Searching and Invoking e-Services in Multichannel Information Systems

Doctoral Dissertation of:
Pierluigi Plebani

Advisor:
Prof. Barbara Pernici

Tutor:
Prof. Franca Garzotto

Supervisor of the Doctoral Program:
Prof. Stefano Crespi Reghizzi

2005 – XVII edition
to my family
Abstract

During the Nineties, several enterprises recognized, even in the early stage, how Internet would affect the way of communicating with the clients. With Internet, indeed, the possible audience grows and, consequently, building new commercial relationships becomes more likely. According to that, terms such as e-commerce, e-business, e-government, and e-procurement have been the result of the efforts to integrate, both vertically and horizontally, the existing information systems inside the same enterprise and among different enterprises. Such an integration requires a big effort both on organizational and on technological standpoint. In fact, solutions about integration should mediate between the necessity to exchange information among the involved actors, and the necessity to leave them a great autonomy.

Focusing on the technological aspects, Web Services represent one of the most important challenges in last years for realizing distributed systems and, among the others, is going to solve the problems related to the information systems integration introduced above. Nowadays the lavished efforts are visible in the several models, methodologies, and specifications proposed to describe, compose, and invoke a Web Service. Actually, what is available so far is only the starting point of a more complex scenario, where a new generation of information systems is involved. The growing standardization, acceptance, and diffusion of novel devices (e.g., PDA, Smartphones) and network infrastructures (e.g. Wi-Fi, UMTS, Bluetooth) require an overall shift of the paradigms for design and development of traditional information systems in order to enable the access of services from users exploiting several distinct devices. In other words, traditional information systems are evolving to multichannel information systems in which a service is provided on several distinct channels, being the latter an abstraction of the network infrastructure and of the device used to access a service. For this reason, in this thesis work we adopt the term e-Service to extend the Web Service paradigm where the communication does not rely only on http protocol but also on different protocols involving devices like smartphones, PDAs, or special interfaces for person with disabilities.

Starting from this situation, the number of services available in the
next future is growing, so that any user can find an e-Service able to satisfy his requirements. But, the higher is the number of the available services, more difficult is to identify the desired service if the user is not effectively helped. Tools as UDDI and ebXML Registry, for example, organize Web Services in a set of pre-defined categories the user exploits to browse the registries. Unfortunately, these approaches are considered insufficient for a real scenario since they do not provide a support for queries based on functionalities and qualities of services. What is considered useful, indeed, is a registry able to receive as input a set of requirements, returning the set of services able to satisfy such constraints. The main goal of the present thesis is to propose Urbe (Uddi Registry By Example), a Service Directory which deals with both the functional and non-functional aspects and allows the user to query the registry according a query by example approach, specifying which are the characteristics of the desired service. Therefore, in the present thesis, a service description model is presented where the current models are enriched also considering quality features in multichannel environments. This means a model in which the influence of the communication channel between the service provider and the service user is considered as well. Not necessarily, in fact, the quality measured at provider side corresponds to the quality really perceived by the final user. Moreover, the thesis, relying on this service description model, provides a set of matchmaking mechanisms able to evaluate the functional similarities among e-Services, and their compatibility about non-functional aspects. With respect to the functional similarity, the approach the thesis proposes is based on a semantic analysis of the terms used for describing the e-Services functions. Exploiting the functionalities provided by Urbe, this thesis introduces an architecture able to invoke e-Services in a flexible and adaptive way.

The thesis is structured as follows. Chapter 1 discusses the main characteristics of multichannel information systems and the role hold by e-Services. In Chapter 2 the state of the art about the existing technologies related to the Web Services is presented, as well as an overview of the approaches strictly related to the service discovery. Chapter 3 introduces the functional and quality models, according to which an e-Service is described by both the provider and user standpoint. Chapter 4 provides the details about how the service discovery is performed describing the matchmaking algorithm. Finally, Chapter 5 discusses in depth about Urbe implementation and how Urbe can be exploited in a framework for providing and executing e-Service in a flexible way.
Durante gli anni Novanta, diverse aziende riconobbero già nelle fasi iniziali l’enorme influenza che Internet avrebbe potuto esercitare nella comunicazione con i propri clienti. Attraverso Internet, infatti, il potenziale pubblico cresce e, di conseguenza, diventa più probabile instaurare nuove relazioni commerciali. Per questo motivo, termini come e-commerce, e-business, e-government ed e-procurement rappresentano il risultato degli sforzi fatti per integrare, sia verticalmente che orizzontalmente, sistemi informativi esistenti all’interno della medesima azienda e tra diverse aziende. Tale integrazione richiede un grosso sforzo sia dal punto di vista organizzativo che tecnologico. Infatti, soluzioni per l’integrazione dovrebbero mediare tra la necessità di scambio di informazioni tra gli attori coinvolti e la necessità di lasciare ampia autonomia agli stessi.

Focalizzando l’attenzione sugli aspetti tecnologici, i Web service rappresentano una delle più importanti sfide degli ultimi anni nella realizzazione di sistemi distribuiti e, tra gli altri, punta a risolvere i problemi relativi all’integrazione di sistemi informativi presentati in precedenza. Al momento, gli sforzi profusi sono visibili nei numerosi modelli, metodologie e specifiche proposte per descrivere, comporre ed eseguire un Web service. In realtà, quanto disponibile oggi, è solo il punto di inizio di uno scenario molto più complesso dove è coinvolta una nuova generazione di sistemi informativi. Le crescenti standardizzazioni relative sia a nuovi dispositivi (e.g., PDA, Smartphone), sia a nuove infrastrutture di rete (e.g., Wi-Fi, UMTS, Bluetooth) richiedono un generica revisione delle modalità di progettazione e sviluppo di sistemi tradizionali allo scopo di abilitare l’accesso ai servizi da parte di utenti con diversi dispositivi. In altre parole, i sistemi informativi tradizionali si stanno evolvendo in sistemi informativi multicanale in cui un servizio è fornito su diversi canali, visti come una astrazione delle infrastrutture di rete e dei dispositivi per l’accesso ai servizi. Per questo motivo, in questo lavoro di tesi si adotterà il termine e-Service per estendere il paradigma dei Web service considerando una comunicazione non legata al solo protocollo http ma anche a protocolli differenti dedicati a dispositivi come smartphone, PDA o interfacce speciali destinate a persone disabili.

Partendo da questa situazione, il numero dei servizi che nel prossimo
futuro saranno disponibili è in continuo aumento, di modo che ogni utente possa trovare un e-Service in grado di soddisfare i propri requisiti. Ma, più alto è il numero di servizi a disposizione, maggiore è la difficoltà di identificare i servizi desiderati se l’utente non è adeguatamente supportato. Strumenti come UDDI ed ebXML Registry, per esempio, organizzano i Web service secondo insiemi di categorie predefinite che l’utente può sfruttare per navigare all’interno del registry medesimo. Sfortunatamente, questi approcci sono poco efficaci visto che non forniscono un supporto per interrogazioni basate sulle funzionalità e le qualità dei servizi. Viene considerata utile, infatti, la possibilità per il registry di ricevere come input un insieme di requisiti e restituire l’insieme dei servizi che soddisfano tali requisiti. Per questa ragione, obiettivo principale della tesi è proporre Urbe (Uddi Registry By Example) un Service Directory che considera sia gli aspetti funzionali che non funzionali e permette di interrogare l’archivio dei servizi secondo un approccio di tipo query by example specificando le caratteristiche del servizio desiderato. Pertanto, nella presente tesi, è proposto un modello di descrizione del servizio che arricchisce gli attuali modelli, in quanto considera anche aspetti di qualità in ambiente multicanale. Ciò si traduce in un modello in cui è considerata l’influenza del canale di comunicazione tra fornitore e utente finale. Non necessariamente, infatti, la qualità misurata al lato fornitore risulta essere identica a quella realmente percepita dall’utente finale. La tesi, inoltre, basandosi su questo modello di descrizione del servizio, fornisce una serie di meccanismi di matchmaking in grado di verificare la similitudine funzionale tra e-Service e la loro compatibilità circa gli aspetti non funzionali. Rispetto alla similarità funzionale, l’approccio proposto nella tesi si basa su una analisi semantica dei termini utilizzati per la definizione delle funzioni messe a disposizione dagli e-Service. Sfruttando le funzionalità fornite da Urbe, la tesi propone inoltre una architettura in grado di eseguire e-Service in modo flessibile ed adattativo.

La tesi è strutturata nel seguente modo. Il Capitolo 1 presenta le caratteristiche principali di un sistema informativo multicanale e il ruolo degli e-Service. Nel Capitolo 2, il lavoro di tesi viene collocato all’interno dello stato dell’arte relativo allo strumenti dei Web service e ai meccanismi legati alla ricerca di servizi. Il Capitolo 3 presenta il modello funzionale e di qualità, secondo i quali un e-Service è descritto sia dal punto di vista dell’utilizzatore sia del fornitore. Il Capitolo 4 descrive in dettaglio i meccanismi di ricerca dei servizi e l’algoritmo di matchmaking utilizzato. Infine, il Capitolo 5 discute in modo approfondito come è strutturato Urbe e come possa essere sfruttato in un sistema che fornisce ed esegue e-Service in modo flessibile.
Acknowledgments

I would like to thank you all the people with whom I was in contact during these three years. First of all many thanks to my advisor prof. Barbara Pernici, for her guidance, support, patience and encouragement throughout my research work. I am also very grateful to Maria-grazia Fugini and Maristella Matera who convinced me to apply for the doctoral program more than three years ago. I would like to thank the “Fondazione Silvio Tronchetti Provera” which supports my work during the first two years and especially dr. eng. Lucio Pinto. A special thank to prof. Mathias Weske for his comments on this thesis and his suggestions for improving this work.

I would like to express my thanks to all my colleagues, who brightened my doctoral experience with their humour and friendship both at the University Department in Milan and around Italy; especially Alessandro, Andrea, Carlo, Cinzia, Daniele, Danilo, Devis, Enrico, Enzo, Fabio, Luciano, Massimo, Marco, Michele, Monica, Paola, Stefano, Stefania, and Tony. I wish to thank all the people I had the pleasure to work with during my exciting internship at IBM in NY, especially Axel, Bill, Francisco, Ignacio, Matt, Nirmal, Rania, Sanjiva, Stephan, and Thomas. Obviously, I don’t forget all my friends for their support taking my mind off work worries during the train commuting and the spare time; especially Alex, Andrea, Chiara, Consuelo, Dafne, Dario, Elena P.I., Elena P., Federico, Lorenzo B., Lorenzo M., Lucio, Marco, Maria, Manuel, Paolo, Raffaella, Roberto B., Roberto M., Sem, Simon, Valeria.

Finally, I would like to thank my family for always being by my side throughout my studies, for loving me unconditionally, for understanding me, and for constantly inspiring me to reach my goals.

Part of this thesis work has been supported by Italian MIUR-MURST under the VISPO (Virtual district Internet-based Service Platform) and MAIS (Multichannel Adaptive Information Systems) contracts.
Contents

1. Service Oriented Computing in a Multichannel Environment 1
   1.1. e-Service logical components 6
       1.1.1. Application logic 7
       1.1.2. Presentation logic 8
       1.1.3. Channel 9
   1.2. Adaptivity 10
       1.2.1. Channel adaptation 10
       1.2.2. e-Service adaptation 10
       1.2.3. Process adaptation 11

2. Related work 13
   2.1. Web Service description 14
       2.1.1. Web Service Description Language (WSDL) 15
       2.1.2. OWL-S 17
       2.1.3. Web Service Modeling Framework (WSMF) 18
   2.2. Web service quality 21
       2.2.1. QML (Quality of service Modeling Language) 22
       2.2.2. WSLA (Web Service Level Agreement) 23
       2.2.3. WSOL (Web Service Offering Language) 24
       2.2.4. Ws-Policy 25
   2.3. Web Service discovery 26
       2.3.1. UDDI (Universal Description Discovery Integration) 28
       2.3.2. ebXML Registry/Repository 31
   2.4. Projects 33
       2.4.1. VISPO Project 33
       2.4.2. MAIS Project 36

3. e-Service model 39
   3.1. System model 41
       3.1.1. Actors 41
       3.1.2. Communities 42
       3.1.3. Objects 42
   3.2. Channel 43
   3.3. e-Service functional model 46
       3.3.1. Context 47
Chapter 1.

Service Oriented Computing in a Multichannel Environment

Service Oriented Computing (SOC) is a new emergent paradigm where services are considered as building blocks to build a new kind of applications [83]. In this new environment, applications run in a distributed system where the different modules composing the application, i.e., the services, are under control of several actors. Considering the Web as the communication channel used by these actors for data exchange, the provided services are usually called Web Services. This approach is very close to the distributed computing approach developed during the Nineties, which has produced technologies such as DCOM, CORBA. Due to this closeness, a great debate about the real differences between these two worlds took place in the last years. [45] presents an interesting analysis where Web Services are compared to CORBA (Common Object Request Broker Architecture) [101]: an open standard for distributed object computing defined by the OMG (Object Management Group). Such a work presents the analogies, the differences, and how these two approaches can work together. Analogously, [102] tries to clarify several widely held beliefs when Web Services are considered equivalent to the Distributed Object paradigm, dealing with statements as: “Web Services are just like distributed objects”, “Web Services are RPC for the
Internet”, and “Web Services need HTTP”.

Even if it is not in contrast with the previous position, the reference model provided by W3C [14] better clarify the relationship between these two approaches. In more detail, a “distributed system is defined as diverse, discrete software components that must work together to perform some tasks. Furthermore, the components in a distributed system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks over a network. According to this definition, distributed object systems are distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries, marshall and unmarshall method argument data, etc”.

On the other side, Service Oriented Computing is defined introducing the Service Oriented Architecture (SOA), a conceptual architecture on which SOC is based. More precisely [14] SOA is a form of distributed systems architecture that is typically characterized by the following properties:

- **Logical view**: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.

- **Message orientation**: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be “wrapped” in message handling code that allows it to adhere to the formal service definition.

- **Description orientation**: A service is described by machine-processable meta data. The description supports the public nature of SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
• **Granularity**: Services tend to use a small number of operations with relatively large and complex messages.

• **Network orientation**: Services tend to be oriented towards use over a network, though this is not an absolute requirement.

• **Platform neutral**: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

Figure 1.1 shows the typical representation of the SOA where nearby the typical client-server interaction a new actor, called Service Directory or Service Broker, is introduced, and all the properties expressed above holds.

• **Service Provider**: who is in charge of building and making available a service on the basis of its mission and available resources. Through the publish operation a service is advertised, storing a document summarizing the service features in a public registry. Once the service is published, the Service Provider waits for contacts from the users interested in the service.

• **Service Directory or Service Broker**: this component is in charge of maintaining the Service Registry into which the description of the services are stored. The user of such registry can use it to find the services more suitable with respect to their needs. The Service
Chapter 1. Service Oriented Computing in a Multichannel Environment

Figure 1.2.: Web Service instance of the Service Oriented Architecture

Directory could also define a set of access policies to limit the user accessibility for security or privacy reasons.

- **Service Requestor**: represents a potential user for the published services. With the `find` operation, the user interacts with the Service Directory to obtain the most suitable service with respect to his own needs. Once the service is identified, the Service Requestor communicates with the related Service Provider (`bind`) and starts to interact with the service (`use`) on the basis of the service description provided by the service provider.

It is worth noting that, due to the platform neutral properties of the SOA, the three actors just described can be distributed and can rely on different platforms. In any case, even if the SOA should be network oriented, all the actors have to use the same communication channel during the interaction. Considering the nowadays available channels, several implementations of SOA can be obtained. Indeed, mobile devices using wireless connections, web-based channels, as well as e-mail systems can be used as possible communication channels. In particular, we define an e-Service as an instance of SOA where an electronic channel identifies the communication channel, whereas for the Web Service the communication channel is represented by the Web. Figure 1.2 shows this particular instantiation and the three main technologies, the role of
SOAP [46], WSDL (Web Service Description Language) [21], and UDDI (Universal Description Discovery Integration) [98] are identified.

SOAP is an XML based protocol able to define an interaction pattern among remote components on the Web. Even if one of the earlier purposes of SOAP is to support RPC (Remote Procedure Call) on Web, this protocol can support asynchronous, or message based, communications as well. Actually SOAP allows to invoke not only objects but several kind of remote elements and, for this reason, starting from the version 1.2, W3C does not associate the SOAP acronym to ‘Simple Object Access Protocol’ anymore. Even if SOAP represents the most common protocol used to invoke the Web services, other options such as, for example, SMTP and RMI exist.

Due to the great interest around the SOC paradigm, even SOA has been extended in order to include new issues as composition, conversation and management. Figure 1.3 takes up what discussed in [83], where the basic SOA expands in a third dimension able to consider the use of Web services in a more complex scenario. Here the classical SOA is represented as a part of the bottom layer, where the service is also described.
Chapter 1. Service Oriented Computing in a Multichannel Environment

in terms of the offered quality. The middle layer not only considers the problem of composing several service according to a process, but also takes into account the problem of coordinating and monitoring the composed services during the process execution. Finally, at the top layer service management, as well as contracting among the involved parties, are the main issues.

In the recent period, some efforts have been done to further extend the SOA with respect to the communication channel used by the involved actors. Multichannel Information Systems take into account such an extension where the same service can be invoked through different channels. As an example, we can consider the “Mobile TV” service, recently launched on the Italian marketplace, where the same TV shows can be also viewed on UMTS Smartphones. In this scenario, it is more useful to consider the e-Service rather than the Web service. The former, indeed is more general and it does not rely only on the Web channel unlike the Web service does.

The present thesis focuses on the role of the service discovery proposing Urbe: a service registry which extends the functionalities provided by UDDI in order to support query based on both functional and non functional constraints. The next section presents, at higher level, which are the main elements characterizing an e-Service which will be deeper discussed and modeled in Chapter 3. It is worth noting that such a model holds a central role about in the Urbe organization since it puts the basis of the mechanisms adopted to match the e-Service description against the user constraints as introduced in Chapter 4.

1.1. e-Service logical components

Extending the discussion to e-Services, the main subject of this thesis, and maintaining the analogies with the distributed object paradigm, some of the available specifications can be reused to express the characteristics of a software component invocable through several channels.

Taking the channel into account, the e-Service model can be considered as a composition of two main elements as shown in Figure 1.4: the application logic and the presentation logic [1]. The application logic refers to “what the service does”, whereas the presentation logic refers to “how can I interact with the service”. With respect to a typical software component the presentation logic is not a priori defined, but can be influenced by the user preferences about the channel, and the availability of the channel itself.

Depending on the client, the service provider can offer (i) only the
1.1. e-Service logical components

Figure 1.4.: e-Service components

application logic or (ii) both the application logic and the presentation logic. In the first case, client of the e-Service is a software able to respect and understand the exchanged messages. In fact, only a software can easily understand the meaning of exchanged messages if they are encoded accordingly to a specific protocol. In the second case, the presentation logic is provided assuming a human being as a client. In this way, the user can easily understand and interact with the e-Service according to the provided User Interface (UI).

In any case, it is worth noting that the service provider usually cannot manage the channel. In other words, the provider is not responsible about the network and the device functioning.

1.1.1. Application logic

The application logic of an e-Service is not different from the application logic of a Web Service, therefore, we can rely on the same model, i.e., WSDL. Besides the syntactic issues, according to [16] a WSDL document can be divided in two logical layers:

- *interface definition*: specifies the operations available in terms of incoming and outgoing messages. Such operations are logically grouped inside a *portType* which represents the abstract view of a service. This abstraction is justified by the absence of information about how such a service can be invoked. In other words, the user knows what the service does, but does not know how the service can be contacted.

- *implementation definition* (binding): specializes a *portType* in a *port* where for each provided operation an end-point is defined
through the binding process. Now, the user is able to connect to the service knowing where the service is and which protocol governs the interaction (e.g., HTTP, SOAP, SMTP, RMI).

The application logic of a service is strictly related to the interface definition, whereas the implementation definition is more related to the channel through which the service can be invoked. In this way, part of the application logic can be factorized and specified independently from the particular channel.

Besides the definition of the operation a service provides, an explicit definition about the relationship among such operations, i.e., the behavior, is required as well. Even if a service exports a set of operations, according to the status of the service, only a subset of them could be invoked. Moreover, the invocation process should follow a precise order.

Orthogonally to the functional and behavioral aspects, the specification of the non-functional aspects in terms of quality parameters is required. Here, the user and the provider are divided by the channel which is not necessarily handled by one of these actors. For this reason, the quality measured at provider side can be different from the quality perceived by the user, due to the influence exercised by the channel. As suggested by [100], it is useful to differentiate the quality offered by the provider, i.e., Quality of Service (QoS), from the quality at the user side called Quality of Experience (QoE). Even if QoE could suggest the idea that the user can learn something about the quality using the service, in this work, as also done in the literature [100], we consider the QoE as the quality perceived by the user in a context where the channel could affect the quality.

1.1.2. Presentation logic

In traditional systems the presentation logic is in charge of organizing the information provided by the application logic in a way usable for the final user. Typically, in Web systems, the presentation logic can be performed using scripts at both server side (e.g., ASP and JSP) and client side (e.g., JScript). In this way, most of the work is done at server side, whereas the browser is in charge of executing only simple scripts to improve the readability of the pages. In some other cases, the browser can rely on more sophisticated tools (e.g., applets) to improve the presentation.

Considering a multichannel environment, part of the presentation logic of an e-Service can be influenced by the channel. For example, given an image, a PC can easily show it on a single page, while it might happen that the same image could be too big for a PDA screen. For
1.1. e-Service logical components

This reason, the browser on PDA should initially present a thumbnail of the same image and, only if explicitly requested by the user, the original image should be completely viewed. In the same way, according to the bandwidth of the network, the e-Service can decide to show the same image at different resolution levels to improve the overall performance.

The presentation logic can operate during two moments: (i) before sending the information through the channel, and (ii) before displaying the information on the device. In the first case, the presentation logic can be defined knowing all the elements composing the channel. On the contrary, when the information is being displayed on the device, the presentation logic can be based only on the characteristics of the device itself.

Analysis of the presentation logic, as well as its relationship with the channel characteristics, are more related to the Human Computer Interaction community and are out of the scope of this work.

1.1.3. Channel

Due to the nature of the Multichannel Information Systems, the channel represents one of the most important element of the discussion. At higher level, a channel is composed by everything is needed to transport information back and forth between two communicating parties. In this way, between sender and receiver, some systems and human beings are properly organized in order to effectively carry the information. Giving an example, let us consider a typical service where a bank offers to its customers the possibility to manage their accounts. Nowadays, the customer can do the same operation through two channels: either using the usual branch or using, where available, the banking-online applications. The former is a traditional channel where the information stored in the bank information systems are provided to the final user by the bank employee. The latter is a new kind of channel, where Internet is exploited to directly connect the two communicating parties.

Concentrating on the purpose of this work, we consider only channels composed by automatic systems where the human beings are involved only during the systems maintenance. In other words, a typical system is composed by two systems connected together by a totally automated channel. The information exchanged are not necessarily in a form a human can understand. As said above, the presentation logic, i.e., element of the service able to transform the machine-readable information in a way really comprehensible by the user, is not considered.
1.2. Adaptivity

Adapting the e-Services execution means awareness of the environment in which the e-Service is executing, and reacting to the possible environment changes in case these modifications can influence the execution [84]. In particular such changes can derive from both internal and external reasons.

Internal reasons could be: (i) the failure of an e-Service or a part of it, or (ii) e-Service quality level lower than the quality level promised by the provider. In these cases, the system should be able to substitute the failed part with a compatible one, or to modify the provisioning in order to satisfy the user requests. About the quality levels, we suppose that the agreement between the parties is already done before starting the service invocation.

On the other hand, the user is responsible of the external changes. During e-Service invocation, indeed, the user can decide to modify the channel according to its preferences or the context in which the user operates. For this matter, the system should be able to inform the user about both the environment and the set of tools able to switch between the channels without affecting the state of the running e-Service.

This thesis work presents, in Chapter 5, an architecture able to support an adaptive and flexible execution of a multichannel information system. Such an architecture describes a possible use of Urbe exploiting the functionalities the registry provides. In the following a brief overview about adaptation mechanisms considered are described.

1.2.1. Channel adaptation

When the same e-Service can be invoked through several channels, the user can decide to switch among them to exploit the e-Service in different contexts. On the other hand, the provider can also decide to impose a channel switching when (i) the current one is no longer available, or (ii) the quality provided is lower than the quality promised. These situations have sense, in particular, when the e-Service communication occurs over a long time span, in which the user context can vary. In other cases the user, for example with disabilities or with contextual inabilities (e.g., preferring voice interaction to written text while driving), prefers using different devices during the same e-Service interaction.

1.2.2. e-Service adaptation

e-Service adaptation is performed through substitution mechanisms. During the e-Service execution, it might happen a service failure, or the e-
1.2. Adaptivity

Service cannot be reached because out of the current context, or a better equivalent e-Service becomes available. In these situations, the current e-Service can be substituted with another one. For example, let us suppose that the final user usually relies on a flight reservation service which covers only the European zone. In case the flight requested was directed to an American city, the system should able to identify a compatible service be able to handle this destination.

Hence, the goal is to substitute an e-Service without affecting the observable behavior of the cooperative process [109].

1.2.3. Process adaptation

A more complex situation can take place when even the service substitution is not possible since a compatible e-Service is not available [22, 70, 23]. In this case, the adaptation strategy directly involves the process.

The problem is to reconfigure the process in a new and equivalent one, where the activities performed by the failed service are modified with new activities requiring different kinds of service. For example, in case no flight reservation service is able to find a flight cheaper than a specified budget, and the final destination is reachable only by air, then the involved process sub-process can be redefined considering a train reservation to a intermediate city and then a flight to the final destination.

In a multi-channel information system, process adaptation can be affected by the variable network configurations [8]. Assuming that the process is executed in partitioned networks, mechanisms for transforming a process into a set of coordinated subprocesses with additional coordination activities has been proposed. The partitioning is based on the concept of coordinators for subprocesses and of patterns for partitioning the process for each of the flow constructs defined in BPEL4WS [95]. The result is a modified process, composed of a set of choreographed independent processes [30].
Chapter 2.
Related work

Several research communities, in the recent period, have concentrated their efforts around the Service Oriented Computing and the Web service technologies as their most common applications. First of all, the Middleware community has seen in this topic the possibility to extend the existing component-based architectures allowing the use of the Web as a new communication channel. The Information System community has considered Web services as a possible way to wrap the existing information systems in order to provide a standard interface invocable by the external partners in a non-pervasive way. Even the Agent community has considered Web services as the natural extension of the concept of agent in a web environment [60].

Due to this really big interest, a lot of definitions are nowadays available to describe a Web service. To avoid misunderstanding, in this work, without prejudice towards other definitions, a Web service is defined as stated in [89], i.e., "A Web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and support direct interactions with other software applications using XML based messages via Internet-based protocols".

Goal of this chapter is to introduce the main proposals that, starting
from the above definition, are able to describe what a Web service is and in which way it can be used. Considering the goal of this thesis, the approaches here described concentrate both on syntactic and on semantic standpoints. Such a description is completed also considering the proposals about a Web service quality model. Solutions about the service discovery, based on the presented service descriptions approaches, are discussed as well. Finally, a brief overview about two Italian projects, focused on the flexible and adaptive Web services invocation, concludes the chapter.

2.1. Web Service description

Describing a Web service means defining what is the purpose of the service and how such a service should be used. According to the loosely coupling philosophy, which inspires the Web service technologies, this information is required to effectively allow all the potential users to correctly interoperate with the Web services.

WSDL (Web Service Description Language) [21] represents the best known specification to describe a Web service in terms of location, characteristics of the operation provided, exchanged messages, and supported transmission protocols. Its structure is very close to IDL (Interface Definition Language) [81] used in CORBA [101, 1] to describe a component. WSDL is one of the main result of a consortium of enterprises including, among the others, BEA, IBM, Microsoft, SAP, aiming at defining standards and interoperable specifications.

Focusing only on the syntax, WSDL does not deal with the semantics of terms used to describe a Web service. In this way, misunderstanding about terms between the client and the service provider might occur. This situation becomes more probable if the two communicating parties are not human beings but software applications. The Semantic Web deals with this kind of problems, aiming at introducing a set of languages able to describe, in a machine-processable way [11], the semantics of generic Web resources. According to this idea, OWL-S [68] (Web service Ontology Language) (formerly DAML-S [2]), provides a Web service description model complementary to WSDL. OWL-S is not only focused on the syntax aspects, but also on the semantic aspects. In the same way, WSFM [41] (Web Service Modeling Framework) proposes a complete framework totally based on the Semantic Web approach to interact with Web services.

According to the main goal of the present thesis, i.e., to search for an e-Service, both the syntax and the semantic related to an e-Service
should be taken into account. About the syntactic perspective, WSDL is now the standard de-facto and all the Web services available nowadays provide such a description. As introduced in the next section, WSDL can also describe part of the elements characterizing an e-Service but they are not enough to cover all the aspects defining a provisioning channel. On the other hand, from a semantic perspective, even if several approaches are now available, it is not easy to find an e-Service really described according to such languages. For this reason, we rely on several ontologies, defined using the mentioned languages, in order to create a mechanisms to derive a semantic-oriented e-Service description directly from the WSDL definition.

2.1.1. Web Service Description Language (WSDL)

WSDL is an XML based language able to define what a Web service provides in terms of invocable operations. Given a WSDL specification the service provider states how the messages involved during the communication are composed, which are the functionalities provided in terms of incoming and outgoing messages, which protocols are supported by the service, and where the service is located inside the Web. The version 1.1. is nowadays the most used to describe a service and it represents the version adopted in this work. Currently, the W3C Web Service Description Working Group is going to release a new version, i.e, version 2.0 \[104\], with the same scope but with a different structure, which will be probably approved during the 2005.

Entering in detail, Figure 2.1 shows the structure of a WSDL document. Tag `<types>` is used to define new data types according to XML schema \[40\]. Those data types will be used in the rest of the specification to specify the nature of the messages. With a set of tags `<message>`, the messages involved during the communication in both the directions are defined. Each `<message>` is composed by several `<part>`s, each of them representing a data requested or provided by the Web service. In WSDL, the most important element is represented by `<portType>` tag, which groups a set of `<operation>`s. Each `<operation>` represents a functionality provided by the Web service and available to the user. An `<operation>` must refer to one of the following four predefined patterns, since both synchronous and asynchronous invocation are supported:

- **One_way**. The operation is composed by only one incoming message with respect to the service provider.

- **Request_response**. After a request performed by the client, the service responds.
Chapter 2. Related work

Figure 2.1.: WSDL document structure

• *Solicit Response*. Here the provider starts the communication and waits until the response from the client arrives.

• *Notification*. Only an outgoing message from the service composes the pattern.

The tags introduced so far deal with the description of what the Web service provides and which is the syntax the user should adhere to. The remaining tags enrich this description defining (i) where the service is located inside the Web, and (ii) according to which protocol the communication has to take place. In particular the tag `<binding>` is in charge of defining the protocol (e.g., SOAP, HTTP) supported by the `<portType>` previously defined. It is worth noting that the same `<portType>` can be associated to several `<binding>`s. Association between a `<portType>` and one of the binding results in a `<port>` that defines the end-point at which the service responds. Usually, the end-point is defined by an URL.
2.1. Web Service description

or according to the Ws-Addressing [25] specification. All the defined <port>s are included in the <service> tag, which describes the purpose of the Web service and specify URL at which all the documentation about the Web service can be found.

As Figure 2.1 highlights, the tags defined in WSDL can be grouped in two layers represented by the interface definition and the implementation definition. The first group of tags are closely related to “what the service does”, whereas the second one to “how the service can be reached”. The cardinality between a <portType> and <binding>, if we consider each <binding> as a channel, is the basis of a multichannel service definition.

Although WSDL provides a complete description of a Web service, it relies only on a static standpoint. What is missing, is an explicit definition of the dynamic vision of the Web service, i.e., the behavior. It specifies how the Web service works, which are the available operations according to the Web service status, and in which order the operations have to be invoked. For this purpose, several languages have been proposed. Among the others, WSCL (Web Service Conversation Language) is an XML based specification, which describes how a user can converse with a Web service. WSCL defines the service as a state-finite machine [7], where the operations represent the states and the transition between such states are defined. A more ambitious project is represented by BPEL4WS (Business Process Execution Language for Web Service) [95], which defines a service composition language that can also be used to specify the behavior of a single service.

2.1.2. OWL-S

From a functional and syntactical standpoint, WSDL provides only the basis to really describe Web services. In fact, this approach works only if the Web service and its client interpret in the same way the terms used during the service exploitation. Goal of OWL-S [68] is to improve the WSDL specification, providing an XML based language which enables the creation of ontologies. As usually defined, an ontology is a formal and consensual specification of conceptualizations that provide a shared and common understanding of a domain, an understanding that can be communicated across people and application systems. OWL-S is defined starting from OWL [61], a general purpose ontology language based on description logics [77]. Usually, systems relying on ontologies avoid the eventual misunderstanding due to erroneous interpretation of terms during the service invocation.

According to OWL-S, the ontology is composed of Web services described according not only to a syntactical perspective, but also to a
Chapter 2. Related work

semantic one. In this way, OWL-S aims at providing mechanisms to locate, select, employ, compose, and monitor Web services automatically. Actually, despite WSDL where a Web service is well defined by the functionalities provided, in OWL-S a Web service is defined as any Web accessible program, sensor, device. In more detail, three are the main purposes of OWL-S:

- **Automatic service discovery**: using OWL-S, the advertised Web services are semantically described in a computer-understandable way.

- **Automatic service invocation**: OWL-S markup of Web services provides a declarative, computer-interpretable API for executing the function calls.

- **Automatic service composition and interoperation**: OWL-S provides declarative specifications of the prerequisites and consequences of individual services.

The upper level of an ontology described by OWL-S can be sketched according to the elements shown in Figure 2.2. Here any service in the ontology can be defined by its ServiceProfile, ServiceModel, and ServiceGrounding.

The ServiceProfile defines “what the service does” where, in addition to the capabilities of the service, the need of the service itself are specified as well. The ServiceModel tells “how the service works” to better explain the requirements of the service with respect to the user invocation (i.e., in which order some functionalities should be invoked). This aspect can be also used to compose service descriptions from multiple services to perform a specific task. Finally, the ServiceGrounding specifies the technical details about how the client can reach and invoke the service. Due to its closeness, WSDL is usually included inside the ServiceGrounding.

2.1.3. Web Service Modeling Framework (WSMF)

Sticking around the Semantic Web community, it is worth introducing WSMF [41]. As the author states, WSMF is a “fully-fledged modeling framework that provides the appropriate conceptual model for developing and describing Web services and their composition”. Unlike WSDL, WSMF deals with a more complete Web service description which, among the others, considers:

- **Document types.** Describing the structure of the information the Web service exchanges with the client. Both the communicating parties should adhere to such a structure to correctly interoperate.
2.1. Web Service description

- **Semantics.** The creation of the exchanging documents, performed by the service and the client according to the structure defined, only ensures the syntactic correctness. The values and the concepts used in these documents must be correctly interpreted as well. In order to support the semantic correctness, the ontologies are introduced, in order to provide a means for defining the concepts and to allow the same interpretation of the concepts by all the parties involved.

- **Transport binding.** Several transport mechanisms are available like HTTP/S, S/MIME, FTP or EDIINT. Service requester and service provider have to agree on the same transport mechanism that will be used when the invocation takes place.

- **Exchange sequence definition.** If a service provides several functionalities, it might happen that such functionalities have to be invoked in a precise order. Considering both sides, the service exploitation underlies on a well defined sequence of messages.

The vision of WSMF about the relationship between Web services and Semantic Web is shown in Figure 2.3. Semantic Web enabled Web Service (SWWS) is considered as an improvement of the Web service approaches done so far. Introducing the semantic aspect, indeed, allows to mechanize what in [17] are considered the elements necessary to enable efficient inter-enterprise execution, i.e., public process description and advertisement, discovery of services, selection of services, composition of services, and delivery, monitoring and contract negotiation.
Chapter 2. Related work

![Diagram of Static Resources vs. Dynamic Resources and Syntactic vs. Semantic Interoperability]

Figure 2.3.: Semantic Web enabled Web Services (SWWS) vision

The philosophy of WSMF is based on the principle of maximal decoupling complemented by scalable mediation service. The de-coupling reflects the need to hide the internal structure of the enterprise, exporting to the user only the set of functionalities the user can invoke. The interoperability among the clients and the services takes place only considering this public interface. The mediation is required to solve differences among the terms used by the two communicating parties as well as the different interaction styles.

Briefly, a model based on WSMF consists of four main elements:

- **ontologies** that provide the terminology used by other elements;
- **goal repositories** that define the problems that should be solved by Web services;
- **Web services descriptions** that define various aspects of a Web service;
- **mediators** which bypass interoperability problems.

Project WSMO (Web Service Modeling Ontologies) [106] is derived from WSMF and provides a conceptual model for the description of Semantic Web Services. WSMO represents the basis for the Web Service Modeling Language (WSML) [105]: a formal language to describe Web services based on Description Logics. The same groups are also developing an execution environment called WSMX [107]. This is an implementation of WSMO to provide dynamic discovery, selection, mediation, invocation, and interoperability, of the Semantic Web Services.
2.2. Web service quality

Quality of Service (QoS) and related issues have been the topic of several researches and standardization efforts crossing distinct communities during the last years. For the purposes of this work, it is worth mentioning the Web and Web service community (e.g., [86, 64, 110]), the networking and internetworking communities (e.g., [50, 51, 24, 48]) as well as the middleware community (e.g., [111, 67]). Even if different in nature, the objectives of these efforts were at least intended (i) to identify the relevant measurable characteristics affecting the quality of the services provided by a given “object” (e.g., a Web service, a network infrastructure, a middleware platform) and (ii) to define means (e.g. architectures, paradigms, components, and protocols) to implement an “object” whose values of its measurable characteristics satisfy some quantitative constraints. We name the measurable characteristics of objects QoS parameters.

For instance, focusing on the Web service community, Mani and Magarajan in [64] identify the QoS parameters deemed useful for service providers to characterize the services. In some sense, Ran in [86] integrates this proposal devising means to extend service discovery on the basis of QoS-related information in addition to interfaces. Zeng et al. [110] proposes a methodology enabling the evaluation of the overall QoS of a composite service, i.e., a service obtained by composing several distinct services, provided that a description of the QoS parameters of the component services is available. WSOL [97] proposes an XML-based language to define the quality aspect and constraints for a Web service. Around this language, a set of tools able to monitor and evaluate what a WSOL document states have been developed [96]. Unfortunately, several of such parameters (e.g., availability, performance, and so on) are difficult to define and to evaluate in an end-to-end manner since the networks or the devices used to let a service and users interact are even partially out of the control of the provider. For this matter, in these proposals services are labelled by providers with a set of parameters in order to enable ordering and selecting them on the basis of QoS. This approach is limited when considering that usually the QoS perceived by users depends on factors out of the control of the provider, e.g., service availability depends on (i) provider availability, (ii) network availability, and (iii) user device availability.

To address this kind of issues, van Moorsel in [100] introduces the relevant distinction between Quality of Experience (QoE) and QoS in the

\[1\] Several other names have been proposed in the literature on the topic, e.g. *dimension, attributes* [42].
web environment, by pointing out that while QoS parameters are under full control of the service provider (e.g., throughput, server availability), QoE parameters, even if closely related to QoS parameter, may be influenced (i) by subjective elements related to user history and preferences, and (ii) by any system interposed between the provider and the user. Evidences of this important distinction are presented in [57], experimentally showing that the QoE of a user surfing the web using a browser through an Internet Service Provider is lightly affected by network latency and highly affected by network bandwidth. Similar work, e.g., Siller and Woods in [91], show how to relate QoS of service and network providers to QoE for specific services and applications. In other words, QoE can be expressed as the user perception of end-to-end QoS [52].

Putting the basis for evaluating QoE of services is a first class issue, as it would allow coping quite straightforwardly with other relevant issues such as (i) the definition of service level agreements [93, 56] based on the actual quality of service perceived by users, (ii) the clear and unambiguous assignment of responsibilities for enforcing each quality parameter value (and to possibly define the corresponding penalties in case of unsatisfactory levels) [74, 53], and (ii) the definition of integrated architectures among service, network, and device providers, cooperating to enforce QoE parameters [62, 54].

Comparing the current proposals with the goal of this thesis, in a multi-channel information system it is highly desirable to provide users with some guarantees about end-to-end QoS, in order to enable service selection and comparison not only on the basis of functional requirements, but also using information about non-functional aspects of the service. This requires service providers to carefully evaluate how the QoS they are able to offer in the basis of their internal resources is modified along the path connecting the user to the service itself. To this aim, the existing approaches are not enough and we deem it necessary to develop a quality model that enables reasoning on how channels (and associate providers) affect non-functional aspects of a service. Once such a model is available, services can be labeled with end-to-end QoS information, thus enabling QoS-enhanced service offering (by providers) and service lookup (by users) in a multichannel information system.

In the following, we present the subset of the approaches listed above which have inspired our quality model.

2.2.1. QML (Quality of service Modeling Language)

QML [42] tries to model the QoS as independent as possible of the specific domain in which the service works. For this reason such a specification
2.2. Web service quality

relies on the definition of QoS parameters organized according to the object-oriented paradigm concepts. Among the others, QML specification lists a set of elements that every QoS documents should consider in order to provide a complete and consistent specification about the quality. In particular:

- QoS specification should be syntactically separated from the other portions of service specification, such as interface definitions;
- it should be possible to specify both the QoS properties required by the user and the QoS properties about the service provisioning;
- there should be a way to determine how the QoS specification can match the user QoS requirements;
- it should be possible to redefine and to specialize an existing specification, analogously to the inheritance in the object-oriented programming.

According to these requirements, QML provides three main abstraction mechanisms for QoS specification: contract type, contract, and profile. While a contract type defines the dimensions that can be used to characterize a particular QoS aspect, a contract is an instance of a contract type and represents a particular QoS specification. In particular a contract type defines a collection of dimensions, each of them associated to a range of allowed values. A contract redefines these constrains according to given needs. A profile associates the contracts to the service interfaces operations, operation arguments, and operation results.

On the contrary QML, does not specify either how to enforce and to monitor the QoS or how the responsibilities are distributed among the involved actors.

2.2.2. WSLA (Web Service Level Agreement)

WSLA [56] is an XML-based, extensible language used to define a contract between a Web Service provider and respective Web Service user. Analogously to QML, WSLA defines QoS levels according to a set of different quality parameters but, unlike the QML, also considers the responsibility about the monitoring and enforcement of the quality. Briefly, a WSLA document is composed by three main sections:

- parties description: who is involved in the contract;
- service definition: what are the parameters describing the QoS, what are the metrics related to them, and for each parameter who is in charge to monitor the values;
Chapter 2. Related work

- obligation: the range of values the parameters have to respect and the action to be taken in case of violation.

Due to its native purpose, WSLA is strictly related to the Web Service provisioning, and it is not able to specify the QoS when the same service is provided through a channel different from the Web.

2.2.3. WSOL (Web Service Offering Language)

WSOL (Web Service Offerings Language) is an XML-based language, compatible with WSDL 1.1, for formal specification of constraints, management statements, multiple classes of service for Web Services and contracts between Web Services [97].

This language proposes a method to define and attach one or more quality document to a WSDL description. Each quality document, called service offering, represents the quality offered by the Web service and it is defined as a combination of various constraints and statements. WSOL supports the specification of constraints, such as functional constraints (pre-conditions, post-conditions, and future-conditions), non-functional constraints (a.k.a. QoS constraints), and access rights. WSOL supports specification of statements, such as price, price default, penalty, penalty default, subscription, external operation call, management responsibility. In more detail, WSOL is composed by five main constructs:

- **Service offering**: defines the quality of the Web service to which it is attached. In particular, it specifies the quality offered by the service. In case the offering is already negotiated with the service user, it also represents the service level agreement (SLA). A service offering is defined in terms of constraints and management statements.

- **Constraint**: is a boolean expression that states some conditions to be evaluated. This expression can refer both to the functional and quality aspects and its evaluation can take place before or after the service invocation. A constraint on the functional aspect could be pre- and post-condition statements, or a statement which must be verified some time after the service invocation. On the other hand, quality constraints describe properties such as performance, reliability, and availability.

- **Management statements**: mainly defines the responsibility about the constraints previously defined. In other words, who is responsible to measure the parameters included in the constraints, and
which are the penalties in case the quality measure is lower than the agreed quality.

- **Reusability constructs**: even if the quality definition is domain dependent, some of the parameters used to define the quality can be reused in several domains. For this reason a set of constructs able to define service offering templates is considered.

- **Dynamic relationships between service offerings**: allows to define situation in which a service offering can be replaced at run-time in case some constraints is not verified.

### 2.2.4. Ws-Policy

Ws-Policy [90] is a general purpose container of policy assertions, where an assertion refers to a quality aspect and is defined by external bodies. By means of two main operators, i.e., *exactlyOne* and *all*, a Ws-Policy document can combine in several ways assertions related to different quality domains such as security, transactionality, and reliable messaging. So far, Ws-SecurityAssertion [75] and Ws-RMPCI [9] represent the only public assertion specifications available, even if some other specifications about the transaction aspects as well as the system performance are going to be released [73]. A Ws-Policy document can be defined by both the service provider and requester to specify, namely, the quality offered and the quality required. Once the Ws-Policy document is written, it can be attached to a Web service according to the Ws-PolicyAttachment specification [47]. Since a policy definition can be useful not only to define the quality of a Web service, but also to characterize any element involved during the service life cycle (provider, user, platforms, network), Ws-PolicyAttachment specifies a generic mechanisms to attach policies to XML documents.

Figure 2.4 shows a Ws-Policy document example about the quality offered by a Web service running on a given system. The messages such a Web service will exchange, either have to be signed (*<integrity>* ) or encrypted (*<confidentiality>* ). Besides the security mechanism selected, the Web service is transactional, whereas the reliable messaging does not rely on the *atMostOnce* approach. Considering the several configurations allowed for one assertion, as well as the set of assertions which will be defined in the future, it is easy to figure out how a Ws-Policy document can be rich enough to describe the Web service quality.
Chapter 2. Related work

2.3. Web Service discovery

Extending the definition provided in [14] the Web service Discovery can be defined as the act of locating the machine-processable descriptions of the Web services that may have been previously unknown and that meet the criteria expressed by the service requestor. Depending on the approach used to describe Web services, we can classify the existing service discovery approaches in two main classes: structural and semantic [92]. The first case concentrates on the syntax aspects and the request has to be expressed using the same structure in which the Web service is described. In the second case, the goal is twofold. On one hand, analyzing the terms semantic we aim at overcoming possible misinterpretations of terms, on the other hand we want to discover services described with terms different to those used in the request, but semantically related to them. In any case, the service discovery relies on a set of techniques to assess the similarity among the available Web services. Actually, these techniques are used before the service discovery phase takes place, in order to classify the Web services registered in a way that facilitates the subsequent discovery. Usually, such a classification is done according to the functionalities.

None of the approaches listed below are used by UDDI, that represents well-known Web services registry together with ebXML Registry [78]. In fact, UDDI relies on a static classification performed by the publishers
2.3. Web Service discovery

themselv es. As described in detail in the following, UDDI organizes the Web services according to pre-defined and standard taxonomies, such as UNSPSC [37], NAICS [76]. Actually, the service can be indirectly classified also using the tModel, a document that can specify the guidelines about a service. Web services, built according to a given tModel, have to explicitly declare, during the publication time, such a compliance. In this way, the tModel is used to classify the published Web services. Such a flexibility UDDI provides, will be used in this thesis work to apply the approaches described in the following inside an existing and well known product. In fact, this thesis aims at dealing with a content-based query, the current registries does not provide. At this stage several approaches are now available in order to support part of the desired functionalities. We aims at applying some of the mechanisms adopted in order to realize a complete environment.

About the structural matching, the service discovery relies only on the syntax of the Web service description and compares the signature of the requested service with respect to the signatures of the existing Web services. This approach is closely related to the approaches studied in the reusable components retrieval literature [27]. In this field, there are two types of methods to address this problem: the signature matching [108] and the specification matching [109]. In particular, the signature matching [108] considers two level of similarity introducing the exact and relaxed signature matching. The specification matching [109] enriches the similarity analysis also considering the component behavior, i.e., the order in which the provided methods have to be invoked. Here the use of pre- and post-conditions models the behavior and allows a more effective comparison among Web services. Woogle [33] uses this approach and provides a service search engine \(^2\) based on the functionalities provided by the Web services. In [10], fuzzy logics are used to classify the components registered in a repository in order to identify which are the elements providing the same functionalities. This classification can be considered as an initial phase. It can be followed by a phase where the signature/specification matching is applied only to the components belonging to the same cluster.

On the other hand, semantic matching occurs when the analysis takes into account the meaning of the terms used to describe a service. In this area, thesaurus and ontologies are often used to avoid the misinterpretation of terms by the involved actors. Fensel et al. [58] suggest the use of semantic annotation of Web service descriptions to allow automatic service discovery, composition, invocation and interoperation. In partic-

\(^2\)http://data.cs.washington.edu:8080/won/wonServlet

27
Chapter 2. Related work

Singular, [55] proposes an extension of WSDL, that is called WSSP (Web Service Semantic Profile), to encode semantic information in WSDL by annotating I/O entities by means of domain ontologies written in RDF [59], OWL and by expressing constraints on inputs and outputs expressed using an RDF-Rule Markup Language [88]. Semantic annotation is used both when a new service is registered into the UDDI Registry and when a service request is formulated. A semantic matchmaker is then used to perform matchmaking among a service request and the registered service descriptions. The matchmaker is inspired by LARKS [94] algorithm and the OWL-S matchmaker (formerly DAML-S matchmaker) [82].

Several efforts have been also dedicated to the classification of e-Services based on the use of ontologies in order to enhance their discovery. Puyal et al. [85] takes into account the selection and execution of remote services on mobile devices and proposes a system, remote, where a service ontology stores information about the available services in a frame-based way simply using service categories and keywords to classify services in a taxonomy (only is-a relationship is taken into account). When a service discovery request is formulated, a new concept (service) is created and positioned in the taxonomy with the support of a Description Logic reasoner. The child nodes of this new concept, once it is positioned in the ontology, are the services presented to the user. Mostéfaoui and Hirsbrunner [72] proposes an architecture for gathering and processing contextual information exploited to enhance service selection after the application of the discovery mechanism based on functional description of services. In [110] a service ontology is proposed to obtain a common service language agreed by a community (e.g., air transfer). Here, a service ontology specifies a domain, a set of synonyms to allow a flexible search for the domain and a set of service classes to define the properties of services, further specified by its attributes and operations. A service class represents a set of services providing the same capabilities (operations). The service ontology also specifies a service quality model that is used to describe non functional aspects. Even in [69] service selection mechanisms are based on service ontology. In this case, an additional ontology about the quality of service aspects is introduced to enrich the matchmaking mechanisms.

2.3.1. UDDI (Universal Description Discovery Integration)

UDDI Registry [98] is the best known implementation of a service directory. The main task of a UDDI Registry is to organize Web services with respect to three different aspects: who provides the Web service, which functionalities the Web service performs and how the Web ser-
2.3. Web Service discovery

dvice can be invoked. In this way, the UDDI Registry, besides a classical keyword-based search, can be browsed according to three different modalities, namely White Pages, Yellow Pages and Green Pages. The first two modalities reflect the typical browsing based on the name of the provider and the type of business. On the contrary, Green Pages represent one of the enhancement of UDDI, since they allow the user to look for a Web service with respect to its functionalities.

Several implementations of the UDDI Registry are now available\(^3\), but extensions have been proposed due to the lack of a semantic-based classification and search of services.

Providing a global coverage, several UDDI registries (called nodes) can be distributed around the world according to a peer-to-peer architecture. In this way, even if each node can host only a subset of the existing Web services, it can communicate with the other peers to a complete and exhaustive search. In order to ensure the interoperability, each node, managed by the operator, must be realized according to the specifications [103] defined by the UDDI working group. Such specifications define how the information must organized, and the minimum set of API the registry must implement.

On these basis, each node can be represented according to what the Figure 2.5 shows, where two are the main components:

- **UDDI Business Registry** further subdivided as:

  - **Service Type Registry**: it contains the *tModels* (technical Model). A *tModel* represents a document where a concept is formalized. Whenever it is reasonable to define a common knowledge about any aspect involving the services, a *tModel* is written. For example, a *tModel* can be defined to describe which are the admissible currencies and, eventually, the relationships among them. A *tModel* can be also defined as a reference point in building services defining a service type. A business community, for example, can define how a shipping service should be built. All the service provider joining to this community will rely on this specification whenever they are going to implement such a service.

  - **Business Registry**: containing the information about the service provider and the service the provider makes available. Each entry is called *businessEntity*.

Chapter 2. Related work

Figure 2.5.: UDDI Architecture

- *API* to access, search, and maintain the stored information.

Figure 2.6 introduces a sample entry in the UDDI Registry v.2.0, where *businessEntity*, *contact*, *businessService*, *bindingTemplate*, and *tModelInstanceInfo* are stored in the *Business Registry*. On the other side, the *tModels* are stored in the *Service Type Registry*.

Starting from the *tModels*, in the considered example, two of them define two taxonomies used to classify the service, i.e., NAICS [76] and UNSPSC [37], whereas the third one refers to a service specifications defined inside RosettaNET\(^4\). The *businessEntity* stores all the information about the service provider, eventually relying on other elements such as *contact*. For each service, the provider makes available an instance of *businessService* where an higher level description of the service is presented. For a more technical details about the service, the *businessService* refers to a *bindingTemplate* and *tModelInstanceInfo* in which a link to a more formal specification, such as WSDL, is defined. The service, as well as the service provider, is classified inside a pre-defined categories using the *categoryBag* field.

\(^4\)http://www.rosettanet.org
2.3. Web Service discovery

Introducing the tModel, the Green Pages represent the added value of UDDI. Here the search phase is based on the information stored in the tModels where both information about the providers and the functionalities of the service are specified. Moreover, the information stored in the businessEntity can be exploited to allow search based on the geographic position of the provider rather than the sphere in which the provider operates.

Complementary to UDDI, WS-Inspection [6] represents a way for distributing the service provider, and service related information. UDDI, indeed, even if it can be distributed among several nodes, it is logically a centralized solution. With WS-Inspection, the authors aims at proposing a a means for aggregating references to descriptions about the same service, and at referencing this aggregation at the service provider side.

2.3.2. ebXML Registry/Repository

ebXML [35] is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. ebXML provides companies with a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes. In some sense what ebXML proposes is a set of specifications comparable
to all the specifications described so far. Among the others, a description language, as well as a registry, are proposed to define and store services available on the Web. Concentrating on ebXML Registry, the Figure 2.7 shows the overall organization highlighting the role of the registry with respect to the other components [38].

Trading Partner identifies a business involved in the business process which exports the functionalities according to what declared in the Business Service Interface. Such a document mediates between the private processes, characterizing the internal structure of the business, and the external businesses with which it is going to communicate.

When a Trading Partner is going to adhere to one of the available business processes, as stated in the available Business Process Specification [34], it has to define a CPP (Collaborative Partner Profile) [36]. This document specifies: the Trading Partner role, the features requested by the Trading Partner to the other communicating parties.
2.4. Projects

The present thesis work is mainly developed inside two Italian projects in which the e-Service substitution and adaptivity are the main issues. In particular, VISPO (Virtual district Internet-based Service Platform) is about the analysis of cooperative processes inside the Italian productive district where the enterprises belonging to the same district interact each other relying e-Service technology. In some sense, MAIS project extends the VISPO approach considering multichannel information systems.

2.4.1. VISPO Project

A virtual district is defined as a consortium of independent member enterprises which operate in an integrated and organic way to exploit business opportunities. From a technological standpoint, the information

...
systems of the such enterprises have to communicate following a process called cooperative process.

Within the Italian vispo project ⁵ we aiming at creating a flexible environment supporting the design of service-based cooperative processes and the dynamic composition of e-Services, by considering not only the design of complex e-Services starting from simple ones, but also the dynamic substitution of failed or modified e-Services. The structure of the cooperative process and the modalities of interaction are designed according to the nature of interactions in a virtual district, ranging from completely hierarchical structures to open markets.

Cooperative processes in vispo are defined on the basis of cooperating e-Services and they are based on the following models:

- *an e-Service Model*: the characteristics of e-Services are specified in terms of interface, behavior, and quality of service;
- *an Orchestration Model*: the interactions between e-Services and their coordination are specified;
- *a Business Transaction Model*: the orchestration is defined on the basis of pre-defined business transaction patterns, specifying the characteristics of interactions between organizations and provided e-Services.

The actors and tools involved during the cooperative process design are depicted in Figure 2.8.

Two main sets of e-Services are considered:

- *district specific e-Services*: oriented to the cooperative processes in the district and provided by the organizations belong to the virtual district;
- *general purpose e-Services*: realized to be used in several environments.

Given a cooperative process specification, an Application Programmer composes the available e-Services in order to realize the requirements.

Considering the available e-Services as black boxes exporting a set of methods invocable in a precise order, the cooperative process created by the Application Programmer can be defined as a set of cooperating e-Services. In such a way, two different behaviors constitute the cooperative process:

⁵http://vispo.casaccia.enea.it
Figure 2.8.: VISPO reference scenario

- *Non-observable* behavior: related to the behavior of the composing *e*-Services. Since an *e*-Service can be considered as a black box, we cannot modify its internal structure, but only use the provided methods, following its specifications.

- *Observable* behavior: related to the existing cooperation among the organizations which participate in the process.

Moreover, in our approach an *e*-Service uses Internet only for the information exchange. The activity defined inside the *e*-Service can be performed indifferently by either humans or applications. The work of the Application Programmer is supported by the following three knowledge bases, maintained and used by a *Domain Expert*:

- *Domain Text Ontology*: to discover relationships between terms used in the organizations belonging to the district as, for example, homonymy or hyponymy.

- *Domain Service Ontology*: to discover relationships between the district specific *e*-Services in terms of compatibility.

- *Domain Specific Knowledge Base*: to store typical patterns related to the district and used by both the *Domain Expert* to appropriately populate the above ontologies, and the *Application Programmer* to match the different service interfaces.
Chapter 2. Related work

2.4.2. MAIS Project

The goal of the MAIS project \(^6\) is the development of models, methods and tools that allow the implementation of multichannel adaptive information systems able to provide services with respect to different types of networks and access devices. Therefore, the design and implementation multichannel adaptive information systems presents various cross-disciplinary research problems, as they concern hardware and software technologies, as well as communication and network technologies, and also development methodologies [63]. Concentrating on the service oriented computing, in this project the final e-Services offered to the user will be handled in a flexible way by the platform, with negotiation of context parameters at application level. Attention is also be devoted to methodological aspects related to the development of multichannel adaptive information systems. The analysis of dynamically varying characteristics of the use of services, networks and devices is at the basis of context information and user profiles needed for pursuing adaptivity of services.

The MAIS architecture is designed to allow adaptivity at all levels in a distributed information system, in which services are provided in a multichannel environments, e.g. web-based, with call centers, and via short messages or e-mail, with the user interacting with the system with different types of devices in a ubiquitous way. The main elements of the adaptivity platform are:

- e-Service composition: adaptivity at application level is obtained through flexible e-Service composition, that allow modifying e-Services according to the context of invocation.

- Interaction enabling platform: the interaction enabling platform allows transforming events registered at technological level into application level events. For instance, the position registered by a GPS might be mapped into a town, or into a fine grained location, depending on the needs of the application. User profiling obtained through data mining of user accesses will support event identification and selection of services.

- Reflective architecture: The reflective architecture provides basic and advanced services. Basic services allow observation and control of technological properties of the system (e.g., modifying the audio volume). Advanced services provide also strategies for adaptation

\(^6\)http://www.mais-project.it
of the environment in ways which do not need control from the application level.

- **Network:** The network level provides network services at different levels of service, controllable via software (radio software), possibly negotiated through the reflective architecture.

- **Hardware:** Hardware adaptivity is being studied in terms of power consumption (adaptive low power consumption processors) and memory structures on portable devices (adaptive microdatabase technology).
As discussed in Chapter 1, Service Oriented Computing shares with Distributed Computing the underlying conceptual model, i.e. building applications based on components remotely available. Thus, the e-Service life cycle is quite similar to the software component one. As shown in Figure 3.1, the service developer is in charge of both realizing the e-Services and making them available to the user deploying it. It is worth noting that according to the EJB approach [31], we can assume that the service developer focuses only on the application logic and, by means of a service descriptor [87], specifies the channel in which such a service should be invoked. In this way, the middleware on which the service will be deployed, needs to verify if the requested channels are supported and, if so, it needs to export the application logic through them. Supposing several platforms able to understand such a descriptor exist, the same e-Service can be deployed in different locations. According to the supported channels and the available resources, the same e-Service deployed in different platform can be invoked through different channels and with different quality levels.

Operatively, considering a WSDL document, the service developer is in charge of defining the WSDL interface document, i.e., the set of functionalities. When the service is deployed, the WSDL interface document
Chapter 3. e-Service model

Figure 3.1: e-Service life cycle

is associated to a WSDL implementation document where the real service location is defined.

Once an e-Service is deployed on a given platform, the e-Service becomes available and the publication step takes place. This means that the service provider publishes into a service registry the e-Service specification (usually a WSDL document). It is worth noting that, according to the Service Oriented Architecture (SOA) introduced in Chapter 1, the service registry represents the connection between the service provider and the service user, since the latter exploits the functionalities provided by the registry to look for the desired e-Service and, once the desired e-Service is discovered, the e-Service invocation can take place. It might happen that who is in charge of discovering the service, i.e., service requester, is different to who really will use the service, i.e., service user. In this work, we assume that the service requester will also use the retrieved e-Service.
3.1. System model

So far, the service life cycle has taken into account only the actors usually involved in a software component life cycle. In this thesis work, we extend the typical scenario deeper analyzing the service realization phase. In this way, we aim at proposing a model, i.e., the system model, in which a general and shared vision about how an e-Service is defined in terms of functionalities, quality, and channel through which it can be invoked. Such a model includes actors, objects described in the following.

Roughly speaking, actors represent who directly perform some task during the implementation whereas objects represents the elements on which the actors operate. Communities, a particular type of actor, are introduced to produce a general framework in which several service providers can share information.

3.1.1. Actors

Actors are entities responsible of performing actions on objects or interested in the execution of some operations. In particular, extending the set of actors involved in the SOA and according to the model presented above, we consider service providers, network providers, device providers, and service users. With respect to the service life cycle shown in Figure 3.1, service providers represent both the service developer and the system the provider uses to realize and deploy the service. Network providers and device providers are introduced to capture who is in charge of controlling what happens between the service providers and the service users during the e-Service execution.

In more detail, service providers build e-Services and publish their description within a service registry. Network providers offer networking services and manage the underlying network infrastructure, e.g., cellular phone companies, Internet Service Providers (ISP). Device providers model industries producing devices as PDAs, PCs, SmartPhones, and so on. Service users, using the service directory, are able to select, to locate, and to invoke services by analyzing functional specifications.

Due to the nature of the e-Service, all the objects produced by these actors are strictly related. For example, service providers need to know in which way the network works, in order to evaluate if a particular channel can be exploited. In the same way, device providers needs to define which are the characteristics of the object they build, in order to inform the user about the available features. Indirectly, even the service providers need to know which are the limitations of the devices the user is going to adopt in order to present the information appropriately.
3.1.2. Communities

The strict relationships among the actors requires a sort of standardization efforts able to define shared information. For example, considering a device provider which is going to release a new cellphone, in order to produce a usable object, the embedded software has to respect all the requirements defined by the GSM specifications.

Extending this approach, in our model we also consider communities. A community is a group of actors which aims at proposing a specification for a group of objects with some relevant common characteristics. Even if a community could be considered an actor as well, we separate these two concepts for clarity.

An actor in the community implements objects compliant to the specifications defined by the community. In particular, service providers standardize the minimum functional specification, i.e. set of methods, a service must implement to be included into a particular service community. Considering, for example, the Video-on-Demand (VoD) service community, the minimum functional specification will include the play, stop, pause functionalities. Such functionalities do not reflect a precise set of invocable methods, but they represent an higher level description about what the service does.

Network communities define the main characteristics of networks by means of network specifications. Analogously, providers of same class of devices (e.g., PDA, Smartphone, PC) identify a minimum set of features (e.g., display size, color depth, audio capabilities, and network interfaces) that members of a device community must support. Figure 3.2 shows the relationship among kind of communities and kind of actors. Any community is defined for the service users.

Besides the functional aspects, specifications defined by the communities can also includes the quality aspects. In this way, given two objects performing the same functionalities according to the related specification, we can state which is the best one analyzing the quality level provided.

3.1.3. Objects

e-Service and channel are the first class objects of our system model and a complete description of such objects is provided in the rest of this chapter. As previously introduced, both these objects are handled by the respective actors and specified by communities.

Figure 3.3 shows the complete system model where all the components are included. Independently of the kind of community, both the functional and the quality specifications are defined. Once a provider,
3.2. Channel

The channel represents the link between the service provider and the service user during the service invocation and its definition involves both the network and the device provider. Given a service provider network member of the community, is going to build an object such specifications are adopted. Therefore, the object is an implementation of the functional specification. As discussed in the next section, the quality of the object is derived from the quality parameters which are included in the quality specification.

In the rest of the chapter, the objects involved in the system model are detailed, and the quality model adopted is introduced as well.

3.2. Channel
Chapter 3. e-Service model

and the network of its user, we can have several available channels, each of them characterized by different functionalities and quality. In particular the functionalities mainly derive from the application protocol on which the communication relies, whereas the quality mainly depends on the underlying network infrastructure.

Before introducing the channel, it is worth discussing the network. Every e-Service has a provider and a user by definition. These two actors should not know each other in advance, but they can communicate if each of them has a network infrastructure and has a device on which a software able to handle an application protocol, e.g., smtp, soap, http, is installed. Concentrating on the network infrastructure of one of the two sides (we do not care about the side, we refer to a generic 'actor'), the network is composed by a sequence of network segments. It is important noting that a network segment could not correspond to a physical network link, but it represents a set of network devices managed by the same network provider. The last segment in the sequence can have two meanings:

- the point beyond which it is not possible to know how is the network structure. Usually, Internet represents the last segment often characterized by the involved ISP (Internet Service Provider, here a network provider) and by quality provisioning estimations.

- the network device at which some devices are connected to. In this way we want to state that the actor can be reached only by a well-defined set of machines.

Considering the definitions introduced above, a channel is defined by two networks sharing the same application protocol. Thus, a channel is defined by both the underling infrastructures able to transport information and to support the protocols coordinating the end-to-end communications. In this way, we are able to track, or estimate, what happens starting from the system in which the service is running to the device used by the service client, and viceversa.

Giving an example, let us consider a service provider having a machine on which both a HTTP server and a SOAP server are available (Figure 3.4). Such a machine can be reached through two networks. The first one is composed by an Ethernet segment followed by an Internet connection made available by means of a generic ISP. The second one is composed by the only segment ending with the switch. On the other side, the service user has a machine on which only the SOAP client is available and it communicates with a wireless access point connected by means of a DSL modem to Internet. The user can rely on a ISP network provider which can be different from that used by the service provider. In this
example, we can realize that at least one of the end points of both sides are compatible, i.e. Internet, so they can be physically interconnected. Considering the application protocol, they can communicate only using SOAP.

Figure 3.5 defines the channel model using an UML class diagram. A Channel between two actors is defined by two Networks and the Application Protocol used during the communication. The Application Protocol is one of the protocols supported by the two network by means of the software installed in the Device. These devices can be PC, SmartPhones, or servers. DMTF [32] proposes the CIM model, a complete description of a device considering all the possible components starting from the type CPU clock to the number of USB ports. With respect to the goal of this work, here a device is composed by the following main elements:

- input device: the set of tools by means of which the operator can interact with the device (e.g., keyboard, mouse, braille keyboard). The input devices are important especially for the end-point device;

- CPU: defined in terms of model and clock.

- Memory: specifies the total amount and the available amount.

- Operating system.

- Applications: the set of installed programs.
Chapter 3. e-Service model

Figure 3.5.: Channel model

- Output device: how the user can obtain the information by the device; the display is the typical output device.

- Device Provider: the device manufacturer.

A network is composed by a sequence of Network Segments. Each Network Segment is managed by a single Network Provider who can estimate the quality of the network and its main functional characteristics.

3.3. e-Service functional model

According to the loosely coupling approach underlying Service Oriented Computing, e-Services description for service providers is different from the description for service users. While the user is interested in the public view of the e-Service, i.e., the functionalities available, the service provider also needs to know how the service works. Thus, our functional model is divided in two main components according to these two standpoints, where:
3.3. e-Service functional model

- The provisioning perspective specifies who provides the e-Service, what the e-Service does, and how to invoke its functionality.

- The request perspective specifies who requires the e-Service, its profile, and the context in which the requester operates.

Nearby these two descriptions, a model defining the context is introduced as well. The context, indeed, it will be used by both the service provider and the service used to characterize an e-Service.

3.3.1. Context

Considering portable devices like SmartPhones as well as laptops, means considering the possibility for the user to move around during the e-Service invocation. Depending on the user position, the e-Service available can be different. In the same way, given the same e-Service it can be invoked through different channels with respect to the user location. The context collects this information able to identify both where an e-Service is available and where the user is located. Being aware about the context can help the user to choose the best channel. Giving an example, if we consider an e-Service which describes the paintings in a museum, it will be able to automatically talk about the painting closest to the user. On the opposite, the context can be used to explicitly declare that this e-Service is available only in that particular museum.

A more formal definition of context is provided starting from previous work in the literature, such as UWA [99]. Here, the context is defined considering all the elements characterizing the user environment, such as the position and the positions of possible other users. The context can also be enriched considering the user history, which includes the set of activities already performed in order to better estimate the future actions.

Figure 3.6 shows the context model adopted in this work. According to that, we aim at capture the user mobility in order to identify, given a position, the set of available channels and the set of available services. Tracking the user history, the system can also collects information useful to personalize the service provisioning.

In particular, the context is defined by the user position, time information, and the devices the user can utilize. About the first two elements, the model considers the Location, the Time, and the Time Zone.

Location can be defined according to different granularities, depending on the precision requested. From the finer to the coarser the model considers, Geo Positions (i.e., latitudes and longitudes), Districts, (e.g.,
Chapter 3. e-Service model

![Context Model Diagram](image)

Figure 3.6.: Context model

special-interest areas), *Towns* and *Countries*. To add more expressiveness to the *Location*, a set of *Properties* can be associated to it. This is a general-purpose mechanism to add further information to the context description. For instance, we could use a *Property* to specify weather conditions. On the other hand, *Time* specifies the current time, whereas the *Time Zone* the offset from Greenwich mean time, and the daylight saving time.

The context is strictly related to the channels available to the user. Independently of the invoked e-Service, in a given location and time, the user can operate with several *Devices*. Each of them, according to the channel model presented above, is linked to a set of *Networks* and, thus, supports a set of *Application Protocols*. In this way, when the user invokes a particular e-Service, the set of available channel can be defined according to the network and the *Application Protocols* supported at provider side.

### 3.3.2. e-Service provider perspective

Figure 3.7 shows the service provider perspective, where two main blocks are considered: *functional decomposition* and *provisioning*. 

48
3.3. e-Service functional model

About the functional decomposition, following the WSDL specification, an e-Service is defined in terms of invocable Operations. Depending on the operation type, it can require some Inputs and returns a set of Outputs. An Operation, or a group of them, is associated to a Functionality. Not all the Operations can be invoked by the user, but only the Visible ones.

An Operation can be associated to a set of PreConditions and PostConditions defined according to a First Order Logic, that usually predicate on the related Inputs and Outputs. PreConditions must be verified before the execution of the operation, while PostConditions must be satisfied after the execution. If these conditions refers to other Operations then it is possible to define in which order the Operation should be invoked. Since we have two kinds of operations, i.e. visible and internal, according to [109], we can define two behaviors: observable and non-observable. The observable behavior defines the order in which the visible operation should be invoked by the user, whereas the non-observable behavior define the order in which all the operation should be invoked.

Whereas the functional decomposition identifies how an e-Service is built, the provisioning defines how the e-Service is related to the other actors. First of all, e-Services can be clients of other e-Services. In this way an e-Service can be simple or composite. In the latter case, in order to perform the functionalities offered, it relies on other e-Services according to a composition framework [4, 15, 95]. These e-Services can be provided by the same Service Provider or by external ones. In any case, an e-Service is installed on a Device and, according to the channel model presented above, it can be invoked through different Application Protocols. Finally, the Context defines where and when the e-Service is available, and which are the Application protocols available.

3.3.3. e-Service requestor perspective

From a service requestor perspective (see Figure 3.8), the functional model is mainly interested on the User definition, and on the way followed by the user to requests for an e-Service.

About the User definition, the Profile represents its preferences and its practise. In more detail, the profile has a static and a dynamic component: the first one to render those properties that are statically set by the user (usually during the registration phase), the second one to characterize all information that is collected during the e-Service invocation. Static profile is defined by means of a set of user preferences such as Role, which identifies the role played by the user while using the application, its Expertise and Ability on the application, and a set of Generic Pref-
Chapter 3. e-Service model

Figure 3.7.: Service provider perspective model.

erences to add application-specific characterizations to the profile. The hypothesis is that roles or expertise define the minimum profile, which can always be enriched with further information: each Generic Preference has a name and value to let the designer render and “quantify” any property.

Independently of the user profile, the request takes into account the current context in which the user operates and the used e-Services. In more details, a user can invoke one or more e-Services according to the visible operations and the observable behavior as they are defined in the provider perspective.

The user can be further characterized by the set of contexts in which the user can operate. In a particular location, at a given time, the user is in one of the available contexts. As defined above, each context also determines which are the available devices and the application protocols
supported. Matching this information with the application protocol of the service provider, a set of available channels is obtained. In particular, when the user identifies the e-Service to invoke, first of all we have to verify that the application protocol supported by the e-Service is also available on the user devices. As a second step, the network referring to the application protocol on both sides should have at least one common element.

### 3.4. e-Service quality model

A quality model is desirable to provide a description about how well the e-Service works. Typical approaches introduce a set of quality parameters [20] that the service provider uses to describe the offered quality, but the quality parameters definition is not enough to effectively provide to the user a way to state the real quality of service.
Chapter 3. e-Service model

In fact, the user can assign a different importance to these parameters according to its needs. For example, “price”, a typical quality parameter, can be considered very important by a user who is looking for a cheap e-Service, whereas it can be even ignored in case the user utterly needs from an e-Service. Moreover, the quality parameters are not independent each other, but relationships among them can exist. In fact, if we consider the “urgency” as a quality parameter we can say that more important is the urgency, less important becomes the price. A third aspect the quality model should deal with, is strictly related to multichannel information systems, where the channel can influence the quality parameters. In this way it might happen that the quality offered by the service provider differs from the quality really perceived by the user.

According to that, it is necessary to develop a quality model able to define an end-to-end quality of service and that enables reasoning on how channels (and associated providers) affect non-functional aspects of an e-Service. Once such a model is available, services can be labelled with Quality of Experience (QoE) [100] information, i.e. the quality really perceived by the user. In addition, the quality model puts the basis for quality certification, monitoring, and the definition of contracts among the parties [18]. In particular, by clearly stating how each metric is affected by service, network, and device providers, the proposed model supports the definition of metrics, monitoring means, and responsibilities of service level enforcement within a service level agreement framework [93]. Finally, the model supports the design of user-side and provider-side channel adaptation strategies based on QoS [65], i.e., user criteria to trigger channel switches upon detecting unsatisfactory levels of quality while accessing the service from a given channel, or provider suggestions about channel switches for maximizing user quality, respectively.

The quality model [66] proposed in this work is composed by a set of roles and rules that the elements of the system model respect and use to enable the eventual evaluation of end-to-end QoS.

The basic idea underlying the quality model is thus to extend specifications of objects with the quality information necessary to evaluate end-to-end QoS. Further, roles define who is responsible of defining quality parameters and quality sets for each object and each specification, while rules dictate how to combine quality sets associated to services, networks, and devices to evaluate end-to-end QoS. We support the presentation using a running example, i.e., the implementation of a VoD service in a multichannel information system.
3.4.1. Quality parameters, sets, and rules

A quality parameter models relevant and measurable non-functional aspects of an object. Quality sets are compounds of quality parameters that can be associated to objects. Quality rules define how to combine quality sets associated to distinct objects to evaluate their global effects on their non-functional aspects. In the following, we introduce the notation deemed necessary to deal with these concepts.

**Quality parameters**

A quality parameter $qp$ is a pair

$$qp = (\text{name, admissible\_values})$$

where name identifies the parameter and admissible\_values is an ordered set of typed values within which the parameter is admitted to vary. Examples of quality parameters are:

- $(\text{bandwidth}, [1Kps..512Kps])$
- $(\text{encryption}, [40bit; 64bit; 128bit])$
- $(\text{resolution}, [320 \times 200; 800 \times 600; 1024 \times 768; 1240 \times 748])$,
- $(\text{latency}, [10ms...500ms])$.

Given a quality parameter $qp$, $qp.name$ returns the name field, and $qp.values$ returns the admissible values. The functions best(name) and worst(name) are introduced to evaluate the admissible value corresponding to the best and the worst quality value of a quality parameter $qp : qp.name = name$, respectively. The values of these functions clearly depends on the semantics of the parameter, e.g., best(bandwidth) = 512Kbps, best(latency) = 10ms.

**Quality sets**

Quality sets are classes of quality parameters. Given a quality set $QS$, $QS.names$ returns the set of names of the quality parameters belonging to $q$. We say that a quality set $QS_2$ is contained within a quality set $QS_1$ if and only if (i) $QS_1$ and $QS_2$ have the same set of names, and (ii) each quality parameter in $Q_1$ has admissible values ranging within the admissible values of the corresponding parameter in $Q_2$. More precisely:
Chapter 3. e-Service model

Figure 3.9.: Example of the effect of containment between quality sets

\[ QS_1 \subseteq QS_2 \quad \text{iff} \]
\[ (QS_1.\text{names} = QS_2.\text{names}) \land \]
\[ (\forall qp_1 \in QS_1, qp_2 \in QS_2, qp_1.\text{name} = qp_2.\text{name} \Rightarrow \]
\[ qp_1.\text{values} \subseteq qp_2.\text{values}) \]

Quality sets can be associated to both objects and specifications. Therefore, provided that object \( O \) is the implementation of a specification \( S \), we distinguish between specification quality sets \((S.QS)\) and object quality sets \((O.QS)\). Using \( S.QS \) a community defines relevant quality parameters and their admissible values. These values represent all the possible values that any object implementing the specification can offer. As a consequence we require that if \( O \) implements (or is compliant to) \( S \), then \( O.QS \) is contained within \( S.QS \). Figure 3.9 illustrates the effect of containment between quality sets on the framerate quality parameter of a VoD service (expressed in frame per seconds - fps).

Quality rules

Actors and communities first associate quality parameters and sets to objects and specifications, and then they relate quality sets using quality rules. A quality rule can be defined as follows: given two quality sets \( QS_1, QS_2 \), a quality rule for \( QS_1 \) on \( QS_2 \) is a function \( qr_{12}(QS_1, QS_2) = QS_3 \) such that: (i) \( QS_3 \) is a quality set, and (ii) \( QS_3 \) is contained in \( QS_1 \). This implies that independently from the set of quality parameters of \( QS_2 \), \( QS_3 \) has parameters whose names are the same of those contained in \( QS_1 \) and whose values are contained in the corresponding parameters of \( QS_1 \). Indeed, these rules are used to model the effects on the quality
3.4. e-Service quality model

parameters of an object/specification $A$ having a quality set $QS_1$ when it is coupled with another object/specification $B$ having a quality set $QS_2$, i.e., to evaluate how the quality of $A$ is affected by $B$. Further, we require that quality rules satisfy the following containment property:

$$\text{if } (qr_{12}(QS_1, QS_2) = QS_3 \text{ is a quality rule}) \text{ then}$$
$$\text{for any } QS_1', QS_2' : QS_1' \subseteq QS_1, QS_2' \subseteq QS_2,$$
$$qr_{12}(QS_1', QS_2') = QS_3' \text{ is a quality rule, and } QS_3' \subseteq QS_3.$$

It is easy to see that the quality rule containment property implies that if a quality rule is defined among specification quality sets, it can be used to evaluate effects on quality of the corresponding object implementations.

Remarks.

The main objective of the quality model is to analyze how the relationships among the elements of a multichannel information system affect the quality perceived by users. To this aim, the containment relationship forces actors to implement objects with quality attributes compliant to the specification, while quality rules (which have a similar containment relationship) dictate how to evaluate quality modifications of a service upon varying the channel used to access the service itself. Further, being the focus on relationships, we do not enter the details of a language for specifying quality parameters and sets, and we limit to mention a good candidate, i.e., the Quality Modelling Language (QML, [42]).

3.4.2. Roles for quality management

Exploiting quality parameters, sets, and rules communities and actors operate in two different stages, namely the specification definition phase and the object implementation phase, in order to specify quality aspects of objects and specifications and to analyze their relationships [19, 43, 44]. In particular, first communities define the specification quality sets for services, networks, and devices. Once these information are available, during the object implementation phase, each actor can produce objects compliant to specifications according to the containment relationships, can define quality rules enabling to evaluate the end-to-end quality of a service for each channel, and in particular service providers can evaluate information on the end-to-end quality of service as a function of the channel on which the service is provided.
Chapter 3. e-Service model

Communities roles

During the specification definition phase, communities are in charge of defining quality specifications (i.e., quality sets). In particular, communities identify the name and the admissible values of all the quality attributes identifying all the relevant measurable non-functional aspects of any object that will comply to the specification. In particular, (i) each service community associates to each service the service specification quality set \( S.QS_{QoS} \), (ii) each network community associates to each network the network specification quality set \( S.QS_{QoN} \), and (iii) each device community associates to each device the device specification quality set \( S.QS_{QoD} \). Examples of these specifications can be found in [51, 24] for networks, and in [32] for devices. Concerning services, we remark that current proposals, e.g., [86, 110, 74], commonly model non-functional aspects of services (e.g. performance, availability, and so on), assuming either that they can be evaluated almost independently from networks and devices or including the end-to-end effects of a single channel within the service quality set specification. In both cases, they result not suitable for multichannel information systems. It is also important to note that quality parameters in the service specification quality set shall not consider channels and end-to-end aspects. For example, the service response time should be the time elapsed to produce a reply since the arrival of a request at the provider site; analogously, service availability should be evaluated at the provider site without considering network and device effects. This feature highly simplifies the definition of these service specifications. For each quality parameter identified, the communities set the limits of \textit{admissible value} to values which include all the possible values independently from the implementation of a particular actor. When precise values cannot be identified, both or one of the two \textit{admissible value} limits can be set to \( \pm \infty \). For example, the VoD community, according to the video encoding softwares now available, can set the \textit{admissible value} of the framerate quality parameter in the VoD specification to \( [5\text{fps}..40\text{fps}] \). On the contrary, considering the latency, the network community define the \textit{admissible value} as \( [0\text{sec}..+\infty] \) since some implementation could be follow a best effort approach about this parameter. Finally, a service community can relate the specification quality set \( S.QS_{QoS} \) of a service with specification \( S \) to a set of channels. To attain this, for each channel \( c_i = (N,D) \), the community defines a quality rule

\[
qr_{s,c_i}(S.QS_{QoS}, qr_{n,d}(S.QS_{QoN}, S.QD_{QoD}))
\]

Due to the quality rule containment property and to the quality set
3.4. e-Service quality model

containment relationship, such a quality rule enables every service provider, aiming to offer a service with specification $S$ on channel $c_i$, to evaluate the QoE. This evaluation is obtained by simply substituting the object quality sets $O.QS_{QoS}$, $O.QS_{QoN}$, and $O.QS_{QoD}$ within the rule, provided that object quality sets are contained within those of the corresponding specification. Thus, the Quality of Experience for a service $s$, invoked through a channel $c_i$ will be:

$$QoE_{s,c_i} = qr_{s,c_i}(O.QS_{QoS}, qr_{n,d}(O.QS_{QoN}, O.QD_{QoD}))$$

where:

- $O.QS_{QoS} \subseteq S.QS_{QoS}$
- $O.QS_{QoN} \subseteq S.QS_{QoN}$
- $O.QS_{QoD} \subseteq S.QS_{QoD}$

In order to define quality rules, communities exploit engineering knowledge, e.g., physical dependencies among quality parameters. As example, the bandwidth parameter of networks significantly affects the framerate, color depth and resolution quality parameters of a VoD service which are also affected by the colors quality parameters of a device.

**Actors roles**

During the object implementation phase, actors implement objects compliant to specifications and associate them object quality sets that satisfy containment relationships with the corresponding specification. Adhering to specifications is important to let service providers evaluate end-to-end QoS using (i) their object quality sets, and (ii) the set of quality rules dictated by the service community (if any) as described above. Let us note that “adhering to specifications” in our vision is equivalent to say that object quality sets are included in specification quality sets. To attain this, it is enough to let values of quality parameters of specification quality sets vary in interval sufficiently large for any possible implementation. Note also that if there is no quality rule for given a channel on which the provider aims to offer a service, the actor is free to define a quality rule for the channel and to propose it to the service community for approval. Therefore we can assume that exploiting the proposed model, service provider are eventually able to label service with the quality that will be perceived by users accessing the services from all the channels over which the service is provided.
Chapter 3. e-Service model

Remarks

A relevant issue about the proposed quality model concerns the quality of its instances, this meaning answering the following question: “How good is this instance of the quality model in capturing quality aspects of this particular multichannel information system?” Answering this question relates to the attention paid by communities and actors in defining quality sets and quality rules. In this sense, the proposed model is a first step whose main contribution is in enabling the creation of good instances by a clear definition of the underlying system model, tools and responsibilities. The process to build instances of the proposed model is applied to a case-study in the following section.

Figure 3.10.: System model instance.

3.5. The VoD Example

The focus of this section is to sketch the use of the quality model presented in this paper mainly in order to clarify the phases and tools introduced. Figure 3.10 shows all the main objects and actors used in this example. As aforementioned, the Video-on-Demand is a service that allows a user to watch video-streams on several heterogeneous devices, e.g. traditional PCs and SmartPhones. Let us remark that this service has been recently introduced in the European mobile phones marketplace and currently provides users with short streams containing relevant pieces of football matches. Therefore, in this simple example, the communities involved are: the VoD service community (SC), the network community NC, and two device communities (PCC for the PCs and
3.5. The VoD Example

SMC for the SmartPhones).

First of all, recall that the model assumes that communities define the functional specifications and identify the set of quality parameters deemed relevant with respect to the services they provide. Therefore we assume communities to define the following quality sets.

SC defines the quality set $V_{oD}^{spec}.Q_{S_{QoS}}$ as follows:

$$V_{oD}^{spec}.Q_{S_{QoS}} = \{ \langle \text{framerate}, [5 \text{fps}..40 \text{fps}] \rangle, \langle \text{colordepth}, [2 \text{bit}..24 \text{bit}] \rangle, \langle \text{resolution}, [320 \times 200; 800 \times 600; 1024 \times 768] \rangle \}$$

NC defines the quality set $MyNet^{spec}.Q_{S_{QoN}}$ as follows:

$$MyNet^{spec}.Q_{S_{QoN}} = \{ \langle \text{bandwidth}, [10\text{Kbps}..512\text{Kbps}] \rangle, \langle \text{loss}, [0..0.01] \rangle, \langle \text{latency}, [5\text{ms}..500\text{ms}] \rangle \}$$

PCC defines the quality set $Pc^{spec}.Q_{S_{QoD}}$ as follows:

$$Pc^{spec}.Q_{S_{QoD}} = \{ \langle \text{colors}, [2..24\text{bit}] \rangle, \langle \text{nw\_int}, [802.11b; 802.3] \rangle \}$$

SMC defines the quality set $SmartPhone^{spec}.Q_{S_{QoD}}$ as follows:

$$SmartPhone^{spec}.Q_{S_{QoD}} = \{ \langle \text{weight}, [150\text{gr}..300\text{gr}] \rangle, \langle \text{colors}, [2..16\text{bit}] \rangle, \langle \text{nw\_int}, [\text{GPRS}; \text{UMTS}; \text{GSM}] \rangle \}$$

Communities are also responsible of defining the quality rules relating quality parameters. These rules originates from the analysis of the physical dependencies among the service, network, and device parameters. Considering the VoD example, and considering very simplified versions of the actual mathematical relationships, it is possible to state that:

$$\text{framerate} \times \text{colordepth} \times \text{resolution} = K \times \text{bandwidth}$$

According to this relationship we can also identify the following relationships which affects the admissible range of the quality parameters.

$$\text{best}(\text{framerate}) = \frac{\text{best}(\text{bandwidth})}{\text{worst}(\text{colordepth}) \times \text{worst}(\text{resolution})}$$

$$\text{worst}(\text{framerate}) = \frac{\text{worst}(\text{bandwidth})}{\text{best}(\text{colordepth}) \times \text{best}(\text{resolution})}$$
Chapter 3. e-Service model

\[ \text{best(colordepth)} = \min(\text{best(colors)}, \text{best(colordepth)}) \]

\[ \text{worst(qoe.colordepth)} = \min(\text{worst(colors)}, \text{worst(colordepth)}) \]

With similar reasonings, it is possible to derive how each service quality parameter is affected by (i) channel quality parameters and (ii) device quality parameters, i.e. to define quality rules. Therefore we abstract out from the detailed sequence of steps taking to the precise derivation of quality rules, which are the main objectives of some ongoing works. We limit to mention that observing mathematical relationships as those described above, it is possible to precisely characterize these rules. In particular, it is possible to evaluate two end-to-end quality rules, namely \( qr_1() \) and \( qr_2() \), that take a service quality set as input parameter and return the service quality perceived by users accessing the service through network \( MyNet \) and using either a PC or a smartphone, respectively.

Let us now suppose that four different actors (namely, a service provider \( S_1 \), a network provider \( N_1 \), a PC provider \( D_1 \), and a smartphone provider \( D_2 \)) implement all the specifications described above. According to actor roles, besides the implementation of the functional specification, actors are in charge of defining an object quality set that satisfies the containment property with respect to related specification. Therefore we assume these providers to define the following object quality sets.

\( S_1 \) defines the \( S_1.QS_{QoS} \) quality set as follows:

\[ S_1.QS_{QoS} = \{ \langle \text{framerate}, [5\text{fps}..30\text{fps}] \rangle, \langle \text{colordepth}, [2\text{bit}..24\text{bit}] \rangle, \langle \text{resolution}, [320 \times 200; 800 \times 600] \rangle \} \]

\( N_1 \) defines the \( N_1.QS_{QoN} \) quality set as follows:

\[ N_1.QS_{QoN} = \{ \langle \text{bandwidth}, [10\text{Kbps}..128\text{Kbps}] \rangle, \langle \text{loss}, [0..0.01] \rangle, \langle \text{latency}, [10\text{ms}..500\text{ms}] \rangle \} \]

\( D_1 \) defines the \( D_1.QS_{QoD} \) quality set as follows:

\[ D_1.QS_{QoD} = \{ \langle \text{colors}, [2..24\text{bit}] \rangle, \langle \text{nw_int}, [802.3] \rangle \} \]

\( D_2 \) defines the \( D_2.QS_{QoD} \) quality set as follows:
3.6. Languages

$D_2.\text{SmartPhone}_{\text{spec}}.QS_{QoD} = \{\langle \text{weight}, [170\text{gr..180gr}] \rangle, 
\langle \text{colors}, [2..8\text{bit}] \rangle, 
\langle \text{nw}_{\text{int}}, [\text{UMTS}] \rangle \}$

These specifications, together with the quality rules $qr_1$ and $qr_2$, allow the user to derive the end-to-end quality of service for each available channel. Note that, in this simple example, the admissible channels are two (obtained combining network $MyNet$ with the devices $D_1$ and $D_2$). For each of these channels the user can evaluate how the service quality set $S_1.\text{QS}_{QoS}$ is perceived through these channels applying the quality functions. Such an evaluation generates, for each channel, a new quality set $U_i.\text{QS}_{QoS} \subseteq S_1.\text{QS}_{QoS}$ ($i$ identifies the channel). Note that $U_i.\text{QS}_{QoS}$ is composed by the same quality parameters belonging to $S_1.\text{QS}_{QoS}$ with an admissible range restricted if necessary. Let us also finally remark that the same evaluation can be performed by service providers in order to provide users with a precise characterization of the actual end-to-end quality of service they are going to perceive upon accessing the service. This feature can be used perform service composition basing on dynamic, end-to-end quality of service specifications that depend on the overall path followed by service information.

3.6. Languages

In this section we present how some of the languages described in Chapter 2 can be used inside the e-Service model presented above. In particular, WSDL is used to define an e-Service, whereas WSOL defines the quality of the object, i.e., e-Service and channel, involved. Ad-hoc XML languages are introduced to describe the networks and the context.

3.6.1. Communities specifications

Following the structure of the system model, we want to describe a shipping service invocable using both a PDA connected to a Wireless Lan and a traditional PC connected to Internet. Before introducing the service definition, we assume that five communities exist:

- a service community which defines the functionalities a shipping service usually performs;
- a network community which defines the main characteristics of Internet;
- a network community which defines the main characteristics of a Wireless Lan;

61
Chapter 3. e-Service model

- a device community which defines the characteristics of PDA;
- a device community which defines the characteristics of PC;

Each of these communities, according to our system model, provides the functional and quality specifications of the related object. In particular, the service community produces a WSDL Interface document which lists the available operations.

```xml
<wSDL:definitions xmlns:wSDL="..."
  targetNamespace="...">
  ...
  <wSDL:message name="deliveryRequest">
    <wSDL:part name="Description" type="xsd:string"/>
    <wSDL:part name="PickUpLocation" type="my:address"/>
    <wSDL:part name="Destination" type="my:address"/>
  </wSDL:message>
  ...
  <wSDL:message name="trackingRequest">
    <wSDL:part name="TrackingNo" type="xsd:string"/>
  </wSDL:message>
  ...
  <wSDL:message name="trackingResponse">
    <wSDL:part name="Location" type="my:address"/>
  </wSDL:message>
  ...
  <wSDL:portType name="MyCourier">
    <wSDL:operation name="Delivery">
      <wSDL:input message="deliveryRequest"/>
      <wSDL:output message="deliveryResponse"/>
    </wSDL:operation>
    <wSDL:operation name="Tracking">
      <wSDL:input message="trackingRequest"/>
      <wSDL:output message="trackingResponse"/>
    </wSDL:operation>
  </wSDL:portType>
</wSDL:definitions>
```

Service community also defines the relevant QoS parameters using a WSOL document. For each parameter the admissible range is defined as well.

```xml
<wsol:WSOLdefinitions xmlns:wsol="..."
  xmlns:expressionSchema="..."
  xmlns:shipping="...">
  <wsol:serviceOffering name="ShippingQos" service="shipping:shippingservice">
    <wsol:statement name="price" xsi:type="shipping:price"
      service="shipping:shippingservice"/>
  </wsol:serviceOffering>
</wsol:WSOLdefinitions>
```
3.6. Languages

...
Chapter 3. e-Service model

The device community, as occurred for the network communities, defines the most relevant quality parameters describing a PDA and a PC using the same syntax used by the network communities.

All the specifications introduced above are published in a UDDI registry. In this way all the providers interested in realizing an object can query the registry to obtain the interested specifications. More specifically, these specifications are available as tModel in the UDDI Registry. The following example refers to the tModel related to the shipping service specification.

<tnModel tModelKey="UUID:A4643DFD-8487-4734-6632-EBBA7467AAC4">
  <name>ShippingServiceModel</name>
  <description xml:lang="en">
    This specification describes how a shipping service should be built
  </description>
  <overviewDoc>
    <description xml:lang="en">
      WSDL Service Interface Document
    </description>
    <overviewURL>http://www.servicecommunity.com/shipping.wsdl</overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="UUID:C1AC7G6D-9672-4404-9B70-39976E622B4" keyName="uddi-org:types" keyValue="wsdlSpec" />
    <keyedReference tModelKey="uuid:cd153257-086a-4237-b336-6bdcb7cc6634" keyName="Shipping" keyValue="83.11.17.00.00" />
  </categoryBag>
</tnModel>

3.6.2. Provider side

A provider can be characterized using the businessEntity structure used by UDDI. Besides that, we are also interested in the way in which such a provider can be reached. For this reason, using the specification provided
by the networks community we can define the network available to the provider.

```
<networks>
  <network name="Internet">
    <device name="fip" type="pc"/>
    <applProtocols>
      <applProtocol name="soap" version="1.1"/>
      <applProtocol name="http" version="1.2"/>
    </applProtocols>
    <segments>
      <segment name="toRouter"/>
      <segment name="toInternet"/>
    </segment>
  </network>

  <network name="Wlan">
    <device name="fip" type="pc"/>
    <applProtocols>
      <applProtocol name="http" version="1.2"/>
    </applProtocols>
    <segments>
      <segment name="toAccessPoint"/>
    </segment>
  </network>
</networks>
```

Let us suppose that this provider is going to realize a shipping service. According to our system model, before starting the development, the service provider has to query the UDDI service looking for the tModel specifying how such a service should be. Obviously, the tModel expresses only a set of recommendations, and the provider might adopt even only a part of it. For this reason, the names used to identify the service components, i.e. the methods, could be different, as well as the messages structure. The service specification is completed defining the WSDL implementation document in which are defined both the supported application protocols and the devices on which the service can be invoked. About the quality aspects, the admissible range values defined in the WSOL specifications are re-defined according to the features of the platform on which the service is installed. Finally, the context defines the constraints about the position of the user during the service invocation.

```
<wsdl:definitions xmlns:wsdl="..."
  xmlns:shipping="..."
  targetNamespace="...">
  ...

  <wsdl:binding name="MyCourierSOAPInternetBinding">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                  style="document"/>
    <wsdl:operation name="shipping:Delivery"/>
  </wsdl:binding>
</wsdl:definitions>
```
Chapter 3. e-Service model

WSDL specification can be extended also including information about the context in which the user should be during the invocation.

A WSOL document is created to define the quality of the service provided. This specification relies on the WSOL defined by the service communities redefining the admissible value ranges of the quality parameters involved. In the example above, only the operation “delivery” has taken into account on both the networks.
3.6. Languages

Once the service is available, the complete WSDL specification such as all the other documents about the quality and the context are published in the UDDI registry. Such specification are included in a businessEntity item where information about the provider are specified.

```xml
<wsol:statement name="price" xsi:type="shipping:price"
    service="shipping:shippingservice"
    portOrPortType="myshipping: MyCourierSoapPort"
    operation="shipping:delivery">
    <expressionSchema:arithmeticComparator type="between">
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticUnitConstant>
                <wsol:number value="20"/>
                <wsol:unit type="Dollar"/>
            </expressionSchema:arithmeticUnitConstant>
        </expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticUnitConstant>
                <wsol:number value="80"/>
                <wsol:unit type="Dollar"/>
            </expressionSchema:arithmeticUnitConstant>
        </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:arithmeticComparator>
</wsol:statement>

<wsol:statement name="price" xsi:type="shipping:price"
    service="shipping:shippingservice"
    portOrPortType="myshipping: MyCourierHttpPort"
    operation="shipping:delivery">
    <expressionSchema:arithmeticComparator type="between">
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticUnitConstant>
                <wsol:number value="20"/>
                <wsol:unit type="Dollar"/>
            </expressionSchema:arithmeticUnitConstant>
        </expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticUnitConstant>
                <wsol:number value="90"/>
                <wsol:unit type="Dollar"/>
            </expressionSchema:arithmeticUnitConstant>
        </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:arithmeticComparator>
</wsol:statement>

<wsol:serviceOffering>
    ...
</wsol:serviceOffering>

<wsol:WSDLdefinitions>
    ...
</wsol:WSDLdefinitions>
```
Chapter 3. e-Service model

<name>MyProvider</name>
<description xml:lang="en">Some data</description>
<contacts>
  <contact>
    <description xml:lang="en">Manager</description>
    <personName>john Stewart</personName>
    <phone useType=""+1 (555) 555 5555"/>
    <email useType=""><manager@myprovider.com</email>
  </contact>
</contacts>

<businessServices>
  <businessService
    serviceKey="UUID:5452388F-77FC-66FA-CFBF175711ACF"
    businessKey="UUID:BBA0422-5362-0983-EBFF-73E2PEFAC3C5">
    <name>eShipping</name>
    <description xml:lang="en">This service allow to send a shipping order using http and soap over a vlan and internet</description>
  </businessService>
</businessServices>

<bindingTemplates>
  <bindingTemplate
    serviceKey="UUID:AEAE34F6-87BC-434F-EFAC-541985FC3FA2"
    bindingKey="UUID:5452388F-77FC-66FA-CFBF175711ACF">
    <accessPoint URLType="http">http://www.fia.com/invocation</accessPoint>
    <tModelInstanceDetails>
      <tModelInstanceInfo
        tModelKey="UUID:A4642EFD-9457-4734-6632-EBFB7467A0C4">
        <instanceDetails>
          <overviewDoc>
            <overviewURL>http://www.fia.com/webservices/shipping.wsdl</overviewURL>
          </overviewDoc>
        </instanceDetails>
      </tModelInstanceInfo>
    </tModelInstanceDetails>
  </bindingTemplate>
</bindingTemplates>

<categoryBag>
  <keyedReference
    tModelKey="UUID:cd35267-0066-4237-b336-66549bdc3063"
    keyValue="03.11.17.00.00" />
</categoryBag>
</businessEntity>
3.6.3. User side

From the user perspective we are only interested on its technological capabilities in order to figure out if the user can invoke the service. For this reason, a user is characterized by the networks and the devices the user has available, and the contexts in which the user can be. This information together with the user expertise are stored in the profile specification.

```xml
<profile>
  <ability>expert</ability>
  ...
  <contexts>
    <context name="home">
      <device name="mypc" type="pc"/>
      <applProtocols>
        <applProtocol name="http" version="1.2"/>
      </applProtocols>
      <segments>
        <segment name="DSLModem"/>
      </segments>
      <location>
        <town>milan</town>
      </location>
      <time zone="GMT+1" from="08:00" to="20:00"/>
    </context>
    ...
  </contexts>
</profile>
```
Chapter 4.

e-Service discovery

Who is looking for an e-Service, according to the model presented in the previous chapter, defines the characteristics of the desired service and submits them to the service directory. Such requirements can be expressed considering both the functional and non-functional aspects of the e-Service, and they rely on the same languages adopted to describe the e-Services themselves. For this reason, a user request can be defined using WSDL to express the functional requirements, whereas a WSOL document can be used to express the quality constraints. It is worth noting that the service requester could define the requirements using even fragments of these specifications. For example, the requested e-Service can be described using a WSDL document where only the <portType> is specified. This means that the requester minds only about e-Services which provide the specified functionalities without any details about how such a functionality can be exploited, i.e., operations, messages, or communication pattern.

In order to support the e-Service publication and discovery process, we developed Urbe (Uddi Registry by Example). Urbe extends the current Uddi implementation providing a new way to search for e-Service based not only on keywords, but also specifying functionalities and quality constraints the desired e-Services must satisfy. The present chapter
discusses the discovery mechanisms on which Urbe is designed whereas the next one, deeply describes the modules composing Urbe.

Concentrating on the discovery process, three are the main steps Urbe performs (Figure 4.1):

- finding the e-Services which perform the functionalities required by the user,
- among those, extracting the e-Services operating in same context, and hence with the same channel, of the user,
- restricting the resulting set of e-Services, only considering the e-Services which guarantee the quality requested by the user.

Service Oriented Computing approach requires that the several actors involved, in this case the service provider and the service user, might not know each other in advance. The connection between them is represented by the languages used to describe the e-Service, which unfortunately provide only a common syntax. For this reason, it might happen that some semantic mismatches describing the e-Service components occurs. With the Semantic Web [11], some techniques have been developed to solve these situations, but it appears that it is not available nowadays a solution able to solve every mismatches in an automatic way.
In this work, in order to automate as much as possible the discovery phase, we rely on two assumptions. First of all the registry does not store every kind of e-Service, but it concentrates only on e-Services belonging to a given application domain. In this way, semantic mismatches become less likely since it is reasonable that, inside the same domain, a common knowledge is shared. Secondly, a domain expert is involved during the classification phase. This new actor is in charge of both (i) solving the terminological mismatches the system is not able to automatically solve, and (ii) tuning the semantic tools adopted as described in the following. At discovery time, the domain expert has only to supervise the system and possibly tuning it.

Figure 4.2 shows a logical architecture of Urbe. The entire system
Chapter 4. e-Service discovery

relies on a Domain knowledge base in which two ontologies are included. In more detail, a term ontology is used to analyze the term semantics and to evaluate their similarity. This ontology can be initially derived by a general purpose ontology such as WordNet and updated by the domain expert. The domain service ontology is created and updated whenever a new e-Service is published in Urbe. The next section will introduce the mechanisms adopted to organize such an ontology. The four modules in the middle, i.e., functional similarity, context inclusion, channel compatibility, and quality satisfiability, are in charge of discover which e-Services satisfy a particular kind of requirements. Finally, the requester interpreter receives as an input a list of requirements using the same set of languages which define services, contexts, and networks. Thus, this module dispatches the user request to the related module, in order to answer with the list of e-Services satisfying the expressed requirements.

With respect to the existing approaches, our model considers not only the e-Service functionalities but also the non-functional aspects. Moreover, focusing on the functional aspects, our approach does not require any additional descriptions but the WSDL specification. For this reason, we can apply our approach on all the existing e-Services nowadays available, despite semantic-oriented approaches which need for additional description (e.g., OWL-S) not easily to deduce for complex e-Services. Such an advantage is compensated for a less precise retrieve as stated in [12], in which the deductive based systems, as WSMO, are considered the best ones.

4.1. Notation

To better explain the mechanisms used during the service discovery, the following notation is adopted:

- \( \Sigma = \{ \sigma_i \} \) represents all the e-Services stored in the Registry.
- \( \sigma_i = ( \{ f_{i,j} \}, \{ c_{i,k} \}, \{ ch_{i,l} \}, \{ q_{i,m} \} ) \) is a single e-Service defined in terms of functionalities provided \( ( f_{i,j} ) \), contexts in which the service operates \( ( c_{i,k} ) \), channels used to transport the information \( ( ch_{i,l} ) \), and quality levels offered \( ( q_{i,m} ) \).
- \( f_{i,j} = \{ op_{i,j,p}, \{ pre_{i,j,p,q} \}, \{ post_{i,j,p,r} \} \} \) represents the functional decomposition according to the e-Service model presented in the previous chapter. Each operation is further defined in terms of input and output, i.e., \( op_{i,j,p} = ( \{ in_{i,j,p,x} \}, \{ out_{i,j,p,y} \} ) \).
4.1. Notation

- $c_{i,k} = (\{loc_{i,k,s}\}, \{time_{i,k,t}\})$ represents the context in terms of location ($loc$) of user and invocation time ($time$).
- $ch_{i,l} = (\{ap_{i,l,v}\}, \{nw_{i,l,w}\})$ represents the application protocols ($ap$) supported and the networks ($nw$) available according to the channel model.
- $q_{r,m} = (\{params_{r,m,n}\})$ where, for each quality parameter, the range of admissible values is defined.
- $\sigma_r = (\{f_{r,j}\}, \{c_{r,k}\}, \{ch_{r,l}\}, \{q_{r,m}\})$ defines the user requirements, where at least the set of functionalities is required.
- $\Sigma_{\{f_{r,j}\}}$ the e-Services $\in \Sigma$ satisfying the functional requirements.
- $\Sigma_{\{c_{r,k}\}}$ the e-Services $\in \Sigma$ satisfying the context requirements.
- $\Sigma_{\{ch_{r,l}\}}$ the e-Services $\in \Sigma$ satisfying the channel requirements.
- $\Sigma_{\{q_{r,m}\}}$ the e-Services $\in \Sigma$ satisfying the quality requirements.
- $\Sigma_{\sigma_r}$ the e-Services $\in \Sigma$ satisfying all the requirements defined in $\sigma_r$, i.e., the result of the discovery process (Figure 4.3).

For each element $e$ introduced above, a function $e.name$ exists and returns the name of the element. We assume that the name is also the element identifier.
Table 4.1.: WSDL entities and corresponding Descriptor concepts

<table>
<thead>
<tr>
<th>WSDL entity</th>
<th>Descriptor concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortType name</td>
<td>Descriptor name</td>
</tr>
<tr>
<td>Operation name</td>
<td>Operation name</td>
</tr>
<tr>
<td>Input message part name</td>
<td>Input entity name</td>
</tr>
<tr>
<td>Output message part name</td>
<td>Output entity name</td>
</tr>
</tbody>
</table>

4.2. e-Service functional similarity

Similarity among e-Services is performed on the basis of semantic information related to the service interface. The similarity evaluation is based on the computation of coefficients obtained comparing input/output entities that services exchange during their execution (Entity-based similarity analysis) and operations that they are able to perform (Functionality-based similarity analysis). Similar approaches have been widely studied in the field of reusable software components [26, 10] to discover components in a library that match given requirements. These approaches usually rely on a descriptor\(^1\), i.e., a subset of information used to describe an e-Service. In particular, following the notation provided above, a descriptor for an e-Service \(\sigma_i\) is formally described as:

\[
\text{descr}_{\sigma_i} = \{\text{op}_{i,j}.\text{name}, \{\text{in}_{i,j,x}\}, \{\text{out}_{i,j,y}\}\}
\]

Descriptors give a summary abstract view of services taking into account only the name of the operations provided and, for each of them, the input and output information. A descriptor can be automatically extracted from the WSDL specification [29, 28] of the e-Service. Table 4.1 introduces the correspondence between these two service representations, whereas Figure 4.4 shows an example of a descriptor derived from a WSDL portType.

4.2.1. Domain terms ontology

The similarity analysis is supported by a domain ontology used to annotate I/O entities and operation names. In this ontology terms are organized by means of weighted terminological relationships (synonymy, generalization/specialization relationships) both extracted from a pre-existing, domain independent basic ontology (e.g., WordNet) and supplied by the domain expert.

\(^1\)Descriptor here introduced is different from the EJB descriptor
4.2. e-Service functional similarity

Figure 4.4.: WSDL example specification part of the deliver sofa e-Service
Chapter 4. e-Service discovery

The domain ontology is organized as a graph \( \langle N, \mathcal{E} \rangle \), where \( N \) is the set of nodes (i.e., terms) and \( \mathcal{E} \) the set of edges (i.e., relationships between terms). Each edge is represented in the form \( \langle (n_h, n_k), t, w \rangle \), where \( n_h \in N \) is the source node of the relationship, \( n_k \in N \) is the destination node, \( t \) is the kind of relationship and \( w \in (0, 1] \) is a weight associated to that kind of relationship. We call path \( p \) between two nodes \( n, n' \in N \) a finite ordered sequence \( \langle e_1, e_2, \ldots, e_n \rangle \), where the source node of \( e_1 \) is \( n \) and the destination node of \( e_n \) is \( n' \). Between two nodes in the ontology there can exist more than one path. Figure 4.5 shows a view of an ontology where words about transportation are related and weights expressed as well.

We define \( \mathcal{P} \subseteq 2^{\mathcal{E}} \) the set of all possible paths in the ontology. The strength of a path \( p \in \mathcal{P} \) is the value of the function \( W : \mathcal{P} \rightarrow (0, 1] \) that associates to \( p \) the product of the weights of all the relationships belonging to it, i.e.,

\[
W(p) = \prod_{i=1}^{n} w_i \quad \text{where } p = \langle e_1, e_2, \ldots, e_n \rangle
\]

and \( w_i \) is the weight associated to the edge \( e_i \)

We say that two terms \( n \) and \( n' \) have name affinity \( (n \equiv n') \) if at least one path of relationships in the ontology between \( n \) and \( n' \) exists and the strength \( W \) of this path is greater or equal than a given threshold. In case more than one path exists, the one with the highest strength is chosen. According to this, we define the name affinity as

\[
A(n, n') = \max(W(p_k)) \in (0, 1] \quad \text{where } k \text{ is one of the possible path between } n \text{ and } n'
\]

4.2.2. Similarity analysis

The similarity between two e-Services \( \sigma_a \) and \( \sigma_b \), is defined by the Global similarity coefficient \( (G_{Sim}) \) defined as a composition of two other similarity coefficients, i.e. Entity-based similarity coefficient \( (E_{Sim}) \), and Functionality-based similarity coefficient \( (F_{Sim}) \) [13]:

\[
G_{Sim}(\sigma_a, \sigma_b) = w_1 \cdot \text{Norm}E_{Sim}(\sigma_a, \sigma_b) + w_2 \cdot \text{Norm}F_{Sim}(\sigma_a, \sigma_b) \in [0, 1]
\]
4.2. e-Service functional similarity

Figure 4.5.: View on domain terms ontology about transportation

$NormE_{Sim}$ and $NormF_{Sim}$ are respectively the values of $E_{Sim}$ and $F_{Sim}$ normalized to the range $[0, 1]$. In more detail, $E_{Sim}$ is given by the following formula:

$$E_{Sim}(\sigma_a, \sigma_b) = \frac{\sum ja.jb \sum ka.kb \sum xa.xb A(in_{a,ja,ka,xa}, in_{b,jb,kb,xb})}{|in_{a,ja,ka}| + |in_{b,jb,kb}|}$$

$$+ \frac{\sum ja.jb \sum ka.kb \sum xa.xb A(out_{a,ja,ka,xa}, out_{b,jb,kb,xb})}{|out_{a,ja,ka}| + |out_{b,jb,kb}|} \in [0, 1]$$

where $| |$ denotes the cardinality of a given set of input or output entities; the higher the number of pairs of entities with name affinity, the higher the value of $E_{Sim}(\sigma_a, \sigma_b)$. $E_{Sim}$ assumes value 0 when no pairs of I/O parameters with name affinity are found, one from $\sigma_a$ and one from $\sigma_b$, while it is 1 when $\sigma_a$ and $\sigma_b$ have the same input and output parameters.

This equation considers all the I/O parameters defined in the e-Service descriptor, independently from the operation with which a particular parameter is associated. In this way, $E_{Sim}$ measures how much two e-Services are based on the same information set. At this stage, this similarity evaluation is performed only considering the name affinity of the parameter names. Further study will consider the data type as well.

On the other side, $F_{Sim}$ performs a more functional analysis. In this case, each operation in $\sigma_a$ is compared with all the operations in $\sigma_b$ in order to identify which is the operation in $\sigma_b$ more similar to the considered operation in $\sigma_a$. 

79
Chapter 4. e-Service discovery

\[ F_{Sim}(\sigma_a, \sigma_b) = \frac{\sum_{i,j} |\text{ Opsim}(\text{op}_a,i, \text{op}_b,j)|}{|\text{operations}|} \in [0,3] \quad (4.5) \]

where:

\[ \text{Opsim}(\text{op}_a, \text{op}_b) = A(\text{op}_a.\text{name}, \text{op}_b.\text{name}) \]

\[ + 2 \cdot \sum_{x_a,x_b} A(\text{in}_{a,x_a}, \text{out}_{b,x_b}) \]

\[ + 2 \cdot \sum_{y_a,y_b} A(\text{out}_{a,y_a}, \text{out}_{b,y_b}) \]

\[ \in [0,3] \quad (4.6) \]

Such a comparison is performed considering name affinity between the operation names and the affinity among the information set involved, i.e. the I/O parameters. Thus, \( F_{Sim} \) returns a value which collects all the best comparison measures and, in this way, reflects how much the two services perform the same functionalities.

We note that \( \text{Opsim}(\text{op}_a, \text{op}_b) \in [0,3] \) since it is the sum of three elements in the range \([0,1]\). It is 0 when there is no affinity between operation names and I/O parameter names of two operations, while it is 3 when two operations have the same name and the same I/O parameters. It is worth noting that in (4.5) we consider only if the two operations are similar, i.e., if the following conditions hold:

- \( \text{Opsim}(\text{op}_a, \text{op}_b) \geq \lambda_{\text{Opsim}} \), where \( \lambda_{\text{Opsim}} > 0 \) is a similarity threshold set by the domain expert;

- each of the three terms on the right hand side of (4.6) is greater than 0. Consequently, \( F_{Sim}(\sigma_a, \sigma_b) \in [0,3] \), since each term \( \text{Opsim}(\text{op}_a,i, \text{op}_b,j) \in [0,3] \).

Weights in (4.3) \( w_1 \) and \( w_2 \), with \( w_1, w_2 \in [0,1] \) and \( w_1 + w_2 = 1 \), are introduced to assess the relevance of each kind of similarity in computing the \( G_{Sim} \). The use of weights in \( G_{Sim} \) is motivated by the need of flexible comparison strategies. For instance, to state that the \( E_{Sim} \) and \( F_{Sim} \) similarity have the same relevance, we choose \( w_1 = w_2 = 0.5 \).

4.2.3. Abstract and concrete e-Services

\( G_{Sim} \) represents how much a service is closer to another one from a functional perspective, mainly considering the affinity of the terms used to describe the operations. This analysis can be exploited by \( \text{URBE} \) comparing the \( \sigma_r \) with any of the \( \sigma_i \) stored in the registry. Actually, it
4.2. e-Service functional similarity

Figure 4.6.: Compatibility class

is more reasonable to optimize this step, pre-computing the similarity coefficient $G_{Sim}$ among the published e-Services and, in this way the Urbe, can identify groups of similar e-Services. Such a clusterizing of the published e-Services can be performed fixing a threshold $\lambda_{G_{Sim}}$ where a cluster $\gamma$ is a set of e-Services $\gamma = \{\sigma_i\}$, where for each $\sigma_i, \sigma_j \in \gamma$ and $i \neq j$, $G_{Sim}(\sigma_i, \sigma_j) \geq \lambda_{G_{Sim}}$. Clustering mechanism can follows a hierarchical procedure, as explained in [39].

The domain expert is in charge of initially assigning the a value of $\lambda_{G_{Sim}}$. This value can be modified by the domain expert in order to provide a better precision on the cluster construction. In particular, one of the main objective is to maintain a good partition among cluster, having a high $G_{Sim}$ value inside the same cluster, and a very low $G_{Sim}$ value among the clusters.

On these basis, the Registry can be organized in two levels as shown in Figure 4.6. At the bottom level the e-Services published are grouped in clusters following the mechanisms described above. These clusters are composed by the e-Services which performs the same functionalities, and they can be considered semantically compatible. For this reason we
introduce the term *compatibility classes* to define such clusters. On the other hand, the top level is populated by e-Services which act for the compatibility classes at bottom level. Since the e-Services at bottom level really exist, whereas the e-Service at top level are suitably built to represent the cluster, we refer to *concrete services* and *abstract services* to describe such a distinction. In particular:

- **Concrete services** are real directly invocable e-Services published by service providers. Result of the discovery phase is one or more of these e-Services.

- **Abstract services** are not directly invocable e-Services, which represent the capabilities of the concrete services belonging to the same compatibility class. These services are defined by the URB and are obtained according to a semi-automatic procedure supervised by the domain expert. Such a procedure can follow both bottom-up and top-down approach.

According to a bottom-up approach, once the clusters are built using the $G_{Sim}$, the abstract service is defined with a “minimal” set of operations, i.e. the set of operations in the abstract service interface are only those common to all the services in the cluster. It is worth noting that this set of operations is composed only by the semantically common operations. Two operations with synonym names are considered equal. The designer can force additional capabilities to the abstract service also including operations belonging to a large number of e-Services in the cluster.

On the other hand, in the top-down approach, the domain expert defines a set of abstract services on the basis of his knowledge about the domain. When a service provider publishes the e-Service in the Registry, according to the $G_{Sim}$ value, the e-Service is assigned to one of the existing compatibility classes. It might happen that any of the existing abstract service can be closer enough to the publishing service. In this case, the domain expert is in charge of defining a new abstract service.

Actually, a good approach to create and to manage the abstract and concrete service, relies on a mix of the two approaches. When a compatibility class becomes too populated, indeed, it means that it could be useful to split the class in two subclasses, each of them assigned to two abstract services. These abstract service will be two specializations of the former abstract service. According to that, the abstract layer can become more complex and organized as a multi-tier structure.
4.2. e-Service functional similarity

4.2.4. Mapping Information

The structure defined above defines the relationship among services only about the semantics of the operation provided. On the contrary syntactic differences are not considered at all. In order to maintain information not only about the how much is closer a concrete service to the abstract service, but also about in which way the two are different, mapping rules are defined. Giving an example, let us consider two services $\sigma_a$ and $\sigma_b$ belonging to the same compatibility class, where $f_a = (op_{a,i})$ and $f_b = (op_{b,j})$. Assuming that $op_a$ and $op_b$ perform the same activity, let us suppose that the required parameters differ. More precisely, $op_a = (\text{string } a_1, \text{string } a_2, \text{int } a_3)$, whereas $op_b = (\text{string } b_1, \text{int } b_2, \text{string } a_3)$. Due to similarity analysis, the systems also knows that $a_1 \cong b_2$, $a_2 \cong b_3$, and $a_3 \cong b_2$. Goal of the mapping rules is to transform this knowledge in a set of information useful to trace the syntactic differences between e-Services in the same cluster.

This example describes only a simple case, where only the order among the parameter differs. Actually the mapping rules can be more complex when a single parameter in an operation can be split in several parameters in another one. In some other cases a parameter can miss, and the mapping rules should specify how the value of such a parameter can be obtained (e.g., asking directly to the user, or looking for a service which returns that parameter.

At this stage, operations can be compared only if they have an equal number of parameters, which a compatible data type. An improvement of this approach can be done only with the support of a data schema matching system we are going to include in the future.

4.2.5. Three-layers domain service ontology

Exploiting the similarity analysis, as well as the concrete and abstract service structure, a domain service ontology can be defined. Such an ontology properly organizes e-Services on three different layers of abstraction, and it can be exploited to enhance service discovery on the basis of user functional requirements (Figure 4.7).

At bottom level, the concrete services layer, represents a collection of e-Services grouped into compatibility classes. An association link is maintained between each abstract service and the corresponding cluster.

The abstract services layer hosts the abstract service organized in a structure created, according to the possible strategies described above. Two kinds of semantic relationships are added between abstract services:

- **is-a**, that holds when an abstract service offers at least the same
functionalities of another one; we say that, given two abstract services $\sigma_{a1}$ and $\sigma_{a2}$, ($\sigma_{a1}$ is-a $\sigma_{a2}$) iff for each operation $op_{a2}$ in $\sigma_{a2}$ there exists a corresponding one $op_{a1}$ in $\sigma_{a1}$ (with the same name or a synonym), where the set of $op_{a2}$ outputs includes the set of $op_{a1}$ outputs and the set of $op_{a2}$ inputs includes the set of $op_{a1}$ inputs;

- **is-composed-of**, that is obtained when an abstract service can be viewed as the composition of other abstract services; we say that, given three abstract services $\sigma_{a1}$, $\sigma_{a2}$ and $\sigma_{a3}$, ($\sigma_{a1}$ is-composed-of ($\sigma_{a2}$, $\sigma_{a3}$)) iff the sets of operations in $\sigma_{a2}$ and $\sigma_{a3}$ constitute a partition of the set of operations in $\sigma_{a1}$.

At top level, the **subject categories layer** considers a standard classification of services to give users a topic-driven access to the underlying services. We considered the UNSPSC classification [37] to organize services in a commonly accepted service taxonomy in UDDI Registry, but this taxonomy is not the only existing effort for service classification and the choice of a different one could be made, since our approach is general enough from this point of view. Each abstract service is associated to one or more service categories in the taxonomy, maintaining an association link between them.

During the construction of the three-layer service ontology the designer is supported by a software tool environment, called **ARTEMIS** [5], that evaluates by means of semi-automatic techniques the coefficients
4.2. e-Service functional similarity

we introduced. ARTEMIS offers a value-added semi-automatic system that facilitates the ontology construction in a dynamic, highly evolving environment.

4.2.6. Exploiting semantic relationships for service discovery

So far, the similarity functions have been exploited in order to classify and to build a service ontology. Semantic relationships among abstract services in the ontology (is-a and is-composed-of), as well as the similarity analysis ($G_{Sim}$), are also used to support the user in finding services that better fit his requirements. Given the user requirements $\sigma_r$, the goal is to retrieve the set of e-Services $\{\sigma_i\} \in \Sigma$ similar, from a functional perspective, to the requested one. To fulfill this goal, three main steps are performed each of them related to the three level composing the service ontology.

First of all, by means of the user, the category is identified among the category tree characterizing the highest level of the service ontology. Then, the similarity analysis used to classify the e-Services in the ontology, can be performed to identify which are the abstract services closer to the requested one. In fact, once the category at highest level in the ontology has been identified, the $G_{Sim}(\sigma_r, \sigma_{abs})$ is calculated for each abstract service $\sigma_{abs}$ associated to the given category. It might happen that any abstract service is able to satisfy user requirements. In such a case semantic relationships can be used to propose other abstract services related to the previously selected one. In such a case, the is-a and is-composed of relationships will be used to identify possible other abstract services.

The third and final step is performed considering the concrete services belonging to the compatibility class represented by the selected abstract service. Using the $G_{Sim}$ value previously calculated which label the association between the abstract service and the concrete ones, only the concrete services with $G_{Sim}$ greater than a given threshold are selected. The domain expert is in charge of defining the initial value of such a threshold after a training phase. Such a value can be modified during the time, according to the eventual feedback coming from the user. The selected concrete services belongs to the $\Sigma_{f,r,j}$. It might happen that no concrete services in the selected cluster satisfies the user requirements. Even in this case a different abstract service with lower $G_{Sim}$ than the first one is considered. As done before, a threshold is defined about the $G_{Sim}$ among abstract service and only the abstract service with an $G_{Sim}$ greater than the threshold can be used to further investigate among the concrete services it represents.
Chapter 4. e-Service discovery

The approach here presented to state the similarity among e-Service takes into account all the elements in the e-Service functional model, but the behavioral components. In fact, the similarity analysis adopted studies the signatures of the e-Services without any interest about their dynamics. According to the e-Service model presented in Chapter 3, an e-Service can specify the order in which the operations have to be invoked using pre- and post-conditions. Although the e-Service behavior is quite important to state the compatibility among services, at this stage this analysis is out of the scope of this thesis.

4.3. Context inclusion

If the service request $\sigma_r$ also specifies in which context the service has to work ($c_r$) an evaluation about the context inclusion is performed. According to the context model presented in Chapter 3, e-Services in the registry specifies the context according to the provider side perspective, i.e., the position in which the user have to be during the service invocation. On the other hand, the $c_r$ specifies when and where the user is going to invoke the service.

The context at both provider and user side, is defined according to different degrees of precision. When the real position is required then the GeoPosition will be defined, this happens for example in application strictly sensible to the context where the information provided change according to the position. A guided tour service for a museum is one of this kind of service where, depending on the user position, the application automatically starts to describe the paintings closer to the user. On the opposite, the Country, or a set of Countries, will be used when the precision provided by GeoPosition is not required, even if an idea about the user location is needed. Such information can be used, for example, when the service should be aware of the laws of a particular country. In case any Location is defined, (i) at provider side means that the service is available everywhere, whereas (ii) at user side means that the system is not able to identify the location.

The same considerations can be done when we consider the Time. The service provider, indeed, can define the interval of time in which the service is available, and in which the user has to invoke the service. Since the user and provider can be located all over the world, the TimeZone supports the time comparison solving the time-zone differences.

Given a request $\sigma_r$ and $\sigma_i \in \Sigma$

$$\sigma_i \in \Sigma_{c_{r,k}} \iff \exists a, b \mid c_{r,a} \subseteq c_{i,b}$$
4.3. Context inclusion

where the operator of inclusion among context is defined as follows

\[ c_{r,a} \subseteq c_{i,b} \iff \exists x, y, t, z \mid time_{r,a,x} \subseteq time_{i,b,y} \land \]
\[ loc_{r,a,t} \subseteq loc_{i,b,z} \]

The inclusion about the time is defined in a natural way, imposing that \( c_{i,k} \) is up during at least one of the interval of time defined by the \( c_{r,k} \).

About the location, the inclusion has to take into account the different levels in which the location is defined. In this way, \( loc_{i,b,z} \) has to cover at least the area required by \( loc_{r,a,t} \). Thus, we define:

\[ loc_{r,a,t} \subseteq loc_{i,b,z} \iff \]
\[ \begin{cases} loc_{r,a,t} = \phi \\ loc_{r,a,t} = loc_{i,b,z} \land loc_{r,a,t}.country \subseteq loc_{i,b,z}.country \end{cases} \] (4.7)

\[ loc_{r,a,t}.country \subseteq loc_{i,b,z}.country \iff \]
\[ \begin{cases} loc_{r,a,t}.country = \phi \\ loc_{r,a,t}.country = loc_{i,b,z}.country \land loc_{r,a,t}.town \subseteq loc_{i,b,z}.town \end{cases} \] (4.8)

\[ loc_{r,a,t}.town \subseteq loc_{i,b,z}.town \iff \]
\[ \begin{cases} loc_{r,a,t}.town = \phi \\ loc_{r,a,t}.town = loc_{i,b,z}.town \land loc_{r,a,t}.town \subseteq loc_{i,b,z}.town \end{cases} \] (4.9)

\[ loc_{r,a,t}.district \subseteq loc_{i,b,z}.district \iff \]
\[ \begin{cases} loc_{r,a,t}.district = \phi \\ loc_{r,a,t}.district = loc_{i,b,z}.district \land loc_{r,a,t}.geo \simeq loc_{i,b,z}.geo \end{cases} \] (4.10)

where \( loc.country, loc.town, loc.district, loc.geo \) return, namely, the country, the town, the district, and the geographical position defined in the context. As it can be noted, the comparison between the \( loc_{r,a,t}.geo \) and \( loc_{i,b,z}.geo \) does not require a strict equivalence. According to the
Chapter 4. e-Service discovery

Figure 4.8.: A portion of domain ontology used to classify the location.

wanted precision and the tolerance of the GPS instruments, the equivalence can be re-defined.

Even for the context comparison, the UDDI structure, already exploited to build the three-layers ontology, can be used to organize the service with respect to the context. The microsoft-com:geoweb:2000 [71] and ISO 3166 [49] classification, for example, can be used to index the services according to the position they specify. Figure 4.8 shows a portion of the structure used to organize the service, which only considers the country and town level. According to that, higher is the precision requested by the user, deeper the service has to be in the resulting tree.

4.4. Channel compatibility

User request $\sigma_r$ could list the set of channels on which the communication should take place. On the other hand services in the registry $\sigma_i \in \Sigma$ defines which are the available channels. In both cases, according to the channel model introduced in the previous chapter, the channel is defined in terms of application protocols supported over a networks. It is worth noting that the same application protocol can be used over several networks and the same network can support several application protocols.

During the service discovery, goal of the channel compatibility assessment phase is to find at least one pair $(ap, nw)$ included in the channel definition of both $\sigma_i$ and $\sigma_r$. Thus, given $\{ch_{i,l}\}$ the channels offered by the service $\sigma_i \in \Sigma$, and $\{ch_{r,l}\}$ the channels requested by the user in $\sigma_r$ then

$$\sigma_i \in \Sigma_{ch_{r,l}} \Leftrightarrow \exists a, b \mid ch_{r,a} \text{ compatible with } ch_{i,b}$$
4.4. Channel compatibility

According to this, the compatibility among channels is evaluated in two steps. Firstly, the application protocols are taken into account. To compare them, we assume that a knowledge base, able to identify the compatibility among protocols, is present. Such a knowledge base is composed by a set of rules defined a-priori by the domain expert which knows the technical aspects about the application protocols. Figure 4.9 shows a possible organization of the knowledge base where the application protocols are grouped in families. Each family represents a technology, and the members are the available versions. According to the current technologies, protocols in the same families are usually compatible, whereas protocols belonging to different families are not usually compatible.

As a second step evaluating the channel compatibility, the user and service networks are considered. In more detail, two networks are considered compatible if they can be joined, i.e. the last segment at both sides can be considered compatible. As done for the application protocol, also for the network a knowledge base in which a set of rules about network compatibilities exists.
Chapter 4. e-Service discovery

4.5. Quality satisfiability

According to the quality model presented in the previous chapter, the quality of service is defined in terms of quality parameters. Introducing the QoE (Quality of Experience), the model is able to define the quality really perceived by the user according to the selected channel. At discovery time, the user defines in \( q_{r,m} \) of \( \sigma_r \) the admissible values for each listed quality parameter and all the quality user requirements must be satisfied. It is worth noting that both the user requirements and the e-Services quality is defined in terms of QoE. So, given \( \{ q_{i,m} \} \) the quality offered by the service \( \sigma_i \in \Sigma \), and \( \{ q_{r,m} \} \) the quality requested by the user in \( \sigma_r \) then

\[
\sigma_i \in \Sigma_{q_{r,m}} \iff \exists a,b \mid q_{i,a} \text{ satisfies } q_{r,b}
\]

where:

\[
q_{i,a} \text{ satisfies } q_{r,b} \iff \forall x \exists y \mid \text{params}_{i,a,y}.name \simeq \text{params}_{r,b,x}.name \land \text{params}_{i,a,y}.range \cap \text{params}_{r,b,x}.range \neq \emptyset
\]

Similarity among names can be performed exploiting the domain terms ontology. Actually, if the community approach presented in the previous chapter is adopted, all the e-Services of the same type are defined using the same quality of service set simplifying the comparison.

About the admissible values, besides the case in which \( \text{params}_{i,a,y}.range = \text{params}_{r,b,x}.range \), the comparison can produce four situations as shown in Figure 4.10, in which a subrange of values in which the quality offered is compliant with the quality requested exists:

a) \( \text{params}_{i,a,y}.worst \leq \text{params}_{r,b,x}.worst \land \text{params}_{i,a,y}.best \geq \text{params}_{r,b,x}.best \)

b) \( \text{params}_{i,a,y}.worst \leq \text{params}_{r,b,x}.worst \land \text{params}_{i,a,y}.best \leq \text{params}_{r,b,x}.best \)

c) \( \text{params}_{i,a,y}.worst \geq \text{params}_{r,b,x}.worst \land \text{params}_{i,a,y}.best \leq \text{params}_{r,b,x}.best \)

d) \( \text{params}_{i,a,y}.worst \geq \text{params}_{r,b,x}.worst \land \text{params}_{i,a,y}.best \geq \text{params}_{r,b,x}.best \)

In these cases the quality provided does not precisely fit the quality requested. In fact, in some case, i.e., a) and c), the e-Service could be
4.5. Quality satisfiability

Figure 4.10.: Quality parameter admissible values comparisons

able to offer a quality higher than the requested one, whereas in some other case, i.e., b) and d), it might happen that the quality could be lower than the expected one. For this reason a negotiation phase could be requested to define the quality really provided by the $\epsilon$-Service.
Chapter 5.

**URBE (Uddi Registry By Example) and flexible e-Service invocation**

The discovery mechanisms described in the previous chapter have been implemented as a UDDI Registry extension called URBE (Uddi Registry By Example). Since one of the most discussed weaknesses of UDDI is about the available limited retrieval methods, with our implementation we aim at providing a new way of searching for services. In particular, the new searching method allows the user to define the functional and quality aspects of the desired service, in order to obtain the list of services able to perform the requested features. The requirements are expressed according to the languages introduced in Chapter 3.

This chapter describes the URBE architecture discussing about the main modules composing it. With respect to the functional similarity algorithm, results of a preliminary experiment are presented as well. Finally, the chapter describes how URBE can be integrated inside a more complex platform able to provide flexible e-Services.
5.1. Urbe architecture

Figure 5.1 shows the main modules composing the Urbe architecture. Urbe is designed to be completely compliant with the current UDDI v.2 specification. For this matter, the system relies on jUDDI [3], a Java open source implementation of UDDI. Starting from the top, the Urbe API exports all the jUDDI functionalities supporting the service publication and service retrieve. Such functionalities perform the basic operations required in the UDDI specifications and, for this reason, they only support key based queries. Thus, Urbe API also include a new set of methods able to support the advance discovery methods defined in the previous chapter. Since the publication phase is affected as well, even the methods about the service publication exported by jUDDI are re-defined.

In more detail, during the publication phase, Urbe processes the service provider publication requests. Despite the typical UDDI publication request, where only the functional aspects are defined, Urbe can manage a publication request in which the quality, context, and channel specifications are defined as well, according to the languages specified in Chapter 3. Before invoking the jUDDI publication methods, Urbe, relying on the Affinity module, identifies which is the compatibility classes in which the new service has to be included in the Service Ontology. Affinity module, in fact, is in charge of calculating the $G_{Sim}$ value which is at the basis of our clustering mechanism. During the $G_{Sim}$ computation, Affinity module relies on both the Domain Term Ontology and WordNet in order to identify the similarity degree among, namely,
domain specific terms and generic terms. After that, by means of the jUDDI API, the new service is indexed in the underlying UDDI Registry. It is worth noting that when the publication request also includes quality, context, or channel definitions, the modules QoS, Context, and Channel, respectively, are invoked in order to store the related information in internal databases.

When the user is going to search for a service, Urbe provides two kinds of methods. The first one is a replication of the jUDDI retrieve methods which implement the UDDI specifications, whereas the second one allows the user to exploit the new approach provided by Urbe. Focusing on the second kind of methods, the constraints about the functionalities, quality, context, and channel are defined according to the same languages used by the service provider to publish a service. For this reason, supposing that the user request includes all those aspects, it is decomposed in four sub requests, each of them submitted to the related module, i.e., Affinity, QoS, Context, and Channel. In particular, by means of the $G_{Sim}$ value calculated by the Affinity module, Urbe is able to identify the subset of published e-Services able to support the requested functionalities. These selected e-Services are thus compared against the other three sub requests in order to identify a list of e-Service satisfying all the user constraints.

In the current Urbe release, the integration with jUDDI is complete, the Affinity modules has been implemented, whereas the remaining modules are under construction. A details of the already implemented
modules are shown in Figure 5.2. In particular, `it.mais.urbe` represents the URBE entry point in which all the publication and retrieve methods are exported. Some of these functionalities are directly performed by `org.apache.juddi`, whereas the URBE specific ones, defined in `it.mais.urbe.function`, relies on the `it.mais.urbe.ServiceOntology`. Finally, `it.mais.urbe.descriptor`, given a WSDL specification, returns the related service descriptor described in Chapter 3, whereas `it.mais.urbe.affinity` provides the method to calculate the $G_{Sim}$ formula.

Focusing on the `it.mais.urbe.affinity` package, a preliminary experiment to validate how well our approach works has been performed. Unfortunately, at this stage, it is quite difficult to find a group of available e-Services, described by WSDL, which perform the same set of functionalities. For this reason, analyzing several flight companies websites, we built a set of WSDL specifications which simulates a hypothetical set of e-Services. Appendix A presents these WSDL file in details. Functionalities specified in the WSDL file reflects what really the web application performs in terms of both information the user has to insert, and information the website returns. Among the others, `Abstract.wsdl` simulates the abstract service generated by the domain expert to represent the compatibility class, whereas `Expedia.wsdl` is about a touristic service which does not perform any flight seat reservation.

Table 5.1 shows the $G_{Sim}$ values calculated for each service pair. In particular, for each row, the value indicates how an e-Service is similar to the e-Service in the column. It is worth noting how the Expedia service has the lower values, whereas the Abstract service the greater ones. About the latter, those values reflect the role of the abstract service which should be closer to all the member of the compatibility class it represents. About the former, the $G_{Sim}$ values can suggest to the domain expert to move the service in a different compatibility class.

### 5.2. Flexible e-Service invocation platform

As described in Chapter 1, using an adaptive information system means the possibility to execute a process in a flexible way. Here a service can be substituted with another one in case the former fails, or the user can switch among different channels inside the same session. In this scenario, URBE functionalities become very useful at both design and execution time. The compatibility classes defined during the e-Service similarity analysis, indeed, identify a set of e-Services which can be substituted each other. For this reason, the functionalities provided by URBE can be
5.2. Flexible e-Service invocation platform

Table 5.1: $G_{Sim}$ experimental results

<table>
<thead>
<tr>
<th></th>
<th>Abstract</th>
<th>Alitalia</th>
<th>Ba</th>
<th>Delta</th>
<th>Expedia</th>
<th>Lufthansa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>–</td>
<td>0.666</td>
<td>0.656</td>
<td>0.590</td>
<td>0.181</td>
<td>0.402</td>
</tr>
<tr>
<td>Alitalia</td>
<td>0.666</td>
<td>–</td>
<td>0.656</td>
<td>0.590</td>
<td>0</td>
<td>0.222</td>
</tr>
<tr>
<td>Ba</td>
<td>0.656</td>
<td>0.656</td>
<td>–</td>
<td>0.584</td>
<td>0</td>
<td>0.166</td>
</tr>
<tr>
<td>Delta</td>
<td>0.590</td>
<td>0.590</td>
<td>0.584</td>
<td>–</td>
<td>0</td>
<td>0.465</td>
</tr>
<tr>
<td>Expedia</td>
<td>0.181</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>0.202</td>
</tr>
<tr>
<td>Lufthansa</td>
<td>0.402</td>
<td>0.222</td>
<td>0.166</td>
<td>0.465</td>
<td>0.202</td>
<td>–</td>
</tr>
</tbody>
</table>

integrated inside a more complex platform. Aim of such a platform, developed inside the MAIS Project (Multichannel Adaptive Information Systems) \(^1\) is to provide a way to invoke $e$-Services in a flexibly. This means that, once the most suitable $e$-Service is discovered and invoked, the platform is able to monitor and eventually to substitute the running $e$-Service. $e$-Services invocation could be adapted after being selected, due to one of the following events:

- during execution, the service fails or cannot be reached;
- a better service could be provided by a new $e$-Service;
- a new version of a selected $e$-Service is available;
- a different channel offers a better quality with respect to the user context.

Goal of the platform is to support such an adaptivity without affecting the observable behavior of the process \([109]\).

According to the system model presented in Chapter 3, the platform, which includes URBE, can be considered as an application server on which $e$-Services are deployed. Since URBE can be considered as a generic UDDI node, published $e$-Services can be not only those installed and running on the same application server, but also $e$-Services available on external application servers which use URBE to advertise them. Nonetheless, after the user submits the required $e$-Service specifications, URBE can returns three kinds of $e$-Services:

- External $e$-Services, if the $e$-Services are only published in URBE but available on external application servers.

\(^1\)http://www.mais-project.it
• Internal e-Services, if the e-Services are published and deployed on the application server.

• Internal flexible e-Services, if the e-Services are not real services published but the abstract representation according to the ontology structure presented in the previous chapter.

In the first case, the user, obtained the e-Service endpoint, invokes the e-Service independently of the application server in which Urbe is involved. On the contrary, about the second and third case, the application server is directly involved during the service invocation. Both internal e-Services and flexible e-Services can be simple or composite, i.e., standalone or based on a process in which the activities are performed by e-Services. Even the e-Services composing the process can be external or internal ones.

Flexible e-Services represent the peculiarity of this platform. If a user invokes such an e-Service, the platform is in charge of managing the substitution if one the situations listed above occurs. An internal flexible e-Service, indeed, does not really exist but it represents a group of e-Services able to provide the same functionalities. Even if from a user perspective the e-Service remains the same, the application server can decide at run-time which is the best internal service.

When the e-Service is selected, the application server is in charge of monitoring the e-Service availability and measuring the effective quality of service. In case such a measured quality does not satisfy what the user requested or the e-Service is no longer available, then an adaptation can take place in two ways. First of all, if the service is available on several channels, the one available with respect to the user context and able to satisfy the quality requested is selected. If any alternative channel is available, the running e-Service is substituted with a compatible one. Before discussing when and in which way such an adaptation can take place, an overview of the module composing the platform is introduced.

Referring to the e-Service life cycle presented in the Chapter 3, as shown in Figure 5.3 the flexible invocation platform provides the functionalities about the service deployment, invocation, and registry. Service developer uses such a platform to deploy its services, the requester looks for an e-Service querying the registry, whereas the service user invokes the selected e-Service exploiting the flexibility provided by the service invocation module.

From a logical standpoint, the platform is composed by three main modules (Figure 5.4). First of all, an e-Service deployer is included to support the service provider to make available its service in the platform.
5.2. Flexible e-Service invocation platform

With respect to the current application server, this platform requires a modified descriptor where channel information is included as well.

Secondly, Urbe provides the e-Service publishing and discovering functionalities, where e-Services are organized according to the classification described in the previous chapter. In particular, in order to present how this platform allows a flexible invocation, in the rest of the chapter we suppose that the requester always selects a flexible invocation e-Service.

Finally, an e-Service invoker represents the entry-point of the platform providing a way to invoke in a flexible way the selected e-Services. As deeply described in the related section, even the e-Service invoker relies on the service discovery functionalities of Urbe in order to obtain, given an e-Service, the e-Services compatible with it.

5.2.1. e-Service deployer

Once the service developer decides to deploy the built service into the platform, the e-Service deployer is in charge of including such an e-Service inside the set of internal e-Services. Among the others, the effect
of such a deployment is twofold. On one hand, the WSDL of the deploying service is completed with the implementation document specifying the available channel. On the other hand, such a specification is published in Urbe.

About the WSDL completion, the e-Service deployer refers to the descriptor submitted together with the service code. Inside such a descriptor, besides the typical information, the specification about the networks on which the service should be available is specified as well. Obviously, such a list of networks should be compliant with the set of networks the platform makes available. Considering the specification in Figure 5.5, we assume that the platform is able to inform the service developer about the possibility to provide its e-Services relying on two networks, i.e., Internet and Wlan. Even both of them have the device named fip as the end point, they rely on two different sets of network segments and support different application protocols. Actually, names used in the example are only a reference to a more detailed description about the network, where network segments, reachable machines are defined as well as protocols supported and quality aspects do. Obviously, it might happen that the same application protocol can be exploited over several networks, and the same network can host several application protocols. In case the service descriptor is not compliant with what the platform offers, the deployment is aborted and the developer informed.

Extending the EJB descriptor notation, as shown in Figure 5.6, the e-Service deployer is able to define a set of configurations which defines the networks on which the service should be invoked. It is worth noting that the descriptor can specify only the list of networks and, for each of them, the list of application protocols. The device as well as the network
5.2. Flexible e-Service invocation platform

segments could not be selected since they represent a physical and not modifiable characteristic of the network.

Now, the first step performed by the e-Service deployer is to install the service inside the platform, creating all the stubs required for each application protocol requested. Such stubs will be installed on the devices related to the networks on which the service should be available. Among the others, the e-Service deployer is also in charge of publishing the e-Service inside Urbe when the WSDL specification is completed with the network information.

Concentrating on the WSDL completions, it is worth remembering that what the e-Service developer submits, is a WSDL composed only by the interface documentation, without any details about the application protocols. Aim of the e-Service deployer is to define a binding tag for each pair (network, application protocol). The network, indeed, affects the location on which the service will be available, whereas the application protocol will define the way in which the information should be encoded and exchanged.

In the same way, the service developer is in charge of defining the service quality. Such an information can be derived from the features of the underlying platform on which the e-Service will run. Since the
quality the user perceives also depends on the channel adopted, it is worth noting that the service deployer has to define the quality for each channel previously identified.

Referring to the example in Figure 5.7, two binding specifications are present, since two pairs (network, application protocol) are defined in the descriptor (Figure 5.6). It is worth noting that all the bindings direct to the same machine since both the networks rely on the same device. The only difference is represented by the application protocol specified in the URL which allows the machine to select which is the protocol the user needs to use. About the service quality, the e-Service definition is completed also considering a WSOL specification in which the set of interesting quality parameters are defined. Such a specification is not directly included in the WSDL file but, by means of the portOrPortType attribute, it can be linked to the interesting functionalities. In particular, as shown in Figure 5.8, the two channels differ only with respect to the price. In this case, an invocation based on http communication will cost less than 10 dollars, whereas a soap based invocation results more expensive, i.e., between 10 and 100 dollars.

5.2.2. e-Service invoker

Core of the flexible invocation platform is represented by the e-Service invoker. Aim of this module is to dispatch the user requests to the e-Services running on the platform and to monitor their execution in order to support the required flexibility. Since the user can invoke an e-Service according to several application protocols, we suppose that an instance of e-Service invoker exists for each application protocol offered by the platform. Considering the example in Figure 5.6, where the networks
supported by the platform are defined, an instance of the e-Service invoker supports a SOAP communication, whereas another one HTTP. Each of these instances has a list of e-Services needing to communicate according to the protocols the instance supports.

Independently of the supported protocol, Figure 5.9 shows in details how the e-Service invoker is composed. Among the others, a reflective architecture allows the system to know how the system itself works. In particular, the reflective architecture is used to obtain measures about the quality dimensionS which characterize the running e-Services. For this reason, we suppose that all the information about the service execution, i.e., service availability, channel availability, bandwidth, and so on, can be obtained querying the reflective architecture. It is worth noting that part of the reflective architecture might be installed at client-side, too. In this way, the platform is able to obtain information about the user context and the user profile while the user is invoking the e-Service.

At highest level, the *Concrete Service Invoker* is in charge of communicating with the user according to the selected application protocols. In case the invoked e-Service is a composite one, then the *Process Orchestrator* is involved to manage the workflow execution. As discussed above, a flexible e-Service is the abstract representation of a compatible
class, populated by concrete services, according to the structure of the domain service ontology. Aim of the Concretizer is to identify, querying URBE, which is the best concrete service given a flexible one.

Concretizer

Concretizer represents the module able to provide the flexibility offered by the overall platform. By definition, a flexible e-Service is one of the abstract services, included in the domain service ontology, which represents a set of concrete e-Services able to perform the same set of functionalities. Whenever a user is invoking a flexible e-Service, the Concretizer is in charge of identifying which is the best concrete service.
5.2. Flexible e-Service invocation platform

Exploiting the mechanisms adopted in URBE, the Concretizer can obtain a set of the candidate e-Services, everyone able to satisfy the user constraints, among which the best one is selected. Such a selection is performed taking into account both the quality requested by the user and the context in which the user is operating.

Both Concrete Service Invoker and Process Orchestrator calls the Concretizer. In the first case, the Concrete Service Invoker needs to know, given a flexible e-Service, which is its best concrete representation, whereas in the second case Process Orchestrator invokes the Concretizer if the process, it has to orchestrate, includes at least one flexible e-Service. In the second case, the Process Orchestrator sends to the Concretizer the entire process specification since it might happen that some global constraints are specified. In this situation the service selection performed by the Concretizer becomes more complex since both global and local constraints should be satisfied.

Considering a simple process composed by two sequence activities performed by two flexible e-Service, let us assume that both global and local constraints are represented by the price. In particular, the overall process should not cost more than 100 euros, and any single e-Service should not cost more than 60 euros. When the Process Orchestrator submits such a specification, the Concretizer is in charge of define an execution plan in which, for each flexible e-Service a concrete service is selected. Now, let us assume that the Concretizer retrieves two concrete e-Services, costing 50 euros each. This solution is fine if the service execution works correctly and each of them satisfy the constraints. Otherwise, if some-
thing goes wrong, the *Concretizer* is in charge of re-optimizing the process selecting other e-Services. For example, if after the execution of the first e-Service the system realizes that the final cost is 55 euros, the *Concretizer* has to figure out a new e-Service, for the second flexible service, which costs 45 euros.

**Concrete Service Invoker**

The first task performed by the *Concrete Service Invoker* when a message arrives, is to identify which is the e-Service the user is invoking. According to the way in which the URLs of the e-Services are built and stored in the WSDL, the radix of such URL has, besides the name of the protocol, the name of the device on which the Concrete Service Invoker is running followed by the service identifier. Considering the URL in Figure 5.10, `soap://www.fip.com/invocation/mycourier/delivery` represents the protocol on which the communication can take place. Continuing on the URL, the server name represents the machine on which the e-Service invoker is running and, thus, where the Concrete Service Invoker is waiting for the incoming messages. The last part defines the e-Service installed in the platform the user is invoking.

![Figure 5.10: e-Service URL example](soap://www.fip.com/invocation/mycourier/delivery)

Assuming that the user is invoking a flexible e-Service, such a service is not directly invokable by definition. In fact, it corresponds to an abstract service in the domain service ontology included in *URBE*. For this reason, *Concrete Service Invoker* relies on the *Concretizer* to find the best concrete service able to perform the functionalities requested. Once the *Concretizer* returns the selected service, it might happen that the two interfaces can be different. In fact, the compatibility among the abstract service and the concrete service only depends on what the functionalities do. Therefore, the way in which the functionalities must be invoked could differ. Relying on the mapping information defined at publication time, when the concrete service had been assigned to a compatibility class, the domain expert has to implement the specific interface that defines wrapper methods. *Concrete Service Invoker* is in charge of retrieving and executing such a wrapper able to transform the invocation between the two interfaces. The wrapper methods, used
by the *Concrete Service Invoker* to perform effective service invocation, allow the system for:

- Locating concrete service access points.
- Translating abstract operations inputs into concrete operations inputs.
- Translating concrete operations outputs into abstract operations outputs.

In some case, it might occurs that a concrete operation requires more input parameter than the abstract one. This means that it is not possible to invoke this operation only with an abstract parameters translation, but it is necessary to retrieve all missing parameters by asking them directly to the user. The domain expert is in charge to create wrappers able to retrieve additional needed information. It is worth noting that using the wrappers the user does not realize which is the real e-Service invoked, even if all its requirements are satisfied.

*Concrete Service Invoker* is also in charge of monitoring the e-Service execution in order to verify if the quality of service promised is really offered to the user. The monitoring relies on the reflective architecture which returns the current values of the quality parameters related to the platform. In case the quality decreases to a value lower than the promised one, the *Concrete Service Invoker* has to invoke the *Concretizator* to obtain a new concrete service able to solve the situation. In case any e-Service are able to increase the quality, the *Concrete Service Invoker* raises an exception to the user. The reflective architecture can be also installed at client side, in this way the *Concrete Service Invoker* is able to know the profile and the context of the user. Based on this information, the *Concrete Service Invoker* is able to interact with the user in order to switch among the channels.

**Process Orchestrator**

*Process Orchestrator* is involved during the e-Service execution if the invoked service is not a simple service but a composite one. In this case, the invoked e-Service will be specified by a BPEL process in which several other e-Services are called to reach a common goal (Figure 5.11). *Concrete Service Invoker*, once it realizes that the user is requesting a composite service, informs the *Process Orchestrator* to start a new process which will be based on the BPEL specification of the invoked e-Service. According to the e-Service model adopted in this work, the
client is not aware about the process specification which involves other several e-Services.

Aim of the Process Orchestrator is to schedule the composing e-Services invocation according to the process specification, to maintain the status of the process, and to verify the correct execution of the process. Recursively, even the e-Services composing the process can be of three types: (i) external e-Services, (ii) internal e-Services, (iii) internal flexible e-Services. In any case, the Process Orchestrator is in charge of invoking the e-Service referring to the URL stored in the BPEL specification. It is interesting to note that in the last two cases the URL directs to the Concrete Service Invoker, i.e., the Process Orchestrator becomes a client of the architecture. For this reason, as occurred in the simple case, the Concrete Service Invoker is in charge of realizing if the Process Orchestrator is invoking a simple or a flexible e-Service or, recursively, a composite e-Service.

Actually, in this case the situation is more complex since the composing e-Services might be flexible ones and the requirements can be local or global. In the first case, the local constraints are about a given activity and express the requirements to a single flexible composing e-Service. In the second case, the constraints are applied to the overall process. In order to satisfy both these kinds of constraints, the Process Orchestrator send the process specification to the Concretizer, which is in charge of selecting, for each flexible composing e-Service, the most suitable concrete service satisfying both the global and local constraints. Such a selection is not returned to the Process Orchestrator but it will be used
5.2. Flexible e-Service invocation platform

by the Concretizator when the Concrete Service Invoker requires the concrete service related to a flexible e-Service requested by the Process Orchestrator.

Figure 5.12 sketched, using an UML sequence diagram, how the three modules in the e-Service invoker communicate in case a composite e-Service, in which at least one composing e-Service is a flexible one, is invoked by the client. Once Concrete Service Invoker realizes that the client is going to invoke a composite service, it informs the Process Orchestrator. The Process Orchestrator forwards the BPEL specification to the Concretizator. The Concretizator builds an execution plan where for each activity specified by a flexible e-Service, a concrete service is identified. Next, Process Orchestrator starts to execute the process invoking the e-Service composing it. If the service is an internal flexible e-Service, the invocation involves the Concrete Service Invoker which will ask to the Concretizator the best concrete service. The Concretizator realizes that such a request is inside the execution plan it has previously defined, and returns to the Concrete Service Invoker the concrete service selected. Now, the e-Service is invoked and the result forwarded to Process Orchestrator since it is the client of such an e-Service. The Process Orchestrator, according to the process specification, continues the execution as far as the activity requires to send the answer to the client.
Chapter 6.

Conclusions

Service Oriented Computing paradigm is driving the development of new applications where the Web service technologies are used to provide services invocable over the Web. In the meanwhile, even the mobile devices, such as smartphone and PDA, can exploit an increasing amount of resources in terms of battery, memory, and computational power. In this scenario, it is reasonable that the same user might invoke the same service using several devices. Multichannel information systems represent a new generation of information systems where Web service technologies are improved taking into account these new devices and the protocols used to communicate with them.

In this thesis work, we have described what is a multichannel information system providing an e-Service model where both functional aspects and quality aspects have been taken into account. Relying on this model, we have introduced Urbe (UDDI Registry By Example), a service registry which extends the current UDDI implementation. Service discovery, indeed, is one the most important emerging key aspects in Service Oriented Computing. The necessity for both a human and an application to find the more suitable e-Services with respect to a set of requirements is discussed in this work. So far, the current tools for service discovery allow only a categorization based search, where the user has to browse the
service registry according to functional taxonomies. With Urbe, now under development, we aim at proposing a tool which performs a content based query, where the user can express its requirements using the same languages with which the published e-Services are described. The service discovery mechanisms are based on both syntactical and semantic analysis. The former allows of evaluating a technological compatibility whereas the latter allows of solving possible mismatches about terms interpretation. Retrieval techniques in Urbe also consider the context of the request and the requested quality level improving the precision of the selection.

Exploiting the functionalities Urbe provides, this thesis has introduced a platform able to support a flexible e-Services invocation. Using such a platform the user, which is invoking an e-Service, can switch among the channels according to its needs. The channel switching can also be driven at provider side, when it realizes that it is not possible to ensure the promised quality of service. If any of the available channels can solve this situation, the platform tries to substitute the running e-Service with a compatible one.

Further research in this area is still needed. First of all, the proposed models and tools needs to be enhanced and verified in real applications. In this direction, we are developing test beds in the areas of risk management and tourism, and we are planning to provide support to mobile information systems for dangerous goods transportation. A selection of the proposed techniques have to be performed for each application areas. Moreover, domain specific support has to be provided, for defining appropriate quality dimensions, users profiles, process, e-Service application logic and presentation logic adaptation.

In addition, some critical issues have emerged during the development of the project. In particular, one of the aspects that needs further development is the need of providing techniques and tools for the management of highly adaptive applications. Such management should not be supported by manual interaction, but we envision that adaptive information systems should be designed to be able to diagnose potential failures and the system should be providing self healing mechanisms to support possible failures. The global system should be perceived by its users as a reliable and stable system, which proactively detects and solve potential and occurring interoperability problems and failures. Further research in this area is needed to characterize e-Service properties, to provide self healing mechanisms, to support the design of diagnosable systems.

Finally, the proposed quality model is a general framework which is useful for both the contract specification and contract enactment. In the first case the quality parameters can be used to specify multi-party
contracts between all involved partners, defining admissible values for each involved object. Negotiation mechanisms are required as well, in order to support such contract definitions. Once the contract is defined, the quality parameters are also used to monitor potential problems and to adapt service provisioning both on provider side and on channel side to context characteristics changed during the enactment.
Bibliography


Bibliography


Bibliography


119


Bibliography


Appendix A.

Flight companies e-Services

WSDL

A.1. Abstract.wsdl

```xml
<wsdl:definitions targetNamespace="http://www.mais-project.it/
flexibleServices/flightservice" ...>

<wsdl:types>
...
</wsdl:types>

<wsdl:message name="ticketPaymentRequest">
  <wsdl:part name="ticket" type="tns1:FlightDetails"/>
  <wsdl:part name="passengerFirstname" type="xsd:string"/>
  <wsdl:part name="passengerLastName" type="xsd:string"/>
  <wsdl:part name="cardType" type="xsd:string"/>
  <wsdl:part name="cardNumber" type="xsd:long"/>
  <wsdl:part name="monthOfExpiration" type="xsd:string"/>
  <wsdl:part name="yearOfExpiration" type="xsd:int"/>
  <wsdl:part name="email" type="xsd:string"/>
  <wsdl:part name="idSession" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="getFlightDetailsRequest">
  <wsdl:part name="flight" type="tns1:Flight"/>
</wsdl:message>

<wsdl:message name="loginUserRequest">
  <wsdl:part name="UserCode" type="xsd:int"/>
  <wsdl:part name="pin" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="ticketPaymentResponse">
  <wsdl:part name="ticketPaymentReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="getUserDataResponse">
  <wsdl:part name="getUserDataReturn" type="tns1:User"/>
</wsdl:message>

<wsdl:message name="getFlightDetailsResponse">
  <wsdl:part name="getFlightDetailsReturn" type="tns1:FlightDetails"/>
</wsdl:message>

<wsdl:message name="findFlightResponse">
  <wsdl:part name="findFlightReturn" type="impl:ArrayOf_tns1_Flight"/>
</wsdl:message>

<wsdl:message name="loginUserResponse">
  <wsdl:part name="loginUserReturn" type="xsd:int"/>
</wsdl:message>
```

125
Appendix A. Flight companies e-Services WSDL

```xml
<wsdl:message name="setPassengerDataRequest">
  <wsdl:part name="ticket" type="tns1:FlightDetails"/>
  <wsdl:part name="firstName" type="xsd:string"/>
  <wsdl:part name="lastName" type="xsd:string"/>
  <wsdl:part name="phone" type="xsd:string"/>
  <wsdl:part name="seat" type="xsd:string"/>
  <wsdl:part name="meal" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="setPassengerDataResponse">
</wsdl:message>

<wsdl:message name="getUserDataRequest">
  <wsdl:part name="idSession" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="findFlightRequest">
  <wsdl:part name="departureAirport" type="xsd:string"/>
  <wsdl:part name="arrivalAirport" type="xsd:string"/>
  <wsdl:part name="departureDay" type="xsd:int"/>
  <wsdl:part name="departureMonth" type="xsd:string"/>
  <wsdl:part name="departureTime" type="xsd:string"/>
  <wsdl:part name="returnDay" type="xsd:int"/>
  <wsdl:part name="returnMonth" type="xsd:string"/>
  <wsdl:part name="returnTime" type="xsd:string"/>
  <wsdl:part name="numberOfAdults" type="xsd:int"/>
  <wsdl:part name="classSeat" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="FlightService">
  <wsdl:operation name="getUserData">
    <wsdl:input name="getUserDataRequest" message="impl:getUserDataRequest"/>
    <wsdl:output name="getUserDataResponse" message="impl:getUserDataResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getFlightDetails">
    <wsdl:input name="getFlightDetailsRequest" message="impl:getFlightDetailsRequest"/>
    <wsdl:output name="getFlightDetailsResponse" message="impl:getFlightDetailsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="setPassengerData">
    <wsdl:input name="setPassengerDataRequest" message="impl:setPassengerDataRequest"/>
    <wsdl:output name="setPassengerDataResponse" message="impl:setPassengerDataResponse"/>
  </wsdl:operation>
  <wsdl:operation name="loginUser">
    <wsdl:input name="loginUserRequest" message="impl:loginUserRequest"/>
    <wsdl:output name="loginUserResponse" message="impl:loginUserResponse"/>
  </wsdl:operation>
  <wsdl:operation name="ticketPayment">
    <wsdl:input name="ticketPaymentRequest" message="impl:ticketPaymentRequest"/>
    <wsdl:output name="ticketPaymentResponse" message="impl:ticketPaymentResponse"/>
  </wsdl:operation>
  <wsdl:operation name="findFlight">
</wsdl:operation>
</wsdl:portType>
```
A.2. Alitalia.wsdl

```xml
<wSDL:operation>
  <wSDL:input name="findFlightRequest"
    message="impl:findFlightRequest"/>
  <wSDL:output name="findFlightResponse"
    message="impl:findFlightResponse"/>
</wSDL:operation>
</wSDL:portType>

<wSDL:binding name="flightserviceSoapBinding" type="impl:FlightService">
  ...
</wSDL:binding>
</wSDL:definitions>

A.2. Alitalia.wsdl

<wSDL:definitions targetNamespace="http://www.alitalia.it/service" ...>
  ...
</wSDL:definitions>

<wSDL:message name="loginMilleMigliaRequest">
  <wSDL:part name="MilleMigliaCode" type="xsd:int"/>
  <wSDL:part name="pin" type="xsd:int"/>
</wSDL:message>
<wSDL:message name="setPassengerDataResponse">
</wSDL:message>
<wSDL:message name="loginMilleMigliaResponse">
  <wSDL:part name="loginMilleMigliaReturn" type="xsd:int"/>
</wSDL:message>
<wSDL:message name="ticketPaymentResponse">
  <wSDL:part name="ticketPaymentReturn" type="xsd:int"/>
</wSDL:message>
<wSDL:message name="setPassengerDataRequest">
  <wSDL:part name="ticket" type="tns1:AlitaliaFlightDetails"/>
  <wSDL:part name="firstname" type="xsd:string"/>
  <wSDL:part name="lastname" type="xsd:string"/>
  <wSDL:part name="phone" type="xsd:string"/>
  <wSDL:part name="seat" type="xsd:string"/>
  <wSDL:part name="meal" type="xsd:string"/>
</wSDL:message>
<wSDL:message name="findFlightResponse">
  <wSDL:part name="findFlightReturn"
    type="impl:ArrayOf_tns1_AlitaliaFlight"/>
</wSDL:message>
<wSDL:message name="findFlightRequest">
  <wSDL:part name="departureAirport" type="xsd:string"/>
  <wSDL:part name="arrivalAirport" type="xsd:string"/>
  <wSDL:part name="departureDay" type="xsd:int"/>
  <wSDL:part name="departureMonth" type="xsd:string"/>
  <wSDL:part name="departureTime" type="xsd:string"/>
  <wSDL:part name="returnDay" type="xsd:int"/>
  <wSDL:part name="returnMonth" type="xsd:string"/>
  <wSDL:part name="returnTime" type="xsd:string"/>
  <wSDL:part name="numberOfAdults" type="xsd:int"/>
  <wSDL:part name="classSeat" type="xsd:string"/>
</wSDL:message>
</wSDL:definitions>

127
Appendix A. Flight companies e-Services WSDL

```xml
<wSDL:message name="getMilleMigliaDataResponse">
  <wSDL:part name="getMilleMigliaDataReturn" type="tns1:MillemigliaUser"/>
</wSDL:message>
<wSDL:message name="ticketPaymentRequest">
  <wSDL:part name="ticket" type="tns1:AlitaliaFlightDetails"/>
  <wSDL:part name="passengerFirstname" type="xsd:string"/>
  <wSDL:part name="passengerLastName" type="xsd:string"/>
  <wSDL:part name="cardType" type="xsd:string"/>
  <wSDL:part name="cardNumber" type="xsd:long"/>
  <wSDL:part name="monthOfExpiration" type="xsd:string"/>
  <wSDL:part name="yearOfExpiration" type="xsd:int"/>
  <wSDL:part name="email" type="xsd:string"/>
  <wSDL:part name="idSession" type="xsd:int"/>
</wSDL:message>
<wSDL:message name="getFlightDetailsResponse">
  <wSDL:part name="getFlightDetailsReturn" type="tns1:AlitaliaFlightDetails"/>
</wSDL:message>
<wSDL:message name="getFlightDetailsRequest">
  <wSDL:part name="flight" type="tns1:AlitaliaFlight"/>
</wSDL:message>
<wSDL:message name="getMilleMigliaDataRequest">
  <wSDL:part name="idSession" type="xsd:int"/>
</wSDL:message>
<wSDL:portType name="Alitalia">
  <wSDL:operation name="getMilleMigliaData">
    <wSDL:input name="getMilleMigliaDataRequest" message="impl:getMilleMigliaDataRequest"/>
    <wSDL:output name="getMilleMigliaDataResponse" message="impl:getMilleMigliaDataResponse"/>
  </wSDL:operation>
  <wSDL:operation name="getFlightDetails">
    <wSDL:input name="getFlightDetailsRequest" message="impl:getFlightDetailsRequest"/>
    <wSDL:output name="getFlightDetailsResponse" message="impl:getFlightDetailsResponse"/>
  </wSDL:operation>
  <wSDL:operation name="setPassengerData">
    <wSDL:input name="setPassengerDataRequest" message="impl:setPassengerDataRequest"/>
    <wSDL:output name="setPassengerDataResponse" message="impl:setPassengerDataResponse"/>
  </wSDL:operation>
  <wSDL:operation name="loginMilleMiglia">
    <wSDL:input name="loginMilleMigliaRequest" message="impl:loginMilleMigliaRequest"/>
    <wSDL:output name="loginMilleMigliaResponse" message="impl:loginMilleMigliaResponse"/>
  </wSDL:operation>
  <wSDL:operation name="ticketPayment">
    <wSDL:input name="ticketPaymentRequest" message="impl:ticketPaymentRequest"/>
    <wSDL:output name="ticketPaymentResponse" message="impl:ticketPaymentResponse"/>
  </wSDL:operation>
  <wSDL:operation name="findFlight">
    <wSDL:input name="findFlightRequest" message="impl:findFlightRequest"/>
  </wSDL:operation>
</wSDL:portType>
```

128
A.3. Ba.wsdl

```xml
<wsdl:definitions targetNamespace="http://www.ba.it/service" ... >

<wsdl:types>...
</wsdl:types>

<wsdl:message name="setPassengerResponse">
</wsdl:message>
<wsdl:message name="getExecutiveClubDataRequest">
  <wsdl:part name="uid" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="ticketPaymentRequest">
  <wsdl:part name="ticket" type="tns1:FlightDetails"/>
  <wsdl:part name="cardIssuer" type="xsd:string"/>
  <wsdl:part name="cardNumber" type="xsd:long"/>
  <wsdl:part name="monthOfExpiration" type="xsd:string"/>
  <wsdl:part name="yearOfExpiration" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="getExecutiveClubDataResponse">
  <wsdl:part name="getExecutiveClubDataReturn" type="tns1:FrequentFlyer"/>
</wsdl:message>
<wsdl:message name="getFlightDetailsRequest">
  <wsdl:part name="flight" type="tns1:Flight"/>
</wsdl:message>
<wsdl:message name="findFlightResponse">
  <wsdl:part name="findFlightReturn" type="impl:ArrayOf_tns1_Flight"/>
</wsdl:message>
<wsdl:message name="getFlightDetailsResponse">
  <wsdl:part name="getFlightDetailsReturn" type="tns1:FlightDetails"/>
</wsdl:message>
<wsdl:message name="findFlightRequest">
  <wsdl:part name="departure" type="xsd:string"/>
  <wsdl:part name="arrival" type="xsd:string"/>
  <wsdl:part name="departureDay" type="xsd:int"/>
  <wsdl:part name="departureMonth" type="xsd:string"/>
  <wsdl:part name="departureTime" type="xsd:string"/>
  <wsdl:part name="returnDay" type="xsd:int"/>
  <wsdl:part name="returnMonth" type="xsd:string"/>
  <wsdl:part name="returnTime" type="xsd:string"/>
  <wsdl:part name="numberOfAdults" type="xsd:int"/>
  <wsdl:part name="classSeat" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="ticketPaymentResponse">
</wsdl:message>
</wsdl:definitions>
```

A.3. Ba.wsdl
Appendix A. Flight companies e-Services WSDL

```xml
<wsdl:part name="ticketPaymentReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="setPassengerRequest">
<wsdl:part name="ticket" type="tns1:FlightDetails"/>
<wsdl:part name="name" type="xsd:string"/>
<wsdl:part name="surname" type="xsd:string"/>
<wsdl:part name="phone" type="xsd:string"/>
<wsdl:part name="seat" type="xsd:string"/>
<wsdl:part name="meal" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="Ba">
<wsdl:operation name="getFlightDetails">
<wsdl:input name="getFlightDetailsRequest" message="impl:getFlightDetailsRequest"/>
<wsdl:output name="getFlightDetailsResponse" message="impl:getFlightDetailsResponse"/>
</wsdl:operation>
<wsdl:operation name="ticketPayment">
<wsdl:input name="ticketPaymentRequest" message="impl:ticketPaymentRequest"/>
<wsdl:output name="ticketPaymentResponse" message="impl:ticketPaymentResponse"/>
</wsdl:operation>
<wsdl:operation name="findFlight">
<wsdl:input name="findFlightRequest" message="impl:findFlightRequest"/>
<wsdl:output name="findFlightResponse" message="impl:findFlightResponse"/>
</wsdl:operation>
<wsdl:operation name="getExecutiveClubData">
<wsdl:input name="getExecutiveClubDataRequest" message="impl:getExecutiveClubDataRequest"/>
<wsdl:output name="getExecutiveClubDataResponse" message="impl:getExecutiveClubDataResponse"/>
</wsdl:operation>
<wsdl:operation name="setPassenger">
<wsdl:input name="setPassengerRequest" message="impl:setPassengerRequest"/>
<wsdl:output name="setPassengerResponse" message="impl:setPassengerResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="serviceSoapBinding" type="impl:Ba">
...
</wsdl:binding>
</wsdl:definitions>

A.4. Delta.wsdl

```xml
<wsdl:definitions targetNamespace="http://www.delta.it/service" ...>
<wsdl:types>
...
</wsdl:types>
```
<wsdl:message name="paymentResponse">
  <wsdl:part name="paymentReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="paymentRequest">
  <wsdl:part name="idFlight" type="tns1:DeltaFlightDetails"/>
  <wsdl:part name="firstname" type="xsd:string"/>
  <wsdl:part name="lastName" type="xsd:string"/>
  <wsdl:part name="cardType" type="xsd:string"/>
  <wsdl:part name="cardNumber" type="xsd:long"/>
  <wsdl:part name="expirationMonth" type="xsd:string"/>
  <wsdl:part name="expirationYear" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="findFlightResponse">
  <wsdl:part name="findFlightReturn" type="impl:ArrayOf_tns1_DeltaFlight"/>
</wsdl:message>

<wsdl:message name="setPassengerDataRequest">
  <wsdl:part name="idFlight" type="tns1:DeltaFlightDetails"/>
  <wsdl:part name="firstname" type="xsd:string"/>
  <wsdl:part name="lastname" type="xsd:string"/>
  <wsdl:part name="phone" type="xsd:string"/>
  <wsdl:part name="seat" type="xsd:string"/>
  <wsdl:part name="meal" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getFlightDetailsResponse">
  <wsdl:part name="getFlightDetailsReturn" type="tns1:DeltaFlightDetails"/>
</wsdl:message>

<wsdl:message name="getFlightDetailsRequest">
  <wsdl:part name="flight" type="tns1:DeltaFlight"/>
</wsdl:message>

<wsdl:message name="setPassengerDataResponse">
</wsdl:message>

<wsdl:message name="findFlightRequest">
  <wsdl:part name="from" type="xsd:string"/>
  <wsdl:part name="to" type="xsd:string"/>
  <wsdl:part name="departureDay" type="xsd:int"/>
  <wsdl:part name="departureMonth" type="xsd:string"/>
  <wsdl:part name="departureTime" type="xsd:string"/>
  <wsdl:part name="returnDay" type="xsd:int"/>
  <wsdl:part name="returnMonth" type="xsd:string"/>
  <wsdl:part name="returnTime" type="xsd:string"/>
  <wsdl:part name="numberOfAdults" type="xsd:int"/>
</wsdl:message>

<wsdl:portType name="Delta">
  <wsdl:operation name="getFlightDetails">
    <wsdl:input name="getFlightDetailsRequest" message="impl:getFlightDetailsRequest"/>
    <wsdl:output name="getFlightDetailsResponse" message="impl:getFlightDetailsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="setPassengerData">
    <wsdl:input name="setPassengerDataRequest" message="impl:setPassengerDataRequest"/>
    <wsdl:output name="setPassengerDataResponse" message="impl:setPassengerDataResponse"/>
  </wsdl:operation>
  <wsdl:operation name="findFlight"/>
Appendix A. Flight companies e-Services WSDL

```xml
<wsdl:input name="findFlightRequest" message="impl:findFlightRequest"/>
<wsdl:output name="findFlightResponse" message="impl:findFlightResponse"/>
</wsdl:operation>
<wsdl:operation name="payment">
<wsdl:input name="paymentRequest" message="impl:paymentRequest"/>
<wsdl:output name="paymentResponse" message="impl:paymentResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="serviceSoapBinding" type="impl:Delta">
...
</wsdl:binding>
</wsdl:definitions>

A.5. Expedia.wsdl

```xml
<wsdl:definitions targetNamespace="http://www.expedia.it/service" ...>
...
</wsdl:types>
<wsdl:message name="getUserDataResponse">
<wsdl:part name="getUserDataReturn" type="tns1:User"/>
</wsdl:message>
<wsdl:message name="hotelRapidSearchResponse">
<wsdl:part name="hotelRapidSearchReturn" type="impl:ArrayOf_tns1_Hotel"/>
</wsdl:message>
<wsdl:message name="bookHotelRequest">
<wsdl:part name="hotel" type="tns1:Hotel"/>
<wsdl:part name="room" type="tns1:Room"/>
<wsdl:part name="contact" type="xsd:string"/>
<wsdl:part name="arrivalDate" type="xsd:string"/>
<wsdl:part name="departureDate" type="xsd:string"/>
<wsdl:part name="preferences" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="loginRequest">
<wsdl:part name="userName" type="xsd:string"/>
<wsdl:part name="password" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="loginResponse">
<wsdl:part name="loginReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="payHotelsRequest">
<wsdl:part name="details" type="tns1:ReservationDetails"/>
<wsdl:part name="cardType" type="xsd:string"/>
<wsdl:part name="cardNumber" type="xsd:string"/>
<wsdl:part name="monthExpiration" type="xsd:string"/>
<wsdl:part name="yearExpiration" type="xsd:string"/>
<wsdl:part name="idSession" type="xsd:int"/>
<wsdl:part name="ownerFirstname" type="xsd:string"/>
<wsdl:part name="ownerLastname" type="xsd:string"/>
</wsdl:message>
```

132
A.5. Expedia.wsdl

```xml
<wsdl:part name="ownerAddress" type="tns1:Address"/>
<wsdl:part name="ownerPhone" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getUserDataRequest">
<wsdl:part name="idSession" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="hotelRapidSearchRequest">
<wsdl:part name="location" type="xsd:string"/>
<wsdl:part name="arrivalData" type="xsd:string"/>
<wsdl:part name="departureData" type="xsd:string"/>
<wsdl:part name="numberOfAdults" type="xsd:int"/>
<wsdl:part name="numberOfKids" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="payHotelsResponse">
<wsdl:part name="payHotelsReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="bookHotelResponse">
<wsdl:part name="bookHotelReturn" type="tns1:ReservationDetails"/>
</wsdl:message>

<wsdl:portType name="Expedia">
<wsdl:operation name="login">
<wsdl:input name="loginRequest" message="impl:loginRequest"/>
<wsdl:output name="loginResponse" message="impl:loginResponse"/>
</wsdl:operation>
<wsdl:operation name="bookHotel">
<wsdl:input name="bookHotelRequest" message="impl:bookHotelRequest"/>
<wsdl:output name="bookHotelResponse" message="impl:bookHotelResponse"/>
</wsdl:operation>
<wsdl:operation name="getUserData">
<wsdl:input name="getUserDataRequest" message="impl:getUserDataRequest"/>
<wsdl:output name="getUserDataResponse" message="impl:getUserDataResponse"/>
</wsdl:operation>
<wsdl:operation name="payHotels">
<wsdl:input name="payHotelsRequest" message="impl:payHotelsRequest"/>
<wsdl:output name="payHotelsResponse" message="impl:payHotelsResponse"/>
</wsdl:operation>
<wsdl:operation name="hotelRapidSearch">
<wsdl:input name="hotelRapidSearchRequest" message="impl:hotelRapidSearchRequest"/>
<wsdl:output name="hotelRapidSearchResponse" message="impl:hotelRapidSearchResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="serviceSoapBinding" type="impl:Expedia">
...
</wsdl:binding>
</wsdl:definitions>
```
Appendix A. Flight companies e-Services WSDL

A.6. Lufthansa.wsdl

```xml
<wsdl:definitions targetNamespace="http://www.lufthansa.it/service" ... >
  <wsdl:types>
    ...
  </wsdl:types>
  <wsdl:message name="setPassengerDataResponse"/>
  <wsdl:message name="getUserDataResponse">
    <wsdl:part name="getUserDataReturn" type="tns1:MilesAndMoreUser"/>
  </wsdl:message>
  <wsdl:message name="reserveRequest">
    <wsdl:part name="ticket" type="tns1:Flight"/>
    <wsdl:part name="firstname" type="xsd:string"/>
    <wsdl:part name="lastname" type="xsd:string"/>
    <wsdl:part name="cardType" type="xsd:string"/>
    <wsdl:part name="cardNumber" type="xsd:long"/>
    <wsdl:part name="monthOfExpiration" type="xsd:string"/>
    <wsdl:part name="yearOfExpiration" type="xsd:int"/>
    <wsdl:part name="email" type="xsd:string"/>
    <wsdl:part name="idSession" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="reserveResponse">
    <wsdl:part name="reserveReturn" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="getUserDataRequest">
    <wsdl:part name="login" type="xsd:string"/>
    <wsdl:part name="password" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="setPassengerDataRequest">
    <wsdl:part name="ticket" type="tns1:Flight"/>
    <wsdl:part name="firstname" type="xsd:string"/>
    <wsdl:part name="lastname" type="xsd:string"/>
    <wsdl:part name="seat" type="xsd:string"/>
    <wsdl:part name="meal" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="Lufthansa">
    <wsdl:operation name="setPassengerData">
      <wsdl:input name="setPassengerDataRequest" message="impl:setPassengerDataRequest"/>
      <wsdl:output name="setPassengerDataResponse" message="impl:setPassengerDataResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getUserData">
      <wsdl:input name="getUserDataRequest" message="impl:getUserDataRequest"/>
      <wsdl:output name="getUserDataResponse" message="impl:getUserDataResponse"/>
    </wsdl:operation>
    <wsdl:operation name="reserve">
      <wsdl:input name="reserveRequest" message="impl:reserveRequest"/>
      <wsdl:output name="reserveResponse" message="impl:reserveResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

134
A.6. Lufthansa.wsdl

<wSDL:binding name="serviceSoapBinding" type="impl:Lufthansa">
... 
</wSDL:binding>
</wSDL:definitions>