Behavioral matchmaking for service retrieval

Juan-Carlos Corrales

Behavioral matchmaking for service retrieval

by

Juan Carlos Corrales

A thesis presented to the
University of Versailles Saint-Quentin-en-Yvelines
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Sciences
Speciality
Computer Science

Versailles, France, January 2008

JURY

Boualem BENATALLAH, Professor University of New South Wales, Australia (Reviewer)
Farouk TOUMANI, Professor University of Blaise Pascal, Clermont-Ferrand (Reviewer)
Bernd AMANN, Professor University of Pierre et Marie Curie-LIP6, Paris (Examiner)
Mokrane BOUZEGHOUB, Professor University of Versailles Saint-Quentin-en-Yvelines (Advisor)
Daniela GRIGORI, Associate Professor of University Versailles Saint-Quentin-en-Yvelines (Co-advisor)

I hereby declare that I am the sole author of this thesis.
I authorize the Versailles Saint-Quentin-en-Yvelines University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Juan Carlos Corrales

I further authorize the Versailles Saint-Quentin-en-Yvelines University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Juan Carlos Corrales

Abstract

The capability to easily find useful services (software applications, software components, scientific computations) becomes increasingly critical in several fields. Current approaches for services retrieval are mostly limited to the matching of their inputs/outputs possibly enhanced with some ontological knowledge. Recent works have demonstrated that this approach is not sufficient to discover relevant components. In this dissertation we argue that, in many situations, the service discovery should be based on the specification of service behavior. The idea behind it, is to develop matching techniques that operate on behavior models and allow delivery of partial matches and evaluation of semantic distance between these matches and the user requirements. Consequently, even if a service satisfying exactly the user requirements does not exist, the most similar ones will be retrieved and proposed for reuse by extension or modification. To do so, we reduce the problem of behavioral matching to a graph matching problem and we adapt existing algorithms for this purpose.

Motivated by these concerns, we developed the WS-BeM platform for ranking web services based on behavior matchmaking, which takes as input two WSCL or BPEL protocols and evaluates the semantic distance between them. The prototype is also available as a web service. Furthermore, an application is described concerning the tool for evaluating the effectiveness of the behavioral matchmaking method.

To my family

Acknowledgements

My foremost thank goes to my thesis advisers Mokrane Bouzeghoub and Daniela Grigori. Without them, this dissertation would not have been possible. I thank them for their patience and encouragement that carried me on through difficult times, and for their insights and suggestions that helped to shape my research skills. Their valuable feedback contributed greatly to this dissertation.

I thank the rest of my thesis committee members: Dr. Boualem Benatallah, Dr. Farouk Toumani and Dr. Bernd Amann. Their valuable feedback helped me to improve the dissertation in many ways.

I thank all doctoral students and staffs in PRiSM Laboratory, especially to SIAL group. Many thanks to my lab mates, past and present, for all the help and friendship shared. Specials thanks to Octavio Ramirez, Parinaz Davari, Veronika Peralta, Xiaohui Xue, Dimitre Kostadinov and Sofiane Abbar. It was nice to work with them in a friendly environment.

I also want to thank all people in the Telematics Engineering Group of the University of Cauca for their support. In the same way, I want thank the Program Alban for the scholarship received during these three years.

I have now the pleasure of acknowledging those who have provided me with the personal foundation which has been so indispensable to my professional life. I owe special thanks with all my heart to my parents, Fredy and Stella, and to my brother David Camilo, for providing the bedrock of support on which my life has been built upon and the fortress of never-ending love which has enabled me to defy all the storms of life. I am also grateful to my girlfriend Carolina for her patience and enriching my life with her love.

Juan Carlos Corrales
January 9, 2008
Versailles, France

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

The area of Web usage has evolved from a sole repository for text and images to a popular means of business process integration giving birth to Web services. Web Services are the self-describing, modular software application that can be marked by URL. It can be defined, advertised, discovered, located and used across Internet by using a set of open standards such as SOAP, XML. The concept of a web service has received much attention in recent years as a promising vehicle for cross-organizational integration of applications based on the web. At its most basic, a web service is a network-connected application, which processes XML documents and exchanges these with its environment. This simple model allows interoperability in heterogeneous networks such as the internet by providing loose coupling of applications and platform-independency.

One of the most appealing aspects of Web services is having the ability to aggregate the functionality of individual Web services by composing them to create Web processes. This leads to the automatization of the capabilities related to publication, discovery of web services that take part in a composition (behavioral interface, choreography and orchestration).

The complexity of the Web process discovery depends directly on user requirements. In a practical situation, user requirements normally consist of multiples web services that take part in a composition. Furthermore, the available candidate Web processes might satisfy only a part of the user requirements. These Web processes may originate from heterogeneous sources and may be represented in different forms.

Further, the increasing number of Web processes descriptions are difficult to manage in open environments such as the Web. The main problem arises when hundreds of different Web processes are created by composing hundreds of thousands of different services. Moreover, web processes are built independently from each other at different locations by different people. Therefore, discovering a Web process that matches user's requirement is time consuming, tedious and clumsy.

In order to make the discovery process efficient, scalable and effective, there exist three types of desiderata for a service description: it has (a) to be *capable* of performing a certain task (i.e., maintain a shopping cart), (b) to expose a particular *interface* (i.e., provide view, addproduct and remove-product) and (c) to *behave* in a certain manner (i.e., ignore any request for product removals if no product additions have been performed yet). Such expectations motivate and guide the searches of the developers through

Web service repositories, as they try to discover and select the service that best matches their needs.

Further, nowadays the capability to easily find useful services (software applications, software components, scientific computations, Web processes) becomes increasingly critical in several fields. Examples of such services are numerous:

- Software applications as web services which can be invoked remotely by users or programs. One of the main problems arising from the current framework of web services is the need to dynamically put in correspondence service requesters with service suppliers, hence allowing the formers to benefit from the more recent offers or updates of the latter. This is specially important for applications interested in dynamic binding to services, depending on their availability, their quality or their current cost, which fits with the dynamic nature of the Web where services are frequently published, removed or released.

- Programs and scientific computations (scientific workflows) which are important resources in the context of Grid systems, sometimes even more important than data [57]. In such environments, data and procedures are first rank classes which can be published, searched and handled. They might be, for example, complex mathematical computations, simulation models or data mining algorithms. Thus, the scientists need to retrieve these procedures to determine whether it is worth to reutilize them or rewrite them again with respect to desired characteristics.

- Pervasive environments emphasize the need for application dynamism and autonomic behaviors. These environments are characterized by the variability and mobility of acting entities on the network. Dynamically composing applications and guaranteeing service continuity to the users is a grand challenge in pervasive computing. In this sense, service selection mechanisms are needed in pervasive architectures in order to overcome service ambiguity, which leads to composition unpredictability [26].

In all these cases, users are interested in finding suitable components in a library or a collection of programs described by appropriate models. User formulates a requirement as a process model; his goal is to use this model as a query to retrieve all components whose respective process models match with a whole or part of this query. If models that match exactly do not exist, those which are most similar must be retrieved. For a given goal, the models that require minimal modifications may be the most suitable ones as the component reuse aims generally to reduce development cost. If the retrieved models have to be tailored to the specific needs, the adaptation effort should be minimal.

In the next section we further motivate the work presented in this dissertation.

1.2 Motivating scenarios

In this section, we present two scenarios requiring behavioral matchmaking. The first example situates in the context of web services integration and consists in retrieving services having compatible behavior. It will be used in section 3.2.7 (An example for the WSCL matchmaking) to illustrate our approach. The second example shows that a behavior matchmaking method and a way to quantify similarities/dissimilarities between two models are needed, not only in the context of service retrieval, but also in other applications, like delta-analysis.

Web services integration. Consider a company that uses a service S to order office supplies. Suppose that the company wants to find retailers (say WalMart or Target) having compatible web services (a new retailer or replacing the current partner). The allowed message exchange sequences are called conversation protocols and can be expressed for example using BPEL abstract processes, WSCL, or other protocol languages (see, e.g., [20]). The specification of the conversation protocol is important, as it rarely happens that service operations can be invoked independently from one another. Thus the company will search for a service having a compatible conversation protocol. After finding retailer services, the most compatible one has to be selected among them. If the service is not fully compatible, the company will either adapt its service or develop an adaptor to conform to the behavior of the retrieved service. In the former case, finding the most similar service allows to minimize the development cost. In the latter case, identifying automatically the differences between protocols is the first stage in the process of semi-automatically developing adapters (see [18]). In both cases, process models have to be automatically compared in order to highlight their similarity or differences in terms of their business protocols.

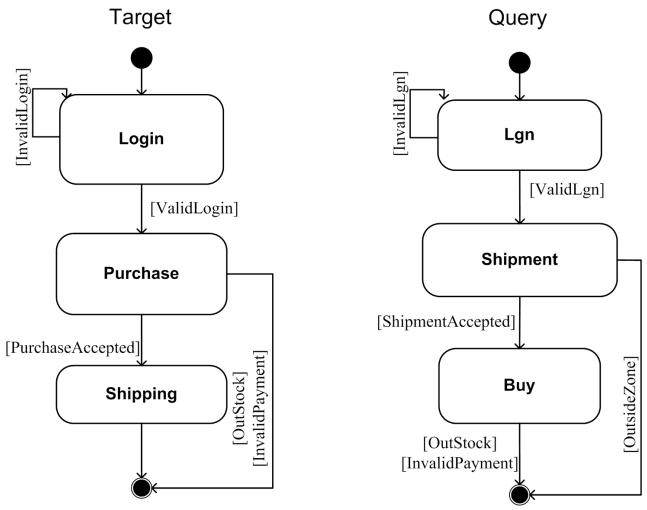

Figure 1.1: Two conversation protocols

To go further in our example, suppose that the protocol of the query model expects to exchange messages in the following order: clients can invoke *Lgn*, then they can send shipping preferences *Shipment* and finally invoke the *Buy* operation. In contrast, the protocol of a given target service allows the following sequence of operations: clients can invoke *Login*, then *Purchase* and finally they receive the shipping information (*Shipping*). Matching the two processes shows that, although they have the same list of actions, the order of actions is not the same. Moreover, the shipping interaction is modelled differently in the two protocols. The user should adapt the query model to the target model by reordering the sequence of actions or generating an adapter as done in [18].

Note that finding retailer services taking into account only inputs and outputs can be used only as a first filter because it does not guarantee that retrieved services support the same order of exchanged messages as done by behavioral matching.

Delta analysis consists in finding the differences between two models. For example, the query model is an implemented model currently running in a given enterprise, while the target model corresponds to a well-known standard in the business area (e.g. RosettaNet PIPs specified by RosettaNet group). The enterprise challenge is to check whether their implemented process conforms to this standard. Thus, they need to compare the conversation model of their existing service with that prescribed by the standards. For large models and especially when the enterprise does not use the standard vocabulary, a tool should help users identifying all the differences between the two models. Based on these differences, the reengineering cost of the existing service could be better evaluated.

1.3 Problem statement

The problem to be addressed in this dissertation is how to support the service discovery process using a matchmaking phase based on the specification of the service behavior. This situation leads to provide formalisms for modelling services and algorithms for reasoning about similarities among instances of such models. Assuming there are appropriate formalisms for specifying behaviors of a query service and corresponding target services, there should also exist correlations between such behaviors. Therefore, determining similarities between service behavior descriptions is necessary for any service searching mechanism.

Further, the service discovery requires new semantics for matchmaking because a simple attribute/value search (see [111]) does not apply to service matching, since these services comprise complex structures which must be compared in order to find a match. Hence, the matchmaking is not simply an equivalence of service attributes, it needs to consider the structural information of the processes included into the services.

1.4 Our proposal

The purpose of this thesis is to develop matching techniques that operate on behavioral models of the services and allow delivery of partial matches as well as an evaluation of the semantic distance between these matches and the user requirements. Consequently, even if it does not exist a service that satisfies exactly the requirements of the user, the most similar ones will be retrieved and proposed for reuse by extension or modification. To do so, we reduce the problem of service behavioral matching to a graph matching problem and we adapt existing algorithms for this purpose.

1.5 Contributions and main results

The main contributions of this thesis are:

- **A detailed analysis of service matchmaking.** The main results of this analysis are: (a) A survey on several viewpoints from which behavioral models for service composition can be captured; (b) A description of formal representations of services; (c) An analysis of the techniques used for service matchmaking, which were clarified in three categories: Service matchmaking based on interfaces, semantics and behavior.

- **The proposal of techniques and algorithms for the service matchmaking.** In this dissertation a solution for service retrieval based on behavioral specification was developed. By using a graph representation formalism for services, we proposed to use an error correcting graph matching algorithm in order to allow an approximate service matching (see [61, 42, 47, 41, 45, 43, 44]).

- **Application of service matchmaking to WSCL protocol.** In this thesis we motivated the need to retrieve services based on their conversation model. We exemplified our approach for behavior matching for conversation protocols expressed using the WSCL model (see [61, 47, 41, 43]). Starting from the classical graph edit distance, we proposed two new graph edit operations to take into account the difference of granularity levels that could appear in two models. The conversation protocol matchmaking process is composed of the following steps: First, the conversations protocols to be compared are transformed into graphs. Next, the graphs are expanded in order to have the same level of granularity in both graphs and the error-correcting graph matching algorithm is applied. The similarity function evaluates the similarity between the graphs. Finally, the granularity levels are compared and the costs corresponding to identified differences are added to the total distance.

- **Application of service matchmaking to BPEL protocol.** Considering the importance of Web processes, in this dissertation we discussed our approach for Behavioral matchmaking, by examining the usage of matching techniques in the context of BPEL behavioral specifications (see [42]). The BPEL matchmaking process is composed of the following steps: first, the BPEL documents to be compared are transformed to graphs. Next, the error correcting graph matching algorithm is applied (considering the decompostion and composition functions during the algorithm execution). Then, the similarity function evaluates the similarity between the graphs.

- **A prototype for behavioral matchmaking for service retrieval.** We developed a prototype called Ws-BeM (Web services-Behavioral Matchmaking), which implements the proposed approaches. The tool allows the execution of the algorithms for matchmaking services (see [45, 44, 46]). In order to validate our approach, the prototype was tested with two application scenarios: the matching of BPEL and WSCL protocols.

- **A prototype for evaluating the effectiveness of our behavioral matchmaking method.** We constructed a tool for evaluating the effectiveness of our behavioral matchmaking method (see [44, 46]). This tool allows to create a user service ranking based on manual comparisons between a query service and the services in the repository. The tool permits to compare the result obtained by the Ws-BeM platform and a ranking defined by users.

1.6 Thesis outline

The remaining of this thesis is organized in five chapters:

Chapter 2 describes the related works to behavioral matchmaking for service retrieval, wich have been clustered into three main categories: (i) As we are focused into matching of composed services, in this category we concentrate on several viewpoints from which behavioral models for service composition can be captured, and the relations between these viewpoints. (ii) Since our approach develops matching techniques that operate on behavioral models of the services, in this category we depict formal descriptions

that offer representation features in order to describe, exchange and execute services.(iii) Finally, we explain the related works with respect to service matchmaking techniques for service discovery.

Chapter 3 explains our graph-based approach to behavior matchmaking and shows an application of this one. First we introduce the graph matching problem explaining its definition and notation. As this PhD thesis concentrates on inexact service matching, a method to measure the similarity of two graphs is depicted. Then, the error-correcting subgraph matching is presented in detail. Finally, we show an application case of this algorithm to WSCL (Web Services Conversation Language) protocols.

Chapter 4 discusses the approach for behavioral matchmaking by examining the usage of matching techniques in the context of BPEL behavioral specifications. First we will introduce the BPEL protocol, then we explain the BPEL to graph transformation. Thereupon, we show the BPEL matchmaking algorithm which is based on the algorithm introduced in previous chapter, but considering the comparison rules for the BPEL metamodel. Finally, an example of the BPEL matchmaking process will be depicted.

Chapter 5 illustrates the practical use of our proposal for Behavioral matchmaking evaluating real services (WSCL and BPEL services). First we describe our prototype and experimentations presenting the We-BeM tool, describing its functionalities, architecture and its user interface. Then, we show a tool for evaluating our matching method. Finally, we present the performance evaluation tests, describing the considered test scenarios, the test strategies and the obtained results.

Chapter 6 presents conclusions and some research perspectives.

Chapter 2

Analysis of Related Work

Web Services (WS) provide an ubiquitously supported framework for application-to-application interaction, based on existing Web protocols and open XML standards. The Web Services framework is one of the newest members in the service-oriented computing area. It is divided into three areas: communication protocol, service description and service discovery. Several specifications have been developed like SOAP [4], Web Services Description Language (WSDL) [6] and Universal Description, Discovery and Integration (UDDI) [5], correspondingly. In addition to these standards, there are a number of proposed standards and on-going work towards standardization that deals with non-functional aspects of Web services. The leading specifications that provide domain independent languages for describing non-functional aspects of Web services are WS-Policy [7] and WS-Agreement [2] . In addition, a number of domain specific vocabularies are being developed to describe various non-functional aspects. For example, WS-Security [3] is a vocabulary for the security domain that defines security tokens, encryption algorithms and other security related artifacts. Other domain specific vocabularies include WS-Trust [8] and WS-Transaction [8].

One of the most appealing aspects of Web services is having the ability to aggregate the functionality of individual Web services by composing them to create Web processes. This leads to the automatization of the capabilities related to publication, discovery of web services that take part in a composition (behavioral interface, choreography and orchestration). However, service composition that enables one to aggregate or compose existing services into a new composite service is still highly complex but critical task in service-oriented technologies. Several key challenges in service composition need to be addressed: How to facilitate the discovery of services? and how to enhance reliability of composite services?.

In this dissertation we argue that, in many situations, the service discovery process requires a matchmaking phase based on the specification of the component behavior. Therefore, the related works to this PhD thesis have been clustered into three main categories:

(i) As we are focused into service matchmaking, in this category we concentrate on several viewpoints from which behavioral models for service composition can be captured, and the relations between these viewpoints.

(ii) Since our approach develops matching techniques that operate on behavioral models of the services, in this category we depict formal representations that allows to describe, exchange and execute service behaviors.

(iii) Finally, we explain the related works with respect to service matchmaking techniques for service discovery.

2.1 Service Behavioral Models

The Service-Oriented Computing (SOC) paradigm refers to the set of concepts, principles and methods that represent computing in Service-Oriented Architecture (SOA) in which software applications are constructed based on independent component services with standard interfaces. The main idea of SOC/SOA is to explicitly separate software engineering from programming, to emphasize on software engineering and to deemphasize on programming. SOC separates software development into three independent parties: Application builders (by software engineers), service providers (by programmers) and service brokers (joint effort from standard organizations, computer industry and government).

- Service providers: They use a traditional programming language such as Java, C++, or C# to write program components. All components will be wrapped with open standard interfaces, called services, or Web services if the services are available over the internet, so that application builders can simply use the services without further communication with the service providers. The same services can be used by many applications.

- Service brokers: Allow services to be registered and published for public access. Help application builders to find services they need.

- Application builders: Instead of constructing software from scratch using basic programming language, the application builders represent the final users who specify the logic application in a high-level language specification, using standard services as components. Therefore, the application builders are software engineers who have a good understanding of software architecture and domain of application.

As the number and types of available services increase, the need for writing new services and the need for programmers will drop. On the other hand, as computer applications move into more and more domains, the need for application builders will increase. Using an analogy example, service providers are hotel and airline proprietaries, while the application builders are tourism plans vendors (architects) who use the hotels and airlines to build millions of different plans-based. We do not need many people who can design different types of hotels and airlines, but we need many architects to build different plans.

Service-Oriented Computing (SOC) utilizes services as the constructers to support the development of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be used in a self-sufficient way. Services can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable and evolvable systems. Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships between possibly multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service. Services reflect a ”service-oriented” approach to programming, based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to

accomplish some task [94]. Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled.

In summary, some of the key aspects of service-orientation are:

- Loose coupling: Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.
- Service contract: service adhere to a communication agreement, as defined collectively by one or more service descriptions and related documents.
- Autonomy: Services have control over the logic they encapsulate.
- Abstraction: Beyond what is decribed in the service contract, services hide logic from the outside world.
- Reusability: Logic is divided into services with the intention of promoting reuse.
- Composability: Collections of services can be coordinated and assembled to form composite services.
- Statelessness: Services minimize retaining information specific to an activity.
- Discoverability: Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

The service composition encompasses necessary roles and functionalities for the aggregation of multiple services into a single composition. Resulting composite services may be used by process as basic services in further service compositions or may be offered as complete applications/solutions to clients. Service aggregators accomplish this task. Service aggregators thus become service providers by publishing the service descriptions of the composite service they create. Service aggregators develop specifications and/or code that permit the composite service to perform functions that are based on features such as meta-data descriptions, standard terminology and reference models and service conformance. Service aggregators perform service coordination to control the execution of the composite services (e.g. processes), services transactions and manage both the dataflow as well as the control flow between composite services. They also enforce policies on aggregate service invocations.

Currently, there are competing initiatives for developing business process definition specifications, which aim to define and manage business process activities and business interaction protocols comprising collaborating services. The terms behavioral interface, orchestration and choreography have been widely used to describe business interaction protocols comprising collaborating services [17].

As we are focused into service matching, in the remainder of this section we concentrate on several viewpoints from which behavioral models for service composition can be captured, and the relations between these viewpoints.

2.1.1 Service Choreography

A choreography model describes a collaboration between a collection of services to achieve a common goal. It captures the interactions in which the participating services engage to achieve this goal and the

dependencies between these interactions, including: causal and/or control-flow dependencies (i.e.. that a given interaction must occur before another one, or that an interaction causes another one), exclusion dependencies (that a given interaction excludes or replaces another one), data-flow dependencies, interaction correlation, time constraints, transactional dependencies, etc.

A choreography does not describe any internal action of a participating service that does not directly result in an externally visible effect, such as an internal computation or data transformation. A choreography captures interactions from a global perspective meaning that all participating services are treated equally. In other words, a choreography encompasses all interactions between the participating services that are relevant with respect to the choreographys goal. Web Service Choreography is more formally described by the W3C Web Services Choreography Working group as; *"...the external observable behaviour across multiple clients (which are generally Web Services but not exclusively so) in which external observable behaviour is defined as the presence or absence of messages that are exchanged between a Web Service and its clients"* [67].

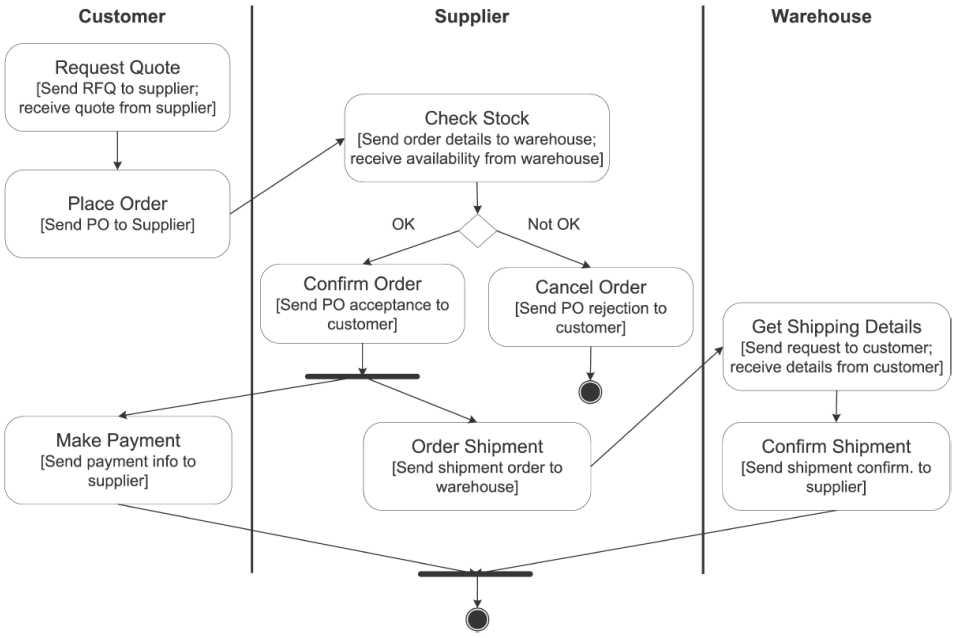

Figure 2.1: Choreography scenario

A choreography of a well known service interaction scenario is shown in the form of an UML activity diagram in Figure 2.1. Three services are involved in this choreography: one representing a *"customer"*, another one a *"supplier"* and a third one a *"warehouse"*. The elementary actions in the diagram represent business activities that result in messages being sent or received. For example, the action *"order goods"* undertaken by the customer results in a message being sent to the supplier (this is described as a textual note below the name of the action). Of course, every message sending action has a corresponding message receipt action but to avoid cluttering the diagram, only the sending or the receipt action (not both) are shown for each message exchange. For example, the action *"send RFQ to Supplier"* in activity *"Request Quote"* implies that there is a corresponding action *"receive RFQ from Customer"* on the Suppliers side,

but this latter action is not shown in the diagram.

Note that Figure 2.1 does not include the activities and alternative paths required to deal with errors and exceptions that one could realistically expect in the scenario in question. Including this information would add considerably to the complexity of the model.

Implementations for choreography standards are currently in the form of the Web Service Choreography Description Language (WS-CDL) [68] and the Web Service Choreography Interface (WSCI) [13]. These specifications have been introduced as part of a service-oriented model aligned with the same W3C working groups.

2.1.2 Behavioral Interface

A Behavioral interface captures the behavioral aspects of the interactions in which one particular service can engage to achieve a goal. It complements structural interface descriptions such as those supported by WSDL, which capture the elementary interactions in which a service can engage, and the types of messages and the policies under which these messages are exchanged.

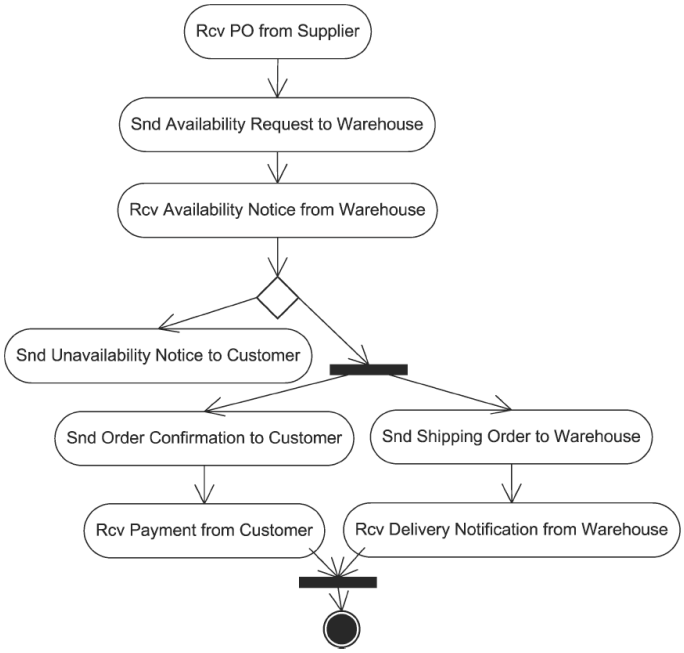

Figure 2.2: Supplier behavioral interface

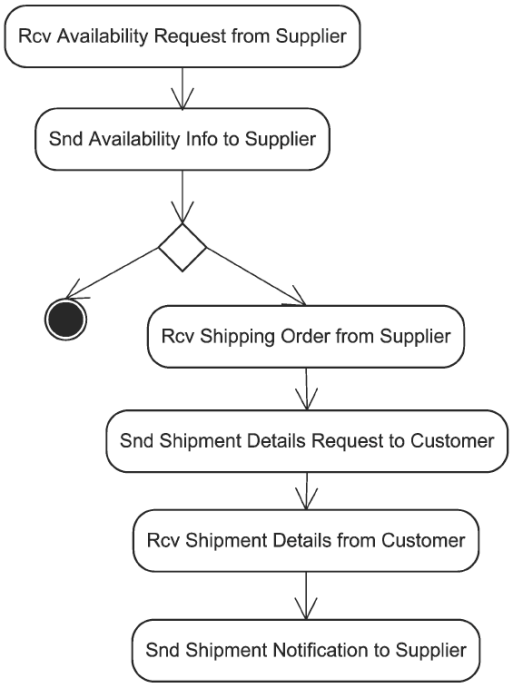

Figure 2.3: Warehouse behavioral interface

A behavioral interface captures dependencies between elementary interactions such as control-flow dependencies (e.g. that a given interaction must precede another one), data-flow dependencies, time constraints, message correlations, transactional dependencies, etc. It focuses on the perspective of one single party. As a result, a behavioral interface does not capture ”complete interactions” since interactions necessarily involve two parties. Instead, a behavioral interface captures interactions from the perspective of one of the participants, and can therefore be seen as consisting of communication actions performed by that

participant. Also, behavioral interfaces do not describe internal tasks such as internal data transformations.

Figures 2.2, 2.3 and 2.4 show examples of behavioral interfaces corresponding to the supplier, warehouse and customer roles in the choreography of Figure 2.1.

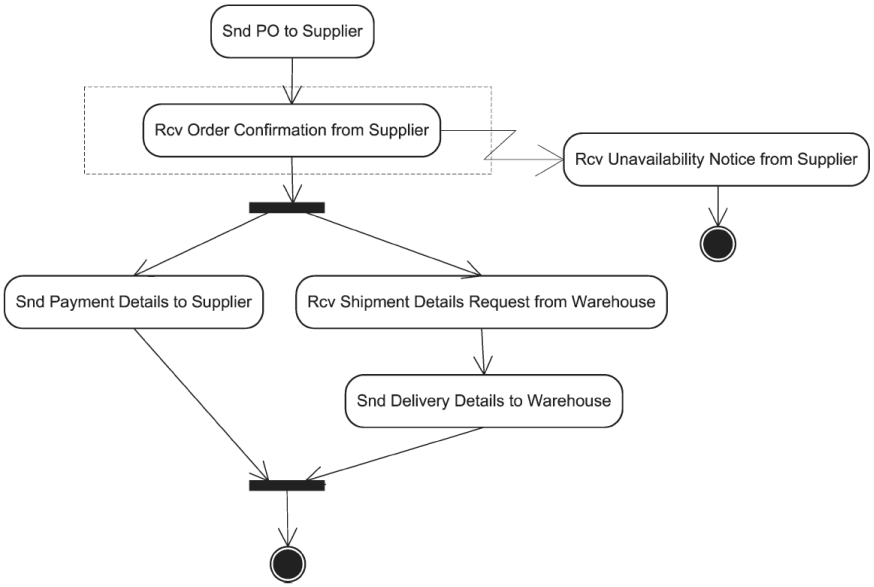

Figure 2.4: Customer behavioral interface

Note that a role defined in a choreography may be associated with multiple behaviours and multiple WSDL interfaces. Moreover, for a given role in a choreography, an arbitrary number of behavioral interfaces may be defined providing the same functionality but not necessarily using the same interactions or the same order of interactions. For example, in Figure 2.3 the shipping order is sent to the warehouse in a parallel thread to the one where the payment details are received from the customer. An alternative would be that payment is received from the customer before the shipping order is sent out.

Depending on whether an interface captures an ”as is” or a ”to be” situation, a distinction can be made between provided and expected (or required) interfaces. A provided (behavioral) interface is an abstraction of the way a given service interacts with the external world. On the other hand, an expected (behavioral) interface captures an expectation of how a service should behave in order to play a given role in a choreography. Thus, an expected interface corresponds to a contract that a given party needs to fulfill to successfully collaborate with other parties. Ideally, the provided and expected interfaces of a service coincide. In practice however, it may happen that the interface provided by a service is different from the interface that it is expected to provide in a given scenario. In this case, the provider of the service is responsible for mediating between the interface that it is expected to provide and the one that it actually implements. This mediation (or adaptation) process has been the subject of several research efforts [105, **?**].

Implementations for behavioral interface are actually in the form of the Web Services Conversation Language (WSCL) [16] wich proposes a simple conversation language standard that can be used for various Web-service protocols and frameworks. It focuses on modeling the sequencing of the interactions

or operations of one interface. It fills the gap between mere interface definition languages that do not specify any choreography and more complex process or flow languages that describe complex global multi-party conversations and processes.

2.1.3 Service Orchestration

Orchestration describes how services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint.

An orchestration model describes both the communication actions and the internal actions in which a service engages. Internal actions include data transformations and invocations to internal software modules. An orchestration may also contain communication actions or dependencies between communication actions that do not appear in any of the services behavioral interface(s). This is because behavioral interfaces may be made available to external parties and thus, they only need to show information that actually needs to be visible to these parties. Orchestration refers to an executable business process that may result in a long-lived, transactional, multi-step process model. With orchestration, the business process interactions are always controlled from the (private) perspective of one of the business parties involved in the process.

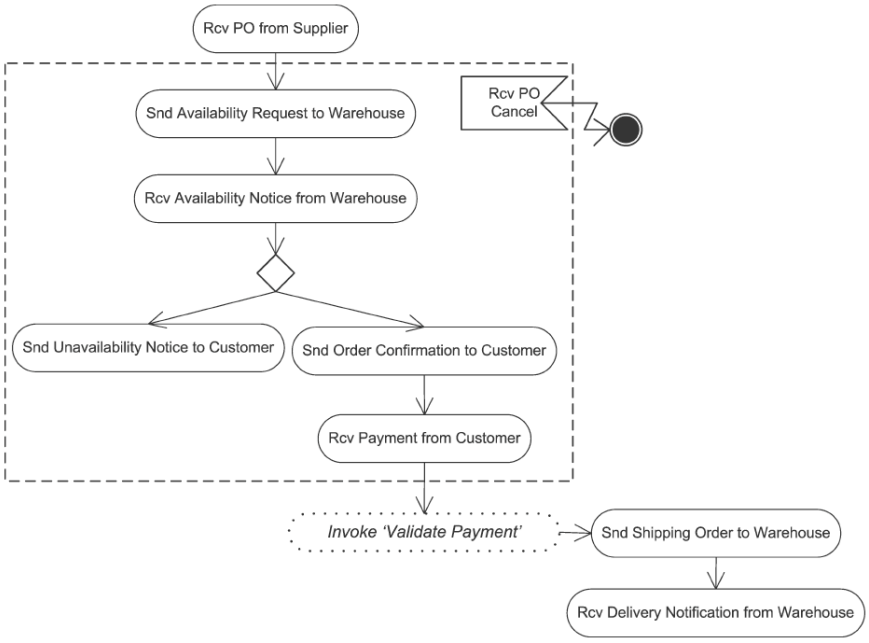

Figure 2.5: Supplier service orchestration

Figure 2.5 shows an orchestration of a supplier service. This orchestration includes an internal action for validating the payment, shown in dotted lines in the diagram. This may correspond for example to an interaction with a service that is not exposed to the outside world. Other internal actions may be included in this orchestration. The orchestration of Figure 2.5 also supports the possibility of an order

cancellation request being received from the customer anytime before the payment, leading to termination of the process.

Orchestration is targeted by a family of XML-based process standard definition languages most representative of which is the Business Process Execution Language for Web Services (WS-BPEL)[l1]. However, BPEL is intended to cover both the orchestration and the behavioral interface viewpoints. Exactly, the abstract process part of BPEL can be used for describing the behavioral interface viepoints and the execution part can be used for describing the orchestration.

2.1.4 Assessment of Service Composition Techniques

Table 2.1 summarizes the service composition viewpoints presented in this sub-section. The symbol (+) means that the property on that column is supported by the viewpoint. The symbol (-) means that the property is not supported. Finally, the sign (+/-) means that the property is moderately supported.

Assessment Parameters	**Composition Viewpoints**		
	Service Choreography	Behavoiral Interface	Service Orchestration
Process is always controlled by one party	-	+	+
The services follow a pre-defined plan	+	-	-
Each service is self-sufficient of the others	+	-	-
Inclusion of internal and external webservices	-	+/-	+
The involved Web Services do not know that they are implicated into a composition	-	+	+
Definition of the public message exchanged between the Web Services	+	+/-	-
Message exchanged must contain all state information needed to evaluate next action	+	-	-
Tracking of the sequence of messages involving multiple parties and sources	+	-	-
Each participant responsible for adaptive behavior for anomaly response	-	+/-	+
Compensation of anomalies in real-time	-	-	+
Description of process flow	-	+	+

Table 2.1: Assessment of composition viewpoints

Orchestration differs from choreography in that it expresses a body of business process logic that is typically owned by a single organization. An orchestration establishes a business protocol that formally defines a business process definition. The workflow logic within an orchestration is broken down into a series of basic and structured activities that can be organized into sequences and flows. More collaborative in nature, choreography tracks the sequence of messages involving multiple parties, where no party truly ”owns” the conversation. It is associated with the public message exchanges that occur between multiple Web services participants that can assume different roles and that have different relationships. If we have an analogy with the urban trafic, the Orchestration is akin to traffic lights where events are controlled centrally, whereas Choreography is more like a roundabout, where each participant is following a prearranged set of rules.

A behavioral interface express the behavior of a particular service provider or service user in its communication with another service provider or user to achieve a particular goal. Since a behavioral interface only describes the behavior of a single service provider, it only comprises one role into the services conversation. A Behavioral Interface can be used as a starting point to generate an ”orchestration” skeleton that can then be filled up with details regarding internal tasks and refined into a full orchestration. This has the advantage that once the orchestration is fully refined, the provided behavioral interface of the service will coincide with the expected behavioral interface. On the other hand, an existing orchestration can be

used to generate the provided behavioral interface of a service by appropriately hiding actions not to be exposed.

As this dissertation is motivated by the idea that an inter-organizational process can be considered as a cooperation of various pre-established processes of several organizations, we focus our efforts on matchmaking of behavioral interfaces, considering that this one captures interactions from the perspective of one of the organizations.

2.2 Formal Models for Representing Processes

This section presents four business process modeling formalisms. These formalisms offer representation features in order to describe, exchange and execute business processes.

2.2.1 Process Algebra Models

Processes can be modeled using process algebras. Examples of process algebra models are calculus of communicating systems (CCS) [87], communicating sequential processes (CSP) [63] and p-calculus [88]. These languages provide algebras for specifying and reasoning about concurrent systems or processes. They provide sets of terms, operators and axioms for writing and manipulating algebraic expressions. The behavior of the systems being modeled can be analyzed based on the defined operators. The pi-calculus is a mathematical model of processes whose interconnections change as they interact [23]. The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes π-calculus particularly suitable for representing where accessible resources vary over time. The π-calculus, in addition to modeling concurrent systems, can also express mobile processes and techniques for analyzing their behavior. Some of the operations supported by these languages are simulation and bisimulation of process instances. For example, given two process descriptions in π-calculus notation, it is possible to check for their equivalence using bisimulation operation.

In [77] the authors argue that BPEL is not equipped with formal semantics (as other existing proposals) and since it includes a large number of aspects, it is difficult to formally reasoning on processes behavior. In light of this observation, the semantics of a BPEL fragment is formally addressed in [77]. The proposal represents a significant contribution in two directions. Firstly, formalizes a novel orchestration language, $web\pi_\infty$ which represents by itself a simplification of WS-BPEL including an unambiguous specification, thus making possible to formally reason on orchestration processes. Secondly, an implementor of an actual WS-BPEL orchestration engine could implement simply this single mechanism providing all the remaining ones by compilation. The language is composed of a small set of operators which can meet the BPEL behaviours, offering a reasonable simplicity to the application designers. This langage includes: a parallel operator allowing explicit concurrency; a restriction operator allowing compositionality and explicit resource creation; a recursion or a process definition operator; a sequence operator allowing causal relationship between activities; an inaction operator which is just a ground term for inductive definition on sequencing; message passing and especially name passing operators allowing communication and link mobility.

2.2.2 Petri Net Models

A Petri Net is a formal and graphical language for modeling systems or processes [99]. It comprises places, transitions, arcs and tokens. Places represent states of a Petri Net and they can contain tokens. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. The current state of the modeled system, called the marking, is given by the number of tokens in each place. The system marking changes when transitions fire, meaning tokens are removed from input places and inserted to output places of a transition. Transitions can only fire if they are enabled, meaning, there are tokens ready to fire in the input places.

Petri Nets provide a tool for describing systems that are characterized as being concurrent, asynchronous, distributed and nondeterministic. In graphical form, Petri Nets can be used as a visual communication aid in a similar way to that the structured design notations from traditional systems analysis and design methodologies. The language of Petri Nets however, provides a solid mathematical basis for the description and analysis of equations of state, algebraic and other mathematical models. This yields a practical notation for describing the behaviour of system processes, such as that given for a simplified alternating bit [92].

A related formalism to Petri Nets is the Workflow Net (WF-Net) [117]. WF-Nets have been used to model interorganizational workflows in [116]. Like Petri Nets, WF-Nets are token-based and contain places and transitions, but in addition, they contain a single initial and final place. They have better computational properties than Petri Nets, but are used in asynchronous communication models were messages may arrive in a different order to that in which they were sent.

The authors of [62] propose a Petri Net-based algebra for composing Web services. The formal semantics of the composition operators is expressed in terms of Petri Nets by providing a direct mapping from each operator to a Petri Net construction. Thus, any service expressed using the algebra constructs can be translated into a Petri Net representation. By means of a set of algebra properties, the approach is able to transform and optimize Web service expressions guaranteeing the same semantics of initial expressions. In addition, the use of a formal model allows the verification of properties and the detection of inconsistencies both within and between services.

2.2.3 Finite State Automata (FSA)

An FSA is defined by a finite set of messages, states, a set of transitions, a initial state and a set of final or accepting states [64]. It can be represented as a graph with a single initial state, where nodes represent states, arcs represent transitions connecting two states. Transitions are labeled with messages drawn from the message set. FSA graphs are traversed from the initial state. Final states are specifically marked with concentric cycles and they represent the acceptance of a message sequence by the FSA. FSAs are closed under intersection and have polynomial time algorithms for emptiness test and intersection. However FSAs in their original form cannot represent mandatory semantics of message sequences. Therefore FSAs do not have parallel execution semantics, as provided by more expressive approaches like Petri Nets.

In [22] the authors describe a conceptual model for representing e-Service behaviour and temporal constraints, based on FSAs. FSAs allows to capture a large class of e-Services and to formally verify important properties of e-Services [122], such as correctness, safety (i.e., at each point in the execution of an e-Service certain logical invariants hold), liveness (i.e., an e-Service is ensured to move towards a point where a goal can be reached). Additionally, this approach introduces a new language, WSTL

(WEB SERVICE TRANSITION LANGUAGE), that relies on such a conceptual model. The novelty of this approach is that WSTL provides constructs to which corresponding elements of FSAs can be straightforwardly mapped.

2.2.4 Graph Representation

Graphs are a general and powerful data structure for the representation of objects and concepts. In a graph representation, the nodes typically represent objects or parts of objects, while the edges describe relations between objects or object parts. Graphs have some interesting invariance properties. For instance, if a graph, which is drawn on paper, is translated, rotated, or transformed into its mirror image, it is still the same graph in the mathematical sense. These invariance properties, as well as the fact that graphs are wellsuited to model objects in terms of parts and their relations, make them very attractive for various applications [30]. In applications such as pattern recognition and computer vision, object similarity is an important issue. Given a database of known objects and a query, the task is to retrieve one or several objects from the database that are similar to the query. If graphs are used for object representation this problem turns into determining the similarity of graphs, which is generally referred to as graph matching.

In [59] the authors present a possible metamodel based on conceptual graphs to represent processes that fulfills corporate memory requirements (Acquisition, storage, evolution and dissemination of knowledge acquired by the organization). The metamodel is composed of three basic concepts: ACTIVITY, PROCESS and EVENT. An activity is defined by its inputs and outputs, the agents that enable the activity and by pre and post conditions. Preconditions define conditions or states that must be verified to fire the execution of the activity; postconditions define states or conditions that will result from the execution of the activity. An event is a point in time that marks the end of an activity; it marks the realization of the postcondition of the activity. A process is defined as a set of events that represent the execution of a set of activities.

In [80], the authors have introduced a transformation from BPEL to EPCs (Event-Driven Process Chains). A flat EPC Schema is defined as a directed and coherent graph with cardinality and type constraints. Built on a conceptual mapping, this approach presents a transformation program that is able to generate EPC models as EPML files (XML-based interchange format for EPCs) from BPEL process definitions automatically. Such a transformation helps to communicate BPEL processes to business analysts that are often involved in the approval of business logic. The EPC visualization focuses on the dynamic behavior of the BPEL model. BPEL constructs (i.e. basic and structured activities) are transformed to blocks of EPC elements that offer equivalent semantics. EPC elements get names that are generated from the names of the corresponding BPEL elements. Furthermore, the program can be used for re-engineering of BPEL processes. Finally, the transformation concept is general in such a way that it can be easily adapted to generate output of another graph-based process language that is encoded in XML.

2.2.5 Assessment of Formal Models

Table 2.2 summarizes the formal representations of processes explained in this sub-section. The symbol (+) means that the parameter on that column is supported by the formal model. The symbol (-) means that the property is not supported. The sign (+/-) means that the property is moderately supported. Finally, the symbol N/A means not applicable, meaning the property does not apply to the model with which it has been associated.

The aim of this comparison is to provide criteria for modelling web service composition (including state of resources etc) through a notation. We consider completeness (a complete set of semantics and maturity of notation), composability (a property that enables reasoning about a composed system on the basis of its constituent parts without any additional need for information about the implementation of those parts), parallelism (a key aspect of formalism which must be fulfilled for accurately modelling web service composition) and the complexity of performing matching (polynomial matching).

Formal representation	**Assessment parameters**			
	Completeness	Composability	Parallelism	Polynomial matching
Process algebra models	+	+	+	N/A
Petri Net models	+	-	+	+
Finite state automata	+	+	+/-	+
Graph representation	+	+	+	+/-

Table 2.2: Assessment of formal models

As this dissertation is focused on matchmaking of behavioral interfaces, and considering that this one captures interactions from the perspective of one of the organizations, we therefore believe that Graph representation model provides a simple and maturity notation for expressing the service's semantics. Moreover the graph representation enables reasoning about a composed system on the basis of its constituents parts without any additional need for information about the implementation of those parts. Furthermore, the parallelism supported by the graph model is a key aspect of formalism which must be fulfilled for accurately modelling web service composition. Despite graph matching is a complex process, techniques can be applied in order to increase its performance (see [40, 124, 27]). Finally, the graph model is used by several approaches (see [80, 96, 59]) as formal representation for business process.

2.3 Service Matchmaking Techniques

In this section we explain the techniques for service matchmaking which are based on process modeling formalisms. As our approach of service matchmaking is founded on graph representation, first we introduce the graph matching algorithms. Next, a state of the art of different service matchmaking techniques is presented. More precisely, we show a classification of the different techniques of service matchmaking. An assessment of existing techniques is given at the end of the section.

2.3.1 Graph Matching Algorithms

Graph matching has become a very active field of research [9]. A wide spectrum of graph matching algorithms with different characteristics have become available meanwhile [30]. In its most general form, graph matching refers to the problem of fiding a mapping f from the nodes of one given graph g_1 to the nodes of another given graph g_2, that satisfy some constraints or optimality criteria. For example, in graph isomorphism detection, mapping f is a bijection that preserves all edges and labels. In subgraph isomorphism detection, mapping f is requested to be injective such that all edges of g_1 are included in g_2 and all labels are preserved. Other graph matching problems that require the constructions of a mapping f with particular properties are maximum common subgraph detection and graph edit distance computation. The standard algorithm for graph and subgraph isomorphism detection is the one by Ullman [114]. In

this approach, a simple tree-search algorithm based on refinement procedure is introduced, which attains efficiency by inferentially eliminating successor nodes in the tree-search. Further, maximum common subgraph detection has been addressed in [78, 72, 97] and several methods for error-tolerant graph matching have been presented in the literature. In this section we present an overview of the principal graph matching approaches.

Classical methods for error-tolerant graph matching can be found in [54, 104, 106, 113, 126]. Most of these algorithms are particular versions of the A* search procedure, (i.e., they rely, at some extent, to tree search incorporating various heuristic lookahead techniques in order to prune the search space).

Additionally, error-correcting graph matching is a powerful concept that has various applications in pattern recognition and machine vision, and its application is focused on distorted inputs. It constitutes a new approach very similar to other graph matching techniques. In [36] this topic is addressed and a new distance measure on graphs that does not require any particular edit operations is proposed. This measure is based on the maximal common subgraph of two graphs. A general formulation for error-correcting subgraph isomorphism algorithms is presented in [75] in terms of adjacency graphs, and [29] presents a study on the influence of the definition of fitness functions for error correcting graph matching which reveals guidelines for defining fitness functions for optimization algorithms in error correcting graph matching. In addition, in [83] an algorithm for error-correcting subgraph isomorphism detection from a set of model graphs to an unknown input graph is introduced.

In approximate, or error-correcting, graph matching one considers a set of graph edit operations, and defines the edit distance of two graphs g_1 and g_2 as the shortest (or least cost) sequence of edit operations that transform g_1 into g_2. A maximum common subgraph of two graphs g_1 and g_2 is a subgraph of both g_1 and g_2 such that there is no other subgraph of g_1 and g_2 with more nodes. Graph edit distance and maximum common subgraph are well known concepts that have various applications in pattern recognition and machine vision. In [28] a particular cost function for graph edit distance is introduced, and it is shown that under this cost function graph edit distance computation is equivalent to the maximum common subgraph problem.

In [56] the authors explain the relationship between two important problems in pattern recognition using attributed relational graphs, the maximum common subgraph and the minimum common supergraph of two graphs. This relation is established by means of simple constructions, which allow to obtain the maximum common subgraph from the minimum common supergraph, and vice versa. On this basis, a new graph distance metric is proposed for measuring similarities between objects represented by attributed relational graphs. The proposed metric can be computed by a straightforward extension of any algorithm that implements error-correcting graph matching, when run under an appropriate cost function, and the extension only takes time linear in the size of the graphs.

Conceptual graphs have been used to model knowledge representations since their introduction in the early 80s. The formalism of conceptual graphs introduced in [110] is a flexible and consistent knowledge representation with a well-defined theoretical basis. Moreover, simple conceptual graphs are considered as the kernel of most knowledge representation formalisms built upon Sowas model. This formalism can capture semantics in the representation of data, and it offers some useful constructs which makes it a likely platform for a knowledge-based system. An extension of this concept to graph matching is introduced in [15], where reasoning in Sowas model can be expressed by a graph homomorphism called projection. This paper presents a family of extensions of this mode, based on rules and constraints, keeping graph homomorphism as the basic operation. Apart from this type of graphs, graph matching has also been proposed, from both a theoretical or a practical view point, in combination with: matching graphs [53],

minimal condition subgraphs [58], finite graphs [14], weighted mean of a pair of graphs [33], median graphs [66] and decomposition approaches [85]. Furthermore, different ways of representing patterns are analyzed in terms of symbolic data structures such as strings, trees and graphs in [31].

Majority of methods presented in this section are guaranteed to find the optimal solution but require exponential time and space due to the NP completeness of the problem. Suboptimal, or approximative methods, on the other hand, are polynomially bounded in the number of computation steps but may fail to find the optimal solution. For example, in [40, 124, 27], probabilistic relaxation schemes are described. Other approaches are based on neural networks such as the Hopfield network [55] or the Kohonen map [128] and the approach presented in [101].

The fact of formulating complex graph matching problems as combinatorial optimization ones is not novel, and many references applying different techniques in this field can be found in the literature. Genetic algorithms are just an example of this [48, 121, 89]. However, all of these approximate methods may get tracked in local minima and miss the optimal solution. Approaches to the weighted graph matching problem using eigenvalues and linear programming, have been proposed in [115] and [10], respectively. As a special case, the matching of trees has been addressed in a series of papers recently [37, 90, 98, 119].

Decision trees have also been applied to graph matching. An example of this is [107], in which decision trees are used for solving the largest common subgraph problem instead of applying queries to a database of models. Another example can be found in [82] where decision trees are applied as a fast algorithm for the computation of error-correcting graph isomorphisms. Decision trees are also been applied to multiple graph matching [84]. The decision tree is created using a set of a priori known model graphs generated from exact subgraph isomorphism detection.

We can see that the correspondences between graphs can be established by a variety of graph relation paradigms. Popular paradigms found today include graph and subgraph isomorphism detection for exact matching approaches. However, in our approach of service matchmaking, the users are interested in finding suitable service. Then, the user formulates a requirement as a process model; his goal is to use this model as a query to retrieve a component whose respective process model matchs with a whole or part of this query. Hence, it is necessary to incorporate the concepts of error correction and inexact matching into service matchmaking.

The term approximate matching means that it is not possible to find an isomorphism between the two services to be matched. This is the case when the number of activities is different in both the query and target service. This may be due to the schematic aspect of the service and the difficulty to segment accurately the service into meaningful entities. Therefore, in these cases no isomorphism can be expected between both services, and the service matching problem does not consist in searching for the exact way of matching activities of a service with activites of the other, but in finding the best matching between them. In that case, the matching aims at finding a non-bijective correspondence between a query service and a target service.

The error correction is the definition of the errors that are to be taken into account into the service matchmaking process. Probably the best known error correction model for matching processes is similar to the model used in string edit distance. It is based on the idea of introducing edit operations. For each possible error type a corresponding list of edit operation is defined. In order to model the fact that certain error types are more likely than others, cost functions are assigned to the edit operations.

Based on the considerations presented above, we therefore believe that the error-correcting subgraph isomorphism detection is the matching technique which is more accurate to our service mathcmaking problem. In the chapter 3 we explain in detail this matching technique.

2.3.2 Service Matchmaking

This section describes the techniques used for service matchmaking. We have clustered the existing techniques in three set: Service matchmaking based on interfaces, semantics and behavior.

Service Matchmaking based on Interfaces. Currently, the algorithms for Web services discovery in registers like UDDI or ebXML are based on a search by key words or tables of correspondence of couples (key-value). To support a more precise service discovery process, mechanisms based on interface matchmaking were proposed (i.e.,[111, 70]). Such approaches of web service matching tend to address syntactic and/or semantic matching. To analyze their merits, it is useful to further classify them as uniform or hybrid. Uniform matching approaches refer to atomic matching techniques that can not be any further decomposed in finer-grained matching techniques. Hybrid matching approaches on the other hand may combine various matching methods (e.g., syntactic and semantic) into a composite algorithm. In [111] the authors discuss a set of complementary methods for assessing the similarity of interface specifications (WSDL), allowing to order the potentially useful services according to their relevance to the developer's query. To assess the similarity between two WSDL specifications, these methods utilize, on one hand, the semantics of the identifiers and of the natural-language descriptions of WSDL specifications, and on the other hand, the structure of their operations, messages and types. More precisely, the authors describe a suite of web-service discovery methods that combine traditional information retrieval and two WordNet-based techniques with a structure-matching algorithm leveraging the structure of the XML-based service specification in WSDL.

In [70] the authors presents the *WSDL-M2* algorithm wich combines two techniques: lexical matching to calculate the linguistic similarity between concept descriptions, and structural matching to evaluate the overall similarity between composite concepts. The overall matching process is implemented in three steps: First, all files from the collection of WSDL specifications are parsed in order to allow extraction of their structured content. In the second step, the parsed document is tagged to enable lexical analysis. In the third step, the tagged WSDL specifications can be further analyzed and subsequently indexed using diferent information retrieval models (VSM: Vector-Space Model and $tf - idf$ measure).To address the major shortcoming of VSM a combination VSM and WordNet is proposed. Finally, the structural matching is treated as Maximum Weight Bipartite Matching to calculate semantic similarity between concept descriptions and to compute similarity of complex WSDL concepts taking into account their constituents (sub-types).

On another front, recently there has been a proliferation of Web service search engines on the Internet. These can be clustered into two types. The first type accepts as input, keywords, which they use to search within WSDL descriptions of services. Bindingpoint, NET XML Web Services Repertory, WebserviceX.NET, Web Service List and SalCental (see [103]). The second type of Web service search engines goes beyond naive keyword matching of WSDL contents by performing a similarity search on WSDL operations of Web services, considering operation name and input/output parameters. An example that uses such a technique to find matching Web services is Woogle (see [51]). In [51] the search engine combines multiple sources of evidence to detect similarity: textual descriptions of the operations and of the entire web services and similarity between the parameter names of the operations. The underlying algorithm is based on a technique that clusters parameter names in the collection of web services into semantically meaningful concepts. These concepts are used in the comparison of input or output parameters.

In summary, the approaches used by Web service search engines can only match simple services, thus

do not handle the execution aspects of services.

Service Matchmaking based on Semantics. Within the framework of the semantic Web, description logics were proposed for a richer and precise formal description of services. These languages allow the definition of ontologies, such as for example OWL-S. The OWL-S [49] proposed an ontology for describing Web services based on the Web Ontology Language (OWL) [50]. OWL-S is structured into three types of knowledge: service profiles, service model and service grounding. Service profiles describe the capability of a Web service. The service model describes services in terms of inputs, outputs, preconditions and effects of invoking a service; processes in OWL-S are described in terms of their states, including information such as initial activation, execution and completion. Service grounding describes how to access the service.

By describing service capabilities using OWL-S, it is possible to find matching Web services from a semantic perspective. Several Web service matchmaking prototypes have been implemented using this approach, for example [39, 74, 93, 24, 21, 112]. Work related to this can be found in [95, 109], where approaches for annotating Web services with semantic information and using this for service discovery are described.

In [93, 21], a published service is matched with a required service when the inputs and outputs of the required service match the inputs and outputs of the published service (i.e., they have the same type or one is a generalization of the other). In [69], independent filters are defined for service retrieval: the name space, textual description, the domain of ontology that is used, types of inputs/outputs and constraints. The approach presented in [38] takes into account the operational properties like execution time, cost and reliability. The authors of [127] provide a lightweight semantic comparison of interfaces based on similarity assessment methods (lexical, attribute, interface and QoS similarity).

The semantic-based approaches mentioned above are very efficient for matching simple services based on semantic descriptions of their capabilities. Besides, is not clear how these approaches can match the complex business processes, which consider the structural information of the processes included into the services.

Service Matchmaking based on Behavior. Service retrieval based on key words or some semantic attributes is not satisfactory for a great number of applications. The tendency of recent work is to exploit more and more knowledge on service components and behavior. The need to take into account the behavior of the service described by a process model was underlined by several researchers [112, 102, 24, 100, 125]. In [24], in order to improve precision of web service discovery, the process model is used to capture the salient behavior of a service. A query language for services is defined which allows to find services by specifying conditions on the activities which compose them, the exceptions treated, the flow of the data between the activities.

Recently, authors in the academic world have published papers that discuss similarity and compatibility at different levels of abstractions of a service description (e.g., [19, 25, 51, 125]). In terms of protocol specification and analysis, existing approaches provide models (e.g., based on pi-calculus, petri nets or state machines) and mechanisms to compare specifications (e.g., protocols compatibility checking).

In [91], the authors have presented an approach that enables dynamic binding for BPEL process (dynamic binding of WSs to WS-flow instances at run time, i.e. the ability to exchange a WS instance participating in a WS-flow instance with an alternative one) that is done in three steps. First, providing

a high-level description of process, second, abstracting the process behavior using symbolic observation graphs (SOG) using workflow nets (Wf-nets) which is a specific form of Petri Nets and finally providing an algorithm for SOG matching used for binding dynamically business processes.

In [71], a pi-calculus representation is used for formalizing a service and query behavior. Specifically, single operations involving message exchanges are expressed in pi-calculus, differentiated by four transmission types, namely one-way, notification, request-response and solicit-response; Also, the constraints between operations of a service or a query that define the allowed order of execution are expressed in pi-calculus. After expressing the service and query behavior using pi-calculus, service matchmaking between a service query and a service description is reasoned through the capability of pi-calculus.

In [125], authors give a formal semantics to business process matchmaking based on finite state automata extended by logical expressions associated to states. Computing the intersection is computationally expensive and thus does not scale for large service repositories. To solve this problem, the authors of [76] present an indexing approach for querying cyclic business processes using traditional database systems.

The choice of finite state automata as a modelling formalism limits the expressiveness of the models, for instance representing parallel execution capabilities can lead to very large models.

A new behavior model for web services is presented in [108] which associates messages exchanged between participants with activities performed within the service. Activity profiles are described using OWL-S (Web Services Ontology Language). Web services are modelled like non-deterministic finite automatons and a new query language is developed for expressing temporal and semantic properties on service behaviors.

To summarize, the need to take into account the service behavior in the retrieval process was underlined by several authors and some recent proposals exist ([108],[76]). The few approaches that exist give a negative answer to the user if a model satisfying exactly his requirements does not exist in the registries, even if a model that requires a small modification exists. Moreover, they assume that services have a common semantics on message names. We do not make this assumption; as companies model differently their services, we try to deal with the heterogeneity of message names and message sequencing. Our objective is to propose an approach for service retrieval based on behavioral specification allowing an approximate match. To the best of our knowledge, there is not another approach allowing to retrieve services having similar behavior and defining a behavior-based similarity measure.

2.3.3 Assessment of the Service Matchmaking Techniques

The table 2.3 summarizes the Service matchmaking techniques presented in this sub-section. The symbol (+) means that the property on that column is supported by the matchmaking technique. The symbol (-) means that the property is not supported. Finally, the sign (+/-) means that the property is moderately supported.

The evaluated parameters for each service technique are: Stateful, Stateless and Semantics. Stateful and stateless are properties that describe whether a computer or computer program is designed to note and remember one or more preceding events in a given sequence of interactions with a user, another computer or program, a device, or other outside element. Stateful means the computer or program keeps track of the state of interaction, usually by setting values in a storage field designated for that purpose. Stateless means there is no record of previous interactions and each interaction request has to be handled based entirely on information that comes with it. Stateful and stateless are derived from the usage of state as a set of conditions at a moment in time.

Assessment Parameters	**Service matchmaking techniques**		
	Based on Interfaces	Based on Semantics	Based on Behavior
Stateless	+	+	+
Stateful	-	-	+
Semantics	+/-	+	+/-

Table 2.3: Assessment of the service matchmaking techniques

In the table 2.3 we can see that service matchmaking tecniques based on interfaces, supports stateless but not stateful services and fairly supports the description of semantic services; Because of search engines can only match simple services, thus do not handle the process aspects of services. Finally, the semantic descriptions are fairly supported, as any search engines like Woogle (see [51]) has a semantics evaluation level.

The Service matchmaking technique based on semantics supports stateless services and the semantics descriptions. Besides, such technique does not support stateful services, due to that it is very efficient at finding simple services by matching semantic descriptions of their capabilities, but is not clear how to translate this to match complex business processes to check for bilateral or multi-lateral collaborations.

The Service matchmaking technique based on behavior supports fairly the semantic description, but some approaches as [108] inserts sematics to the behavioral matchmaking. On the other hand, such technique supports stateless and stateful services, owing to formal descriptions of the services and the used matching technique take into account the order in which services are executed.

2.4 Summary

In this dissertation we argued that, in many situations, the service discovery process requires a matchmaking phase based on the specification of the component behavior. Therefore, the related works to this PhD thesis were clustered into three main categories:

(i) As we were focused into service matchmaking, in this category we concentrated on several viewpoints from which behavioral models for service composition can be captured. More precisely, we presented the following view points: (a) Service Choreography, (b) Behavioral interface and (c) Orchestration. Finally we presented an evaluation of these view points. As this dissertation is motivated by the idea that an inter-organizational process can be considered as a cooperation of various pre-established processes of several organizations, we focused our efforts on matchmaking of behavioral interfaces, considering that this one captures interactions from the perspective of one of the organizations.

(ii) Since our approach develops matching techniques that operate on behavioral models of the services, in this category we depicted formal representations that allows to describe, exchange and execute service behaviors. The formal representations presented were: Process algebra, Petrit Nets, Finite State Automata and Graph Representation. Finally, an assessment of these formal representations was maked. The aim of this comparison was to provide criteria for modelling web service composition. We concluded that Graph representation model provides a simple and mature notation for expressing the service's semantics. Moreover the graph representation enables reasoning about

a composed system on the basis of its constituents parts without any additional need for information about the implementation of those parts. Furthermore, the parallelism supported by the graph model is a key aspect of formalism which must be fulfilled for accurately modelling web service composition.

(iii) Finally, we explain the related works with respect to service matchmaking techniques for service discovery. As our approach of service matchmaking is founded on graph representation, first we introduced the graph matching algorithms. Next, a state of the art of different service matchmaking techniques was presented. We clustered the existing techniques in three set: Service matchmaking based on interfaces, semantics and behavior. Finally, an assessment of existing techniques was given at the end of the chapter. The approaches used by Web service interfaces search can only match simple services, thus do not handle the execution aspects of services. The semantic-based approaches are very efficient for matching simple services based on semantic descriptions of their capabilities. Besides, is not clear how these approaches can match the complex business processes, which consider the structural information of the processes included into the services. To the best of our knowledge, there is not another approach allowing to retrieve services having similar behavior and defining a behavior-based similarity measure.

Chapter 3

Service Behavioral Matchmaking

Service discovery is an essential task in the process of developing service-oriented applications. In a typical service-discovery scenario, the service requester has specific expectations about the candidate service. In general, there are three types of desiderata for a service: it has (a) to be *capable* of performing a certain task (i.e., maintain a shopping cart), (b) to expose a particular *interface* (i.e., provide view, addproduct and remove-product) and (c) to *behave* in a certain manner (i.e., ignore any request for product removals if no product additions have been performed yet). Such expectations motivate and guide the developers searches through web-services repositories, as they try to discover and select the service that best matches their needs.

When these dimensions are not considered in concert, the precision of the discovery process tends to be limited. For example, using only the dimension (a) into a matching method between a service and a request does not guarantee that they are interoperable, so the interoperability between two services depends on its structural information and the service availability. Additionally, when the interface of a service matches a requester interface and the parameter types have not a detailed description, it is not clear how exactly the parameter data and operations match the requester specification. Furthermore, in some cases, interface matching may get confused because the operation signatures are not significantly distinct.

In this dissertation we argue that, in many situations, the service discovery process requires a matchmaking phase based on the specification of the component behavior. After retrieving services having equivalent functionalities (using for example, interface matching, ontology, etc.), several applications require to find among these service candidates the one having the most similar behavior in respect with user needs. While the first discovery step has been the subject of excellent research work ([51, 93], etc.), for the second step, there are no solutions allowing to rank the list of candidates in respect with behavior model.

In this chapter we explain our graph-based approach to behavior matchmaking and show an application of this one. First, we introduce the graph matching problem explaining its definition and notation. As this PhD thesis concentrates on approximate service matching, a method to measure the similarity of two graphs is depicted. Then, the error-correcting subgraph matching is presented in detail. Finally, we show an application case of this algorithm to WSCL (Web Services Conversation Language) protocols.

3.1 The Graph Matching Problem

Graphs are powerful and universal tools widely used in information processing. Numerous methods for graph analysis have been developed and become important in computer science and engineering. Many fields such as computer vision, scene analysis, chemistry and molecular biology have applications in which objects have to be retrieved and identified. When this processing is to be performed by a computer automatically without the assistance of a human expert, a useful way of representing the knowledge is by using graphs. In conclusion the graphs have been proved as an effective way of representing real world objects and in this way the problem of object retrieval turns into determining the similarity of graphs, which is generally referred to as graph matching.

In this section we recall a background on graph matching, then we present a method to measure the similarity of two graphs. Next, we explain the error-correcting subgraph matching. Finally, we depict the extensions for expanding the EC-Algorithm (Error correcting algorithm) to the Service Behavioral Matchmaking.

3.1.1 Background of Graph Matching

In the following we describe the definitions and the notations used within the graph theory.

Definitions and notation

A graph $G = (V, E)$ in its basic form is composed of vertices and edges. V is the set of vertices (also called nodes or points) and $E \subset V \times V$ is the set of edges (also known as arcs or lines) of graph G. The order (or size) of a graph G is defined as the number of vertices of G and it is represented as $|V|$ and the number of edges as $|E|$. If two vertices in G, say u, $v \in V$, are connected by an edge $e \in E$, this is denoted by $e = (u, v)$ and the two vertices are said to be adjacent or neighbors. Edges are said to be undirected when they have no direction, and a graph G containing only such types of graphs is called undirected. When all edges have directions and therefore (u, v) and (v, u) can be distinguished, the graph is said to be directed. In this dissertation we will mainly use directed graphs, but graph matching can also be applied to undirected ones. In addition, a directed graph $G = (V, E)$ is called complete when there is always an edge $(u, u') \in E = V \times V$ between any two vertices u, u' in the graph.

Graph vertices and edges can also contain information. When this information is a simple label (i.e. a name or number) the graph is called *labelled graph*, where $\alpha : V \rightarrow L_V$ is the vertex labelling function and $\beta : E \rightarrow L_E$ is the edge labelling function; therefore a directed labelled graph is defined by a quadruple $G = (V, E, \alpha, \beta)$. Other times, vertices and edges contain some more information. These are called vertex and edge attributes, and the graph is called *attributed graph*. More usually, this concept is further specified by distinguishing between *vertex-attributed* (or weighted graphs) and *edge-attributed graphs*.

Graph matching

Given two graphs -the *Query graph* G and the *Target graph* G' the procedure of comparing them involves to check whether they are similar or not. Generally speaking, the standard concepts in graph matching include graph isomorphism and subgraph isomorphism.

Graph isomorphism. Two graphs are called isomorphic if they have identical structure. More formally, an isomorphism between two graphs G and G' is a bijective mapping between the nodes of G and G' that preserves the structure of the edges. Then:

Let G and G' be graphs. A graph isomorphism between G and G' is a bijective mapping $f : V \rightarrow V'$ such that

- $\alpha(v) = \alpha'(f(v))$ for all $v \in V$
- for any edge $e = (u, v) \in E$ there exists an edge $e' = (f(u), f(v)) \in E'$ such that $\beta(e) = \beta'(e')$ and for any edge $e' = (u', v') \in E'$ there exists an edge $e = (f^{-1}(u'), f^{-1}(v')) \in E$ such that $\beta(e) = \beta'(e')$.

This type of problem is said to be *exact graph matching*.

Subgraph isomorphism is another popular concept in graph comparison. Given two graphs, there exists a subgraph isomorphism if one graph contains a subgraph that is isomorphic to the other. Subgraph isomorphism is useful to find out if a given object is part of another object or even of a collection of several objects. Therefore:

If $f : V \rightarrow V'$ is a graph isomorphism between graphs G and G', and G' is a subgraph of another graph G", i.e. $G' \subset G$", then f is called a subgraph isomorphism from G to G".

When an isomorphism cannot be found between the two graphs to be matched because the number of vertices is different in both the query and target graphs, then the graph matching problem does not consist in searching for the exact way of matching vertices of a graph with vertices of the other, but in finding the best matching between them. This leads to a class of problems known as inexact graph matching.

There exist instances of subgraph isomorphism as maximum common subgraph and the minimum common subgraph. The maximum common subgraph of two graphs G' and G is the largest graph that is isomorphic to a subgraph of both G' and G. Maximum common subgraph is useful to measure the similarity of two objects. Clearly, the larger the maximum common subgraph of G' and G is, the more similar the two graphs are. On other side the minimum common supergraph of a pair of graphs G' and G is the smallest graph that contains subgraphs isomorphic to G' and G (see [35]).

3.1.2 Edit Distance: A Method to Measure the Similarity of two Graphs

A method to measure the similarity of two graphs is graph edit distance. It is a generalization of string edit distance, also known as Levenshtein distance [10]. In graph edit distance, one introduces a set of graph edit operations. These edit operations are used to model distortions that transform a noisy pattern into an ideal object representation. Common sets of graph edit operations include the deletion, insertion and substitution of nodes and edges. Given a set of edit operations, graph edit distance is defined as the minimum number of operations needed to transform one graph into the other. Often a cost is assigned to each edit operation. The costs are application dependent and are generally used to model the likelihood of the corresponding distortions. Typically, the more likely a certain distortion is to occur, the lower is its cost. If a cost is assigned to each edit operation then the edit distance of two graphs, G and G', is defined

as the minimum cost taken over all sequences of edit operations that transform G into G'. Graph edit distance and related similarity measures have been discussed in [l06, 104, 32].

Given a graph G, a graph edit operation δ on G is any of the following:

- substituting the label $\alpha(v)$ of vertex v by l.
- substituting the label $\beta(e)$ of edge e by l'.
- deleting the vertex v from G (for the correction of missing vertices). Note that all edges that are incident with the vertex v are deleted too.
- deleting the edge e from G (for the correction of missing edges).
- inserting an edge between two existing vertices (for the correction of extraneous edges).

Edited graph Given a graph and an edit operation δ, the edited graph $\delta(G)$ is a graph in which the operation δ was applied. Given a graph G and a sequence of edit operations $\Delta = (\delta_1, \delta_2, \cdots \delta_k)$, the edited graph $\Delta(G)$ is a graph $\Delta(G) = \delta_k(\cdots \delta_2(\delta_1(G)))..)$.

As this dissertation concentrates on approximate services matching, then for the sake of completeness, the subgraph edit distance will be formally introduced in the remainder of this section.

3.1.3 Error-correcting Subgraph Matching

Given two graphs G and G', an error-correcting (ec) subgraph isomorphism f from G to G' is a 2-tuple $f = (\Delta, f_\Delta)$ where Δ is a sequence of edit operations and f is a subgraph isomorphism from $\Delta(G)$ to G'.

For each edit operation δ, a certain cost is assigned $C(\delta)$. The cost of an ec-subgraph isomorphism $f = (\Delta, f_\Delta)$ is the cost of the edit operations Δ, i.e., $C(\Delta) = \sum_{i=1}^{k} C(\delta_i)$. Usually, there is more than one sequence of edit operations such that a subgraph isomorphism from $\Delta(G)$ to G' exists and, consequently, there is more than one ec-subgraph isomorphism from G to G'. We are interested in the ec-subgraph isomorphism with minimum cost. Then:

Subgraph edit distance. Let G and G' be two graphs. The subgraph distance from G to G', $ed(G, G')$ is given by the minimum cost taken over all error-correcting subgraph isomorphism f from G to G'.

Algorithm of error-correcting sub-graph isomorphism detection

In this section we present a well-known algorithm for the problem of error-correcting subgraph isomorphism detection [81].

The sub-graph isomorphism detection is based on a state-space searching using an algorithm similar to A* [106]. The basic idea of a state-space search is to have states representing partial solutions of the given problem and to define transitions from one state to another, thus, the latter state represents a more complete solution than the previous state. For each state s there is an evaluation function $f(s)$ which

Algorithm 1 Error-correcting sub-graph isomorphism detection $(G(V), G_I(V_I))$

1: Initialize OPEN: map the activity node in V onto each activity node in V_I (call Node-mapping), i.e. create a mapping p. Calculate the cost of this mapping $C(p)$ and add p to $OPEN$.
2: IF $OPEN$ is empty THEN Exit.
3: Select p of $OPEN$ such that $C(p)$ is minimal and remove p from $OPEN$
4: IF $C(p) > Accept\ threshold$ THEN Exit.
5: IF p represents a complete mapping from G to G_I THEN output p. Set $acceptthreshold$ = $C(p)$. Goto 2.
6: Let p=$\{(v_1,w_i),...,(v_k,w_j)\}$ be the current mapping that maps k nodes from G.
7: FOR each $Interaction\ node\ w\ in\ V_I$ that has not yet mapped to a corresponding node in V

7.1: extends the current mapping p to p' by mapping the v_{k+1} node of V to w, p'=$\{(v_1,w_i),...,(v_k,w_j),(v_{k+1},w)\}$ and calculate the cost of this mapping $C(p')$

7.2: add p' to $OPEN$

8: Goto 2

describes the quality of the represented solution. The states are expanding themselves according to the value of f. In the case of each sub-graph isomorphism detection, given a query graph G and an target graph G_I, a state s in the search space represents a partial matching from G to G_I. Each partial matching implies a number of edit operations and their cost can be used to define the evaluation function $f(s)$.

In other words, the Algorithm 1 starts by mapping the first node of G with all the nodes of G_I and chooses the best mapping (with minimal cost). This represents a partial mapping that will be extended by adding one node at a time. The process terminates when either a state representing an optimal ec subgraph isomorphism (error-correcting subgraph isomorphism matching) from G to G_I has been reached or all states in the search space have edit costs that exceed a given acceptance threshold. The cost of the mapping $C(p')$ represents the cost of extending the current mapping p with the next node in the query graph. Extending the mapping by mapping a vertex v (in the target graph that has not yet mapped) to a vertex w in the query graph (that does not belong to the current mapping) implies node edit operation and edge edit operations. First, the attributes of v must be substituted by attributes of w, and secondly, for each pair of already mapped nodes (v', w') it must be ensured that any edge (v', v) in the query graph can be mapped to an edge (w', w) in the target graph by means of edge edit operations ([81]).

Example. Consider the graphs g_1 and g_2 in Figure 3.1 and the problem of finding the ec subgraph isomorphism from $g1$ to g_2. Notice that the vertices of g_1 and g_2 are labeled with letters of the Latin alphabet while the edges are unlabeled. Additionally, the vertices of g_2 are uniquely numbered for the purpose of identification. The costs of the edit operations are defined as follows. The substitution of a label l_1 by a label l_2 is defined to be the distance of the letters l_1 and l_2 in the order of the alphabet. For instance, if $l_1 = a$ and $l_2 = g$ the cost for substituting a by g, or g by a, is 6. The cost for deleting a vertex is constantly set to 3 while the cost for deleting and inserting an edge is set to 1. The search space that is expanded by the traditional algorithm is displayed in Figure3.1. On the left of the search space, the graph g_1 is redrawn with its vertices given in the order in which they are matched. Each state s is indicated by a partial matching $(v)[c]$ where (v) denotes the number of the vertex of g_2 that is matched with the

corresponding vertex of g_1 and $[c]$ is the total cost that is implied by the partial matching represented in the state s and its predecessor states. The states are expanded according to the cost c. In order to represent the deletion of a vertex, the symbol \$ is introduced.

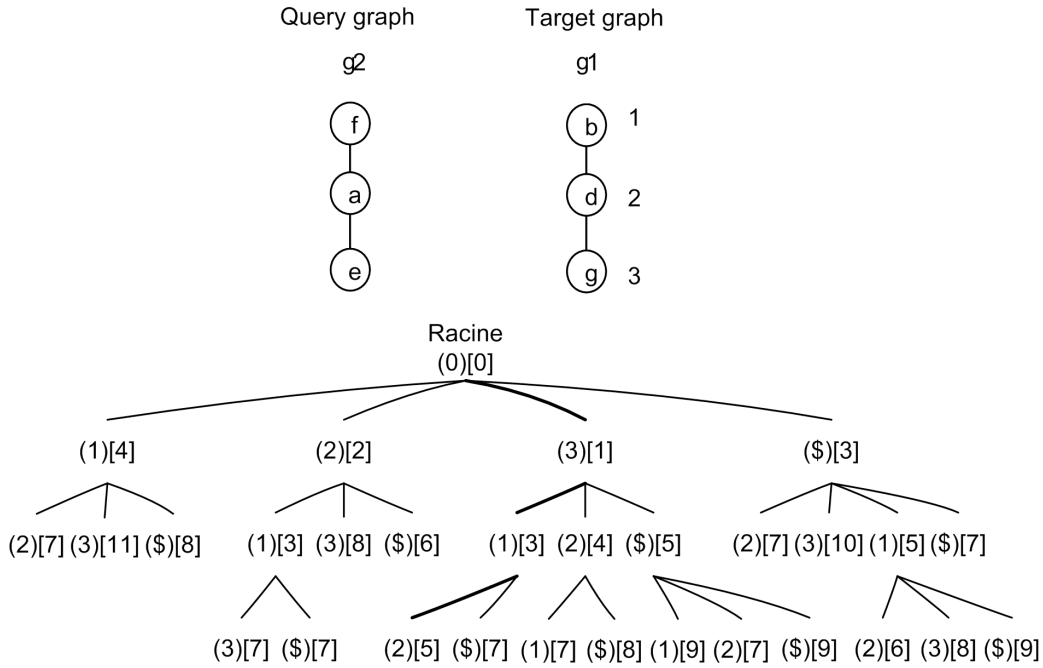

Figure 3.1: Example of Error-correcting subgraph isomorphism detection

In the beginning, there is only the root state, $(0)[0]$, for which all successors states are generated. Next, the state with the lowest costs, namely the third state from the left, $(3)[1]$ is expanded, and so on. The path to the state representing the ec subgraph isomorphism is depicted in bold face in Figure3.1. Note that the ec subgraph isomorphism from g_1 to g_2 matches the vertex f onto the vertex g, the vertex a onto the vertex b and the vertex e onto the vertex d. The edge between a and f in g_1 is deleted and an edge between f and e in g_1 inserted. The total costs of the edit operations amounts to 5 units.

3.1.4 Extensions of the EC-Algorithm

In order to expand the Ec-Algorithm to the service behavioral matchmaking we have considered two extensions. The first extension focuses on granularity level of services. In order to rank the target services, the second extension refers to the similarity measures. Such measures have to take into account the number of interactions or the number of interaction sequences in the target service that were covered by the query service. In the following we describe these extensions.

Extension of the sub-graph edit distance

Given that the matched models can have different granularity levels for achieving the same functionality, new edit operations are required. For example, one service has a single operation (activity) to achieve certain functionality, while in other service the same behavior is achieved by composing several operations.

Given a graph G, we extend the definition of edit operation δ on G by adding two operations:

- decomposing a vertex v into two vertices v_1, v_2.
- joining two vertices v_1, v_2 into a vertex v.

We limit ourselves to a simple case of decomposition, when a vertex is decomposed into a sequence of two vertices. This simple type of decomposition is sufficient for applications that we analyzed. A more general decomposition operation would be to decompose a vertex into a connected subgraph, this is subject of future work.

The operation of decomposing a vertex v into two vertices v_1, v_2 is executed in the following way :

- all the edges having as destination the vertex v will have as destination the vertex v_1;
- all edges having as source the vertex v, will have as source the vertex v_2;
- an edge between the vertex v_1 and v_2 will be added.

The joining operation is executed in a similar way. These two new edit operations allow to model one-to-many dependencies among vertices of two graphs (i.e., a vertex in one graph correspond to two vertices in the second graph). The classical edit operations take into account only one-to-one mappings between vertices of the two graphs. For example, if a vertex v in the first graph corresponds to the composition of two vertices in the second graph (v_1 followed by v_2), a matching algorithm based on the classical edit distance would map v to v_1 and suppress v_2. It would not be possible to discover that v is mapped to a composition of v_1 and v_2.

Similarity Measure for Behavioral Matching

The subgraph edit distance defined previously expresses the cost of transformation needed to adapt the query graph (Q) in order to cover a subgraph in the target model (T). This distance is asymmetric, it represents the distance from the query graph to the target graph. On the other hand, the similarity of graphs and distance between them are closely related and are often confused. While the term distance is used more precisely in a mathematical sense, the particular meaning of the term similarity often depends on the circumstances and its field of application. Therefore, the similarity measures for service behavioral matchmaking are calculated as following:

- considering that a similarity measure should not only be qualitative, giving information about commonalities and differences between the two graphs, but also quantitative, indicating how much two graphs are similar. So we consider that the similarity between two graphs can be based on distance measure, but they are inversely related. Therefore two graphs are deemed to be similar when there exist a small distance measure between them. The similarity function can be defined as:

$$Sim_{edit}(Q,T) = \frac{1}{1 + editDist(Q,T)}$$

Where $editDist(Q,T)$ is the minimal cost of the edit operations needed to transform the query graph with respect to the target graph.

- in order to rank the target graphs, the similarity measure has to take into account the number of vertices in the target graph that were covered by the query graph. If two target graphs have the same subgraph distance to the query graph ($Sim_{edit}(Q,T)$) but are matched to subgraphs with different number of nodes, the one that matches a subgraph with more nodes will be preferred. Then the similarity fuction is:

$$Sim_{node}(Q,T) = Sim_{edit}(Q,T) * \frac{|N_Q \cap N_T|}{|N_Q|}$$

Where $|N_Q \cap N_T|$ is the number of nodes that appear both in Query and Target graph.

- The third similarity measure is based on the number of different sequences between two graphs.

$$Sim_{seq}(Q,T) = Sim_{edit}(Q,T) * \frac{1}{1 + |Nseq(Q)| + |Nseq(T)| - 2\,|Nseq(Q) \cap Nseq(T)|}$$

In the same way of Sim_{node}, if two target graphs have the same distance measure with respect to the query graph ($Sim_{edit}(Q,T)$) but they are matched to subgraphs with different number of node sequences, the one that matches a subgraph with more node sequences will be preferred. In this similarity function the $Nseq$ parameter represents the number of consecutive nodes ($Nseq(Q)$ for query graph and $Nseq(T)$ for target graph) and $Nseq(Q) \cap Nseq(T)$ represents the mapped sequences into query graph with respect to the target graph. For instance, if we select $Nseq = 3$ (tri sequence), then the function will consider all sequences of three consecutive nodes of the query and target graph and their intersection. Thus, $tri(Q)$, $tri(T)$ and $|tri(Q) \cap tri(T)|$ will be calculated.

3.2 Conversation Protocol Matchmaking

In this section we illustrate the use of the error-correcting graph matching algorithms for conversation protocol matchmaking. We choose to exemplify our approach for business protocol matchmaking by using the WSCL model [16]. The same approach can be applied for other models (Into chapter 4 we will present the matchmaking by using the BPEL model), as long as the conversation protocol can be transformed to a graph in a unique way (equivalent representations of a conversation protocol are reduced to the same process graph).

The conversation protocol matchmaking process is composed of the following steps (see figure3.2). First, the conversation protocols to be compared are transformed into graphs. Next, both graphs are expanded in order to have the same level of granularity. Then, the error-correcting graph matching algorithm is applied. The similarity function evaluates the similarity between the graphs. Finally, the granularity levels are compared and the costs corresponding to identified differences are added to the total distance.

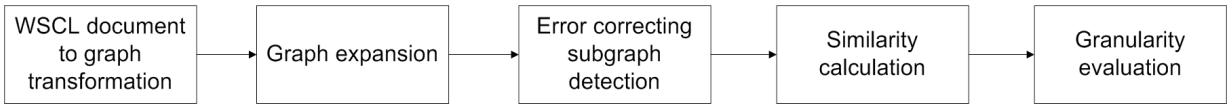

Figure 3.2: WSCL matchmaking process

Next, we first give an overview of Web Services Conversation Language (WSCL), and then we discuss each matchmaking step in detail; finally, we illustrate it using an example.

3.2.1 Web Services Conversation Language: WSCL

WSCL is a simple conversation definition language, which offers the basic constructs to model the sequencing of the interactions or operations of one interface. It complements the interface definition by specifying the invocation order of the operations. A conversation in WSCL is specified using the following basic constructers:

- Document type descriptions specify the types (schemas) of XML documents that the service can accept and transmit in the course of a conversation. The schemas of the documents exchanged are not specified as part of the WSCL specification document; the actual document schemas are separate XML documents referenced by their URL in the XML Document Type elements of the conversation specification (see figure 3.3).

- Interactions model the actions of the conversation as document exchanges between two participants. WSCL supports five types of interactions: *Send* (the service sends out an outbound document); *Receive* (the service receives an inbound document); *SendReceive* (the service sends out an outbound document and then expects to receive an inbound document in reply); *ReceiveSend* (the service receives an inbound document and then sends out an outbound document); *Empty* (does not contain any documents exchanged, but is used only for modelling the start and end of a conversation.) (see figure 3.3). Each interaction specifies the type (schemas) of XML document that is expected as input or is produced as output.

- Transitions specify the ordering relationships between interactions.

Conversations list all the interactions and transitions that compose the conversation. A WSCL document contains additional information about the conversation, including the conversation's name and the name of documents interchanged between the participants. Conversations can be thought of as interfaces or public processes supported by a service. They differ from interfaces as defined by CORBA IDE or Java interfaces because they also specify the possible ordering of operations (i.e. the possible sequences in which documents may be exchanged).

A conversation definition defines the conversation from the perspective of one of the participants. In most cases, a conversation published in a service directory is the one defined from the perspective of the listener; the first interaction to happen is a Receive or ReceiveSend interaction. An initiator can derive its conversation definition from the conversation definition of the listener simply by converting Receive and

ReceiveSend interactions into Send and SendReceive interactions, and vice versa. Two participants can successfully interact if the conversation definitions they use are duals of each other.

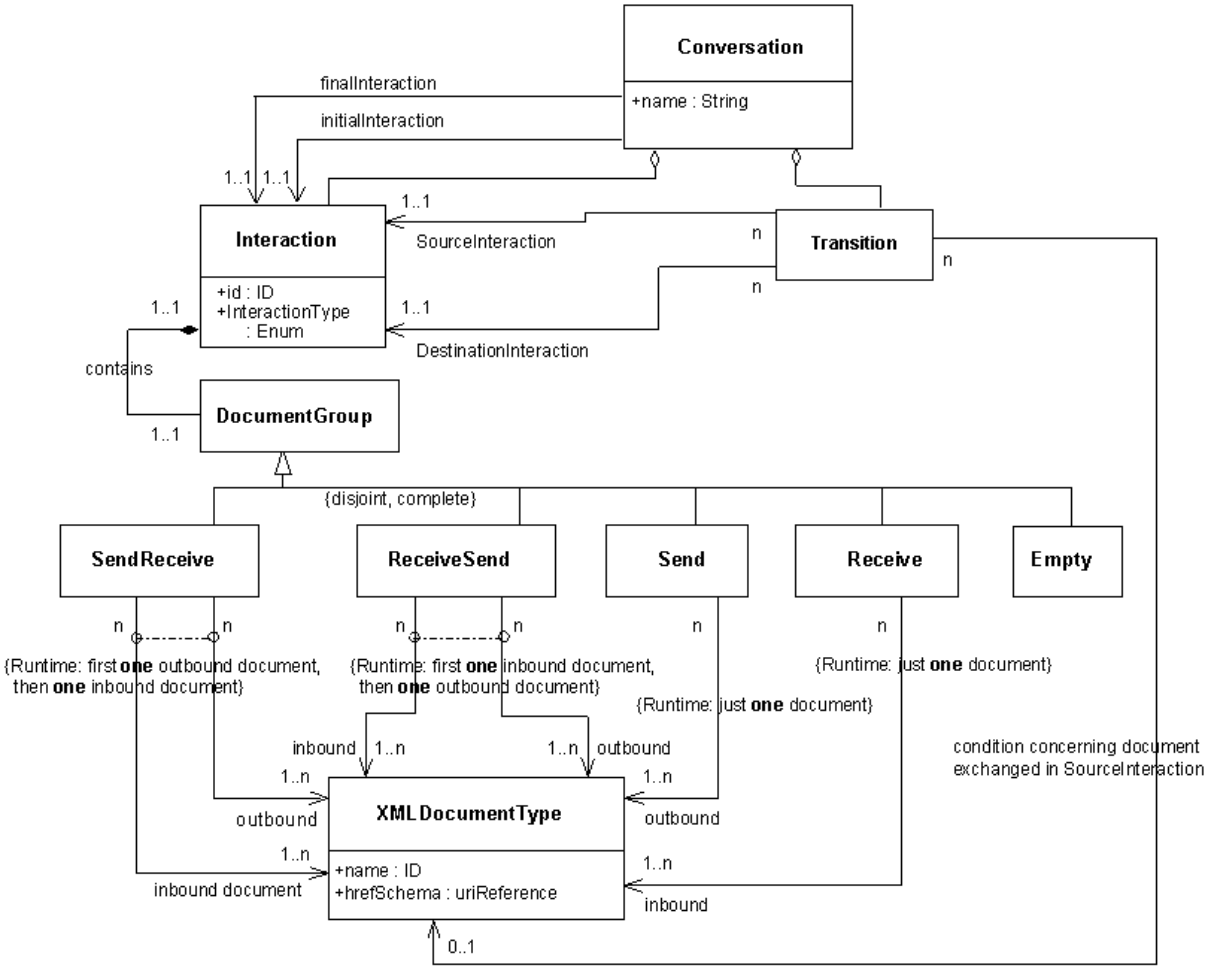

Figure 3.3: UML metamodel of WSCL protocol

The figure 3.4 shows an WSCL conversation. First the service expects a conversation to begin with the receipt of a *LoginRQ* message ($Login$ interaction). The service sends as a response a *ValidLoginRS* or *InvalidLoginRS* document depending on the type and content of the message received. In case of a valid login, the service will expect a *SeatsPreferenceRQ* message ($CheckSeatsAvailability$ interaction). Depending on the content of the received document, the $ChecksSeatsAvailability$ interaction can reply with *ValidPreferencesRS* or *InvalidPreferencesRS*. If the response is a *ValidPreferencesRS*, the service will hope a confirmation *ConfirmationRQ* ($ReserveSeats$ interaction), then if the reservation is valid the conversation await a *PurchasePreferencesRS* message and can send as a reply a *InvalidPaymentRS* or send the payment information (*ValidPaymentRS*) to the user ($PurchaseSeats$ interaction), otherwise the conversation will end.

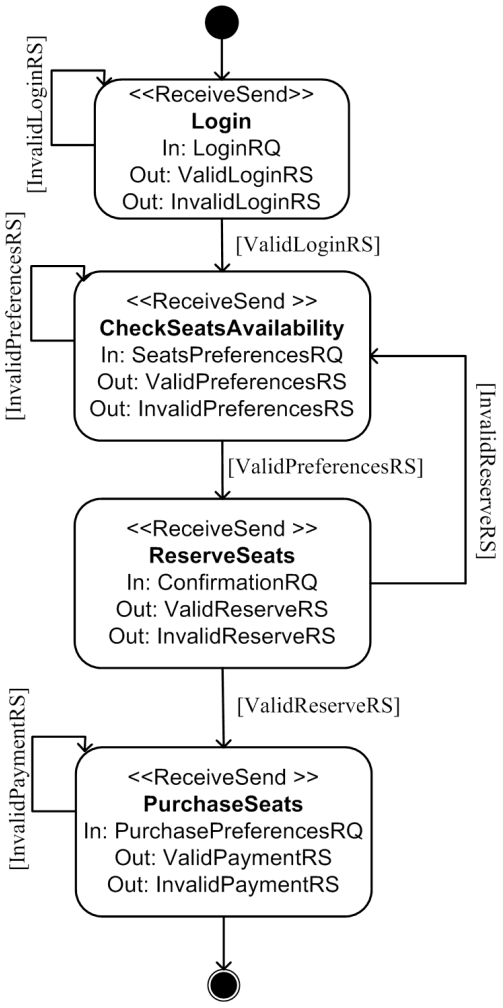

Figure 3.4: Example of a WSCL conversation

The Conversation XML description contains two sub-elements: the interactions list, which contains the conversation interactions and the transitions list have within the transition elements. Another part of defining the possible ordering of interactions is the specification of the first and last interactions of a conversation. In the XML description, this is done by the attributes *InitialInteraction* and *FinalInteraction* of the Conversation element. For the WSCL example presented in figure3.4 the initial interaction is *Login* and de final interaction is *PurchaseSeats*.

A conversation can proceed from one interaction to another as allowed by the permissible sequencing defined in the transition elements. The figure 3.5 shows the XML transition of WSCL example depicted. *SourceInteraction* references an interaction that can precede the *DestinationInteraction* when the conversation is executed. Similarly, *DestinationInteraction* references one of the interactions that can follow the *SourceInteraction* when the conversation is executed. Together, all transitions specify all possible sequences of the interactions. *SourceInteractionCondition* is an additional constraint on the transition. It is

needed when the *SourceInteraction* specifies more than one possible document to be exchanged and the type of document exchanged has an influence on the possible next interactions.

```
<ConversationTransitions>
<Transition>
    <SourceInteraction href="Login"/>
    <DestinationInteraction href="CheckSeatsAvailability"/>
  </Transition>
</Transition>
  <SourceInteraction href="Login"/>
    <DestinationInteraction href="Login"/>
    <SourceInteractionCondition href="InvalidLoginRS"/>
  <Transition>
<Transition>
    <SourceInteraction href="CheckSeatsAvailability"/>
    <DestinationInteraction href="ReserveSeats"/>
  </Transition>
</Transition>
  <SourceInteraction href="CheckSeatsAvailability"/>
    <DestinationInteraction href="CheckSeatsAvailability"/>
    <SourceInteractionCondition href="InvalidPreferencesRS"/>
  <Transition>
<Transition>
    <SourceInteraction href="ReserveSeats"/>
    <DestinationInteraction href="PurchaseSeats"/>
  </Transition>
</Transition>
  <SourceInteraction href="ReserveSeats"/>
    <DestinationInteraction href="CheckSeatsAvailability"/>
    <SourceInteractionCondition href="InvalidReserveRS"/>
  <Transition>
<Transition>
    <SourceInteraction href="PurchaseSeats"/>
    <DestinationInteraction href="PurchaseSeats"/>
    <SourceInteractionCondition href="InvalidPaymentRS"/>
  </Transition>
 </ConversationTransitions>
```

Figure 3.5: XML representation of the WSCL transition

Additionally, the figure 3.6 exemplify the XML representation of $CheckSeatsAvailability$ interaction. Each *ReceiveSend* interaction is the logical unit of receiving a request and then returning a response. The interaction is not complete until the response has been sent. A *ReceiveSend* interaction can specify more than one Outbound XML Document. This ability allows the modeling of the type of cases in which there are multiple types of response messages that a service might return in response to a specific request. Having additional possible responses is mainly used for error messages. If a *ReceiveSend* interaction specifies more than one outbound document type, only one of them is being exchanged at runtime.

```
<ConversationInteractions>
                                        .
                                        .
                                        .
<Interaction interactionType="ReceiveSend" id="CheckSeatsAvailability">
 <InboundXMLDocument id="SeatsPreferencesRQ"
        hrefSchema="http://ariadna.unicauca.edu.co/SeatsPreferencesRQ.xsd">
 <OutboundXMLDocument id="ValidPreferencesRS"
        hrefSchema="http://ariadna.unicauca.edu.co/ValidPreferencesRS.xsd">
 <OutboundXMLDocument id="InvalidPreferencesRS"
        hrefSchema="http://ariadna.unicauca.edu.co/InvalidPreferencesRS.xsd">
</Interaction>
                                        .
                                        .
                                        .
</ConversationInteractions>
```

Figure 3.6: XML representation of a *ReceiveSend* interaction

3.2.2 WSCL to Graph Transformation

The parser function transforms a WSCL conversation model into a graph whose vertices represent interactions and whose edges represent transitions. Each node has the following attributes: name, interaction type and documents. The table 3.1 shows the correspondence between WSCL constructs and graph elements.

WSCL construct	**Graph element**
Interaction	Vertex
Type	Vertex attribute
Inbound document	Vertex attribute
Outbound document	Vertex attribute
Id	Label
Transition	Edge
Transition condition	Edge attribute

Table 3.1: Correspondences between WSCL elements and graph elements

3.2.3 Decomposition of WSCL Interactions

After transforming conversation protocols into graphs, the second step in the behavior matching is graph expansion. The decomposition operations are applied in order to have the same granularity level in both models. The decomposition operation depends on the metamodel of the protocols to be matched. For instance, for WSCL metamodel, it is possible that in one protocol an interaction is modelled as a *SendReceive* interaction, while in the second protocol the same functionality is achieved by having a *Send*

interaction followed by a *Receive* interaction. Thus, the decomposition will transform interactions of type *SendReceive* or *ReceiveSend* in atomic interactions: *Send* and *Receive*.

A *SendReceive* interaction is decomposed into a *Send* interaction followed by a *Receive* interaction in the following way:

- all edges having as destination the *SendReceive* interaction will have as destination the *Send* interaction
- all edges having as source the *SendReceive* interaction, will have as source the *Receive* interaction
- an edge will be added from the *Send* interaction to the *Receive* interaction
- if the *SendReceive* interaction has outbound document a and inbound document b, then the *Send* interaction will have a as outbound document and the *Receive* interaction will have b as inbound document.

In a similar way, a *ReceiveSend* interaction is decomposed into a *Receive* interaction followed by a *Send* interaction.

A decomposition function could also be applied to deal with difference of granularity of exchanged message; that is, if a message *D* can be decomposed into n atomic messages, then an interaction sending the document *D* (*Send D*) could be decomposed into *n Send* interactions corresponding to the *n* documents parts. However, in this approach we do not analyze message content and thus we propose only the decomposition operation described above.

This decomposition function is specific to WSCL model. For other applications, user can specify a different decomposition function. The decomposition function has always the same signature: it takes as argument a vertex and returns two vertices resulting from decomposition (that are supposed to be sequential). The function behavior is specific to the application (metamodel of the protocols to be matched) and consists in specifying how the labels and attributes of the new vertices are obtained from the decomposed vertex.

3.2.4 Comparison Rules of WSCL Graphs

The *Comparison rules* describe all the application-dependent functions allowing to calculate the cost of graph edit operations. These functions are used by the algorithm of ec subgraph isomorphism matching for calculating the distance between the graphs (see section 3.1.3). In order to support applications with specialized cost function, user-defined cost function can be registered as a rule. In the following we explain the cost function used for conversation protocol matchmaking.

The cost for inserting, suppressing edges and vertices can be set to a constant. The cost for editing a vertex is calculated by function VertexMatch (see Algorithm 2). As vertices represent WSCL interactions, this cost expresses the distance between two WSCL interactions. Each interaction has a label (Id) and two attributes: the interaction type ($Type$) and documents set (D) (in or outbound documents). The matchmaking gives priority to type comparison, and if two interactions have the same type, it compares the similarity of the set of documents $TotalSD$; if there is a similarity between them ($TotalSD > 0$), it calculates the similarity of the interaction names ($SimId$).

Algorithm 2 Function VertexMatch

INPUTS: ($Node_i$,$Node_j$)
Nodei: Struct (Id_i,$Type_i$,D_i), Nodej: Struct (Id_j,$Type_j$,D_j)
OUTPUT: $DistanceNode$

if $Type_i \neq Type_j$ (different types) **then**
  Return $DistanceNode = 1$
else
  Calculate document sets similarity $TotalSD$
  if $TotalSD > 0$ **then**
    Calculate Ids similarity $SimId = LS(Id_i, Id_j)$

$$DistanceNode = 1 - \frac{w_d * TotalSD + w_i * SimId}{w_d + w_i}$$

    Return $DistanceNode$
  else
    Return $DistanceNode = 1$
  end if
end if

The function $SD(D_i, D_j)$ where D_i, D_j is the set of documents of $Node_i$ and $Node_j$ respectively, computes the best mapping that can be obtained between the documents of the two sets.

$$SD(Di, Dj) = \begin{cases} Max(SD(Di - I, Dj - J) + LS(I, J)), & Di \neq \phi, Dj \neq \phi, \\ & I \in Di, J \in Dj \\ 0, & Di = \phi \vee Dj = \phi \end{cases}$$

The number of mappings established are $Min(|D_i|, |D_j|)$. Function LS calculates the linguistic similarity between document names and is explained in the next section.

Finally, the total similarity of the document sets is:

$$TotalSD = \frac{SD(D_i, D_j)}{k}$$

Where, k = Number of documents of set D_i. The result of applying the function SD is normalized with respect to the number of documents of the node belonging to the target graph.

Weights w_d and w_i indicate the contribution of $TotalSD$ (similarity of documents being exchanged) and $SimId$ (similarity of interaction names) respectively in the total DistanceNode score ($0 \leq w_d \leq 1$ and $0 \leq w_i \leq 1$).

3.2.5 Linguistic Comparison of WSCL Attributes

The *Linguistic comparison* calculates the linguistic similarity between two labels based on their names [95]. The labels are often formed by a word or by a combination of words and can contain abbreviations. To obtain a linguistic distance between two strings, we use existing algorithms: *NGram*, *Check synonym*, and *Check abbreviation*. The *NGram* algorithm estimates the similarity according to a number of common *qgrams* between label names [12]. The *Check synonym* algorithm uses a linguistic dictionary (e.g. Wordnet [86] in our implementation) to find out the synonyms between the label names while the *Check abbreviation* uses an abbreviation dictionary according to the application domain.

First, strings are tokenized on the basis of punctuation and capitalization. Then unnecessary words are removed from the list of tokens, using a stop-word list. If these individual tokens cannot be matched, they are stemmed using $porter\ stemmer$ algorithm and try to match them using $NGram$ technique. If any of these algorithms return a full match, i.e. 1 on scale of 0 to 1, then a match score of 1 for linguistic similarity is returned. On the other hand, if all the algorithms return 0, it means that there is no matching between labels. If the $NGram$ value and the $Check\ abbreviation$ value are equal to 0, and $Check\ Synonym$ is between 0 and 1, the total linguistic similarity value will be equal to the Check Synonym one. Finally, if the three algorithm values are between 0 and 1, the similarity LS ([95]) is the average of them:

$$LS = \begin{cases} 1 & if\ (m1 = 1 \vee m2 = 1 \vee m3 = 1) \\ m2 & if\ (0 < m2 < 1 \wedge m1 = m3 = 0) \\ 0 & if\ (m1 = m2 = m3 = 0) \\ \frac{m1+m2+m3}{3} & if\ m1, m2, m3 \in (0, 1) \end{cases}$$

Where, m1 = Sim(NGram), m2=Sim(Synonym Matching) and m3= Sim(Abbreviation Expansion).

There are other possible ways to measure name similarity: Levenshtein edit distance algorithm , techniques borrowed from the information retrieval area like TF-IDF or a combination of these techniques. However, defining a clever function for syntactic similarity is outside the scope of this approach, since the focus of our work is on behavior similarity.

3.2.6 Granularity Level Comparison of Mapped WSCL Interactions

The ec subgraph isomorphism (error-correcting subgraph isomorphism matching) is applied to graphs that were expanded, i.e., contain only atomic *Send* or *Receive* interactions. The granularity comparison checks whether the interactions that were mapped by the ec subgraph isomorphism algorithm have the same granularity level in both models. For instance, suppose that in the target graph we have a *SendReceive* interaction. This was decomposed by the decomposition function in a *Send* interaction followed by a *Receive* interaction that were mapped with two corresponding interactions in the query graph (by the ec subgraph isomorphism algorithm). If these interactions were atomic in the query graph, the cost of joining operation has to be added to the total graph distance (line 5 in the Table3.2).

The costs for granularity differences that have to be taken into account for the total distance graph for all cases of figure (atomic versus non atomic interactions in the query and target graph) are summarized in the Table 3.2.

Interaction type of query graph	**Interaction type of target graph**	**Granularity Diff. Cost**
S atomic	S atomic	0
R atomic	R atomic	0
SR	SR	0
RS	RS	0
SR (or RS)	S atomic + R atomic	c_j
SR (or RS)	S nonat. + R nonat.	$c_d + c_j$
SR (or RS)	S nonat. + R atomic or S atomic + R nonat.	$c_d/2 + c_j$
S atomic	S nonatomic	$c_d/2$
R atomic	R nonatomic	$c_d/2$

S=Send, R=Receive, SR= SendReceive, RS=ReceiveSend

Table 3.2: Cost for granularity differences

For the sake of clarity, the table does not present the cases for interactions that have no correspondence in the other graph. If the mapped interactions have the same granularity level (they are both atomic or non atomic) there is no cost to be added to the subgraph edit distance.

A more complicated case (line 7 in the Table 3.2) is when a *SendReceive* interaction SR_I in the target graph is mapped with an atomic *Send* interaction S_M followed by a *Receive* interaction R_M that is non atomic (belongs to a *SendReceive* SR_M) in the query graph. In this case, the cost is $c_j + c_d/2$ ($c_j =$ cost of joining S_M and R_M; $c_d/2 =$ cost for obtaining R_M by decomposing SR_M interaction in the query graph).

3.2.7 An Example for the WSCL Matchmaking

Suppose that we would like to find the similarity between two purchase services whose conversations have been described using WSCL language. We consider again the example presented in section 1.1 that we describe in more detail.

- The first conversation (target service) of the figure 3.7 expects a conversation to begin with the receipt of a *LoginRQ* message ($Login$ interaction). The service sends as a response a *ValidLoginRS* or *InvalidLoginRS* document depending on the type and content of the message received. In case of a valid login, the service will expect a *PurchaseRQ* message ($Purchase$ interaction). Depending on the content of the received document, the $Purchase$ interaction can reply with *PurchaseAccepted RS*, *InvalidPaymentRS* or *OutOfStockRS*. If the response is a *PurchaseAcceptedRS*, the service will send the shipping information (*ShippingInformation, type Send*) to the user ($Shipping$ interaction), otherwise the conversation will end.

- Similarly, the second conversation (query service) of the figure 3.7 expects a *LgnRQ* message and can send as a reply a *ValidLgnRS* or *InvalidLgnRS* document (Lgn interaction). In case of a valid login, client can send shipping preferences and the service will return a *ShipmentAcceptedRS* or *OutsideZone* document according to the content of document received ($Shipment$ interaction, type SendReceive). In the first case, the conversation continues with a Buy interaction, otherwise it ends.

Our system converts each WCSL document into a graph (target graph and query graph, Figure3.7). Next, the graphs are decomposed according to the interaction type (Decomposed target graph and query graph, Figure 3.7) using the decomposition function. The documents are assigned according to the type of the decomposed interaction. Thus, for a decomposed Receive interaction an Inbound XML Document is assigned and for a decomposed Send interaction an Outbound XML Document is set.

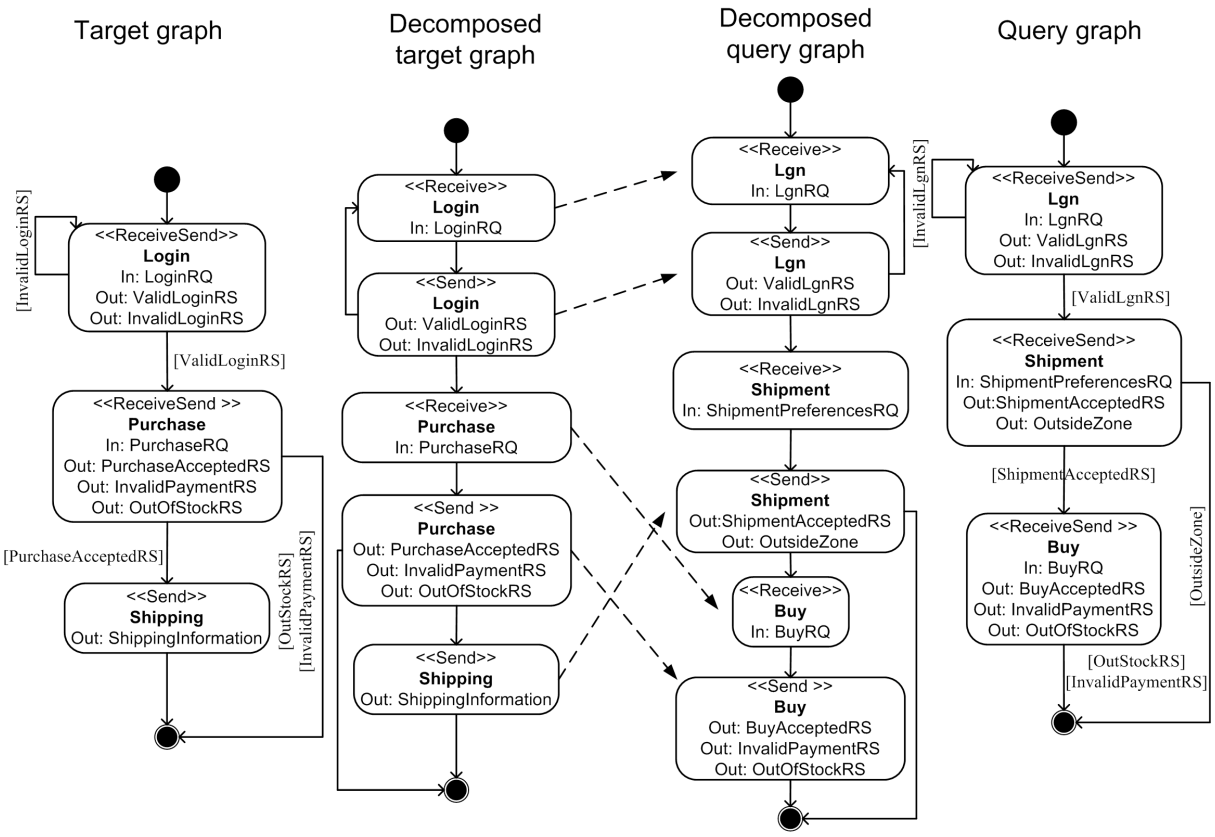

Figure 3.7: WSCL matchmaking example

Then the graph matchmaking algorithm is applied on the decomposed graphs. The function $Vertex\ Match$ will be invoked for comparing nodes. For each pair of interactions, first the function will verify if there exists a similarity between its documents. For example, when comparing *Buy* and *Purchase* interactions (Type: Send), the best mapping between the outbound documents of each interaction is found and a total similarity ($TotalSD$) is calculated. As there exist a similarity between them, the linguistic similarity (LS) is calculated between the interaction names (Id: *Buy* and Id: *Purchase*) and finally, the total distance is calculated. For this example we have considered that the similarity of interaction names (w_i) has the same importance than the similarity of documents being exchanged (u_{il}). The dotted lines in Figure 3.7 represent the mappings found by the system between the two graphs.

Finally, the cost of granularity differences is added to the total graph distance. Thus, the *Shipment* interaction (type ReceiveSend) in the query graph has to be decomposed into two interactions to match

the *Shipping* interaction (type Send) into the target graph (see the dotted lines of Figure3.7). Therefore, the cost of line 8 of Figure 3.2 is added. For the other mappings no granularity cost is added.

In conclusion, the edit script will show that the two graphs are similar, but have the following structural differences: for the mapping (*Shipment*,*Shipping*) (see figure 3.7), the *Shipment* is a Send non atomic interaction (was obtained by decomposing a SR interaction) and the *Shipping* is a send atomic interaction. (Hence the system will add the granularity cost $C_d/2$ to the total distance between the two graphs.) There is not a corresponding node for the Shipment (R) into the target graph. On the other side, the interactions for purchasing and for shipping are executed in different order in the two models, therefore the system will add to the total distance the costs of necessary edit operations for reordering them. Thus, the script of the graph edit operations is the following:

- Decomposing the node *Shipment(SR)* from query graph
- Deleting the node *Shipment(R)* from query graph
- Deleting the edge (*Shipment*,*Buy*) from query graph
- Deleting the edge (*Lgn*,*Shipment*) from query graph
- Inserting the edge (*Lgn*,*Buy*) into query graph
- Inserting the edge (*Buy*,*Shipment*) into query graph

3.3 Summary

In this chapter we explained our graph-based approach to behavior matchmaking and showed an application of this one. First, we introduced the graph matching problem explaining its definition and notation used within the graph theory. As this PhD thesis concentrates on approximate services matching, a method to measure the similarity of two graphs was depicted. Next, the error-correcting subgraph matching (Ec-Algorithm) was presented in detail.

In order to expand the Ec-Algorithm to the service behavioral matchmaking we presented two extensions. The first extension focused on granularity level of services. In order to rank the target services, the second extension refered to the similarity measures. Such measures have to take into account the number of interactions or the number of interaction sequences in the target service that were covered by the query service.

Finally, we showed an application case of this algorithm to WSCL (Web Services Conversation Language) protocols. The same approach can be applied for other models (Into chapter 4 we will present the matchmaking by using the BPEL model). The conversation protocol matchmaking process is composed of the following steps. First, the conversation protocols to be compared are transformed into graphs. Next, both graphs are expanded in order to have the same level of granularity. Then, the error-correcting graph matching algorithm is applied. The similarity function evaluates the similarity between the graphs. Finally, the granularity levels are compared and the costs corresponding to identified differences are added to the total distance.

Chapter 4

Behavioral Matchmaking: Application to Business Process Protocol

Over the last couple of decades, workflow technology has increasingly been used to coordinate activities in inter and intra organizational settings. With the advent of Web services and service oriented architectures (SOA) (see [52]), workflows have transitioned into Web services based processes (called Web processes), which leverage XML based open standards and a loosely coupled distributed computing model of SOA to achieve easier integration of autonomous distributed components. While the preliminary focus of SOA based implementations in the industry have leveraged the ease of integration provided by Web services, the true potential of Web service based solutions are the Web processes.

Web processes are the new generation workflows created using Web services. The Web processes can be implemented using WS-BPEL (Web Services Business Process Execution Language), wich is the industry de facto standard for Web services. BPEL technology has been established as a key contributor to the success of the service environment. BPEL allows to easily compose new services out of existing services. This enables new business models for software and enables non-IT professionals to create services. In the BPEL initiative, the process constructed exploits a classical workflow style composition of services, extended with external message interaction capability.

Considering the importance and extensive utilization of BPEL protocol for services description, in this chapter, we will discuss our approach for Behavioral matchmaking, by examining the usage of matching techniques in the context of BPEL behavioral specifications of the service. The BPEL matchmaking process is composed of the following steps (see figure 4.1). First, the BPEL documents to be compared are transformed to graphs. Next, the error correcting graph matching algorithm is applied (considering the decompostion and composition functions during the algorithm execution). Then, the similarity function evaluates the similarity between the graphs.

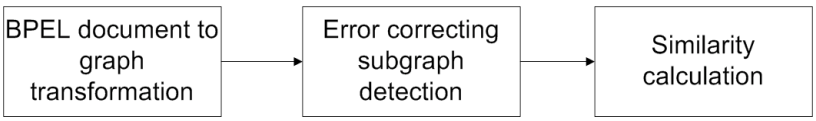

Figure 4.1: BPEL matchmaking process

First we will introduce the BPEL protocol, then we will explain the BPEL to graph transformation. In the section 4.3, we will show the BPEL matchmaking algorithm which is based on the algorithm introduced in previous chapter, but considering the comparison rules for the BPEL metamodel (section 4.4). Finally, an example of the BPEL matchmaking process will be depicted.

4.1 Business Process Execution Language for Web Services: BPEL

Web services are components, which are based on the industry standards WSDL (Web Service Definition Language), UDDI (Universal Description Discovery and Integration) and SOAP (Simple Object Access Protocol). They enable to connect different components even across organizational boundaries in a platform and language independent manner [73]. None of these standars for Web services however provides for the definition of the business semantics of Web services, the Web services are isolated and opaque. Braking insolation means to connect Web services and specify how collections of Web services are jointly used to realize more complex functionality- typically a business process. A business process specifies the potential execution order of operations from a collection of web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, joint exception handling for collections of Web services etc. In this way, **BPEL** [11] has emerged as a standard for specifying and executing web services-based processes. It supports the modelling of two types of processes: executable and abstract processes. An *abstract process* is a business protocol, specifying the message exchange between different parties from the perspective of a single organization (or composite service), whitout revealing the internal behavior. An *executable process*, in contrast, specifies the actual behavior of a participant. On the other side, a BPEL process is constituted of the following components:

- **Variables:** In BPEL variables are used to store workflow data and messages that are exchanged with Web Services. Variables have to be declared in the header part of a BPEL process.
- **PartnerLinks:** Partner links represent a bilateral message exchange between two parties. Via a reference to a *partnerLinkType* the *partnerLink* defines the mutual required *portTypes* of a message exchange: it holds a *myRole* and a *partnerRole* attribute to define who is playing which role. *PartnerLinks* are relevant for basic activities that involve Web Service requests (see figure4.2).
- **Basic Activities:** Basic activities define the operations which are performed in a process (see figure 4.3). These include operations involving Web Services like:
  - The *receive* construct allows the business process to do a blocking wait for a matching message to arrive.
  - The *reply* activity allows the business process to send a message in reply to a message that was received through a *receive*. The combination of a *receive* and a *reply* forms a request-response operation on the WSDL portType for the process.
  - The *invoke* construct allows the business process to invoke a one-way or request/response operation on a *portType* offered by a partner.
  - The *assign* activity can be used to update the values of variables with new data. An *assign* construct can contain any number of elementary assignments.

- The *throw* construct generates a fault from inside the business process.
- The *wait* activity allows you to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified.
- The *empty* construct allows you to insert a ”no-op” instruction into a business process. This is useful for synchronization of concurrent activities, for instance.

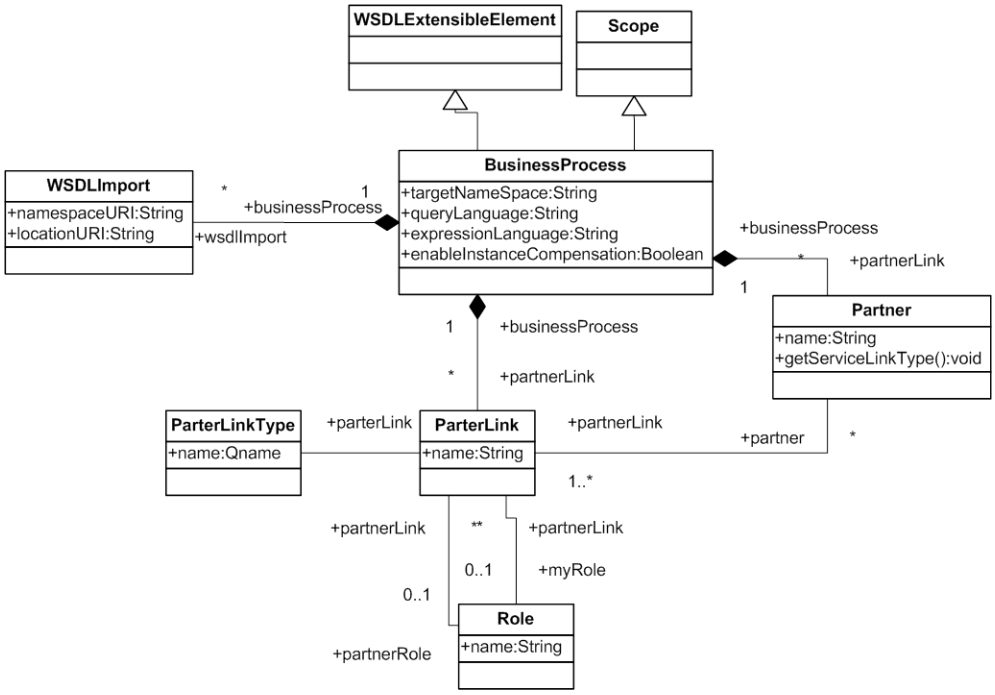

Figure 4.2: BPEL metamodel

- **Structured Activities:** BPEL offers structured activities for the definition of control flow, alternative branches or sequential execution (see figure 4.3). These structured activities can be nested. Next, we present the structured BPEL activities:
  - The *sequence* construct allows to define a collection of activities to be performed sequentially in lexical order.
  - The *switch* activity allows to select exactly one branch of activity from a set of choices.
  - The *pick* construct allows to block and wait for a suitable message to arrive or for a time-out alarm to go off. When one of these triggers occurs, the associated activity is performed and the *pick* completes.
  - The *flow* activity allows to specify one or more activities to be performed concurrently. Links can be used within concurrent activities to define arbitrary control structures.
  - The *scope* construct allows to define a nested activity with its own associated variables, fault handlers, and compensation handler.

- The *compensate* structured activity is used to invoke compensation on an inner scope that has already completed normally. This construct can be invoked only from within a fault handler or another compensation handler.

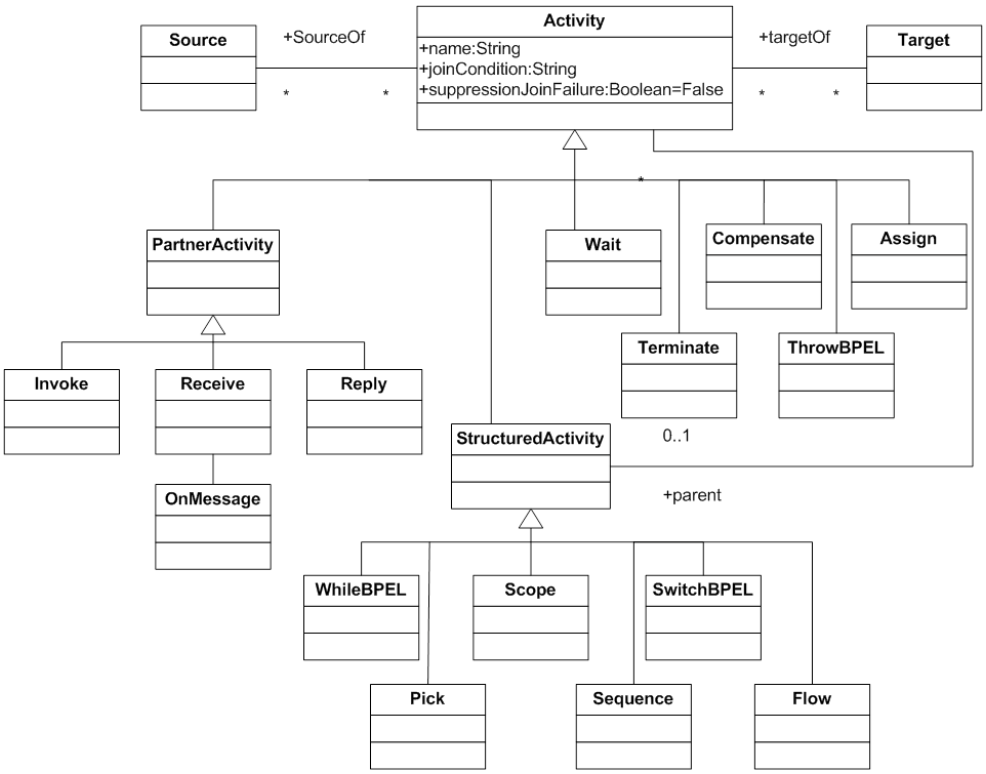

Figure 4.3: BPEL activities hierarchy

BPEL builds on IBM's WSFL and Microsoft's XLANG and combines thus the features of a block structured language (XLANG) with those for directed graphs (WSFL). As a result there are sometimes two equivalent ways to implement a desired behavior. For exemple, a sequence can be accomplished using a *sequence* or a *flow* with a link between activities, a choice based on certain data values can be done using the switch or flow elements, etc.

As this dissertation focuses on matching of behavioral interface, next we present an example of a BPEL abstract process. The figure 4.4 presents a BPEL abstract process for a shopping portal. The business strategy of the Web shopping portal has two aspects: On one hand, the portal is open to all $Online\ Shops$. On the other hand, the portal requires the participating shops to build their services according to a standardized protocol, specified in terms of an abstract BPEL process model. The BPEL process models the following behavior: First, the customer should place his order ($ReceiveOrder$ activity, type: receive). Then, the shop may send zero or more questions ($OrderIncomplet$ activity, type: while) to the customer concerning his order ($SendQuestion$ activity, type:invoke) and await his update ($ReceiveUpdate$ activity, type: receive). Eventually, the shop sends the invoice ($SendInvoice$ activity, type: invoke) and re-

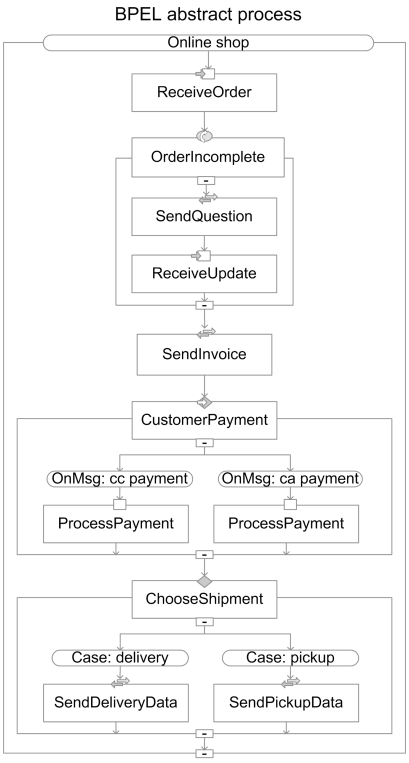

Figure 4.4: Example of a BPEL process

quires the customer to pay ($CustomerPayment$ activity, type: pick), either with credit card (cc payment) or out of his checking account (ca payment). Finally, the shop finishes the process by sending the delivery data or pickup data accordingly to its business strategy ($ChooseShipment$ activity, type: switch). The formats and the channels of messages being exchanged are defined in the Web service description WSDL (Operation, Portype, etc).

There are two major sections in the XML description of the abstract business process for the Online Shops service:

- the *partnerLinks* section defines the different parties that interact with the business process in the course of processing the order.

```
<partnerLinks>
  <partnerLink name="Customer" portType="ep:CustomerPT"/>
  <partnerLink name="InvoicingProvider" partnerLinkType="ep:InvoicingProPLT"/>
  <partnerLink name="PaymentProvider" partnerLinkType="ep:PaymentProPLT"/>
  <partnerLink name="ShippingProvider" partnerLinkType="ep:ShippingProPLT"/>
</partnerLinks>
```

Figure 4.5: Example of a ParterLink description

The four partnerLinks shown here correspond to the sender of the order (customer), as well as the providers of price (InvoicingProvider), Payment (PaymentProvider) and the shipment (Shipping-Provider). Each partner link is characterized by a partner link type. This information identifies the functionality that must be provided by the business process and by the partner service for the relationship to succeed, that is, the *portTypes* that the purchase order process and the partner need to implement (see figure 4.5).

- the process definition section contains the description of the normal behavior for handling a purchase request. The structure of the main processing section is defined by the outer $< sequence >$ element, which states that the five activities contained inside are performed in order ($ReceiveOrder$, $OrderIncomplet$, $SendInvoice$, $CustomerPayment$ and $ChooseShipment$). The figure 4.6 shows the XML description for the *Pick* activity.

```
<pick createInstance="yes" name="CustomerPayment">

   <onMessage name="cc payment" partnerLink="Customer" portType="ep:CustomerPT"  operation="PayCC">
     <invoke name="ProcessPaymentCC" partnerLink="PaymentProvider"
          portType="ep:PaymentProPLT" operation="PaymentCreditCard"/>
   </onMessage>

   <onMessage name="ca payment" partnerLink="Customer" portType="ep:CustomerPT" operation="PayCA">
     <invoke name="ProcessPaymentCA" partnerLink="PaymentProvider"
          portType="ep:PaymentProPLT" operation="PaymentCheckingAccount"/>
   </onMessage>

</pick>
```

Figure 4.6: Example of the XML description of a pick activity

The *onMessage* element is used to receive a particular message from a partner via a port type and operation that the process provides. In this way, the $CustomerPayment$ activity (*pick* activity) waits for a credit card ($cc\ payment$) or checking account ($ca\ payment$) payment incoming message. Whenever one of the specified messages is received, the *pick* activity is completed, and the *ProcessPaymentCC* or *ProcessPaymentCA* activity is executed.

In the remainder of the chapter, we concentrate on the matchmaking of BPEL abstract processes, but the same approach can be adapted to matching executable processes.

4.2 BPEL to Graph Transformation

The parser function presented in this section has the same functionality of parser explained into the section 3.2.2. For instance for BPEL model the parser transforms a behavior model into a process graph. A BPEL graph has at least one start nodes and can have multiple end nodes. The graph has two kinds of nodes : regular nodes representing the activities and connectors representing split and join rules of type XOR or AND. Nodes are connected via arcs which may have an optional guard. Guards are conditions that can evaluate to true or false.

Algorithm 3 Function FlatteningBCF

INPUT: $BPELdocument$
OUTPUT: $BPELgraph$

ADD $Start$ node to $BPELgraph$
ADD End nodes to $BPELgraph$
for each SA_i **do**
  Calculate $SAgraph_i = BCFtransform(SA_i)$
  ADD $SAgraph_i$ to $BPELgraph$
  Calculate the edge $(SAgraph_i, SAgraph_j) = tc(SA_i, SA_j)$
  ADD the edge $(SAgraph_i, SAgraph_j)$ to $BPELgraph$
end for
return $BPELgraph$

We implemented the flattening strategy presented in [80] to transform a BPEL document to a BPEL graph. The general flattening idea is to map Structured activities (SA) to respective BPEL graph fragments (see Algorithm 3). This algorithm traverses the nested structure of BPEL control flow (*BCF*) in a top-down manner and apply recursively a transformation procedure specific to each type of structured activity. First, the *Start* and the *End* nodes are identified into the BPEL document, then for each structured activity the *Structured activity graph* ($SAgraph_i$) is calculated executing the $BCF\ transform$ function on SA_i (see Algorithm 4). Next, the $SAgraph_i$ is added to $BPELgraph$. For each $SAgraph_i$ added to $BPELgraph$ the Algorithm 3 calculates its edges with others structured activity graphs ($SAgraph_j$) considering the mapping tc, which is defined according to the transition conditions of Structured Activites *(SA)* into the BPEL document.

The algorithm 4 shows the $BCFtransform$ procedure which is reinvoked recursively on nested elements. In this algorithm the first parameter represents the structured activity to be processed followed by the *predecessor* and *successor* node of the output Structured Activity Graph ($SAgraph$) between which the nested structure is hooked in (i.e. predecessor and successor).

The $BCFtransform$ procedure starts checking whether the current structured activity (SA) serves as target and source for links. If so, respective connectors are added at the beginning and the end of the current structured activity block. In this way the links into BPEL control flow are mapped to arcs and the respective $join$ and $split$ connectors are added around the nested basic activities. There exist five sub-procedures to handle the five structured activities: *Sequence, Flow, Switch, While, and Pick*. Here, it is assumed that *Pick* is only used to model alternative start events. The transformation of Scopes simply calls the procedure for its nested activity. *Empty* activities map to an arc between predecessor and successor nodes. Terminate is mapped to an end event. Moreover, each BPEL basic activity ($Receive$, $Reply$, $Invoke$, $Wait$) is transformed to a node into BPEL graph. Since our approach is interested into basic BPEL control flow, the *Assing* activities are mapped to an edge between predecessor and successor nodes. Besides this PhD thesis focuses on messages exchanged by partners engaged in a bussines conversation and not in the private behavior of each partner, hence the handling of failures and compensation are not covered in this dissertation. Future works may address activities that interfere with the control flow (e. g. $throw$).

The procedures *BCFtransformSeq*, *BCFtransformFlow*, *BCFtransformPick*, *BCFtransformSwitch* and

BCFtransformWhile generate the BPEL graph elements that correspond to the respective BCF structured activities. *BCFtransformSeq* transforms a *Sequence* by connecting all nested activities with the graph edges. Although not explicitly defined, this transformation requires an order defined on the nested activities. For each sub-activity the *BCFtransform* procedure is invoked again. The graph representation of *Switch (BCFtransformSwitch procedure)* into BPEL graph consists of a block of alternative branches between an XOR split and an XOR join. The branching conditions are associated to the edges.

Algorithm 4 Function BCFtransform

INPUTS: SA
OUTPUT: $SAgraph$

Calculate $BCFtransform(SA, Predecessor, Successor, Connector)$
if $\exists(Link_i, SA)$ **then**
  ADD a $SplitConnector_i$
  ADD an edge between the $Predecessor\ Activity$ and the $SplitConnector_i$
end if
if $\exists(SA, Link_j)$ **then**
  ADD a $JoinConnector_j(Link_j)$
  ADD an edge between the $JoinConnector_j$ and $Successor\ activity$
end if
if $activity \in Seq$ **then**
  Calculate *BCFtransformSeq(SA,Predecessor,Successor)*
else if $activity \in Flow$ **then**
  Calculate *BCFtransformFlow(SA,Predecessor,Successor,AND)*
else if $activity \in Pick$ **then**
  Calculate *BCFtransformPick(SA,Predecessor,Successor,XOR)*
else if $activity \in Switch$ **then**
  Calculate *BCFtransformSwitch(SA,Predecessor,Successor,XOR)*
else if $activity \in While$ **then**
  Calculate *BCFtransformWhile(SA,Predecessor,Successor,XOR)*
else if $activity \in Basic$ **then**
  Calculate *(Activity,Predecessor,Successor)*
else if $activity \in Empty$ **then**
  Calculate *(Predecessor,Successor)*
else if $activity \in Assign$ **then**
  Calculate *(Predecessor,Successor)*
else if $activity \in Terminate$ **then**
  Calculate *(Predecessor,End)*
end if
return $SAgraph$

The *Pick* activity has some similarities to the *Switch*. Yet, instead of evaluating an expression it waits for the occurrence of one out of a set of events and executes the associated activities. These events may be related to time or to message receipts. Syntactically, the *BCFtransformPick procedure* maps to the

same control flow elements as the *Switch*. In the case of *OnMessage* conditions the message is specified with noncontrol flow elements similar to a *Receive* activity. In the case of an *OnAlarm* event the time is modelled similar to the *Wait* activity. Each alternative event is followed by nested activities merged with an XOR join. The *BCFtransformFlow procedure* is transformed to a block of parallel branches starting with an AND split and synchronized with an AND join. For the *While* activity, *BCFtransformWhile* creates a loop between an XOR join and an XOR split, the condition is added to the edge. The transformation of Scopes simply calls the procedure for its nested activity.

The nodes that represent the basic activities have the following attributes: Operation and PortType. The connector nodes have two attributes: ConnectorType (AND-split, AND-join, XOR-split, XOR-join) and ActivityType (the BPEL structured activity from which it was transformed). Figure 4.7 shows the correspondence between BPEL constructs and graph elements.

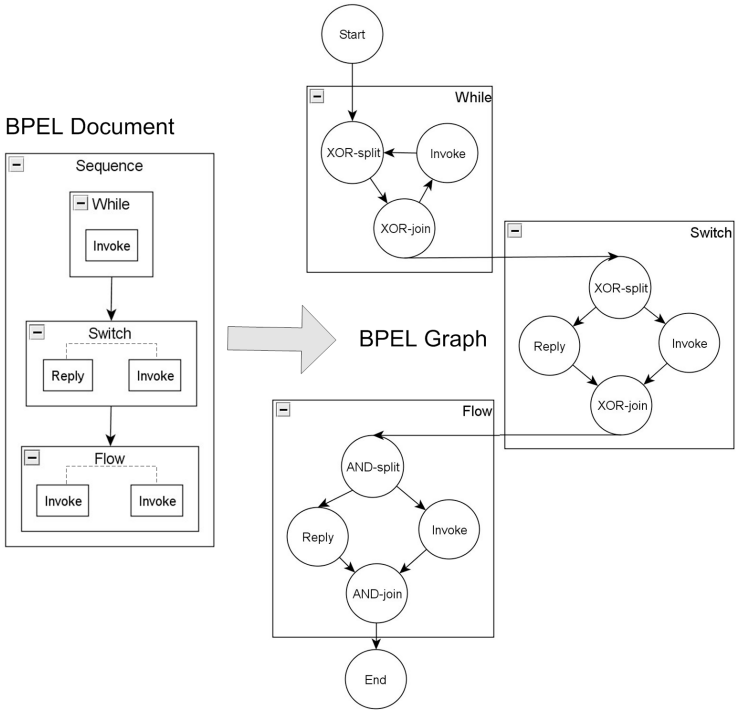

Figure 4.7: Correspondences between BPEL elements and graph elements

4.3 BPEL Graphs Matchmaking

The algorithm presented in this section is based on ec subgraph isomorphism matching for calculating the distance between two graphs (see section 3.1.3). Next, we present the modification that we have made to this algorithm for working with BPEL model.

With the goal of reducing the search space, before executing the matchmaking algorithm the nodes

of two graphs (G and G_I) are well-arranged in sets according to the activity types. In this way only the nodes that belong to the same activity type into G (query graph) and G_I (target graph) respectively are compared (i.e., $Invoke_{syn}$ set, $Invoke_{asyn}$ set, $Receive$ set, $Reply$ set, $Wait$ set). The algorithm starts by mapping the first node included into the first set of G with all the nodes of the same set of G_I and chooses the mapping with minimal cost (Algorithm5, line 1). This represents a partial mapping that will be extended by adding one node at a time (line 7). The process terminates when either a state representing an optimal ec-subgraph isomorphism from G to G_I has been reached or all states in the search space have edit costs that exceed a given acceptance threshold.

The cost of the mapping $C(p')$ (line 7.1) represents the cost of extending the current mapping p with the next node in the query graph. Extending the mapping by mapping a vertex v (in the target graph that has not yet mapped) to a vertex w in the query graph (that does not belong to the current mapping) implies node edit operation and edge edit operations. On the other hand, a process graph has two kinds of nodes: activities and connectors. In contrast with activities, connectors do not represent business functions, they express control flow constraints. In this way first, the label and attributes of v must be substituted by label attributes of w, and secondly, for each mapping $(v', w') \in p$ it must be ensured that any edge (v', v) or any connector between (v', v) in the query graph can be mapped to an edge (w', w) or a connector between (w', w) in the target graph by means of edge and connector edit operations.

Algorithm 5 Error-correcting sub-graph isomorphism detection $(G(V), G_I(V_I))$

1: Initialize OPEN: For each activities set, map the activity node in V onto each activity node in V_I (call ActivityMatch), i.e. create a mapping p. Calculate the cost of this mapping C(p) and add p to OPEN.
2: IF OPEN is empty THEN Exit.
3: Select p of OPEN such that C(p) is minimal and remove p from OPEN
4: IF C(p)> Accept threshold THEN Exit.
5: IF p represents a complete mapping from G to G_I THEN output p. Set accept threshold = C(p). Goto 2.
6: Let p=$\{(v_1,w_i),...,(v_k,w_j)\}$ be the current mapping that maps k nodes from G.
7: FOR *each activity node w in V_I that has not yet mapped to a corresponding node in V*

7.1: extends the current mapping p to p' by mapping the v_{k+1} node of V to w, p'=$\{(v_1,w_i),...,(v_k,w_j),(v_{k+1},w)\}$ and calculate the cost of this mapping C(p')

7.2: CHECK if v_{k+1} and w nodes represent basic or wait activities; THEN execute Function $BasicActivityMatch$ or $WaitActivityMatch$ respectively on (v_{k+1},w). VERIFY if minimal $DistanceNode >$ composition or decomposition threshold, and apply decomposition or decomposition operation if necessary. Finally, ADD $DistanceNode$ to C(p').

7.3: CHECK if $\exists$ an Edge or a Connector between (w',w) mapped nodes; THEN execute Function $EdgeCost$ or $ConnectorCost$ respectively. ELSE $SuppressionCost$ on (v',v) is applied. Finally, ADD $Distance(Edge, Connector\ or\ Supp)$ to C(p')

8: Goto 2

In line 7.2 we consider the cost of mapping two basic ($Invoke$, $Receive$, or $Reply$) or two $wait$ activities (We make the activities distinction as the attributs of $wait$ activity and the other activities are different). Exactly, this line calls the algorithms that implement these cost functions. Then in the same

line 7.2 the algorithm verifies if minimal $DistanceNode >$ composition or decomposition threshold, and next the decomposition or decomposition operation is applied if necessary. The decomposition operation is applied on *Invoke* activity of type request/response. The composition operation is applied on asynchronous communication pattern [*Invoke*(one way) + *Receive*] consecutive. Finally, $DistanceNode$ is added to C(p'). On the other hand the matching process compares the connectors in a manner similar to edges, when mapping edges between two activity nodes, we map also the possible connectors binding directly to the nodes. That is, first the algorithm verifies if there exist an edge or a connector between the mapped nodes (w', w) of target graph (Line 7.3) and then, it calls the function that calculates the cost of adding or deleting an edge, or the function that implements the cost of inserting, substituting or suppressing a connector. Finally the costs are added to C(p'). In the next sections we explain in detail how each function is implemented.

4.4 Comparison Rules of BPEL Graphs

In the same way as in section 3.2.4 the *Comparison rules* depict all the application-dependent functions allowing to calculate the cost of graph edit operations. These functions are used by the graph matching module for calculating the distance between the graphs. In the following we explain the cost functions used for BPEL protocol matchmaking.

4.4.1 Matching Edges.

For any mappings $M(v, w)$ and $M'(v', w')$ (where $\{v, v'\}$ nodes $\in$ query graph and $\{w, w'\}$ nodes $\in$ target graph) and given that exists an edge between (v, v') nodes, the Algorithm 6 analyzes if the costs of inserting, substituting an edge or deleting a connector between (w, w') nodes are necessaries.

Algorithm 6 Function EdgeCost

INPUTS: $M(v, w); M'(v', w')$
OUTPUT: $DistanceEdge$

For the mappings $M(v, w); M'(v', w')$ where (v, v') nodes $\in$ query graph AND (w', w) nodes $\in$ target graph; and given an edge between (v, v') nodes then:

if $\exists$ an edge between the nodes (w, w') **then**
  return $DistanceEdge = 0$
else if $\nexists$ an edge between the nodes (w, w') **then**
  return $DistanceEdge = C_{ei}$
else if $\exists$ a connector between the nodes (w, w') **then**
  return $DistanceEdge = C_{cd} + C_{ei}$
end if

Therefore, the algorithm considers three cases:

- If there exist an edge between (w, w') nodes, the $DistanceEdge = 0$ is returned.

- If does not exist an edge between (w, w') nodes, a cost of inserting an edge between (w, w') is returned $DistanceEdge = C_{ei}$.
- If there exists a connector between the nodes (w, w'), the cost of deleting a connector (C_{cd}) and inserting an edge between (w, w') are returned ($DistanceEdge$=C_{cd} +C_{ei}).

4.4.2 Matching Connectors.

For any mappings $M(v, w)$ and $M'(v', w')$ (where $\{v, v'\}$ nodes $\in$ query graph and $\{w, w'\}$ nodes $\in$ target graph) and given that exist a connector between (v, v') nodes, the Algorithm 7 analyzes if the costs of inserting, substituting a connector or deleting an edge between (w, w') nodes are necessaries.

Algorithm 7 Function ConnectorCost

INPUTS: $M(v, w); M'(v', w')$
OUTPUT: $DistanceConnector$

For the mappings $M(v, w); M'(v', w')$ where (v, v') nodes $\in$ query graph AND (w, w') nodes $\in$ target graph; and given a connector between (v, v') nodes then:

if $\exists$ a connector between the nodes (w, w') **then**
  if the connector types between(v, v') and (w, w') are different **then**
    return $DistanceConnector = C_{cs}$
  else if the connector types are same **then**
    return $DistanceConnector = 0$
  end if
else if $\nexists$ a connector between the nodes (w, w') **then**
  return $DistanceConnector = C_{ci}$
end if
if $\exists$ an edge between the nodes (w, w') **then**
  return $DistanceConnector = C_{ed} + C_{ci}$
end if

Therefore, the algorithm considers three cases:

- If there exists a connector between (w, w') nodes, the algorithm analyzes the connector types. If the connector types between(w, w') and (v, v') are different, a cost of connector substituting is returned (C_{cs}) other wise $DistanceConnector = 0$.
- If does not exist a connector between (w, w') nodes. In this case, a cost of inserting a connector between (w, w') is returned $DistanceConnector = C_{ci}$.
- Finally, if there exists an edge between the nodes (w, w'), the cost of deleting an edge (C_{ed}) and inserting a connector between (w, w') are returned ($DistanceConnector$= C_{ed} +C_{ci}).

4.4.3 Suppression Function.

For any mappings $M(v, w)$ and $M'(v', w')$ (where $\{v, v'\}$ nodes $\in$ query graph and $\{w, w'\}$ nodes $\in$ target graph) and given that does not exist neither an edge nor a connector between (v, v') nodes, the Algorithm 8 analyzes if the costs of deleting a connector or an edge between (w, w') nodes are necessary.

Algorithm 8 Function Suppression

INPUTS: $M(v, w); M'(v', w')$
OUTPUT: $DistanceSupp$

For the mappings $M(v, w); M'(v', w')$ where (v, v') nodes $\in$ query graph AND (w, w') nodes $\in$ target graph; and given that does not exist neither an edge nor a connector between (v, v') nodes then:

if $\exists$ an edge between the nodes (w, w') **then**
  return $DistanceSupp = C_{ed}$
else if $\exists$ a connector between the nodes (w, w') **then**
  return $DistanceSupp = C_{cd}$
else if $\nexists$ neither a connector nor an edge between the nodes (w, w') **then**
  return $DistanceSupp = 0$
end if

Therefore, the algorithm considers three cases:

- If there exists an edge between (w, w') nodes, the cost of deleting an edge is returned ($Distance\ Supp = C_{ed}$).
- If there exists a connector between (w, w') nodes, the cost of deleting a connector is added ($Distance\ Supp = C_{cd}$).
- If does not exist neither a connector nor an edge between (w, w') nodes, so $Distance\ Edge$ =0.

4.4.4 Matching Basic Activites.

The cost for editing a basic activity vertex (*receive, invoke, reply*) is calculated by function BasicActivityMatch (see Algorithm 9). This cost expresses the distance between two BPEL basic activities. Each activity has two attributes: the Operation name (Op) and the PortType (PT). The matchmaking gives priority to operation comparison, and if two operations are similar ($SimOperation > 0$), it compares the similarity of the $PortType$ and calculates the distance between activities ($DistanceNode$).

Weights w_{op} and w_{pt} indicate the contribution of Op (similarity of Operations) and PT (similarity of PortTypes) respectively in the total $DistanceNode$ score ($0 \leq w_{op} \leq 1$ and $0 \leq w_{pt} \leq 1$).

Further, the *Reply* activities must always be preceded by a *Receive* activity with the same partner link, portType and (request/response) operation, such that no reply has been sent for that *Receive* activity. So, given two mapped *Receive* activities, the function $BasicActivityMatch$ will be applied only on *Reply* activities that correspond to the previously mapped *Receive* activities.

Algorithm 9 Function BasicActivityMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (Opi, PTi), Nodej: Struct (Opj,PTj)
OUTPUT: $DistanceNode$

Calculate Operation Similarity $SimOperation = LS(Opi, Opj)$
if $SimOperation = 0$ (different Operations) **then**
  Return $DistanceNode = 1$
else
  Calculate PortType Similarity $SimPortType = LS(PTi, PTj)$
  Calculate $DistanceNode$

$$DistanceNode = 1 - \frac{w_{op} * SimOperation + w_{pt} * SimPortType}{w_{op} + w_{pt}}$$

end if

4.4.5 Matching Wait Activities.

This function (see Algorithm 10) calculates the cost for editing a vertex which represents a *wait* activity. Each wait vertex has two attributes: a delay for a certain period of time (F) or until a certain deadline is reached (U). The function checks if two $ForExpressions$ or two $UntilExpressions$ are similar, and gives a result for $DistanceNode$ respectively. The time similarity function (TS) calculates the resemblance between the time expressions giving preference to the closest one. The following time expressions are considered: $PnYnMnDTnHnMnS$ for FOR expressions (i.e. 1 year, 3 months, 5 days, 8 hours, 45 minutes, and 10 seconds = $P1Y3M5DT8H45M10S$) and $CCYY - MM - DDThh : mm : ssZ$ for UNTIL expressions (i.e. 10:25 p.m. and 12 seconds UTC, January 31, 2005 = $2005 - 01 - 31T22 : 25 : 12Z$).

Algorithm 10 Function WaitMatch

INPUTS: (Nodei,Nodej)
Nodei: Struct (Fi, Ui), Nodej: Struct (Fj, Uj)
OUTPUT: $DistanceNode$

if $ForExpression$ there exist **then**
  Calculate ForExpression Similarity $SimFor = TS(Fi, Fj)$
  Calculate $DistanceNode = 1 - SimFor$
else
  Calculate UntilExpression Similarity $SimUntil = TS(Ui, Uj)$
  Calculate $DistanceNode = 1 - SimUntil$
end if

4.5 Compostion and Decomposition of BPEL Basic Communication Patterns

We have seen that the classical set of edit operations of error correcting subgraph detection, consisting of the deletion, insertion, and substitution of nodes and edges, is powerful enough to transform any two given graphs into each other. Despite their theoretical power, it can be argued that these edit operations are not perfectly suit for all problem domains. For instance for the BPEL matchmaking process, it is possible that when we compare two services, in one service a message exchange is modelled as a synchronous interaction, while in the second process is modelled as an asynchronous interaction, therefore a decomposition operation is considered for representing a synchronous interaction as an asynchronous interaction. If we have the contrary case in which the first process has an asynchronous interaction, while into the second process the same interaction is modelled with a synchronous pattern, a composition operation is necessary.

The table 4.1 shows how a BPEL message exchange can be modelled as a basic asynchronous or synchronous interaction for an operation invoked by the process.

Synchronous interaction	**Asynchronous interaction**
Invoke (request/response)	Invoke (one way) + Receive

Table 4.1: Synchronous vs. asynchronous interactions

Exactly, the decomposition and composition operations take importance when the graph matchmaking algorithm does not find an acceptable mapping cost between the BPEL activities that represent a basic synchronous or asynchronous communication pattern.

In this way if the minimal cost of a mapping $M(v, w)$ (where v node $\in$ query graph AND w node $\in$ target graph) exceeds a threshold and v represents an Invoke (request/response), the decomposition operation is applied on w for verifying if the equivalent asynchronous pattern ($w_1 \rightarrow w_2$) is found by the algorithm to have a better value of total distance between the two graphs. This edit operation changes target graph by replacing node w by two new nodes w_1 (*Invoke* (one way)) and w_2 (*Receive*). Moreover, edges between the new nodes or between the new nodes and nodes that existed before the decomposition operation must be inserted. The contrary case is applied for the composition operation.

For instance, the costs tree of figure 3.1 represents the error correcting subgraph isomorphism between the query graph g_2 and the normal target graph g_1. If the node 3 of target graph is decomposed, two new nodes (g' and g'') and one edge between them must be inserted. Therefore the costs tree is expanded in two new branches, one for g' and another for g''. These branches are originated in the same level of the node 3. Then, the algorithm will search for the path with minimal value from the costs tree.

Further, there exists several interaction cases between a BPEL process and another application as: Asynchronous Interaction with Timeout or with a Notification Timer, One Request-Multiple Responses, One Request-One of Two Possible Responses, One Request-a Mandatory Response-and an Optional Response, Partial Processing, Multiple Application Interactions; but our approach considers the basic synchronous and asynchronous interactions only, because the advanced interaction analysis increases the complexity of error-correcting sub-graph isomorphism detection. This means that the proposed solutions to vertex composing and decomposing require comparison not only of the original graph nodes but also of the composing and decomposing nodes that they originate. Assuming the computational complexity of a traditional error-correcting sub-graph isomorphism detection problem to be $O(m^n n)$ (n and m being the number of nodes of the two graphs), the complexity of the composition solution scales to

$O(m^{(n+\xi)}(n+\xi))$ ξ being the number of nodes originated from each decomposition and composing operation respectively.

In the next paragraphs we will present in detail each operation.

Decomposing vertices An *Invoke* activity of type request/response (having m_{in} as input message and m_{out} as output message) can be decomposed in an *Invoke* activity (one way, having message m_{in}) and a *Receive* activity (having m_{out} as message). The new activities *Invoke*(one way) and *Receive* take the Operation and PortType attributes of *Invoke*(request/response) activity.

The algorithm 11 shows the decomposition operation wich is executed during the matching process. This one is organized in the following way:

- first the nodes of two graphs are compared using the function $BasicActivityMatch$
- if the minimal cost of a mapping (v, w) between the *Invoke* (request/response) activity v of query graph and the *Invoke* (request/response) activity w of target graph exceed the decomposition threshold ($Distance\ Node(v, w) >$ Threshold), then the decomposition operation is executed on *Invoke* (request/response) vertex (w) of the target graph.
- afterwards the decomposed activities (w_1 and w_2) are added to OPEN. Therefore the costs tree is expanded in two new branches, one for w_1 and another for w_2.

Algorithm 11 Function Decomposition

INPUTS: $M(v, w)$ where $v = Invoke_{synchronous}$ activity $\in$ query graph; $w = Invoke_{synchronous}$ activity $\in$ target graph
OUTPUT: ($w_1 \rightarrow w_2$) decomposed asynchronous pattern; where w_1 is an *Invoke* activity (one way) and w_2 is a *Receive* activity

Calculate Function $BasicActivityMatch(v, w)$
if minimal $DistanceNode(v, w) >$ Threshold **then**
  Calculate the $Decomposition(w) = (w_1 \rightarrow w_2)$
  ADD the mappings of the ($w_1 \rightarrow w_2$) decomposed pattern to OPEN
else
  EXIT
end if

For instance, suppose that into a BPEL model a submission process requires two activities: an *Invoke* (one way) activity (*SubmitOrder*) and a Receive activity (*ConfirmationOrder*), but in a second BPEL model the same process is modeled as one *Invoke* (request/response) activity (*SubmitOrderRequest*). Since we search a correspondence betwen the two BPEL models, then the *Invoke* (request/response) activity that represents the Order submission in the second BPEL model can be decomposed in one *Invoke* (one way) activity followed of a *Receive* activity.

Composing vertices The vertex composition is the inverse operation to vertex decomposition. This operation replaces a set of connected vertices with one vertex. The new vertex replaces the old ones and inherits their properties. For BPEL metamodel the vertices composition is applied on asynchronous communication pattern [*Invoke*(one way) + *Receive*] consecutive (Table4.1). The new activity *Invoke*(request/ response) takes the Operation and PortType attributes of *Invoke*(one way) activity.

In the same way of decomposition opreration, the composition operation is executed during the matching process. In order to notice the composition of the vertices, the matching process is organized as an iterative process that performs the following actions (see Algorithm12), at each iteration step:

- first the nodes of two graphs are compared using the function $BasicActivityMatch$
- if the minimal $DistanceNode$ between two *Invoke* (one way) activities (v, w) exceeds a composition threshold ($Distance\ Node(v, w) >$ Threshold), then the algorithm verifies if the *Invoke* activity (one way) w in the target graph is followed by a *Receive* activity p.
- if the *Invoke* activity (one way) w is followed by a *Receive* activity p, then the composition operation is executed on asynchronous communication pattern ($w \rightarrow p$).
- afterwards the composed *Invoke* (request/response) activity (c) is added to OPEN. Therefore the costs tree is expanded in one new branch.

Algorithm 12 Function Composition

INPUTS: $M(v, w)$;q where v is an $Invoke_{asynchronous}$ activity $\in$ query graph; w is an $Invoke_{asynchronous}$ activity $\in$ target graph; p is a $Receive$ activity $\in$ target graph.
OUTPUT: (c) composed synchronous pattern; where c is the *Invoke* activity (request/response)

Calculate Function $BasicActivityMatch(v, w)$
if minimal $DistanceNode(v, w) >$ Threshold **then**
  Verify if w is followed by a Receive activity p
  if w is followed by a Receive activity p **then**
    Calculate the $Composition(w, p) = (c)$
    ADD the mappings of the (c) composed pattern to OPEN
  else
    EXIT
  else
    EXIT
  end if
end if

Suppose that a BPEL graph describes a loan process, in this graph an *Invoke* activity (initiate service) initiates the *loan request*. The contents of this request are put into a request variable. This request variable is sent to the asynchronous loan processor Web service. The loan process into Web service then sends the correct response to the *Receive* activity (Wait for call-back) into de BPEL graph. In this example, the *Invoke* (one way) activity that represents the *loan request*, and the *Receive* activity that waits for the *call-back*, can be composed into *Invoke* (request/response) that contains both activities.

4.6 An Example for the BPEL Matchmaking

We will exemplify the BPEL matchmaking by comparing two BPEL processes for hotel reservation.

- Suppose that the first service has the following activities: first, the customer should place his hotel selection *Reservation Request* (Activity type: Receive). Then, either ShowCatalog or ShowAvailability message are expected via *Hotels information* (Activity type: Pick). Next, the (*RequestCatalog*, Activity type: Invoke) or ShowAvailability information (*RequestAvailability*, Activity type: Invoke) activities are executed respectively. Afterwards, a confirmation (*UserConfirmation* Type: Reply) with the reserve information is sent to user. Finally, the hotel reservation service expects for the credit card payment *PaymentCC* (Type: Receive).

- The second service model has the following activities sequence: first, the customer should place his *Reservation* (Activity type: Receive) preferences. Then the hotel reservation service receives the customer reservation dates (*Show Availability* Type: Receive) and verifies the hotels availability (*CheckAvailability* Type: Invoke); if there are no rooms available for the proposed dates, the last two operations are repeated until finding available rooms. Next, a confirmation (*Confirmation* Type: Reply) is sent to user. Finally, the hotel reservation service requires the customer to pay (*Payment* Type: Switch), either with credit card (are *PaymentCC* Type: Receive) or out of his checking account (*PaymentCA* Type: Receive).

Our system converts a BPEL document into a graph (query graph and target graph in Figure4.8) using the BPEL parser function. Next, the graphs are compared by the graph matchmaking algorithm. The $EdgeCost$, $ConnectorCost$, $Suppression$, $BasicActivity\ Match$ and $WaitMatch$ functions will be invoked for comparing the activities and connectors nodes.

For each pair of activity nodes, first the $BasicActivityMatch$ and $WaitMatch$ functions will verify if there exist a similarity between its operations. For example, when comparing *Reservation* and *ReservationRequest* activities (Type: Receive), the best mapping between the Reservation and ReservationRequest operations is found respectively using the linguistic similarity function LS. As there exist a similarity between them, the LS function calculates the similarity between the PortTypes (portType:$ResvPT$ and portType:$ResvRqPT$), and finally, the total distance is returned. For this example we have considered that the similarity of operation (w_{op}) has the same importance than the similarity of portType (w_{pt}). With the goal of reordering the query graph with respect the target graph, for each mapping found the system applies the $EdgeCost$, $ConnectorCost$ and $Suppression$ functions. The dotted lines in Figure4.8 represent the mappings detected by the system between the two graphs.

In conclusion, the edit script will show that the two graphs have some common activities, but the activities *ShowAvailability*, *CheckAvailability* and *PaymentCC* of the query graph are parts of different structured activities in the target graph. However, the matchmaking algorithm will find similar activities for the right branch of the target graph (*Start*, *ReservationRequest*, *ShowAvailability*, *RequestAvailability*, *UserConfirmation*, *Payment* and *End*). In this example the decomposition and composition operations are not applied.

Finally, the script of the graph edit operation is the following:

- Deleting the edge (*CheckAvailability*,*XOR-Split*) from query graph

- Deleting the edge (*XOR-join*,*ShowAvailability*) from query graph
- Inserting the edge (*XOR-Split*,*ShowAvailability*) into query graph
- Inserting the edge (*CheckAvailability*,*XOR-join*) into query graph

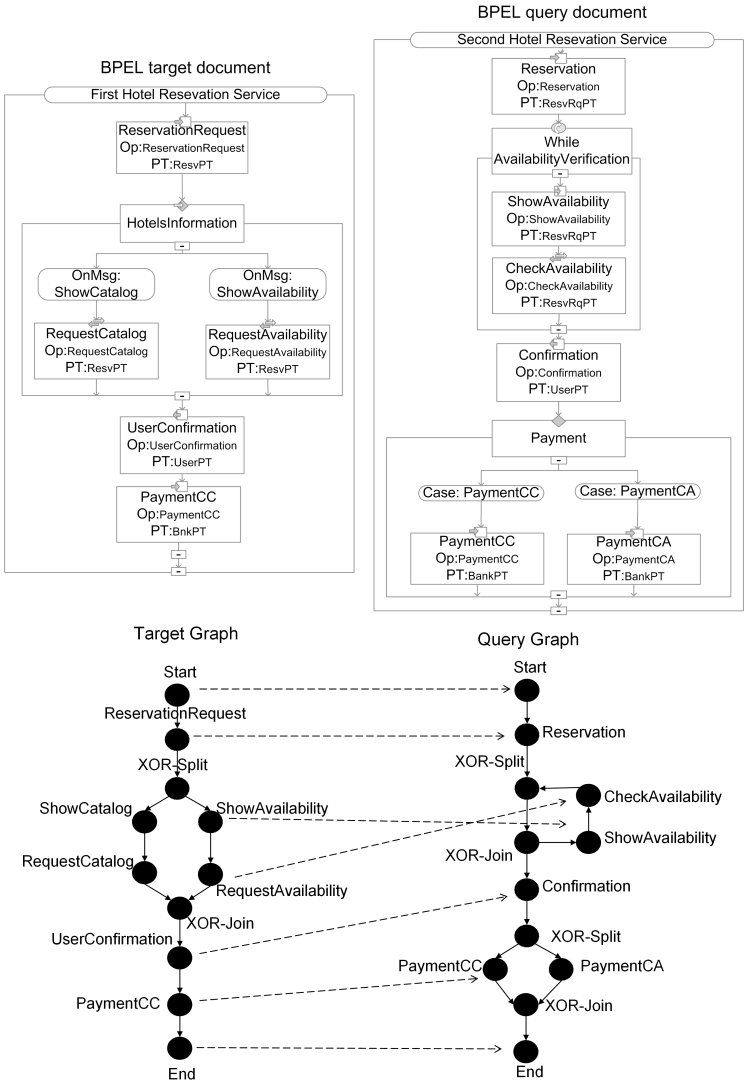

Figure 4.8: BPEL matchmaking example

4.7 Summary

Considering the importance and extensive utilization of BPEL protocol for service description, in this chapter, we discussed our approach for Behavioral matchmaking, by examining the usage of matching techniques in the context of BPEL behavioral specifications of the service. The BPEL matchmaking process is composed of the following steps. First, the BPEL documents to be compared are transformed to graphs. Next, the error correcting graph matching algorithm is applied (considering the decompostion and composition functions during the algorithm execution). Then, the similarity function evaluates the similarity between the graphs. So, in this chapter first we introduced the BPEL protocol, then we explained the BPEL to graph transformation. In the section 4.3, we showed the BPEL matchmaking algorithm which is based on the algorithm introduced in previous chapter, but considering the comparison rules for the BPEL metamodel. Finally, an example of the BPEL matchmaking process was depicted.

Chapter 5

Prototype and Experimentation

Chapters 3 and 4 described our proposal for Behavioral matchmaking for service retrieval. In this chapter we illustrate its practical use with real services. To this end, we have developed a prototype called Ws-BeM (Web services-Behavioral Mathcmaking), which implements the proposed approaches. The tool allows the execution of the algorithms for matchmaking services in the context of service ranking.

In order to validate our approach, the prototype has been tested in two application scenarios: the matching of BPEL and WSCL protocols. For each of these applications, we briefly describe its principle and how Ws-BeM has been used. The validation of our approach in these application scenarios is twofold. Firstly, we want to analyze the matching process quality by using different application scenarios into WS-BeM. Secondly, we want to test the execution time of matchmaking method using these protocols.

Further, we have constructed a tool for evaluating the effectiveness of our behavioral matchmaking method. This is a tool that allows to create a user service ranking based on manually comparisons between a query service and the services in the repository. The tool permits to compare the result obtained by the platform and the ranking defined by users.

The following sections describe our prototype and experimentations: Section 5.1 presents the We-BeM tool, describing its functionalities, architecture and its user interface. Section 5.2 shows the tool for evaluating our matching method, depicting its functionalities, architecture and its user interface. Finally, the section 5.3 presents the performance evaluation tests, describing the considered test application scenarios, the test strategies and the obtained results.

5.1 Platform for Service Matchmaking

We developed a prototype system that implements our approach of Behavioral matchmaking for services retrieval. The tool was built using the Java programming language (JDK 1.6.0) and the Netbeans 5.0 Integrated Development Environment (IDE). We created as desktop prototype, but this one is also available as a web service that takes as input two WSCL or BPEL files and calculates the similarity between them (`http://ariadna.unicauca.edu.co/matching/`). It returns also the script of edit operations required in order to transform the first protocol to conform with the second one.

In the remainder of this section, first we describe the functionalities provided by the system. Then, we describe the system architecture and finally, we depicted the user interfaces provided by the prototype.

5.1.1 System Functionalities

Given a set of published services having equivalent functionalities (corresponding to a given domain, for example, trip reservation services), the goal of WS-BeM platform is to rank services with respect to their suitability in fitting user requirements. We suppose that the user expresses his needs as a service behavior model and the platform will help him identifying the services having the most similar behavior model.

In our platform the service ranking is based on service behavioral matching presented in chapter 3 and 4, which is reduced to a graph matching problem.

The services ranking is constructed taking into account different measures (graph edit distance, simple similarity and similarity based on: number of mapped nodes and number of mapped node sequences). The system is presented in Figure 5.1. This is composed of the following modules:

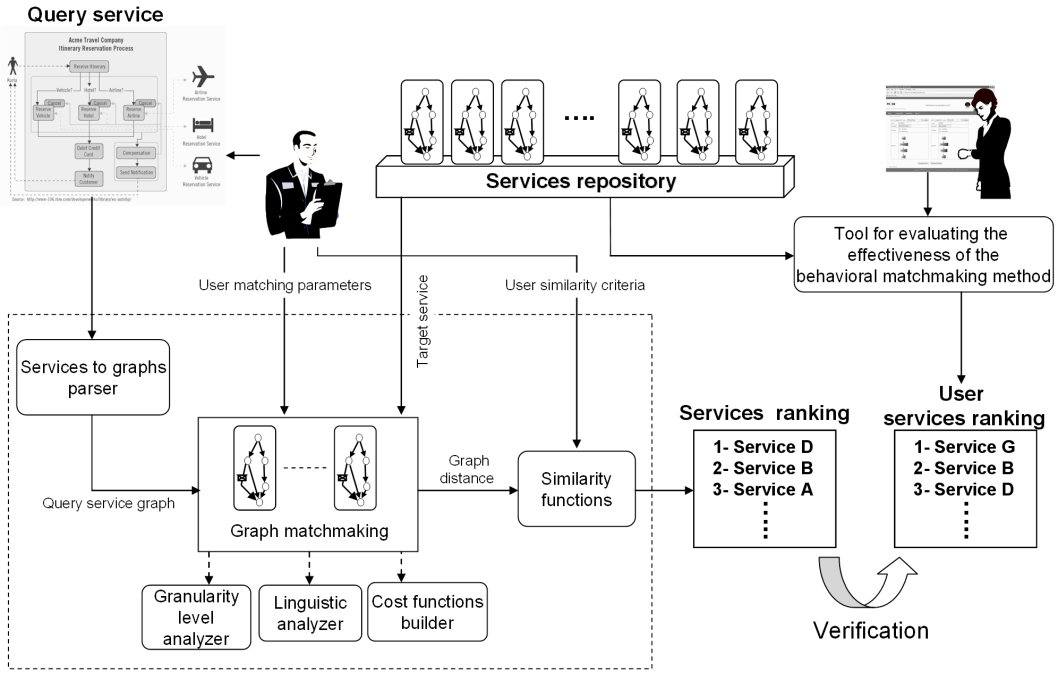

Figure 5.1: Platform for service ranking based on behavioral matchmaking

- Services to graphs parser: This module transforms a service behavior description (e.g., BPEL or WSCL) to a graph.
- Graph matchmaking: This module takes as inputs the two graphs produced by the parser presented above and finds out the semantic distance between them based on the error correcting sub-graph isomorphism with minimal cost.
- Cost functions builder: This module groups the cost functions for the graph edit operations that allow to calculate the distance between graphs. The costs assigned to different graph edit operations reflect the relative importance of dissimilarities between different graph attributes. Thus they depend of service behavior metamodel and on the application domain.

- Linguistic analyzer: This component calculates the linguistic similarity between two strings using the following algorithms: NGram, Check synonym, Check abbreviation and tokenization.
- Granularity level analyzer: It checks whether decomposition/composition operations are necessary and add their cost to the distance measure. These graph edit operations are necessary when the same functionality is modeled at different granularity levels in the two graphs (for example, using two nodes in a graph and only one node in the other graph).
- Similarity functions: This module defines the similarity functions that allows to construct the service ranking. It uses the result of the graph matchmaking module (the edit distance, the node mappings, etc.).
- Tool for evaluating the effectiveness of the behavioral matchmaking method. This is a tool that allows to create a user service ranking based on manually comparisons between a query service and the services in the repository. The tool permits to compare the result obtained by the platform and the ranking defined by users. Given the fact that the parameterization of the cost function is domain dependent and very important for the effectiveness of the matchmaking method, it is important to have a tool allowing to determine the optimal parameters to use for a given domain and similarity criteria. In the section 5.2 we will explain in detail this tool.

5.1.2 System Architecture

The logic architecture presented in figure 5.2 organizes the software classes into packages, subsystems, and layers. This one is implemented in three different layers (Application, Mediation and Foundation [65]) . Figure 5.2 shows the layers of the architecture and the interaction among them, as well as the most relevant packages that compose each layer.

- **The Application layer** manages the packages that implement the prototype functionalities. This layer is composed by the following packages:
  * *Service parser*: this package allows to register the functions that transform the services metamodel consumed by the user to a graph. For instance for We-BeM we have implemented the WSCL and BPEL functions, but it is possible to register other function as WS-CDL.
  * *Graph matchmaking*: this package contains the classes that implement the error correcting subgraph isomorphism detection. This package uses the *Similarity analyzer, Granularity analyzer, Cost function builder, Linguistic analyzer* packages for calculating the matching between the two service graphs.
  * *Granularity analyzer*: contains the decomposition and composition classes for each metamodel.
  * *Cost function builder*: this package allows to register the comparison rules for matching the service metamodels.
  * *Linguistic analyzer*: contains the classes that implement the algorithms for calculating the similarity between the two character strings. For instance for WS-BeM we have used the Ngram, Token, Sinonym and Abbreviation algorithms, but other algorithms can be registered.

* *Similarity analyzer*: has within the functions that allows to calculate the similarity based on the total distance between the two graphs. In the same way of above packages, this one allows to register other similarity functions.
* *Graphical user interface*: In order to achieve a visual representation, this package contains all classes that implement the graphical interfaces of the prototype.

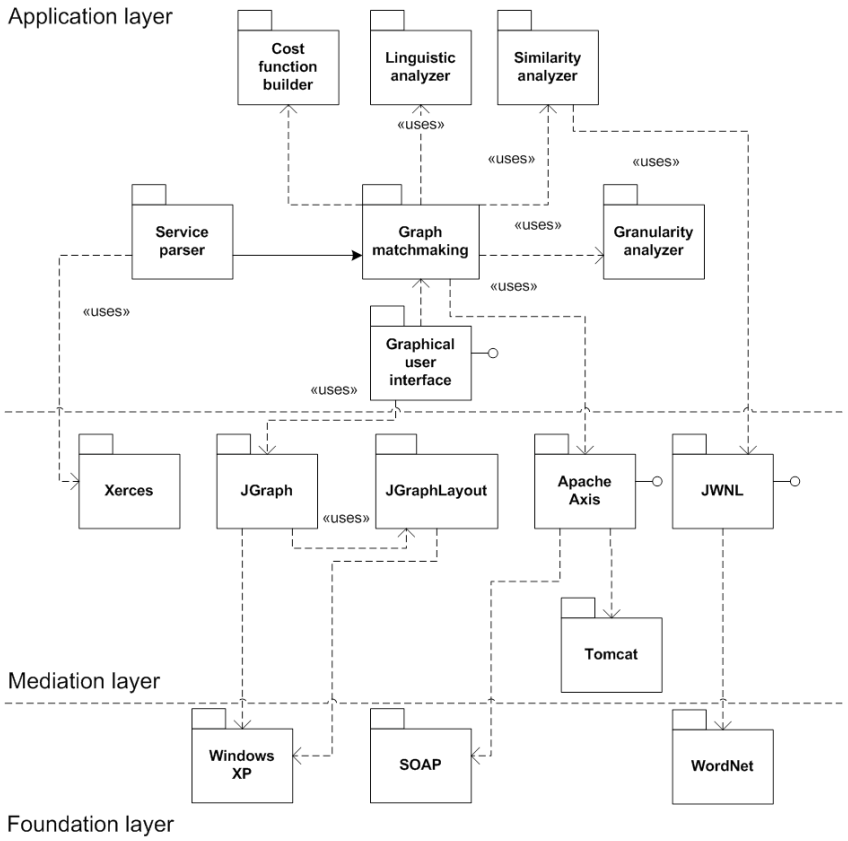

Figure 5.2: Logical architecture of the prototype

- **Mediation layer** contains all application program interfaces (APIs) used by the prototype. The layer is composed by the following packages:

  * *Xerces*: is a family of software packages for parsing and manipulating XML. The library implements a number of standard APIs for XML parsing, including DOM, SAX and SAX2. In this manner, Xerces package supports the service parser package.
  * *Apache Axis*: is an open source, XML based Web service framework. This consists on a Java and C++ implementation of SOAP server, and several utilities and APIs, to generate and deploy Web service applications. Using Apache Axis, the graphs matchmaking can be published as a Web service.

  * *JWNL*: (Java WordNet Library) is an API for accessing WordNet-style relational dictionaries. It also provides functionality beyond data access, such as relationship discovery and morphological processing.
  * *JGraph*: is a Java Graphing framework that fully complies with Swing design principles. It contains all the graph visualization and interaction functionality of a graph library. This package supports the visualization of the graphical user interface.
  * *JGraph Layout*: is a high performance graph layout library for JGraph that automatically positions the graph, diagram, or network in a visually pleasing manner.

- **Foundation layer** includes the basic software that enables the prototype performance . This layer is composed of the following packages:

  * *SOAP*-Simple Object Access Protocol: is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can be built on. This package supports the publication of graphs matchmaking as a Web service.
  * *WordNet*: is a semantic lexicon for the English language. It groups English words into sets of synonyms called synsets; Provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications. This package is used by Linguistic analyzer (through the JWNL package) for verifying the semantic relationship between two words.
  * *Tomcat*: implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. In this manner, Tomcat supports the Apache Axis package.
  * *Windows XP professional*: is the operating system that supports the prototype.

5.1.3 User Interfaces

In this section, we present the graphical user interfaces (GUI) of WS-BeM through two screen snapshots: one showing the panel of WSCL mathcmaking and the other showing the panel of BPEL mathcmaking. The mathcmaking process begins when the user loads the WSCL or BPEL files, then the parser WSCL or BPEL is executed respectively. Next, the user selects the decomposition function, this is a particular step to WSCL metamodel since for BPEL metamodel the decomposition and composition functions are executed during the matching process. The fourth step allows the user to assign the costs for the graph edit operations. Finally, the matching result is presented to the user. The figures 5.3 and 5.4 show the graphical interfaces in which user can choose the WSCL or BPEL documents to be compared. The interface shows also the graphs resulted from parsing the documents. Into the figure5.3 the atomic nodes (Send or Receive) and the non-atomic nodes (ReceiveSend or SendReceive) have different colors. In the same way, the interface of figure 5.4 depicts the connector and basic activity nodes with different colors.

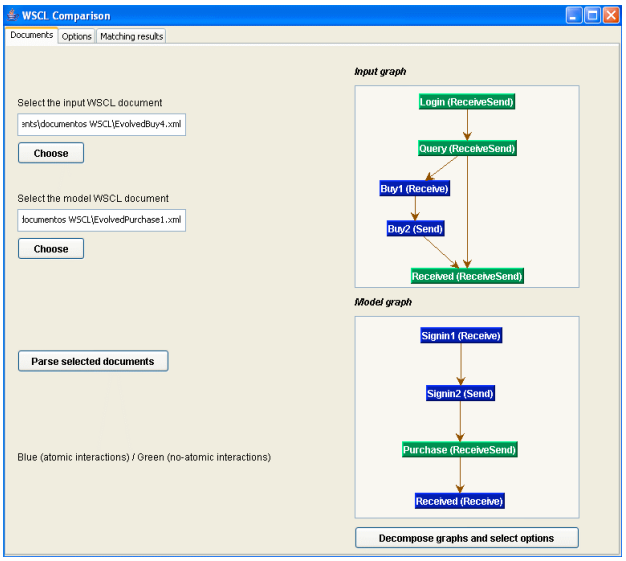

Figure 5.3: WSCL documents interface

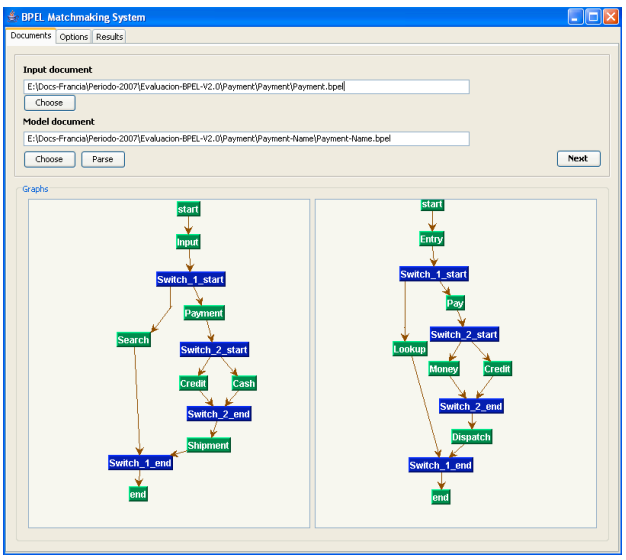

Figure 5.4: BPEL documents interface

In the graphical interface of the figure 5.5 user can set the costs for the WSCL graph edit operations (Interaction and Transitions). The costs of: the granularity, the similarity weight for names and document interactions (W_i and W_d), and the threshold value for the graph edit distance (Acceptable value), are fixed too in this interface. On the other hand, the interface shows the decomposition of input and model graph (first and second graph). In the same way, figure 5.6 allows to insert the cost for the BPEL graph edit operations (Activities, Connectors and Transitions). The costs of: similarity weight for PortType and OperationType (W_{pt} and W_{op}), and the threshold values for the composition and decomposition functions, are fixed too in this interface.

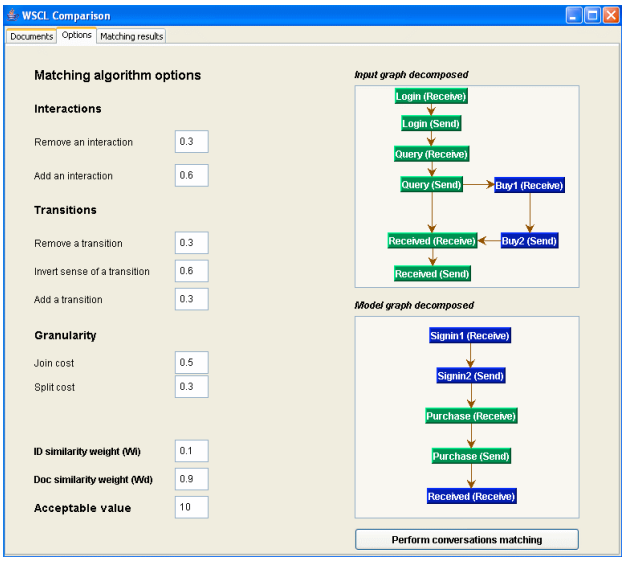

Figure 5.5: WSCL options interface

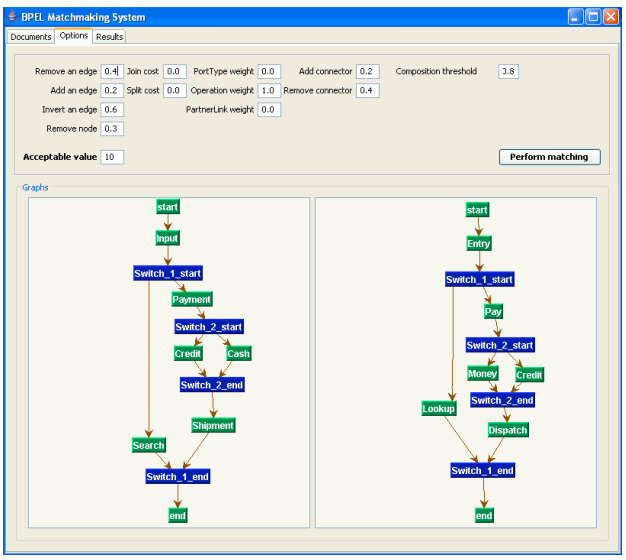

Figure 5.6: BPEL options interface

In [34], authors argue that the edition operation costs are defined in *ad doc* manner, purely guided by heuristics and intuition. Therefore in the section 5.3 we center our effort on the parameterization of costs function to give the user an optimal matching result.

Finally, the figures 5.7 and 5.8 show the results of matching algorithm. These results are displayed in a graphical and textual way.

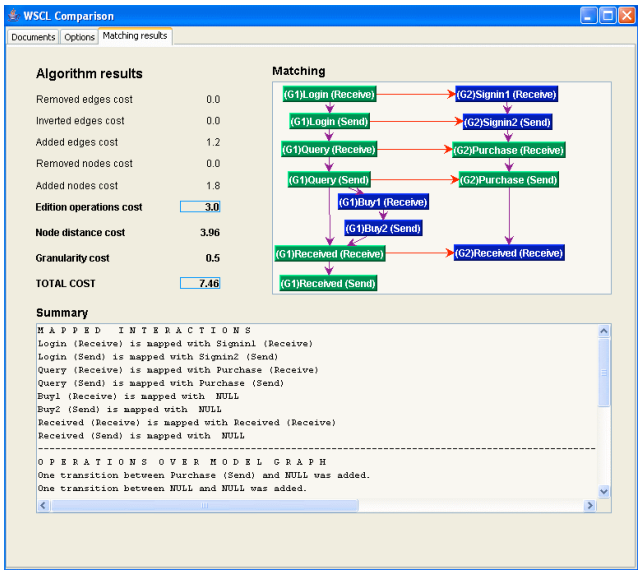

Figure 5.7: WSCL matching results interface

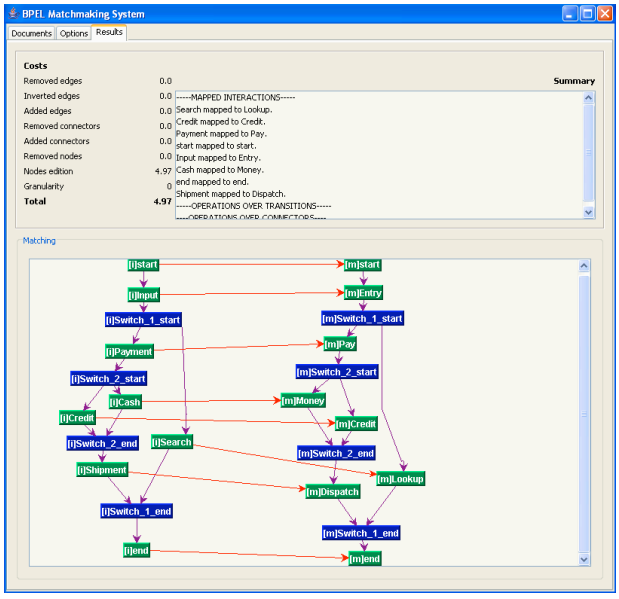

Figure 5.8: BPEL matching results interface

5.2 A Tool for Evaluating the Effectiveness of Behavioral Matchmaking Method

The tool implements a method that allows the users to compare manually the models of a query service and a target service, and subsequently create a service ranking according to the results of the comparison. The models represent the most relevant features of the web processes described in BPEL (abstract process). In this way, the tool enables to compare the result obtained by the platform for service matchmaking and the ranking defined by the users. Given the fact that the parameterization of the cost function is domain dependent and very important for the effectiveness of the matchmaking method, it is important to have a tool allowing to determine the optimal parameters to use for a given domain and similarity criteria. The tool was built using the Java programming language (JDK 1.6.0) and the Netbeans 5.0 Integrated Development Environment (IDE). The system is a Web application wich is available at `http://ariadna.unicauca.edu.co/pertinence`.

In the remainder of this section, first we describe the functionalities provided by the system. Then, we describe the system architecture and finally, we depicted the user interfaces provided by the tool.

5.2.1 System Functionalities

Our implementation includes two base user-levels: the comparator and the administrator user. Next the functionalities are presented for each user.

- Comparator level
    * Make comparison: After logging in the evaluation tool, the system proposes to the user the services to compare. Then he can assign a similarity score (between 1 and 5) for the two services corresponding to the comparison criteria that he finds relevant. The proposed similarity criteria are: service name, service description, activity set and the service structure. For the activity set, the tool allows to compare interaction by interaction.
- Administrator level
    * Users management: The tool allows to create new users or edit already existing users (comparator or administrator users).
    * BPEL document management: The system enables to register target and query BPEL documents into the services repository.
    * Analyzing comparison results: With this functional component the tool can create a ranking for each query service analyzed according with the comparison criteria. This list is a $Top\ n$ (1<n<10) ranking, where the first one is the most similar service.

5.2.2 System Architecture

Figure 5.9 shows the layers of the tool's architecture and the interaction among them, as well as the most relevant packages that compose each layer.

- **The Application layer** manages the packages that implement the tool functionalities. This layer is composed of the following packages:
    * *Comparator user interface*: In order to achieve a visual representation, these packages contain all classes that implement the graphical interfaces of the comparator user.
    * *Administrator user interface*: this package contains all classes that implement the graphical interfaces of the administrator user.
    * *Comparison process*: contains the implementation for comparing manually two services.
    * *Ranking process*: this package calculates the service ranking based on results of comparison process package.
    * *User manager*: this package implements the logic to register and edit the user parameters that interact with the tool.
    * *BPEL documents manager*: contains the classes that implement the logic for managing the BPEL query and target documents.

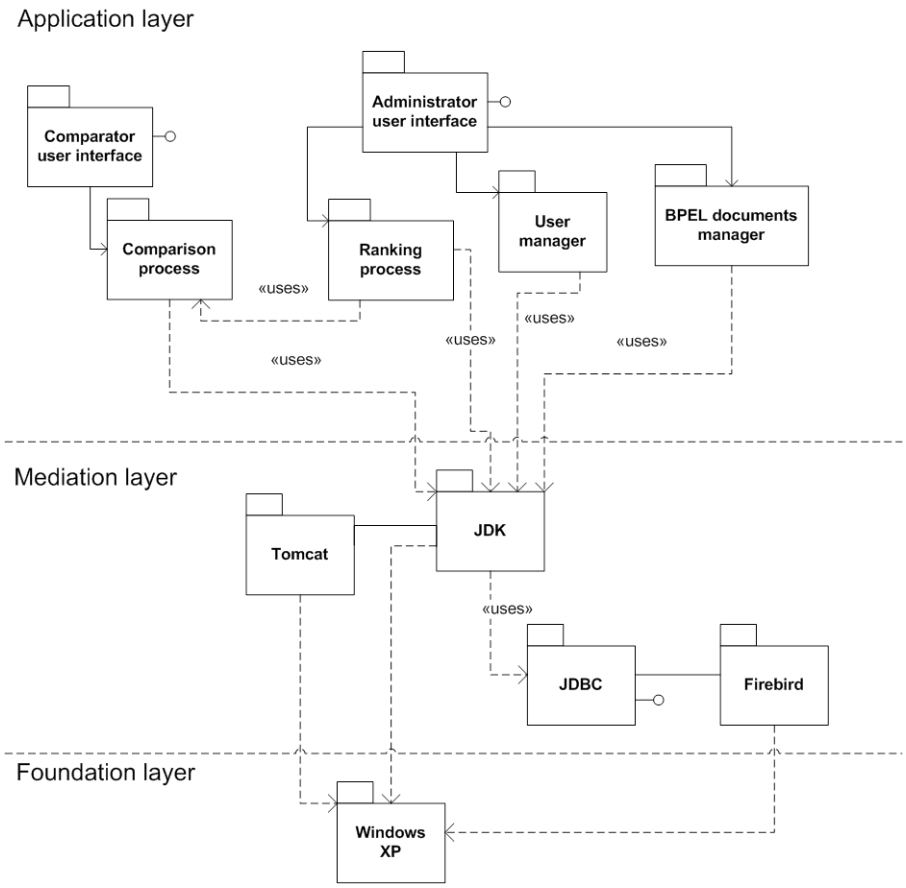

Figure 5.9: Logical architecture of the tool

- **Mediation layer** is composed by the following packages:
  * *JDBC*: is an API for the Java programming language that defines how a user may access a database. It provides methods for querying and updating data in a database. This interface is used by the packages of application layer that implement the logic of application, for storing the data into the Firebird database.
  * *Firebird*: is the database management system (open source of InterBase) used for storing the users data and the analysis of services matching and ranking.
  * *JDK*: Java Development Kit is an integrated development environment (IDE) for writing Java applications. It consists of a runtime environment that sits on top of the operating system layer that allows to compile, debug, and run the effectiveness tool which is written in the Java language.
  * *Tomcat*: provides an environment to run the tool as a web application.
- **Foundation layer** include the basic software that enables to perform the prototype. This layer is

composed of *Windows XP professional* which is the operating system that supports the tool.

5.2.3 User Interfaces

In this section we show the main interfaces of the tool. As we have said, after logging into the evaluation tool, the user selects the query service to analyze. The comparison process begins by comparing the query service against the first target service (see figure 5.10) and finishes when the last target service is compared.

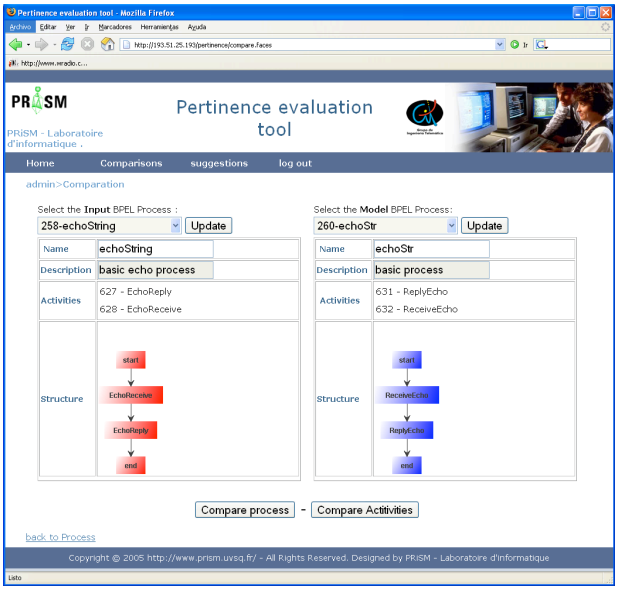

Figure 5.10: Services to compare interface

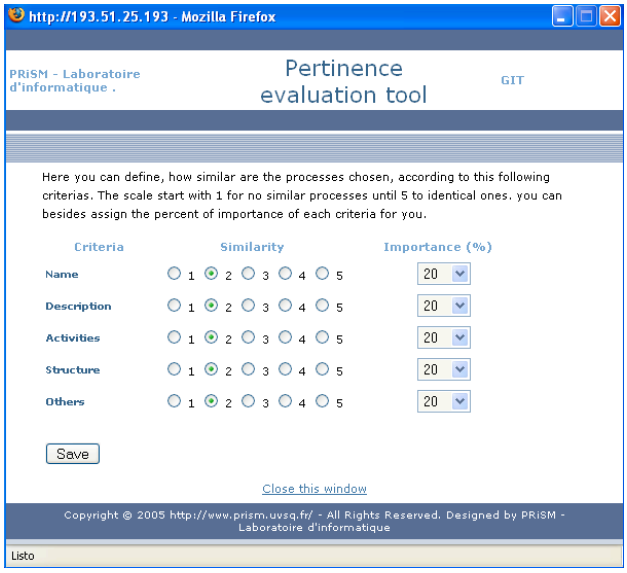

Figure 5.11: Criterion selection interface

Each pair of services is turned into a web-page, which offers a comfortable graphical user interface and permits an accurate definition of survey parameters. As can be seen in Figure 5.11, the subjects are asked to assess the similarity between two processes on a scale from 1 (no similarity) to 5 (identical). With simple radio buttons the users can specify how they have made the assessment: 1. by service name, 2. by service description, 3. by set of activities, 4. by service structure and 5. using other assessment method. Finally, in order to catch the subjective aspect of the similarity measurement, the tool allows to specify the importance of each criteria.

If the user selects as criteria the set of activities, the tool allows to analyze each branch of service graph, comparing interaction by interaction (see figure 5.12). Finally, the tool creates a ranking for each query service analyzed according with the comparison criteria or by a combination of criteria (see figure 5.13). The n level of $Top\ n$ ranking is given by the user. This Top is organized in descending order and if this one is composed by several criteria an average is calculated.

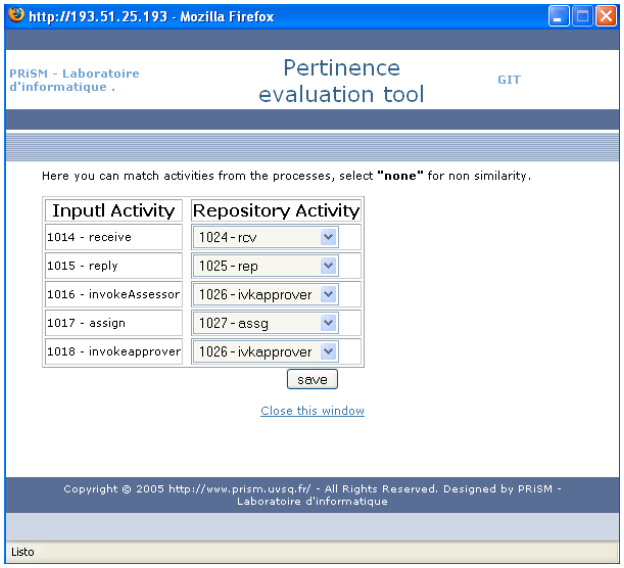

Figure 5.12: Interface of service branch comparison

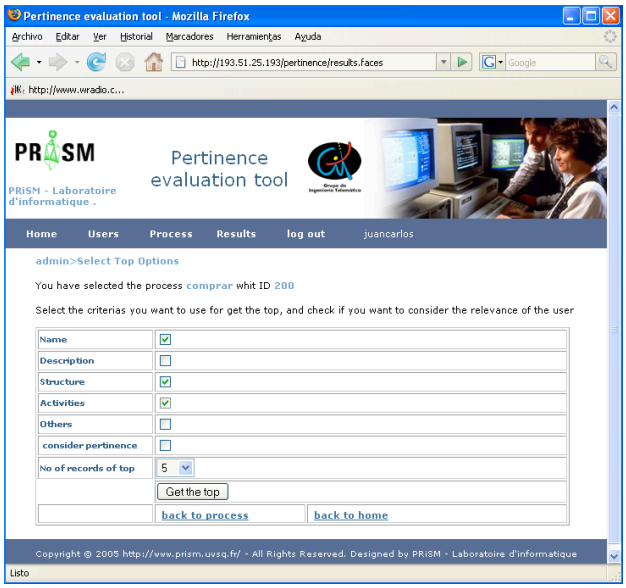

Figure 5.13: Service ranking interface

5.3 Experimental Evaluation

One of the problems in services retrieval evaluation is the lack of a benchmark of service similarity. In a nutshell the problem of developing a benchmark of service similarity is that this one requires a lot of testing data and experimentation which is time consuming, therefore most evaluation methods take place on a statistical system level. Besides, a benchmark should involve the human evaluation, but bechmark of this type are not proposed frequently. Therefore the human interaction must be addressed in service retrieval evaluation if it wants to catch up with reality. So, in this dissertation we made use of the *construct validity technique* for validating the WS-BeM results. This validity is concerned with the relation between theory and observation. It refers to the degree to which a given measure accurately characterizes the construct under study. To carry out this validity we analyzed the effectiveness of behavioral matchmaking method, which is based on comparison of the results obtained by the WS-BeM prototype and a human evaluation carried out by the tool for evaluating the effectiveness of behavioral matchmaking method(see 5.2).

As this dissertation focuses on matching technique and not in service ranking, the experimental evaluation concentrates on measuring the matching quality. Although the evaluation tool allows to create a service ranking, for this experimental evaluation we used only the tool functionality that lets us analyze each branch of service graph, comparing interaction by interaction (see figure5.12). Therefore, we evaluate the matching quality comparing the result obtained by the prototype against the results stored into the evaluation tool. In futur works we will use the tool for evaluating a service discovery prototype.

In this section, first the goals of the evaluations will be described, followed by an overview of the environment of the experiments. The data set used for the experiments, is described as well as results and their analysis. These results show the better cost function for obtaining an optimal matching result. Also

we explore the limits of the algorithm with respect to the execution time.

5.3.1 Experimental Evaluation Goals

The goal of the experimental evaluation was to characterize the performance and quality of matchmaking process. In particular, we wanted to find out the following:

- parameterization of the cost function
- quality assessment of matchmaking process
- performance of matchmaking algorithm
- conformance of matching results calculated by the prototype to intuitive results registered by the user.

To make the parameterization and relevance measurable criteria, we will exemplify the expression of parameterization of the cost function, and in particular the matching quality definition in terms of precision and recall.

Parameterization of the cost function:

An important problem is the definition of the graph edit operations and the corresponding cost function. The choice of a specific edit operation to be applied on the query graph depends on its cost. For example, consider that the node deletion cost has set to 0.5 and that three vertices labels in the query graph are different from the target graph (the distance between the labels being 0.1). Then, the graph edit distance is 0.3. On the other hand, if label error grows beyond 0.5, then the algorithm will consider the possibility of deleting a vertex. The relationship between node deletion cost and node substitution cost strongly influences the result and the behavior of the algorithm. The goal of our experiments is to parameterize the cost function in order to obtain a good quality of the matchmaking algorithm. We first describe how the quality of the matchmaking algorithm can be evaluated.

Measures for match quality:

To provide a basis for evaluating the quality of the matchmaking algorithm, we first have manually performed the match identifying the correspondences between the interactions of two BPEL models (nodes of the graph). To evaluate the quality of the matchmaking algorithm, we compared the matches P returned by our algorithm to the matches provided manually (R). Then, we determined the set of true positives, i.e. correctly identified matches, I, as well as the set of false positives, i.e. false matches, $F = P/I$, and false negatives, i.e. missed matches $M = R/I$. Based on the cardinalities of these sets, the following quality measures are computed:

- *Precision* = $|I|/|P|$ = $|I|/(|I| + |F|)$ estimates the reliability of the match predictions
- *Recall* = $|I|/|R|$ specifies the share of real matches that are found.

- *Overall* = 1 - ($|F| + |M|$)/R= ($|I| - |F|$)/$|R|$= $Recall * (2 - 1/Precision)$ represents a combined measure for match quality (see [79]) taking into account the post-match effort needed for both removing false and adding missed matches.

Precision and *Recall* metrics are intensively used in information retrieval systems. Note that neither *Precision* nor *Recall* alone can accurately assess the match quality. *Recall* can easily be maximized at the expense of a poor *Precision* by returning all possible correspondences. Similarly, a high *Precision* can be achieved at the expense of a poor *Recall* by returning only few (correct) correspondences. On the other side, *Overall* metric proportionally depends on both *Recall* and *Precision*, providing a single metric summarizing match quality.

5.3.2 Experiment Methodology and Result

As we have said, evaluating the WS-BeM prototype needs to compare the matching results against a foundation result. Therefore the system is compared against the matching results of a human evaluation which is done using the tool for evaluating the effectiveness of behavioral matchmaking method. We studied the influence of the cost function, more precisely of the relationship between the deletion cost and substitution cost, over the quality of the matching result. For our evaluation, we used 5 WSCL and BPEL files (For the BPEL files three nesting levels of structured activities are considered) having between 10 and 15 nodes (current web services are relatively simple, with a small number of operations). For each file, we generated distorted copies (that are syntactically different, but achieve the same functionality) in the following way:

- by changing label names with synonyms and abbreviations (case 1)
- by changing label names and changing the order of interactions (deleting, inserting edges) (case 2)
- by changing label names, deleting vertices and inserting new vertices (case 3).

In summary we generated 5 query services and 15 target services as testing data.

Using the tool for evaluating the effectiveness of behavioral matchmaking method we generated a foundation matching result that will allow to evaluate the matching quality of the WS-BeM prototype. Precisely, we took the testing data presented above and then we compared the query service against the target services using the tool functionality that allows to compare interaction by interaction between two service graph. Therefore, five users assessed 5 query services (Q_j) against 15 target services (T_i). Each user covered a set of services. The table 5.1 shows that five users (U_1 to U_5) carried out 15 comparisons for each query service respectively (e.g. the user U_1 assessed the query service Q_1 against 15 target services). In the same way the query service Q_2 was treated). In summary 150 comparisons were analyzed using the tool and two comparison were made to each query service, since one query service was evaluated at least one time by two different users.

In the other side, we evaluated the 5 query services against 15 target services (the same testing data) using the WS-BeM prototype. With the goal of studying the influence of the cost function into the matching result, for each query service analyzed we modified the costs of edit operations. Then, we contrasted these results with the foundation matching result obtained by the evaluation tool to determine the best cost function.

Users	Query service	Comparisons
U_1	Q_1 and Q_2	30
U_2	Q_2 and Q_3	30
U_3	Q_3 and Q_4	30
U_4	Q_4 and Q_5	30
U_5	Q_5 and Q_1	30

Table 5.1: Comparisons set

The experiments were conducted on a Dell machine, with a Pentium 4 processor 2.30GHz clock speed and 1000 MB RAM. The total disk space was 80 GB. The machine was running under Windows XP operating system.

In the remainder of this section, we show the evaluation results obtained for calibrating the system, and the execution time used by the matching process of two WSCL and BPEL metamodels.

WSCL experimentation:

In our experiments, we have changed the cost of deleting a vertex, C_d from 0.1 to 0.9, while the weights of deleting and inserting edges were kept constant to 0.2. The cost for substituting a node label and its associated attributes is defined as explained in the section 4.1. Its value is between 0 (full similarity) and 1 (no match).

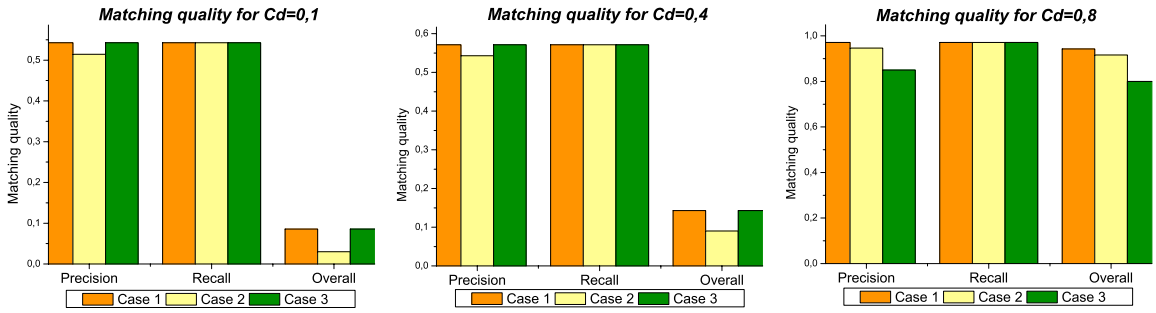

Figure 5.14: Match quality for different cost functions (WSCL system)

The figures 5.14 shows the average *precision*, *recall* and *overall* that we obtained when comparing the WSCL files with their distorted copies for different values of cost of deleting a vertex, C_d (C_d=0.1, C_d=0.4, C_d=0.8). We reported the quality measures separately for the three cases presented above. The experiments show that good quality results can be obtained with the following parameterization of the cost function: vertex deletion cost= 0.8, edge deletion cost=0.2, edge insertion cost =0.2.

Figure 5.15 shows the average execution times for the three cases and the different values for the vertex deletion cost. The experiments show that for the parameterization that produced the best quality results ($C_d = 0.8$), the algorithm takes more time compared to the others. The explanation for this behavior is the following: The relationship between node deletion cost and node substitution cost strongly influences the search space of the algorithm. When the vertex deletion cost is high compared to the insertion cost, before any node deletion is tried,

the algorithm attempts to substitute labels in the query graph by labels in the target graph. As the names matching algorithm is time consuming, the execution time is longer. When the deletion cost is decreased, many of these mappings become too expensive and are no longer considered.

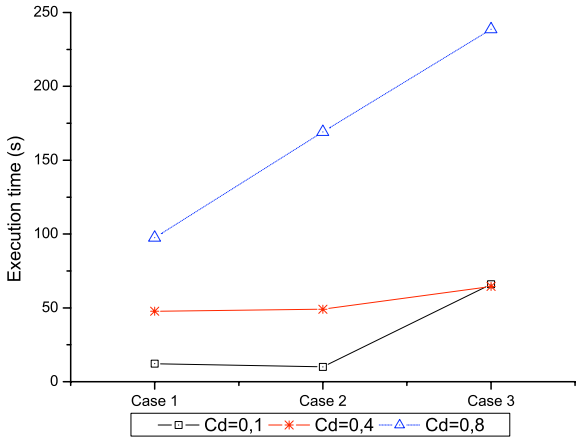

Figure 5.15: Execution time for different cost functions (WSCL system)

In the next experiment we tried to explore the limits of the algorithm with respect to the size of the target graph. The theoretical complexity of the graph matchmaking algorithm [81] is ($O(m^2n^2)$ in the best case (when the distance between the query and the target graph is minimal) and $O(m^n n)$ in the worst case (m = the total number of vertices in the target graph; n = the total number of vertices in the query graph).

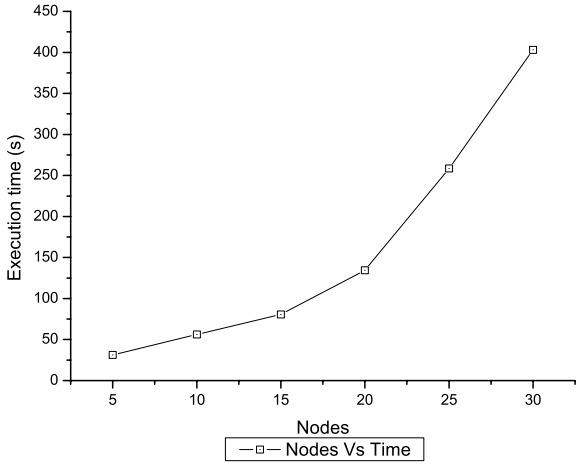

Figure 5.16: Execution time for growing number of nodes (WSCL system)

In the previous experiment we noticed that searching for synonyms in Wordnet leads to the

search of possibly large conceptual graphs, which is time consuming. In order to reduce this time we have introduced an *ad hoc* dictionary in which synonyms search is done with a negligible cost. (This domain-specific dictionary can be enriched with confirmed correspondences at the level of interaction and document names, allowing thus to reuse the match results.) Within this context, given a set of target graphs whose sizes vary from 5 vertices to 30 vertices, the corresponding execution times are presented in Figure 10. Despite the exponential theoretical cost, the graphic shows that the matchmaking algorithm can be used, with a low cost, for WSCL specifications having less than 30 interactions, which corresponds to reasonable problem size.

BPEL experimentation:

In the BPEL experiments we introduced the following relations between the graph edit operations:

- $Cei = Ced$, the cost of inserting and deleting an edge take the same value.
- $Cns < Cnd/Ws$, the parameter Ws controls the weight of a node deletion relative to a node substitution
- $Ced = We * Cnd$, the parameter We allows us to weight the importance of edit operations on the edges relative to nodes deletion operation

The Ws parameter assures that the node substitution operation has more probability of executing, since its cost is smaller with respect to the cost of deleting a node. Therefore, the matching algorithm will consider that the cost of label substituting a node is between zero and Cnd/Ws, where zero represents a perfect mapping and Cnd/Ws the threshold permitted for substituting a label. If $Cns > Cnd/Ws$, then the algorithm will consider that labels are totally different.

$$Cns = \begin{cases} LS & if\ (0 \leq \text{LS} \leq \text{Cnd/Ws}) \\ \infty & if\ \text{LS} > \text{Cnd/Ws} \end{cases}$$

On the other hand the We parameter guarantees that the operations cost on the edge (Ce) are smaller that Cnd, assuring that these ones have more probability of performing.

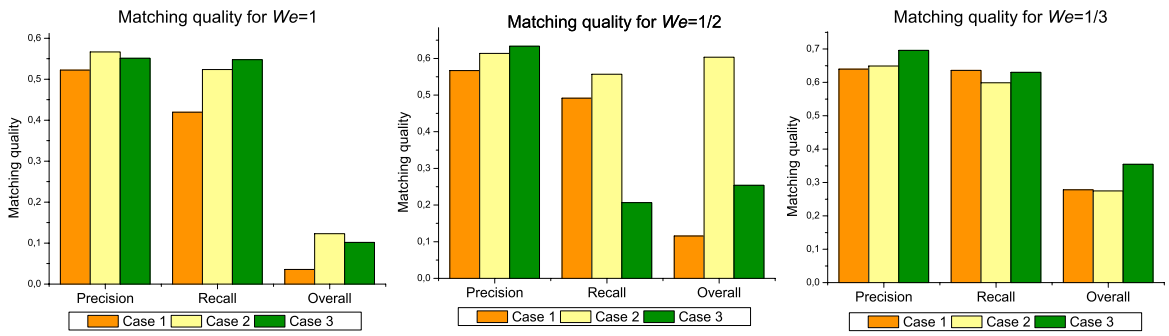

Figure 5.17: Match quality for $We = 1$, $We = 1/2$ and $We = 1/3$ (BPEL system)

In our experiments, we fixed the cost of deleting a vertex Cnd to 2 and the parameter Ws was kept constant to 4. Considering these values, the cost of the Linguistic Similarity (Cns) between two node labels took values between 0 (full similarity) and 1 (threshold permitted for substituting a label). While the weights of deleting and inserting edges were varied according to We parameter. In these experiments we varied We from 0 to 1.

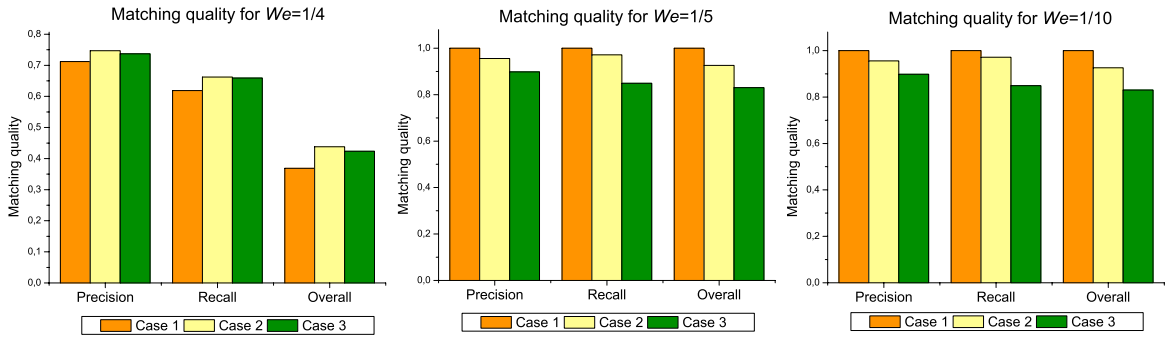

Figure 5.18: Match quality for $We = 1/4$, $We = 1/5$ and $We = 1/10$ (BPEL system)

Figures 5.17 and 5.18 show the precision, recall and overall average achieved after comparing the foundation matching results and the matching results obtained using the WS-BeM prototype. The considered values of We parameter were: We= 1, We =1/2, We= 1/3, We= 1/4, We= 1/5 and We= 1/10.

We reported the quality measures separately for the three cases of distorted service copies (case 1, 2 and 3). Through the figure 5.17 we can conclude that for $We = 1$, $We = 1/2$, and $We = 1/3$ values, the three cases have a similar behavior for the precision, recall and overall measures. The figure 5.18 shows that for $We = 1/4$ there is an improvement in the quality matching, and with We= 1/5 and We= 1/10 we can achieve the best matching.

Finally, in the figure 5.19 we can deduce that good quality results can be obtained from We=1/5 and Ws=4, with vertex deletion cost= 2, where the node substitution threshold=1/2, edge deletion cost=2/5 and edge insertion cost =2/5. Since for connectors are treated as edges, the edit operations on connectors take the same costs assigned to the operations on the edges.

Figure 5.20 shows the average execution times for the three distorted copies of BPEL services and six parameterizations of the cost function (150 comparisons). The experiments show that the first case takes less time compared to the others. The explanation for this behavior is the following: As in both services the structure is the same and the node label names are synonyms or abbreviations, then the algorithm will apply the minimal number of edit operation since the two services are semantically similar. Case 2 shows that the algorithm takes more time for calculating the matching, as the query and the target service have different structure and more edit operation are applied on the query graph. Finally in case 3 the algorithm spends more execution time, therefore the target service has more activities, and then there are more mapping possibilities for each activity into the query graph. On the other hand the optimal

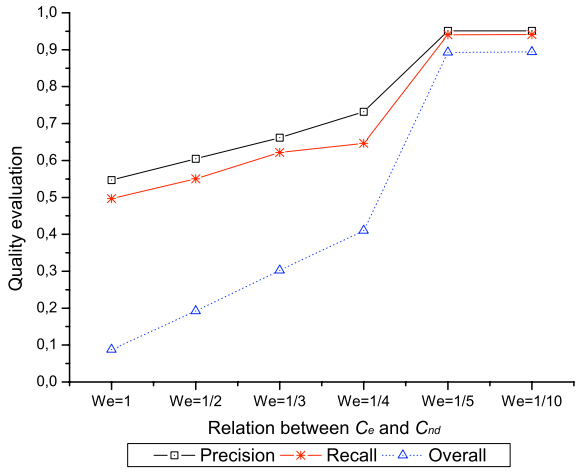

Figure 5.19: Match quality average (BPEL system)

parameterization (with $We = 1/5$ or $We = 1/10$) takes more execution time for calculating the services matching (with respect to the other parameterizations) because of the algorithm prefers to suppress, insert edges and connectors prior to suppress nodes.

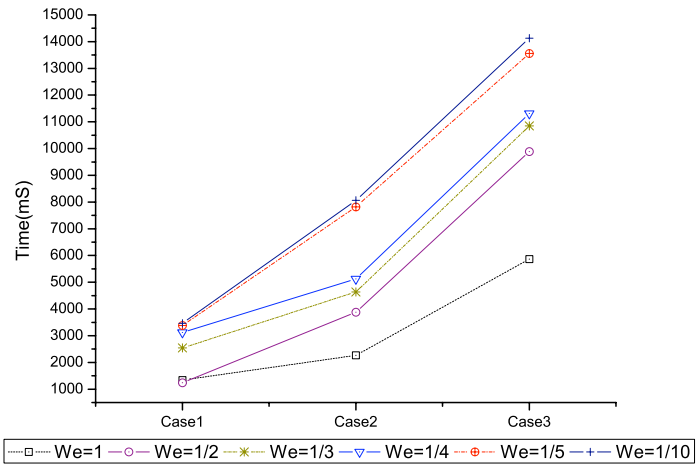

Figure 5.20: Execution time for the different cost functions (BPEL system)

In the next experiment we tried to explore the limits of the algorithm with respect to the size of the target graph. Given a set of target graphs whose sizes vary from 5 vertices to 35 vertices, the corresponding execution times are presented in Figure5.21. The experiment was made for $We = 1/5$ parameter (best matching). Despite the exponential theoretical cost, the graphic 5.21 shows that the matchmaking algorithm can be used, with a low cost, for BPEL specifications having less than 35 basic activities and three nesting levels for structured

activities, which corresponds to reasonable problem size.

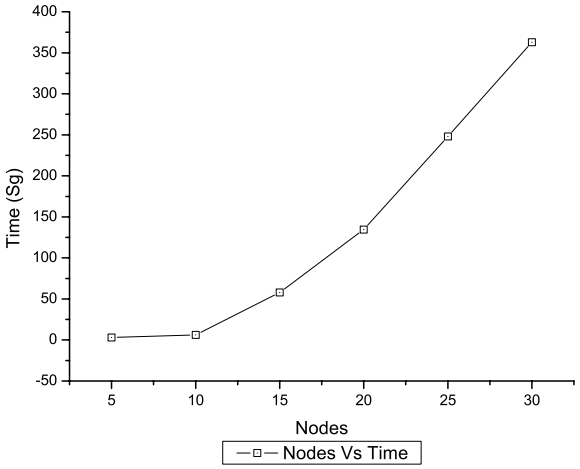

Figure 5.21: Matchmaking two BPEL documents

5.4 Summary

In this chapter we described a prototype called Ws-BeM (Web services-Behavioral Mathcmaking), which implements the proposed approaches into the chapters 3 and 4. The tool allows the execution of the algorithms for matchmaking services in the context of service ranking. Further, we have constructed a tool for evaluating the effectiveness of our behavioral matchmaking method. This is a tool that allows to create a user service ranking based on manually comparisons between a query service and the services in the repository.

The experimental evaluation of our approach was twofold. Firstly, we wanted to analyze the matching process quality by using different application scenarios (WSCL and BPEL) into WS-BeM. Although the evaluation tool allows to create a service ranking, for this experimental evaluation we used only the tool functionality that lets us analyze each branch of service graph, comparing interaction by interaction. Secondly, we wanted to test the execution time of matchmaking method using these protocols. Despite the exponential theoretical cost, the matchmaking algorithm can be used, with a low cost, for WSCL specifications having less than 30 interactions and for BPEL specifications having less than 35 basic activities and three nesting levels for structured activities, which corresponds to reasonable problem size.

Chapter 6

Conclusions

6.1 Achievements of dissertation

In this PhD thesis we proposed a solution for service retrieval based on behavioral specification. The approach uses matching techniques that operate on service behavioral models and allow delivery of partial matches as well as an evaluation of the semantic distance between these matches and the user requirements. Consequently, even if a service satisfying exactly the user requirements does not exist, the most similar ones will be retrieved and proposed for reuse by extension or modification. To do so, we reduced the problem of service behavioral matching to a graph matching problem and we adapted existing algorithms for this purpose. By using a graph representation formalism for services, we proposed to use an error correcting graph matching algorithm in order to allow an approximate service matching.

We exemplified our approach for behavioral matchmaking, by examining the usage of matching techniques in the context of WSCL and BPEL behavioral specifications. Despite the exponential theoretical cost, the matchmaking algorithm can be used, with a low cost, for WSCL specifications having less than 30 interactions and for BPEL specifications having less than 35 basic activities and three nesting levels for structured activities, which corresponds to reasonable problem size.

Aiming to reduce the search space into the BPEL metamodel, before executing the matchmaking algorithm the nodes of two BPEL graphs (G and G_I) were well-arranged in sets according to the activity types. In this way only the nodes that belong to the same activity type into G (query graph) and G_I (target graph) respectively were compared (i.e., $Invoke_{syn}$ set, $Invoke_{asyn}$ set, $Receive$ set, $Reply$ set, $Wait$ set). Furthermore, given two mapped *Receive* activities, the function $BasicActivityMatch$ will be applied only on *Reply* activities that correspond to the previously mapped *Receive* activities.

In spite of the considerations presented above, the number of states in the algorithm search space may grow exponentially in the worst case. Thus, various researchers have proposed to use a more sophisticated evaluation function which considers not only the cost of the represented partial matching, but also the future cost that is implied by this partial matching (see [**?**, 113]). By applying a so-called lookahead procedure in order to estimate the future cost, the number of expanded states can be reduced (see [126]). The lookahead procedure however, is of quadratic complexity and must be executed for each state in the search space. Furthermore, the algorithm continues only matching two graphs at the same time.

Given that in service discovery a query service must be compared against several target services at the same time, our algorithm must be applied on each target service. This may become prohibitive to practical

applications if the number of target services is large. Therefore, in future works it is necessary to create a target service repository and an indexing system that allows to optimize the similarity search process.

As this dissertation focused on matching of behavioral interface, the mathcmaking algorithm was applied on BPEL abstract process, but the same approach can be adapted to matching executable processes (considering the executable BPEL parameters). Furthermore, the dissertation was centered on messages exchanged by partners engaged in a business conversation and not in the private behavior of each partner, hence the handling of failures and compensation were not covered. Future works may address activities that interfere with the control flow (e. g. $throw$).

Finally, we developed a prototype system (Ws-BeM) to implement our approach of behavioral matchmaking for services retrieval: application to BPEL and WSCL protocols. Furthermore, we have constructed a tool for evaluating the effectiveness of our behavioral matchmaking method. The validation of our approach was twofold. Firstly, we analyzed the matching process quality for different application scenarios (WSCL and BPEL) into WS-BeM. Secondly, we tested the execution time of matchmaking method using these protocols.

Although the evaluation tool allows to create a service ranking, for this experimental evaluation we used only the tool functionality that lets us analyze each branch of service graph, comparing interaction by interaction. Therefore, we evaluated the matching quality comparing the result obtained by the prototype against the results stored into the evaluation tool. In this way, the tool for evaluating the effectiveness of our behavioral matchmaking method allowed us to parameterize the costs function to give the user an optimal matching result. The experiments showed that for WSCL portocol, a good quality results can be obtained with the following parameterization of the cost function: vertex deletion cost= 0.8, edge deletion cost=0.2, edge insertion cost =0.2. For BPEL protocol the good quality results were obtained from We=1/5 and Ws=4, with vertex deletion cost= 2, where the node substitution threshold=1/2, edge deletion cost=2/5 and edge insertion cost =2/5. Given that the evaluation tool allows to create a service ranking, in futur works we will use this tool for evaluating a service discovery method.

As summary, the main contributions of this thesis were:

- **A detailed analysis of service matchmaking.** The main results of this analysis were: (a) A survey on several viewpoints from which behavioral models for service composition can be captured; (b) A description of formal representations of services; (c) An analysis of the techniques used for service matchmaking, which were clarified in three categories: Service matchmaking based on interfaces, semantics and behavior.

- **The proposal of techniques and algorithms for the service matchmaking.** In this dissertation a solution for service retrieval based on behavioral specification was developed. By using a graph representation formalism for services, we proposed to use an error correcting graph matching algorithm in order to allow an approximate service matching (see [61, 42, 47, 41, 45, 43, 44]).

- **Application of service matchmaking to WSCL protocol.** In this thesis we motivated the need to retrieve services based on their conversation model. We exemplified our approach for behavior matching for conversation protocols expressed using the WSCL model (see [61, 47, 41, 43]). Starting from the classical graph edit distance, we proposed two new graph edit operations to take into account the difference of granularity levels that could appear in two models. The conversation protocol matchmaking process is composed of the following steps: First, the conversations protocols to be compared are transformed into graphs. Next, the graphs are expanded in order to have the same

level of granularity in both graphs and the error-correcting graph matching algorithm is applied. The similarity function evaluates the similarity between the graphs. Finally, the granularity levels are compared and the costs corresponding to identified differences are added to the total distance.

- **Application of service matchmaking to BPEL protocol.** Considering the importance of Web processes, in this dissertation we discussed our approach for Behavioral matchmaking, by examining the usage of matching techniques in the context of BPEL behavioral specifications (see [42]). The BPEL matchmaking process is composed of the following steps: first, the BPEL documents to be compared are transformed to graphs. Next, the error correcting graph matching algorithm is applied (considering the decompostion and composition functions during the algorithm execution). Then, the similarity function evaluates the similarity between the graphs.

- **A prototype for behavioral matchmaking for service retrieval.** We developed a prototype called Ws-BeM (Web services-Behavioral Mathcmaking), which implements the proposed approaches. The tool allows the execution of the algorithms for matchmaking services (see [45, 44, 46]). In order to validate our approach, the prototype was tested with two application scenarios: the matching of BPEL and WSCL protocols.

- **A prototype for evaluating the effectiveness of our behavioral matchmaking method.** We constructed a tool for evaluating the effectiveness of our behavioral matchmaking method (see [44, 46]). This tool allows to create a user service ranking based on manual comparisons between a query service and the services in the repository. The tool permits to compare the result obtained by the Ws-BeM platform and a ranking defined by users.

In next section we discuss some research perspectives.

6.2 Future work

Future work can be constructed along the following lines: (i) Repository for Business Processes Matchmaking, (ii) Indexing Techniques for Business Processes Matchmaking (iii) Matchmaking of Outsourcing Process Fragments (iv) Ranking of Business Processes.

6.2.1 Repository for Business Processes Matchmaking

The business process matchmaking must be implemented efficiently to support service discovery in a large service repository, like on the Internet scale, addressing the problem of comparing a web process with a set of web processes in a library. Thus, a repository is necesary for storing web processes. In [118] the authors describe a BPEL Repository, which is an Eclipse plug-in built to store business processes together with other XML data. It provides a framework for storing, finding and using these documents. The framework takes care of representing the XML data as EMF objects (Objets of Eclipse Modeling Framework) that are Java objects. Then, it is possible to query the XML files as EMF objects using an object-oriented query language, namely the Object Constraint Language (OCL) that is part of the UML specification. Therefore our perspective is to adapt our Service Matchmaking to this BPEL repository.

6.2.2 Indexing Techniques for Business Processes Matchmaking

Indexing support is needed in large service repositories since the number of business processes to be compared is large, thus scanning whole repositories to compute the matching is computationally expensive [76]. Since our approach of service matchmaking is based on graph matching, and considering that many approaches have been proposed to graph database indexing (see [123, 60, 120]), a future work can explore which is the most appropriate index for working with our matchmaking algorithm.

6.2.3 Matchmaking of Outsourcing Process Fragments

An activity of a process may be implemented by another process. When a process is used to provide a service to be performed by an activity within another process, we can define this one as a subprocess. Outsourcing is a scenario for using a subprocess from another process. In this scenario the functionalities corresponding to the subprocess of an external partner are taken into account for reducing the burden of modelling new ones.

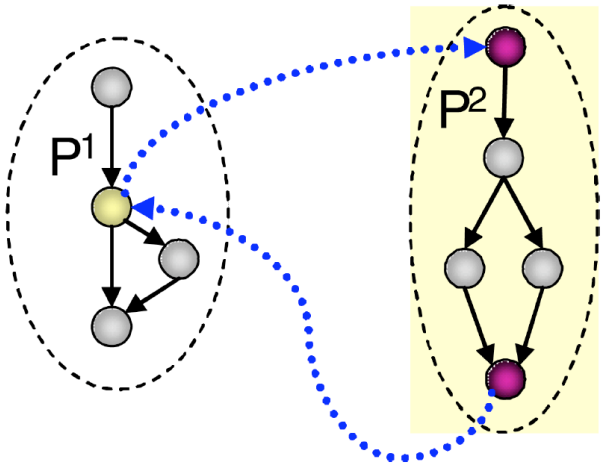

Figure 6.1: Example of the Outsourcing Process Matchmaking

Figure 6.1 gives an example: process P_1 contains an activity that is turned into a subprocess P_2. Our perspective consists in analyzing how the similarity between the activity and the subprocess can be detected automatically.

6.2.4 Processes Ranking

Matchmaking is basically the process of discovering, based on a given request, promising partners for some kind of purpose. It is a common process to several scenarios in the Internet era, spanning from e-commerce to web-services, to grid computing, to human resource management, to actual dating services, to Peer-to-Peer computing. Obviously, main issues emerge when the search is not limited to identity matches but, as in real life, when the objective is finding partners suitable at least to some extent, or – when a single partner cannot fulfill the request – find a pool of cooperating partners able to accomplish it.

As this process may lead to various possible matches, the notion of ranking becomes central, to provide a list of potential partners ordered according to some criteria [l]. In this way, a future work can focus on developing a service ranking for a specific domain based on our service matchmaking approach that take into account others parameters as QoS, execution times, service availability, etc.

Bibliography

[1] 1st international workshop on semantic matchmaking and resource retrieval:issues and perspectives. *http://sisinflab.poliba.it/smr2006/. Last accessed on September 2007.*

[2] Grid resource allocation agreement protocol wg, (ws-agreement). *https://forge.gridforum.org/projects/graap-wg. Last accessed on August 2006.*

[3] Oasis web services security (wss) tc. *www.oasis-open.org/committees/wss/. Last accessed on September 2007.*

[4] Simple object access protocol (soap). *http://www.w3.org/TR/soap/. Last accessed on September 2007.*

[5] Universal description, discovery and integration (uddi). *http://www.uddi.org. Last accessed on September 2007.*

[6] Web service description language (wsdl). *www.w3.org/TR/wsdl. Last accessed on September 2007.*

[7] Web service policy framework (ws-policy). *http://www-106.ibm.com/developerworks/library/ws-polfram/. Last accessed on September 2007.*

[8] Web service transactions specifications (ws-transactions). *http://www-128.ibm.com/developerworks/library/specification/ws-tx/. Last accessed on September 2007.*

[9] Special section on graph algorithms and computer vision. *IEEE Trans. PAMI*, 23(10):1049–1151, 2001.

[10] H. Almohamed. A linear programming approach for the weighted graph matching problem. *IEEE Trans. PAMI 15*, pages 522–525, 1993.

[11] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services, version 1.1. In *Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation*, 2003.

[12] R. C. Angell, G. E. Freund, and P. Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing and management*, 19(4):255–261, 1983.

[13] A. Arkin and S. et al. Askary. Web service choreography interface (wsci) 1.0. *W3C - Web Services Choreography Working Group*, 2002.

[14] R. Bacik. *Structure of Graph Homomorphism*. PhD thesis, Simon Fraser University, Canada, 2001.

[15] J. Baget and M. Mugnier. Extension of simple conceptual graphs: the complexity of rules and constraints. *Artificial Intelligence Research*, 16:425465, 2002.

[16] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web services conversation language (wscl) 1.0. In *W3C*, 2002.

[17] A. Barros, M. Dumas, and P. Oaks. Standards for web service choreography and orchestration: Status and perspectives. In *Proc. of Business Process Management Workshops*, 2006.

[18] B. Benatallah, F. Casati, D. Grigori, H. R. Motahari Nezhad, and F. Toumani. Developing adapters for web services integration. In *Proc. of CAISE*, 2005.

[19] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web services protocols. In *Proc. of ER*, 2004.

[20] B. Benatallah, F. Casati, and F. Toumani. Web services conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, 2004.

[21] B. Benatallah, M.S. Hacid, C. Rey, and F. Toumani. Semantic reasoning for web services discovery. In *Proc. of ESSW*, 2003.

[22] D. Berardi, F. ROSA, L. SANTIS, and M. MECELLA. Finite state automata as conceptual model for e-services. 2003.

[23] J.A. Bergstra, A. Ponse, and S.A (eds) Smolka. Handbook of process algebra. *North Holland, Elsevier*, 2001.

[24] A. Bernstein and M. Klein. Towards high-precision service retrieval. In *Proc. of ISWC*, 2002.

[25] L. Bordeaux, G. Salan, D Berardi, and M. Mecella. When are two web services compatible? In *Proc. of TES*, 2004.

[26] A. Bottaro, A Grodolle, and P. Lalanda. Pervasive service composition in the home network. In *Proc of the 21st International IEEE Conference on Advanced Information Networking and Applications*, 2007.

[27] A. Branca, E. Stella, and A. Distante. Qualitative scene interpretation using planar surfaces. *Autonomous Robots*, 8(2):129139, 2000.

[28] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689694, 1997.

[29] H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):917922, 1999.

[30] H. Bunke. Graph matching: Theoretical foundations, algorithms, and applications. pages 82–88, 2000.

[31] H. Bunke. Recent advances in structural pattern recognition with applications to visual form analysis. In *Proceedings of the Fourth International Workshop on Visual Form IWVF4*, 2001.

[32] H. Bunke and C. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253, 1983.

[33] H. Bunke and S. Gunter. Weighted mean of a pair of graphs. *Computing*, 67(3):209224, 2001.

[34] H. Bunke and X. Jiang. Graph matching and similarity. *Intelligent Systems and Interfaces*, pages 281–304, 2000.

[35] H. Bunke, X. Jiang, and A. Kandel. On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25, 2000.

[36] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255259, 1998.

[37] V. et al. Cantoni. 2-d object recognition by multiscale tree matching. *Pattern Recognition 31*, pages 1443–1455, 1998.

[38] J. Cardoso and A. Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 21:191–225, 2003.

[39] L. C. Chiat, L. Huang, and J. Xie. Matchmaking for semantic web services. In *Proc of Services Computing, 2004 IEEE International Conference on (SCC'04)*, 2004.

[40] W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Trans. PAMI 8*, pages 749–764, 1995.

[41] J.C. Corrales, D. Grigori, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: application to conversation protocols. In *Proc of 22èmes Journées Bases de Données Avancées, BDA*, 2006.

[42] J.C. Corrales, D. Grigori, and M. Bouzeghoub. Bpel processes matchmaking for service discovery. In *Proc. of 14th International Conference on Cooperative Information Systems (CoopIS)*, pages 237–254, 2006.

[43] J.C. Corrales, D. Grigori, and M. Bouzeghoub. Decouverte de services basee sur leurs protocoles de conversation. In *Ingenierie des systemes d'information (ISI)*, volume 12, pages 9–32, 2007.

[44] J.C. Corrales, D. Grigori, and M. Bouzeghoub. Ws-bem: Web services ranking based on behavior matchmaking. In *Proc of 23èmes Journées Bases de Données Avancées, BDA*, 2007.

[45] J.C. Corrales, D. Grigori, M. Bouzeghoub, and J. Burbano. Services matchmaking system. In *Proc of VI Congreso Nacional de Electrnica, Telecomunicaciones e Informtica, ETI 2.006*, 2006.

[46] J.C. Corrales, D. Grigori, M. Bouzeghoub, and J.E. Burbano. Bematch: A platform for matchmaking service behavior models. In *11th International Conference on Extending Database Technology, EDBT*, 2008.

[47] J.C. Corrales, D. Grigori, M. Bouzeghoub, and A. Ordonez. Conversation protocol matchmaking. In *Proc. of Euro American Conference on Telematics and Information Systems.*, pages 67–74, 2006.

[48] A. Cross, R. Wilson, and E. Hancock. Genetic search for structural matching. In *Proceedings of the Computer Vision - ECCV*, pages 514–525, 1996.

[49] M. David, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The owl-s approach. In *Proc of the First International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[50] M. Dean and G. Schreiber. Owl web ontology language reference. In *w3c recommendation*, pages http://www.w3.org/TR/owl–ref. Last accessed on September 2007, 2004.

[51] L. Dong, A. Halevy, J. Madhavan, E. Nemes, , and J. Zhang. Similarity search for web services. In *Proc. of VLDB*, 2004.

[52] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series, second edition, 2005.

[53] L. Eroh and M. Schultz. Matching graphs. *Graph Theory*, 29(2):7386, 1998.

[54] M.A. Eshera and K.S. Fu. A graph distance measure for image analysis. *IEEE Trans. SMC 14*, pages 398–408, 1984.

[55] J. Feng, M. Laumy, and M. Dhome. Inexact matching using neural networks. *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, pages 177–184, 1994.

[56] M. Fernandez and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753758, 2001.

[57] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of Ssdbm*, 2002.

[58] S. Gao and J. Shah. Automatic recognition of interacting machining features based on minimal condition subgraph. *Computer Aided Design*, 30(9):727739, 1998.

[59] O. Gerbe, R. Keller, , and G. Mineau. Conceptual graphs for representing business processes in corporate memories. 1998.

[60] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. In *Proc. of ICPR*, pages 112–115, 2002.

[61] D. Grigori, J.C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval. In *Proc. of The IEEE International Conference on Web Services (ICWS)*, pages 145–152, 2006.

[62] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *Fourteenth Australasian Database Conference-ADC*, pages 191–200, 2003.

[63] C. A. R. Hoare. Communicating sequential processes. *Prentice Hall International*, 1985.

[64] R. Hopcroft, J. E.and Motwani and J. D. Ullman. Introduction to automata theory, languages, and computation. 2001.

[65] I. Jacobson, G. Booch, and J. Rumbaugh. The unified software development process. In *Addison-Wesley*, 1998.

[66] X. Jiang, A. Munger, and H. Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):11441151, 2001.

[67] N. Kavantzas and D. Burdett. Ws choreography model overview. *W3C - Web Services Choreography Working Group*, 2004.

[68] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web services choreography description language version 1.0. *W3C - Web Services Choreography Working Group*, 2004.

[69] T. Kawamura, J.A. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. A preliminary report of a public experiment of a semantic service matchmaker combined with a uddi business registry. In *Proc. of ICSOC*, 2003.

[70] N. Kokash, W. van den Heuvel, and V. D'Andrea. Leveraging web services discovery with customizable hybrid matching. In *Proc. of ICSOC*, pages 522–528, 2006.

[71] Li Kuang, Ying Li, Shuiguang Deng, Jian Wu, Wei Shi, and Zhaohui Wu. Expressing service and query behavior using pi-calculus for matchmaking. In *Proc of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 629–632, 2006.

[72] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–354, 1972.

[73] F. Leymann. Web services: Distributed applications without limits. In *BTW*, pages 2–23, 2003.

[74] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc of WWW 2003*, pages 331–339, 2003.

[75] J. Llados, E. Marti, and J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):11371143, 2001.

[76] B. Mahleko and A. Wombacher. Indexing business processes based on annotated finite state automata. In *Proc. of The IEEE International Conference on Web Services (ICWS'06)*, pages 303–311, 2006.

[77] M. Mazzara and R. Lucchi. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, 2006.

[78] J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software-Practice and Experience*, 12:23–34, 1982.

[79] H. Melnik, S.and Garcia and E. Rahm. Similarity flooding : A versatile graph matching algorithm and its application to schema matching. In *18th ICDE*, 2002.

[80] J. Mendling and J. Ziemann. Transformation of bpel processes to epcs. In *Proc. of the 4th GI Workshop on Event-Driven Process Chains (EPK2005)*, 2005.

[81] B. Messmer. *Graph Matching Algorithms and Applications*. PhD thesis, University of Bern, 1995.

[82] B. T. Messmer and H. Bunke. Error-correcting graph isomorphism using decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(6):721742, 1998.

[83] B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493504, 1998.

[84] B. T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32:1979–1998, 1999.

[85] B. T. Messmer and H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307323, 2000.

[86] G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[87] R. Milner. A calculus of communicating systems. *Springer-Verlag NewYork, Inc*, 1982.

[88] R. Milner. Communicating and mobile systems: the p-calculus. *Cambridge University Press*, 1999.

[89] R. Myers and E. R. Hancock. Least-commitment graph matching with genetic algorithms. *Pattern Recognition*, 34(2):375394, 2001.

[90] K. Oflazer. Error-tolerant retrieval of trees. *IEEE Trans. PAMI 19*, pages 1376–1380, 1997.

[91] N. Ould, A. M'Bareck, and S. Tata. Bpel behavioral abstraction and matching. In *Business Process Management Workshops*, pages 495–506, 2006.

[92] J. Paananen. Tik-110.501 seminar on network security. In *Available at Helsinki University of Technology: http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/intfo.html*, 1995.

[93] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proc. of ISWC*, 2002.

[94] M. Papazoglou and D. Georgakopoulos. Introduction, service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[95] Abhijit Patil, Swapna Oundhakar, A. Sheth, and Kunal Verna. Meteor-s web service annotation framework. In *Proc. of WWW Conference*, 2004.

[96] C. Pautasso and G Alonso. Visual composition of web services. In *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*, pages 92–99, 2003.

[97] M. Pelillo. A unifying framework for relational structure matching. In *Proceedings of the 14th ICPR*, 1998.

[98] M. Pelillo, K. Siddiqi, and S. Zucker. Matching hierarchical structures using associated graphs. *IEEE Trans. PAMI 21*, pages 1105–1120, 1999.

[99] J. L. Peterson. Petri net theory and the modeling of systems. In *Prentice-Hall*, 1981.

[100] G. Piccinelli, G. Di Vitantonio, and L. Mokrushin. Dynamic service aggregation in electronic marketplaces. *Computer Networks*, 2(37), 2001.

[101] D. Riviere, J. Mangin, D. Papadopoulos, J. Martinez, V. Frouin, and J. Regis. Automatic recognition of cortical sulci of the human brain using a congregation of neural networks. *Medical Image Analysis*, 6(2):7792, 2002.

[102] S. S. Bansal and J. M. Vidal. Matchmaking of web services based on the DAML-S service model. In *Proc. of AAMAS*, pages 926–927, 2003.

[103] M. Saboua and J. Panb. Towards semantically enhanced web service repositories. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5:142–150, 2007.

[104] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):353–363, 1983.

[105] H. Schmidt and R. Reussner. Generating adapters for concurrent component protocol synchronisation. In *FMOODS '02: Proceedings of the IFIP TC6/WG6.1*, pages 213–229, 2002.

[106] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 3, 1981.

[107] K. Shearer, H. Bunke, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):10751091, 2001.

[108] Z. Shen and J. Su. Web services discovery based on behavior signatures. In *Proc. of IEEE SCC*, 2005.

[109] K. Sivashanmugam, K. Verma, A. Sheth, , and J. Miller. Adding semantics to web services standards. In *Proc of the the 1st International Conference on Web Services (ICWS'03)*, 2003.

[110] J. F. Sowa. Conceptual structures: Information processing in mind and machine. *Addison-Wesley*, 1984.

[111] E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *Int. J. Cooperative Inf. Syst.*, 14(4):407–438, 2005.

[112] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In *Proc. of SWWS*, 2001.

[113] W.H. Tsai and K.S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern recognition. *IEEE Trans. SMC*, 9:757–768, 1979.

[114] J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing*, 23(1):31–42, 1976.

[115] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. PAMI 10*, pages 695–703, 1988.

[116] W. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. 1999.

[117] W. Van der Aalst and K. V. Hee. Workflow management- models, methods, and systems. In *MIT Press*, 2002.

[118] J. Vanhatalo, J. Koehler, and F. Leymann. Repository for business processes and arbitrary associated metadata. In *Proc of the BPM Demo Session at the Fourth International Conference on Business Process Management*, 2006.

[119] J. et al. Wang. An algorithm for finding the largest approximately commen substructure of two trees. *IEEE Trans. PAMI 20*, pages 889–895, 1998.

[120] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han. Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications. In *Proc of the ACM-SIGMOD International conference on Management of data(SIGMOD)*, 2005.

[121] Y.-K. Wang, K.-C. Fan, and J.-T. Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Trans. SMC 27*, 4:588–597, 1997.

[122] G. Weikum. Towards guaranteed quality and dependability of information services. 1999.

[123] D.W. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *23rd International Conference on Data Engineering (ICDE)*, 2007.

[124] R. Wilson and E. Hancock. Graph matching by discrete relaxation. *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, pages 165–176, 1994.

[125] A. Wombacher, B. Mahleko, P. Fankhauser, and E. Neuhold. Matchmaking for business processes based on choreographies. In *Proc. of EEE*, 2004.

[126] E.K. Wong. Three-dimensional object recognition by attributed graphs. *Syntactic and Structural Pattern Recognition-Theory and Applications*, pages 381–414, 1990.

[127] J. Wu and Z. Wu. Similarity-based web service matching. In *Proc. of IEEE SCC*, 2005.

[128] L. Xu and E. Oja. Improved simulated annealing, boltzmann machine, and attributed graph matching. *In L.Almeida (ed): LNCS 41 Springer Verlag2*, pages 151–161, 1990.