Hosting, http://www.rackspace.com
- S3. Amazon Simple Storage Service. (2012). http://aws.amazon.com/s3/
- Salesforce CRM. (2012). www.salesforce.com/platform

- SAP Business ByDesign. (2012)., www.sap.com/sme/solutions/ businessmanagement/businessbydesign/index.epx
- SearchCloudComputing. (2012). http://searchcloudcomputing.techtarget.com/photostory/2240149038/Top-10-cloud-providers-of-2012/1/Introduction#contentCompress
- Terremark. (2012). http://www.terremark.es/default.aspx
- Thomas. (2010). http://thomas-tin.usal.es/
- UDDI. (2002). UDDI Technical Committee. Universal Description, Discovery and Integration. (UDDI). http://www.oasis -open.org/committees/uddi-spec/
- VMWare (2012). http://www.vmware.com/es/
- W3C. (May, 2000). Simple Object Access Protocol (SOAP) 1.1 W3C Note
- W3C. Christensen E., Curbera F., Meredith G. and Weerawarana S. (March, 2001). Web Services Description Language (WSDL) 1.1
- Web2Py. (2012). http://www.web2py.es/welcome/default/index
- WidowsServer. Microsoft Windows Server. (2012). http://www.microsoft.com/es-es/server-cloud/ws2012/default.aspx
- Windows Azure (Accedido 2012). www.microsoft.com/azure
- WSDL. (2009). http://www.w3.org/TR/wsdl