École Supérieure de Commerce de Tunis
Université de la Mannouba

Institut des Hautes Études Commerciales
de Carthage - Université de Carthage

Master Thesis in Computer Science

Parameterized Semantic Matchmaking and Ranking Framework for Web Service Composition

Fatma Zahra Gmati

Jury

Prof Rim Faiz, *president*
Institute of Advanced Business Studies, University of Carthage, Tunisia

Dr Ahlem Ben Hassine, *reviewer*
National School of Computing, University of Manouba, Tunisia

Supervision Team

Dr Nadia Yacoubi-Ayadi
National School of Computing, University of Manouba, Tunisia

Dr Salem Chakhar
Portsmouth Business School, University of Portsmouth, UK

January 2015

This thesis is dedicated to my wonderful parents.
When there was no hope, they were the light that guided me through the darkness.
I am grateful to them, because they believed in my recovery.

I want also to thank my sister for her precious presence.

Acknowledgments

I want to thank Dr Nadia Yacoubi for her encouragements and sage advices. I want also to thank Dr Salem Chakhar for his tremendous support, immeasurable help and for the hours we spent discussing the essence of research. I am thankful to Dr Afef Bahri for her insightful criticism.

Contents

Contents **i**

List of Figures **iv**

List of Tables **v**

Introduction **1**
General context . . . 1
Motivation and objectives . . . 2
Structure of the document . . . 3

1 Web Services Matchmaking and Ranking: State of the Art **4**
1.1 Introduction . . . 4
1.2 Web service definition and basic concepts . . . 5
1.2.1 Definition . . . 5
1.2.2 Web service standards . . . 5
1.2.3 Web service architecture . . . 6
1.3 Semantic Web service . . . 7
1.3.1 Definition . . . 7
1.3.2 Ontology . . . 8
1.3.3 Semantic Web standards . . . 9
1.3.4 Extension of OWL-S . . . 10
1.4 Web service composition . . . 11
1.4.1 Web service composition approaches . . . 11
1.4.2 Composition approach . . . 12
1.5 Web services matchmaking . . . 13
1.6 Web services ranking . . . 16
1.7 Discussion . . . 17
1.7.1 Strict syntactic matching . . . 18
1.7.2 Capability-based matchmaking . . . 18
1.7.3 Lack of customisation support . . . 18
1.7.4 Lack of accurate ranking of matching Web services . . . 19
1.8 Conclusion . . . 19

2 Web Services Matching **21**

2.1 Introduction 21
2.2 Background 22
2.2.1 Example scenario 22
2.2.2 Basic definitions 22
2.3 Similarity measure 23
2.3.1 Definitions 23
2.3.2 Bellur and Kulkarni's similarity computing algorithm 24
2.3.2.1 Computing the degree of match 24
2.3.2.2 Computing the similarity measure 25
2.3.3 Efficient computation of the similarity measure 27
2.3.4 Accurate computation of the similarity measure 28
2.3.5 Computational complexity 30
2.4 Matching algorithms 32
2.4.1 Trivial matching algorithm 32
2.4.2 Partially parameterized matching algorithm 33
2.4.3 Fully parameterized matching algorithm 35
2.4.4 Computational complexity 37
2.5 Extension of matching algorithm 38
2.5.1 Functional attribute-level disjunctive matching 38
2.5.2 Functional attribute-level generic matchmaking 38
2.5.3 Functional service-level matching 40
2.5.4 Computational complexity 42
2.6 Comparative study 43
2.6.1 Computing of similarity measure 43
2.6.2 Matching algorithms 43
2.7 Conclusion 45

3 Web Services Ranking **46**

3.1 Introduction 46
3.2 Scoring Web services 47
3.2.1 Score definition 47
3.2.2 Score computing algorithms 48
3.2.3 Algorithmic complexity 50
3.3 Score-based ranking of Web services 51
3.3.1 Score-based ranking algorithm 51
3.3.2 Algorithmic complexity 52
3.4 Rule-based ranking of Web services 52
3.4.1 Ranking rules 52
3.4.2 Rule-based ranking algorithm 55
3.4.3 Algorithmic complexity 56
3.5 Tree-based ranking of Web services 56
3.5.1 Principle 57
3.5.2 Tree construction 57

3.5.2.1 Node splitting functions 58
3.5.2.2 Tree construction algorithm 58
3.5.3 Tree-based ranking algorithm 61
3.5.4 Algorithmic complexity 63
3.6 Comparative study 66
3.7 Conclusion 66

4 Implementation and Performance Evaluation 68
4.1 Introduction 68
4.2 System architecture and implementation 69
4.2.1 System design and conceptual architecture 69
4.2.2 Functional architecture 71
4.2.3 Implementation 72
4.3 Performance evaluation framework and metrics 72
4.3.1 Evaluation framework 72
4.3.2 Test collection 73
4.3.3 Evaluation metrics 74
4.3.3.1 Precision and Recall 74
4.3.3.2 Average Precision 75
4.3.3.3 Query Response Time 75
4.3.3.4 Memory Usage 75
4.4 Performance evaluation analysis 75
4.4.1 Comparison of configurations 1 and 2 76
4.4.2 Comparison of configurations 1 and 4 78
4.4.3 Comparison of configurations 5 and 6 78
4.5 Edge criteria order effect on the tree-based ranking 80
4.6 Comparative study 82
4.6.1 Recall/Precision 82
4.6.2 Average precision 83
4.6.3 Query response time 83
4.6.4 Memory usage 84
4.7 Conclusion 85

Conclusion 86
Summary 86
Future work 88
A. QoS-aware semantic matchmaking and ranking of Web services . . . 88
B. Multicriteria-based matching and ranking of Web services 88
C. Fuzzy semantic matchmaking and ranking of Web services 88

Bibliography 90

Glossary 98

List of Figures

1.1 Web service architecture . . . 7
1.2 XML and Semantic Web W3C standards timeline-history . . . 8
1.3 Extended OWL-S upper ontology . . . 10
1.4 Taxonomy of orchestration models in Web service composition . . . 12
1.5 QoSeBroker architecture . . . 13

2.1 Matching examples . . . 26
2.2 Fragment of the vehicle ontology . . . 30

3.1 Tree structure . . . 57
3.2 Illustration of tree construction process . . . 61
3.3 Illustration of tree traversal algorithm . . . 63
3.4 Traversal of the tree boundaries after each phase of the tree construction 65

4.1 Conceptual architecture of the PMRF . . . 70
4.2 Functional architecture of the PMRF . . . 71
4.3 Extract from class Matching . . . 73
4.4 Ontology example about Health Insurance . . . 74
4.5 Configuration 1 vs Configuration 2: Average precision . . . 76
4.6 Configuration 1 vs Configuration 2: Recall/Precision . . . 77
4.7 Configuration 1 vs Configuration 2: Query response time . . . 77
4.8 Configuration 1 vs Configuration 4: R-Precision . . . 78
4.9 Configuration 1 vs Configuration 4: Recall/Precision . . . 79
4.10 Configuration 5 vs Configuration 6: Average precision . . . 79
4.11 Configuration 5 vs Configuration 6: Query response time . . . 80
4.12 Effect of the criteria order: Average precision . . . 81
4.13 Effect of the criteria order: Recall/precision . . . 81
4.14 Comparative study: Recall/Precision . . . 82
4.15 Comparative study: Average precision . . . 83
4.16 Comparative study: Query response time . . . 84
4.17 Comparative study: Memory usage . . . 85

List of Tables

1.1 Web service composition concerns 11
1.2 Comparison of matchmaking framework 19

2.1 Elements of the semantic distance values set V 24
2.2 Weighting system 26
2.3 Improved weighting system 26
2.4 Elements of the semantic distance values set V 28
2.5 Extended weighting system 29
2.6 Improved weighting system 30
2.7 An example Attributes List 34
2.8 An example Criteria Table 36
2.9 Comparison of similarity measure computing algorithms 43
2.10 Comparison of matching algorithms 44

3.1 Weights of similarity degrees 47
3.2 Web services identified by the matching algorithm 48
3.3 Precision of score-based ranking 52
3.4 Ordinal rank of similarity degrees 53
3.5 Comparison of ranking approaches 66

4.1 Description of the evaluated configurations 75
4.2 Order of the criteria considered 80

Introduction

General context

The W3C[1] defines a Web service as ‘a software system designed to support interoperable machine-to-machine interaction over a network’ [80]. The Web service architecture is defined by the W3C in order to determinate a common set of concepts and relationships that allow different implementations working together [18]. The basic Web service architecture contains three elements: (i) *service requester*, which is the software system that requests; (ii) *service provider*, which is the software system that would process the request and provides the data; (iii) *service registry* which contains additional information about the service provider. More information about Web service architecture is given in Section 1.2.

Individual Web services are conceptually limited to relatively simple functionalities modelled through a collection of simple operations. However, for certain types of applications, it is necessary to combine a set of individual Web services to obtain more complex ones, called *composite* or *aggregated* Web services. Service composition can roughly be defined as the process of combining existent Web services in order to obtain new ones. It is considered as a crucial task since it saves money and human efforts, especially if the task is entirely automatized. The composition process raises new issues in line with the evolvement of the Web and other inherent technologies.

Syu et *al.* [78] categorize Web service composition into three core research concerns (service classification, planning, and selection) as well as two crosscutting research concerns (service description and matchmaking), which cut across all core concerns and widely affecting and involving in them. Syu et *al.* [78] also identify two types of matchmaking for Web service composition: matchmaking between service junctions as well as between services. The aim of the first type of service matchmaking is to determine whether two services are eligible to be related together or not. Basically, in this type of matchmaking, a service’s outputs are compared to the inputs of its successor. In the second type of matchmaking, the purpose is to compare the similarity between two services, for example, inputs of the first service are compared to the inputs of the second one. This research project is concerned the matchmaking between Web services.

[1] The W3C (World Wide Web Consortium) is the main international standards organization for the World Wide Web. See: `http://www.w3.org/`.

Motivation and objectives

The *matchmaking* is a crucial operation in Web service composition. The objective of matchmaking is to discover and select the most appropriate (i.e., that responds better to the user request) Web service among the different available candidates. Several matchmaking frameworks are now available in the literature, e.g., [4][50][53][62][78][86][82]. Ludwig [53], for instance, proposes two matchmaking approaches: one that is based on a genetic algorithm, and the other is based on a memetic algorithm to match consumers with services based on Quality of Service (QoS) attributes. Wang et *al.* [82] propose the use of utility function to evaluate each component service based on the definition in [86] and then map the multi-dimensional QoS composite Web service to the multi-dimensional multi-choice knapsack. Finally, they propose a fast heuristic algorithm for solving the problem. However, most of these frameworks present at least one of the following shortcomings:

1. use of strict syntactic matching, which generally leads to low recall and low precision of the retrieved services;
2. use of capability-based matchmaking, which is proven [24] to be inadequate in practice;
3. lack of customisation support;
4. lack of accurate ranking of matching Web services, especially within semantic-based matching.

These shortcomings are discussed in more detail in the next chapter (see Section 1.7). The objectives of this research project are to propose conceptual and algorithmic solutions to jointly deal with previous shortcomings. More precisely, the main aims of this research project are:

1. Refine and improve the matchmaking algorithms proposed in [24][15][16];
2. Propose new algorithms for scoring and ranking Web services;
3. Conceive and develop a prototype supporting the matching and ranking algorithms;
4. Study the complexity, test and evaluate the performance of proposed algorithms.

Several existing frameworks have influenced this research project, especially the proposals of [42][7][64][24] [15][16]. Although that these proposal are based on semantics, they fail to take into account jointly the shortcomings of Web matchmaking discussed earlier. Indeed, the proposal of [7][42][73] do not support any customization while those of [15][16][24] do not propose solutions for ranking Web services. Some proposals including [9][34] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only. The proposal of [15][16] lack effective

implementation of the proposed matchmaking framework. Indeed, the authors discuss very generally and very briefly the technical issues. In addition, the authors do not precise how the similarity degree is computed and how the different matching Web services are ranked before provided to the user. Finally, there is a lack of effective evaluation and performance analysis of matching algorithms.

The solutions proposed in this research project permit to overcome the first, third and fourth shortcomings of semantic matchmaking frameworks. The second shortcoming is not addressed in this research project. However, the QoS-based semantic matching algorithms proposed in [16]—and which still apply here—can solve this issue.

Structure of the document

This document is structured into four chapters, in addition to this introductive chapter and the general conclusion. The following is an outline.

Chapter 1. **Web service matchmaking and ranking: State of the art** This chapter first introduces the basic concepts on Web services. Then, it discusses the problem of Web service composition. Next, it reviews existing approaches of Web service matchmaking and ranking. Finally, the chapter discusses in more details the shortcomings of existing matchmaking frameworks.

Chapter 2. **Web services matchmaking** This second chapter first introduces an improved algorithm for similarity measure computing. Next, it presents a series of semantic matchmaking algorithms with different levels of customization. Finally, the chapter compares the proposed algorithms to some existing ones.

Chapter 3. **Web services ranking** This chapter first presents a technique for computing Web services scores based on the information provided by the user. Then, it details three different algorithms for ranking matching Web services: score-based, rule-based and tree-based. This chapter also studies the computational complexity of all ranking algorithms and discusses and compares the proposed algorithms to existing ones.

Chapter 4. **Implementation and performance evaluation** This chapter first provides the conceptual and functional architecture of the developed prototype. Then, it details the developed prototype. Finally, it tests and evaluates the performance the proposed matching and ranking algorithms and compare their performances to some existing proposals.

Chapter 1

Web Services Matchmaking and Ranking: State of the Art

The objective of this chapter is to provide the basic concepts of Web service composition and matchmaking. A special attention will be given to review the most important semantic-based matchmaking algorithms that have influenced this research project. The chapter also reviews some Web services ranking methods and discusses the shortcomings of existing matchmaking and ranking algorithms.

1.1 Introduction

Web services are reusable software components on the Web which can be discovered, fetched, and invoked [69]. Several Web services are now used in different application domains including e-commerce, marketing, industry, biology, human sciences, etc. Composing individual Web services to construct a new, more complex, Web services is a current solution to dealt with complex situations in organizations. Discovering the most appropriate Web service is a crucial step in Web service composition. An important component of discovery process is the matchmaking algorithm itself [7], which is the core part of Web service discovery process [63].

Traditional Web services are based on strict syntactic matching, which generally leads to low recall and low precision of the retrieved services [54]. To avoid the shortcomings of these frameworks, different advanced techniques and algorithms have been used such that genetic algorithmic [53], utility function [82][86], etc. More recently, the semantic-based matchmaking algorithms have been used (for a survey, see e.g., [56][67]). However, most of proposed semantic-based matchmaking algorithms focus on capability attributes only. Furthermore, they lack the support of customization. But the most critical point of the Web semantic-based matchmaking algorithms is the ranking of matching services [71].

The objective of this chapter is to provide the basic concepts of Web service composition and matchmaking. A special attention will be given to review the most important

semantic-based matchmaking algorithms and discuss their shortcomings. The rest of the chapter is organised as follows. Section 1.2 introduces the basic concepts of Web services. Section 1.4 deals with Web service composition Section 1.5 reviews semantic-based matchmaking algorithms. Section 1.6 deals with Web services ranking. Section 1.7 discuses the shortcomings of matchmaking and ranking algorithms. Section 1.8 concludes the chapter.

1.2 Web service definition and basic concepts

1.2.1 Definition

Several definition of a Web service have been proposed in the literature. The UDDI (Universal Description Discovery and Integration) standard consortium published defines a Web service as follows [22]:

> A Web service is a self contained, modular business application that have open Internet oriented, standard-based interfaces.

According to this definition, a Web service is obviously an autonomous piece of software characterized by its modularity. In other words, a Web service is easily pluggable in more complex applications.

A more specific definition proposed by the W3C is the following [79]:

> A Web service is a software application identified by a Uniform Resource Identifier (URI), whose interfaces and bindings are capable of being defined, described, and discovered as eXtensible Markup Language (XML) artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.

According to this definition, a Web service is a resource obtainable via the Internet, identified by a unique identifier called URI. Web services permit the interactions between different softwares, by providing a machine processable interface in XML format. This ability enhances the interoperability between applications, making the Web easily used by machines and freeing humans from monotonous tasks.

1.2.2 Web service standards

Web services use a set of standards and protocols. Many of these standards are being worked out by the UDDI project. The following descriptions, reproduced from [76], provide the main standards and protocols used in Web services.

Universal Description, Discovery, and Integration The Universal Description, Discovery, and Integration (UDDI) is a protocol for describing available Web services components. This standard allows businesses to register with an Internet directory that will help them advertise their services, so companies can find one another and conduct

transactions over the Web. This registration and lookup task is done using XML and HTTP(S)-based mechanisms.[1]

Simple Object Access Protocol The Simple Object Access Protocol (SOAP) is a protocol for initiating conversations with a UDDI Service. SOAP makes object access simple by allowing applications to invoke object methods or functions, residing on remote servers. A SOAP application creates a request block in XML, supplying the data needed by the remote method as well as the location of the remote object itself.

Web Service Description Language The Web Service Description Language (WSDL) is the proposed standard for how a Web service is described is an XML-based service IDL (Interface Definition Language) that defines the service interface and its implementation characteristics. WSDL is referenced by UDDI entries and describes the SOAP messages that define a particular Web service.

1.2.3 Web service architecture

Figure 1.1 shows a graphical representation of the basic Web service architecture. It consists of three basic entities:

- **Service provider**. The service provider creates or simply offers the Web service. The service provider needs to describe the Web service in a standard format, which is often XML, and publish it in a central service registry.
- **Service registry**. The service registry contains additional information about the service provider, such as address and contact of the providing company, and technical details about the service.
- **Service consumer**. The service consumer retrieves the information from the registry and uses the service description obtained to bind to and invoke the Web service.

Web service architecture is loosely coupled, service oriented. The WSDL uses the XML format to describe the methods provided by a Web service, including input and output parameters, datatypes and the transport protocol, which is typically HTTP, to be used. The UDDI suggests means to publish details about a service provider, the services that are stored and the opportunity for service consumers to find service providers and Web service details. The SOAP is generally used for XML formatted information exchange among the entities involved in the Web service model.

A Web services architecture then requires three fundamental operations: *publish*, *find* and *bind*. These operations, which can occur singly or iteratively, are described as follows [76]:

[1] The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. The Hypertext Transfer Protocol Secure (HTTPS) is the result of simply layering the HTTP on top of the SSL/TLS protocol.

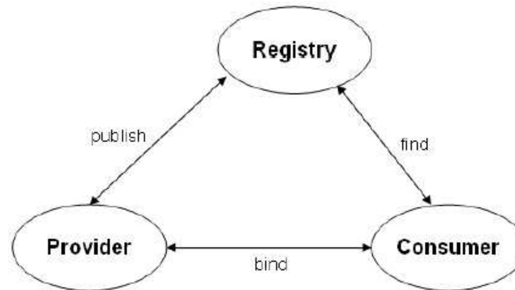


Figure 1.1: Web service architecture

- *Publish*: To to be accessible; a service description needs to be published so that the service requester can find it. Where it is published can vary depending upon the requirements of the application.
- *Find*: In the find operation, the service requester retrieves a service description directly or queries the service registry for the type of service required. The find operation can be involved in the two different lifecycle phases for the service requester: at design time to retrieve the services interface description for program development, and at runtime to retrieve the services binding and location description for invocation.
- *Bind*: Eventually, a service needs to be invoked. In the bind operation the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

These operations are depicted in Figure 1.1 by the keywords 'publish', 'bind' and 'find'.

1.3 Semantic Web service

1.3.1 Definition

The first Web service interfaces was described in WSDL. The WDSL is inherently designed to describe the functional aspects (e.g. service type, port to bind to, the type of parameters, etc.) of a Web service. In addition, the WDSL is not designed to publish the non-functional aspects because WSDL is not designed to take the 'semantic descriptions' of the service [76]. To avoid these shortcomings, a common solution consists in enriching WSDL with semantics support. This permits to enhance the interpretability since adding *semantics* to a Web service description makes it *understandable* by other machines.

A Web service enhanced with semantics is called a *semantic Web service.* The term 'Semantic Web' was coined by Tim Berners-Lee for a Web of data that can be processed by machines (see [10]). The Semantic Web is a collaborative movement led

by international standards body the W3C. Figure 1.2 presents the timeline history of XML and Semantic Web W3C Standards. The W3C defines Semantic Web as follows [81]:

> The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise and community boundaries.

Figure 1.2: XML and Semantic Web W3C standards timeline-history
Source: [11].

The current World Wide Web represents information using natural languages. The information is intended for human readers but not machines. The Semantic Web is 'an extension of the current Web in which information is given well-defined meaning, enabling computers and people to work in better cooperation'. It is a mesh of information that can be automatically processed and understood by machines.

1.3.2 Ontology

Ontology represents knowledge about a particular domain. It includes a set of machine-interpretable definitions of basic concepts in the domain and relations among them. It consists of a set of axioms which place constraints on sets of individuals (called classes) and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided.

Ontology defines a common vocabulary to share information in a domain. Indeed, enabling reuse of domain knowledge was one of the driving forces behind recent surge in ontology research. Additionally, if we need to build a large ontology, we can integrate several existing ontologies describing portions of the large domain. Analyzing domain knowledge is possible once a declarative specification of the terms is available. The main reason to use ontologies in computing is that they facilitate interoperability and

machine reasoning. Indeed, formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them.

Onlology requires the use of a formal language to describe the concepts and relationships in a domain. A common ontology for Web services is the OWL-S (formally DAML-S) ontology, which aims to facilitate automatic Web service discovery, invocation and composition. It describes properties an capabilities of Web services but does not provide details about how to represent other information of Web services (e.g. QoS descriptions).

1.3.3 Semantic Web standards

Web services need to be described in a high-level and abstract manner. This enables their automatic discovery and composition of desired functionality. The Semantic Web involves publishing in languages specifically designed for supporting semantics. The following are some common standards for Semantic Web services representation.

RDF The Resource Description Framework (RDF) is a basic semantic mark-up language for representing information about resources on the Web. It is used in situations where the information needs to be processed by applications rather than humans. RDF Schema (RDF-S) is a language for describing RDF vocabulary. It is used to describe the properties and classes of RDF resources.

OWL The Web Ontology Language (OWL) is a family of knowledge representation languages or ontology languages for authoring ontologies or knowledge bases. The languages are characterized by formal semantics and RDF/XML-based serializations for the Semantic Web. The OWL provides more vocabulary for describing properties and classes of RDF resources than RDF-S. The OWL adds, among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

OWL-S The OWL-S (Web Ontology Language for Services) is an OWL based semantic markup for Web services. It provides a language to describe the properties and capabilities of Web services. OWL-S can be used to automate Web service discovery, execution, composition and interoperation. The OWL-S is required to perform the following tasks automatically:

- Web service discovery: extract the information from the page in order to find a required service.
- Web service invocation: OWL-S along with the domain ontology specifies the invocation methods of a Web service (e.g. necessary inputs, expected outputs).

- Web services composition and interoperation: OWL-S provides declarative way to specify prerequisite and consequences of a service which helps software agents in composing different Web services.

The OWL-S provides Service Profile, Service Model and Service Grounding to represent Description, Functionality and Access Mechanism respectively.

1.3.4 Extension of OWL-S

An extended version of the OWL-S has been proposed in [1]. OWL-S specifies an upper ontology of services that defines the structure of a service description. OWL-S defines that a service presents a *ServiceProfile* (what the service does), is described by a *ServiceModel* (how it works) and supports a *ServiceGrounding* (how to access it) (see Figure 1.3).

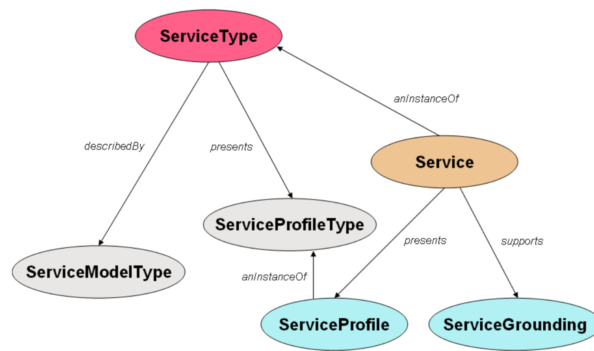


Figure 1.3: Extended OWL-S upper ontology
Source: [1].

The enhanced OWL-S defines a *ServiceType* class hierarchy in addition to the service hierarchy. The *ServiceProfile* of an instance points to the corresponding *ServiceProfileType* for mandatory portion common to all instances. It could, however, add its own precondition and effects. Similarly, it may support additional outputs as well as inputs, in which case it specifies the default values of additional inputs so that the compositions done using *ServiceProfileType* remain valid.

Within the initial OWL-S, the functional requirements of a Web service are expressed through IOPE (Inputs-Outputs-Preconditions-Effects), which captures the transformation performed by this service. OWL-S can also be used to represent non-functional requirements through profile attributes, which may contain parameters other than the functional IOPE. In the extended OWL-S upper ontology, the functional requirements are represented in *ServiceProfileType*. The *ServiceProfile* of an instance inherits these functional requirements from the *ServiceProfileType* and adds the non-functional requirements to it.

1.4 Web service composition

Syu et al. [78] categorize Web service composition into three core research concerns (service classification, planning (also called combination), and selection) as well as two crosscutting research concerns (service description and matchmaking), which cut across all core concerns and widely affecting and involving in them. Table 1.1 provides a brief definition of these concerns. The objective of Web service composition is to aggregate existing elementary services in order to created a new non-existing one. The Web service composition is an important task in SOA, especially if it is entirely automated. This because it it saves implementation efforts and time.

Table 1.1: Web service composition concerns

Service Classification	assign services into different groups based on some criteria
Service Planning	creation of workflows without human influence and efforts
Service Selection	selection of an appropriate service for each activity in the workflow
Service Description	description and expression of services
Service Matchmaking	identifying services that are compatible with the user specifications

1.4.1 Web service composition approaches

Different Web service composition approaches have been proposed in the literature. Most of research investigations on Web service composition were centred on the definition of the orchestration model [65]. The orchestration model defines the techniques adopted to construct the workflow. Petrova and Dimov [65] present a taxonomy of Web service composition approaches, which is summarised in Figure 1.4. They distinguish the following composition approaches:

- Signature Matching (SM) techniques; e.g. [61].
- Artificial Intelligence (AI) Planning approaches; e.g. [41][5][52] [37].
- Planning as Model Checking (PCM) approaches ; e.g. [85][66].
- π-calculus which is a process algebra using orchestration and choreography languages; [19] e.g. [68][13].
- Graph Model (GM) based approaches; e.g. [2][17].
- PetriNets model; e.g. [21].
- State chars models.

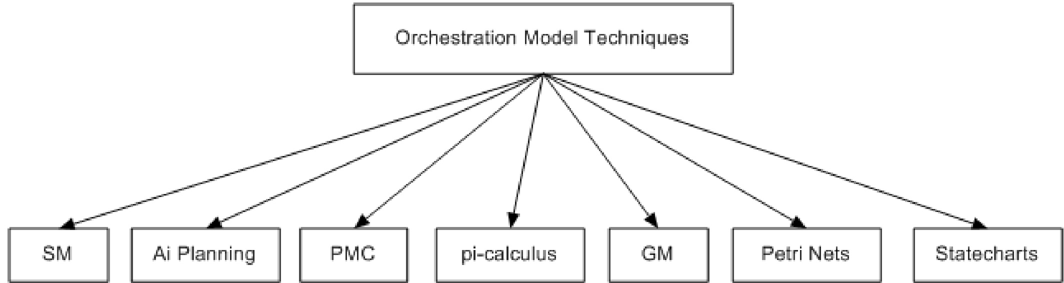


Figure 1.4: Taxonomy of orchestration models in Web service composition

1.4.2 Composition approach

The result of this research project will be integrated in Web service composition system QoSeBroker (for QoS-Enhanced Broker) proposed in [14]. The key elements of the composition approach adopted in QoSeBroker are: the composition graph, potential executable plans and executable plan. The *composition graph* is an abstract representation of functional requirements provided by the user. It models the *invocation* relationships between the individual Web services contained in the composite Web service. The set of *potential executable* plans is the composite *service instances* which are obtained by replacing each services type in the composition graph by its instances using a set of transformation rules. The objective of the transformation operation is to include the different semantics of the BPEL[2] constructors. Among the different potential executable plans only one called *executable* plan should be selected and transformed to a workflow for effective execution.

The composition operation starts by user specification of functional and non-functional requirements and leads to an executable plan that can be handed off to runtime environment for execution. The proposed approach to support the composition operation contains three phases:

1. *Logical composition*: First, the functional requirements provided by the user are used to generate the composition graph.

2. *Physical composition*: Second, the composition graph is transformed to obtain the set of potential executable plans.

3. *Evaluation and selection*: Third, the different potential executable plans are evaluated and compared in order to select one executable plan. The latter is then transformed into a workflow and then deployed, discovered and invoked.

The service composition approach is implemented by a layered system called QoSeBroker (for QoS-enhanced Broker). The architecture of QoSeBroker is given in Figure 1.5. A detailed description of QoSeBroker is given in [14].

[2]The BPEL (Business Process Execution Language) is a standard executable language for specifying actions within business processes with Web services; See [12].

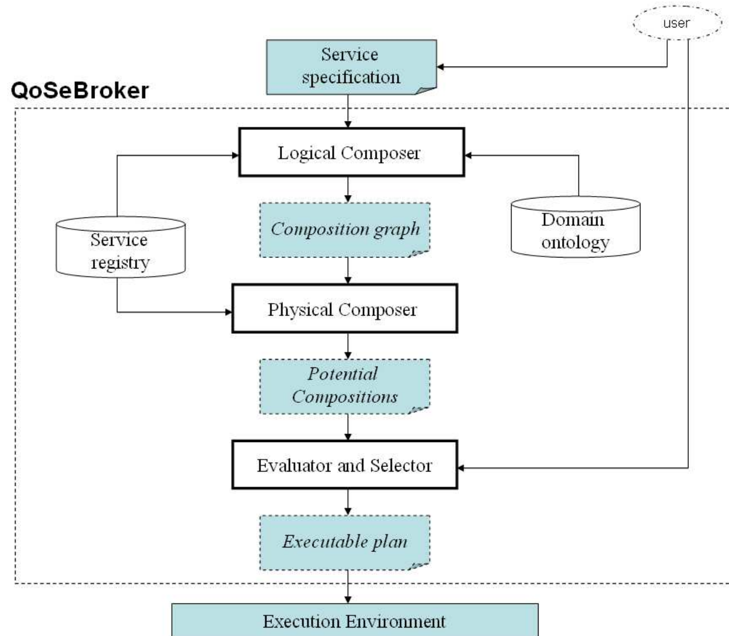


Figure 1.5: QoSeBroker architecture
Source: [14].

1.5 Web services matchmaking

The matchmaking can be defined as the process of identifying services that are compatible with the user specifications. In other words, given a service repository and a specification of a requested service, how to find the most pertinent service in response to the requester needs. The first attempts to resolve this issue have been entirely based upon the Web service interface description, precisely the WSDL files. The first matchmaking frameworks, (e.g. as Jini [4], Konark [50] and Salutation [62]) are based on strict syntactic matching, which generally leads to low recall and low precision of the retrieved services [54]. Klusch [39] classifies the matchmakers into three categories, according to the employed techniques: logic-based, non-logic-based and hybrid techniques. The analysis of all these categories showed that logic-based techniques generate the lowest precision and recall rate [39]. In the next section, we provide a brief review of logic-based matchmaking.

To overcame the shortcomings of traditional matchmaking frameworks, several authors proposed to include the semantics in the matchmaking process [69]. The first semantic matchmaker have been proposed by Paolucci et *al* [64]. This semantic matchmaker adopts a greedy approach. The idea is to assign a degree of match between service concepts and then aggregate the results to obtain an overall service match. The degree of match in [64] is computed according to Algorithm 1.

During the matching process, the request inputs R_{in} and outputs R_{out} are matched to the service inputs S_{in} and outputs S_{out}, respectively. A degree of match between

each concept is then calculated. A greedy algorithm is used to aggregate the matching results in order to obtain the overall service matching degree. Finally the services are ranked according the following order of preference:

$$\textbf{Exact} \rangle \textbf{ Plug-in} \rangle \textbf{ Subsumes} \rangle \textbf{ Fail}.$$

Algorithm 1: Degree of Match (as defined by [64])

```
  Input  : C_R, // first concept.
           C_A, // second concept.
  Output: exact, plug-in, subsumes, fail
1 if (C_R ⊑_1 C_A) then          /* direct child/parent or equivalence relation */
2 |  return exact ;
3 else if (C_R ⊏ C_A) then                     /* indirect child/parent relation */
4 |  return plug-in ;
5 else if (C_R ⊐ C_A) then           /* direct or indirect parent/child relation */
6 |  return subsumes ;
7 else
8 |  return fail ;
9 endif
```

The first drawback of Paolucci et *al.* [64] approach is the way the degree of match is computed. Indeed, calculating a subsumes relation over a large ontology can be time consuming due to the inference process. The second drawback, as noticed by [7], concerns the problem of false positives and false negatives in the final results. A false negative is a relevant service tagged as irrelevant, and a false positive is an irrelevant service tagged as relevant. In other words, the algorithm of Paolucci et *al.* [64] generates a low precision and recall rate.

Several semantic matchmakers inspired by [64] are now available in the literature, see e.g., [7][9][24][29][34][42][51][73][77]. Bellur et *al.* [7] propose an algorithm that improves considerably the algorithm of [64]. Bellur et *al.* define four degree of match (Exact, Plug-in, Subsumes and fail as default). Then, the R_{in} and R_{out} are matched to S_{in} and S_{out} by constructing a bipartite graph where the vertices correspond to concepts associated with the attribute. The vertices in the left side of the bipartite graph correspond to advertised services while those in the right side correspond to the requested service. The edges correspond to the semantic relationships between concepts in left and right sides of the graph. Then, they assign weights to each edge (Exact: w_1, Plug-in: w_2, Subsumes: w_3, Fail w_4 with $w_4 > w_3 > w_2 > w_1$). Finally, they apply the Hungarian algorithm [48] to identify the complete matching that minimizes the maximum weight in the graph. The final returned degree is the one corresponding to the maximum weight in the graph. Then, the selected assignment is the one representing a strict injective mapping, such that the maximal weight is minimized. To obtain the optimal assignment.

The computation of the degree of match as detailed in [64], consumes time and memory. However, the use of a bipartite graph reduces considerably false positives and false negatives in the final results.

The SPARQLent [73] is a semantic matchmaker designed to help agents in the discovery of OWL-S services. The agent's goals and the advertised services' preconditions,

postconditions, inputs and outputs are expressed by means of SPARQL. Then, using a shared knowledge base, the RDF graphs (corresponding to SPARQL queries) are matched and the degree of satisfiability is calculated. Sbodio et *al.* [73] propose also an approximate matchmaking version of SPARQLent where the services IOPE attributes are divided into basic graph patterns and optional graph patterns. The constraints in the optional graph patterns are then relaxed in order to satisfy the agent goal. Services are then ranked according to the number of satisfied constraints in the optional graphs.

The first disadvantage of SPARQLent is the use of SPARQL. In fact, querying a large Knowledge base can be time consuming since SPARQLent uses OWL-DL entailment regime and supports inference at all levels [72]. The second drawback is the ranking process. Indeed some services do not contain preconditions and effects; thus they are disadvantaged during the matchmaking leading to bias in the ranking results.

The iSeM [43] [42] is an hybrid matchmaker offering different filter matchings: logic-based, approximate reasoning based on logical concept abduction for matching Inputs and Outputs. iSeM uses SVM classifier to aggregate the results of several matching modules, each adopting a different approach. These approaches are based on logic, text similarity, ontology structure and approximate logic. The iSeM also filters services using a Precondition/ Effect matching plugin. In what follow, we only present the logic-based module since we perform only logic-based matching in our framework.

The iSeM-logic-based-module generates a degree of similarity between the request and a service $MatchIO(R, S)$ using the following approach. First, the request inputs (outputs) R_{in} (R_{out}) are matched to the service inputs (outputs) S_{in} (S_{out}). This assignment is obtained using a bipartite graph. It is defined as $BPG_X(C, D)$ with C the request concepts, D the service concepts and $X = \{\equiv, \sqsupseteq_1, \sqsubseteq_1, \sqsupset, \sqsubset\}$ where:

- $\equiv$: equivalence relation;
- $\sqsupseteq_1$: direct parent child relation;
- $\sqsubseteq_1$: direct child parent relation;
- $\sqsupset$: parent child relation;
- $\sqsubset$: child parent relation.

The weight of an edge expresses whether the X relation is fulfilled or not. In detail, for $c \in C$ and $d \in D$, if cXd is true then $weight(c, d) = 1$ otherwise $weight(c, d) = 0$. Next, the degrees are calculated according to the following definitions:

$MatchIO(R,S) \in \{$**Exact, Plug-in, Subsumes, Subsumed-by, LFail**$\}$ with

Exact: $BPG_{\equiv}(S_{in}, R_{in}) \neq \emptyset$
$\wedge \ \forall (I_S, I_R) \in BPG_{\equiv}(S_{in}, R_{in}) : I_S \equiv I_R$
$\wedge \ BPG_{\equiv}(R_{out}, S_{out}) \neq \emptyset$
$\wedge \ \forall (O_R, O_S) \in BPG_{\equiv}(R_{out}, S_{out}) : O_R \equiv O_S$

Plug-in: $BPG_{\sqsupseteq}(S_{in}, R_{in}) \neq \emptyset$
$\wedge \ \forall (I_S, I_R) \in BPG_{\sqsupseteq}(S_{in}, R_{in}) : I_S \sqsupseteq I_R$
$\wedge \ BPG_{\sqsupseteq_1}(R_{out}, S_{out}) \neq \emptyset$
$\wedge \ \forall (O_S, O_R) \in BPG_{\sqsupseteq_1}(R_{out}, S_{out}) : O_R \sqsupseteq_1 O_S$

Subsumes: $BPG_{\sqsupseteq}(S_{in}, R_{in}) \neq \emptyset$
$\wedge \ \forall (I_S, I_R) \in BPG_{\sqsupseteq}(S_{in}, R_{in}) : I_S \sqsupseteq I_R$
$\wedge \ BPG_{\sqsupseteq}(R_{out}, S_{out}) \neq \emptyset$
$\wedge \ \forall (O_S, O_R) \in BPG_{\sqsupseteq}(R_{out}, S_{out}) : O_R \sqsupseteq O_S$

Subsumed-by: $BPG_{\sqsupseteq}(S_{in}, R_{in}) \neq \emptyset$
$\wedge \ \forall (I_S, I_R) \in BPG_{\sqsupseteq}(S_{in}, R_{in}) : I_S \sqsupseteq I_R$
$\wedge \ BPG_{\sqsubseteq_1}(R_{out}, S_{out}) \neq \emptyset$
$\wedge \ \forall (O_S, O_R) \in BPG_{\sqsubseteq_1}(R_{out}, S_{out}) : O_R \sqsubseteq_1 O_S$

LFail None of the above logical filters is respected

Finally, the degrees are ranked according to this order of preference:

Exact $\rangle$ **Plug-in** $\rangle$ **Subsumes** $\rangle$ **Subsumed-by** $\rangle$ **LFail**.

The main weakness of iSeM-logic-based-module is its low-precision rate. As indicated in [42], it generates 45% average precision rate according to tests performed over OWLS-TC test collections. In fact, this low-precision rate is due to the coarse-grained produced degrees. Indeed, there is only five degrees as stated above.

An important shortcoming of SPARQLent and ISEM matchmakers is the lack of accurate ranking. Indeed, the ranking of matching Web services is crucial when the number of Web services that match the user request is high. It allows the user to choose among the different alternatives without restricting him to few choices. In fact, most logic-based approaches cited in literature [7][64] preform basic classification of Web services leading to coarse grained ranking results.

1.6 Web services ranking

Provide the user with an ordered list of Web services is important for two main reasons. First, in an automated Web service discovery, the top ranked Web service can be automatically chosen. Second, many Web services may match the user query, the

ranking of these services helps the user to identify the most suitable Web service to deploy.

In what follows, we discuss the main characteristics of the ranking approaches cited in literature. These approaches can be classified into three groups, along with the nature of the information that they use: (i) ranking approaches based on the Web service description information; (ii) ranking approaches based on the information external to the Web service description; and (iii) ranking approaches based on the user preferences.

Ranking approaches based on the Web service description The ranking can relay only on the information available in the web service description, which generally provides information about the service capability (IOPE attributes), the service quality (QoS attributes) and/or the service property (additional information). This type of ranking is far more practical than ranking on the basis of external information, since all needed data is available.

Among the approaches adopting this strategy, we cite [58] where the authors combine the QoS and the fuzzy logic and propose a ranking matrix. However, this approach is centered only on the QoS and discards the other Web service attributes. We also mention the work of [75] where the authors propose a ranking method that computes a dominance score between services. The calculation of these scores requires a pairwise comparison that increases the time complexity of the ranking algorithms.

Ranking based on external information Several ranking approaches use both the Web service descriptions and also other external information to the Web service description (see, e.g., [47][55][57]). For example, the authors in [74] extract the context of Web services and employs it as an additional information during the ranking. In [44], the authors rank Web services on the basis of the user past behavior. However, the use of external information can only be performed in some situations where the data is available, which is not always the case in practice.

Ranking approaches based on the user preference In this type of ranking, the ranking algorithm uses the user preferences. In [6], for instance, the authors use some constraints specified by the user. A priority is then assigned to each constraint or group of constraints. The algorithm proposed by [6] uses then a ranking Tree to perform the matching and ranking procedure.

1.7 Discussion

Several matchmaking frameworks have been proposed in the literature. However, most of these frameworks present at least one of the following shortcomings: (i) use of strict syntactic matching; (ii) use of capability-based matchmaking; (iii) lack of customisation support; and (iv) lack of accurate ranking of matching Web services. I what follows, we discuss each of these shortcomings. Table 1.2 summarizes this discussion.

1.7.1 Strict syntactic matching

The first and traditional matchmaking frameworks, such as Jini [4], Konark [50] and Salutation [62], are based on strict syntactic matching. Such syntactic matching approaches only performs service discovery and service matching based on particular interface or keyword queries from the user, which generally leads to low recall and low precision of the retrieved services [54].

Some advanced techniques and algorithms (e.g., genetic algorithmic as in [53], utility function as in [82][86]) have been used to overcome the problem of syntactic matching.

In order to overcome the limitation of strict syntactic matching, many authors propose to include the concept of semantics as in [7][9][29][34][42][51][64][73][77]. The use of ontology eliminates the limitations caused by syntactic difference between terms since matching is now possible on the basis of concepts of ontologies used to describe input and output terms [8].

1.7.2 Capability-based matchmaking

Most of existing matchmaking frameworks such as [51][64][77][9][34] utilize a strict capability-based matchmaking, which is proven [24] to be inadequate in practice. Some recent proposals including [9][34] propose to use semantics to enhance the matchmaking process but most of them still consider capability attributes only.

Chakhar [15] distinguish three types of service attributes (i) capability attributes that directly relate to working of the service, (ii) quality attributes related to the QoS and property attributes including all attributes other than those included in service capability or service quality.

Chakhar et al. [16] extend the work of [24][15] and propose different matchmaking algorithms devoted to different types of attributes (capability, property and QoS). For instance, non-functional QoS matching categorize Web services into different ordered QoS classes. The user should then select one Web service from the highest QoS class for implementation.

1.7.3 Lack of customisation support

An important shortcoming of most of existing Web service matchmaking frameworks is the lack of customisation support. To deal with this shortcoming, several authors allow the user to specify some parameters. Doshi et al. [24], for instance, present a parameterized semantic matchmaking framework that exhibits a customizable matchmaking behavior. One important shortcoming of [24] is that the sufficiency condition defined by the authors is very strict since it requires that all the specified conditions hold at the same time. This seems to be very restrictive in practice, especially for attributes related to the QoS.

Recently, [16] extend the work of [15] and propose a series of algorithms for the different types of matching. These algorithms are designed to support a customizable matching process that permits the user to control the matched attributes, the order in

which attributes are compared, as well as the way the sufficiency is computed for all matching types.

1.7.4 Lack of accurate ranking of matching Web services

In Semantic Web service, a service rank is a quantitative metric that shows the importance of a service within the process of service selection mechanism. It is known that semantic based service discovery concerns on the matchmaking process between customer's requirement and service profile or description. Its semantic matchmaking process plays a role as a ranking mechanism in service selection process. However ranking based on semantic similarity does not suit for efficient service selection. Because, from customers perspective, it is always not true that a Web service with high semantic similarity is suitable than a Web service with lower similarity. The other difficulty with semantic similarity is that the users find it hard to distinct which service is better suitable between a pool of similar services [70].

To achieve better ranking performance many ranking algorithms have been proposed in the literature. One such approach is to integrate more information besides semantic information. The information may range from time, place, location [47], customer and providers situation [55], etc. The limitation with this approach is that the system becomes more complicated when new constraints are added. To overcome this, the authors in [57] have proposed a social collaborative filtering method for ranking. This method makes use of learning other users previous experiences. This method is used successfully in all kinds of recommendation systems but the limitations with this method are information distortion and independence of service selection.

Table 1.2: Comparison of matchmaking framework

Matchmaker	Matching type	Matched attributes	Customization support	Ranking	Markup Language
Jini [4]	Syntatic	Capability attributes	No	No	No Markup Language
Konark [50]	Syntatic	Capability attributes	No	No	XML
Salutation [62]	Logic-based	Capability attributes	No	Yes	OWL-S
MatchMaker [77]	Syntactic	Capability attributes	No	No	DAMS/UDDI
RACER [51]	Syntactic	Capability attributes	No	No	DAML-S
PSMF [24]	Logic-based	Capability attributes	Yes	No	DAML-S/WSDL/UDDI
SPARQLent [73]	Logic-based	Capability attributes	No	Yes	OWL-S
iSeM-logic-based [43]	Logic-based	Capability attributes	No	Yes	OWL-S/SAWSDL
QoSeBroker [16]	Logic-based	Capability/QoS/ Property attributes	Yes	No	OWL-S

1.8 Conclusion

In this chapter, we mainly discuses several existing matchmaking frameworks and we concluded that these frameworks present at least one of the following shortcomings: (i) use of strict syntactic matching; (ii) use of capability-based matchmaking; (iii) lack of customisation support; and (iv) lack of accurate ranking of matching Web services. Although that there are some recent and intersecting matchmaking frameworks [42][7][64][24][15][16], they fail to take into jointly the shortcomings of Web matchmaking discussed earlier. In the rest of this report, we will propose several solutions to deal

with the above-cited shortcomings.

The next chapter, we improves the matchmaking algorithms proposed in a series of semantic matching algorithms supporting different levels customisation. These algorithms permit to fully overcome the first and third shortcomings of semantic matchmaking frameworks that we have addressed in Section 1.7. The second shortcoming is not addressed in this research project. However, the QoS-based semantic matching algorithms proposed in [16] can solve this issue. The fourth shortcoming will be addressed in the Chapter 4.

Chapter 2

Web Services Matching

This chapter first introduces two improved algorithms for computing the similarity measure between two Web service attributes. Then, it presents three algorithms for functional matching. The trivial matching algorithm supports no customization. The partially parameterized matching algorithm allows the user to specify the set of attributes to be used in the matching. The fully parameterized matching algorithm allows the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed. The chapter also discusses the extension of the proposed matching algorithms to other types of matching and compare them some existing ones.

2.1 Introduction

In [15][16], the authors distinguish three types of service matching: (i) *functional attribute-level matching* that implies capability and property attributes and consider each matching attribute independently of the others; (ii) *functional service-level matching* that considers capability and property attributes but the matching operation implies attributes both independently and jointly; and (iii) *non-functional matching* which focuses on the attributes related to the QoS. Furthermore, the authors in [15][16] identify three types of functional attribute-level matching: (i) conjunctive matching based on the use of "AND" connector, (ii) disjunctive matching based on the use of "OR" connector, and (iii) complex matching that uses different logical connectors, especially "AND", "OR" and "NOT" operators.

In this chapter, we focalize mainly on functional attribute-based conjunctive algorithm. More specifically, we propose three algorithms for functional attribute-based conjunctive matching supporting different levels of customization. The trivial matching algorithm supports no customization. The partially parameterized matching algorithm allows the user to specify the set of attributes to be used in the matching. The fully parameterized matching algorithm allows the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is com-

puted. The chapter also discusses the extension of the proposed matching algorithms to other types of matching and compare them some existing ones. These algorithms generalize and improve the ones proposed in [7][15][16][24].

A common operation in the matching algorithms concerns the computing of the similarity measure between two Web service attributes. In the first part of this chapter, based on the work of [7], we introduce two improved algorithms for computing the similarity measure. The first algorithm affects the precision of [7]'s algorithm but improves considerably its complexity. The second algorithm enriches the semantic distance values used in [7]'s algorithm but ameliorates considerably its precision.

The chapter is structured as follows. Section 2.2 presents some basic definitions. Section 2.3 introduces the different similarity measure computing algorithms. Section 2.4 details the different functional attribute-based conjunctive matching algorithms. Section 2.5 extends the matching algorithms to other types of matching. Section 2.6 compares the proposed algorithms to some existing ones. Section 2.7 concludes the chapter.

2.2 Background

2.2.1 Example scenario

For the purpose of illustration, we consider a Web service use case concerning travel reservation. This example is freely inspired from a use case scenario described in the WSC Web Services Architecture Usage Scenarios [35].

A company (travel agent) offers to people the ability to book complete vacation packages: plane/train/bus tickets, hotels, car rental, excursions, etc. Service providers (airlines, bus companies, hotel chains, etc) are providing Web services to query their offerings and perform reservations.

The user gets the location of a travel agent service via an unspecified way (search engine, service directory, etc).

The user provides a destination and some dates to the travel agent service. The travel agent service inquires airlines about deals and presents them to the user.

2.2.2 Basic definitions

We introduce some basic definitions of a service and other service-specific concepts. Some ones are due to [24].

Definition 2.1 (Service) *A service S is defined as a collection of attributes that describe the service. Let $S.A$ denotes the set of attributes of service S and $S.A_i$ denotes each member of this set. Let $S.N$ denotes the cardinality of this set.*$\diamond$

Example 2.1 *The travel agent company provides a Web service,* ***bookVacation****, that is defined by the following attributes:* service category*,* input*,* output*,* preconditions*,* postconditions*,* response time*,* availability*,* cost*,* security*, and* geographical location*.*$\diamond$

Definition 2.2 (Service Capability) *The capability of a service $S.C$ is a subset of service attributes ($S.C \subseteq S.A$), and includes only functional ones that directly relate to its working.*$\diamond$

Example 2.2 *The capability of **bookVacation** is:* $S.C = \{$input, output, preconditions, postconditions$\}$.$\diamond$

Definition 2.3 (Service Quality) *The quality of a service $S.Q$, is a subset of service attributes ($S.Q \subseteq S.A$), and includes all attributes that relate to its QoS.*$\diamond$

Example 2.3 *The Service Quality of **bookVacation** is:* $S.Q = \{$response time, availability, cost, security$\}$.$\diamond$

Definition 2.4 (Service Property) *The property of a service, $S.P$, is a subset of service attributes ($S.P \subseteq S.A$), and includes all attributes other than those included in service capability or service quality.*$\diamond$

Example 2.4 *The property of **bookVacation** is:* $S.P = \{$service category, geographical location$\}$.$\diamond$

2.3 Similarity measure

In this section, we provide two algorithms for computing similarity measure. These algorithms improve the one proposed in [7], which is presented in Section 2.3.2.

2.3.1 Definitions

A semantic match between two entities frequently involves a *similarity measure*. The similarity measure quantifies the semantic distance between the two entities participating in the match. A similarity measure is defined as follows.

Definition 2.5 (Similarity Measure) *The similarity measure, μ, of two service attributes is a mapping that measures the semantic distance between the conceptual annotations associated with the service attributes. Mathematically,*

$$\mu : A \times A \rightarrow \quad V$$

where A is the set of all possible attributes and V the set of possible semantic distance measures.$\diamond$

There are different possible definition of set V. In [24][15], for instance, the mapping between two conceptual annotations may take one of the following values: `Exact`, `Plugin`, `Subsumption`, `Container`, `Part-of` and `Disjoint`.

A preferential total order may now be established on the above mentioned similarity maps.

Definition 2.6 (Similarity Measure Preference) *Let V be the set of all possible semantic distance values such that $V = \{v_1, v_2, \cdots, v_n\}$. Preference amongst the similarity measures is governed by the following strict order.*

$$v_1 \succ v_2 \succ \cdots \succ v_n,$$

where $a \succ b$ means that a is preferred over b.$\diamond$

Definition 2.7 (Degree of Match) *The degree of match is a function that defines a semantic distance value between two conceptual annotations. Mathematically,*

$$\delta : CA_1 \times CA_2 \rightarrow V$$

where CA_1 denotes the first conceptual annotation and CA_2 denotes the second conceptual annotation and V the set of all possible semantic distance values.$\diamond$

This generic definition of similarity measure extends the one proposed by [24]. Other matchmaking frameworks (e.g. [7][64][77]) utilize an idea similar to μ, but label it differently. The main difference between the above-cited work concerns the way the degree of match is computed. In sections 2.3.3 and 2.3.4, we provide two algorithms for computing the similarity measure. These algorithms improve the one proposed by [7], which is presented in the next section.

2.3.2 Bellur and Kulkarni's similarity computing algorithm

2.3.2.1 Computing the degree of match

Bellur and Kulkarni [7] define the elements of the semantic distance values set according to Table 2.1.

Table 2.1: Elements of the semantic distance values set V

V	Semantic distance value
v_1	`Exact`
v_2	`Plugin`
v_3	`Subsume`
v_4	`Fail`

The degree of match between two conceptual annotations δ is established according to Algorithm 2 where:

- $\equiv$: Equivalence relationship;
- $\sqsupset_1$: Direct parent/child relationship;
- $\sqsupset$: Indirect parent/child relationship;
- $\sqsubset$: Direct or indirect child/parent relationship.

Algorithm 2: Degree of Match $\delta(\cdot,\cdot)$ as defined in [7]

```
   Input  : CA_a, // first concept.
            CA_b, // second concept.
   Output: degree of match
1  if (CA_a ≡ CA_b) then
2  |  return Exact;
3  else if (CA_a ⊐_1 CA_b) then
4  |  return Plugin;
5  else if (CA_a ⊐ CA_b) then
6  |  return Plugin;
7  else if (CA_a ⊏ CA_b) then
8  |  return Subsume;
9  else
10 |  return Fail;
11 endif
```

2.3.2.2 Computing the similarity measure

The computing of the similarity measure over two attributes is modeled by Bellur and Kulkarni [7] as a *bipartite graph matching*. Let first introduce some concepts.

Definition 2.8 (Bipartite Graph) *A graph $G = (V_0 + V_1, E)$ with disjoint vertex sets V_0 and V_1 and edge set E is called bipartite if every edge connects a vertex of V_0 with a vertex of V_1 and there are no edge in E with both endpoints are in V_0 or in V_1.*$\diamond$

Definition 2.9 (Matching in a Bipartite Graph) [7] *Let $G = (V, E)$ be a bipartite graph. A matching M of G is a subgraph $G' = (V, E')$, $E' \subseteq E$, such that no two edges $e_1, e_2 \in E'$ share the same vertex. A vertex v is matched if it is incident to an edge in the matching M.*$\diamond$

Let S^R be a service request and S^A be a service advertisement. The matching between $S^R.A_i$ and $S^A.A_j$ (i.e. $\mu(S^R.A_i, S^A.A_j)$) is computed as follows:

1. *Construction of the bipartite graph.* Let CA_0 and CA_1 be the sets of concepts corresponding to the attributes $S^R.A_i$ and $S^A.A_j$, respectively. These two sets are the vertex sets of the bipartite graph G. In other words, $G = (V_0 + V_1, E)$ where $V_0 = CA_0$ and $V_1 = CA_1$. Considering two concepts $a \in V_0$ and $b \in V_1$. An edge between a and b is valued with the degree of match between a and b, i.e., $\delta(a, b)$. Then, Bellur and Kulkarni [7] assign a numerical weight to every edge in the bipartite graph as given in Table 2.2. These weights verify the following constraints:

$$w_1 \leq w_2 \leq w_3 \leq w_4. \tag{2.1}$$

2. *Selection of the optimal matching.* A matching is selected only if it fulfills an injective mapping between V_0 and V_1. Figure 2.1 presents two matchings. The first one respects the injective mapping between V_0 and V_1 while the second does

Table 2.2: Weighting system

Degree of match	Weight of edge
`Exact`	w_1
`Plugin`	w_2
`Subsume`	w_3
`Fail`	w_4

not. A bipartite graph G may contain several possible matchings. In this case, we need to identify the *optimal matching*. According to Bellur and Kulkarni [7], the optimal matching is the one that minimizes $\max(w_k)$ with $\max(w_k)$ is the maximum weighted edge in the matching. The final returned degree corresponds to the label of the edge that maximizes $\max(w_i)$ in the obtained matching.

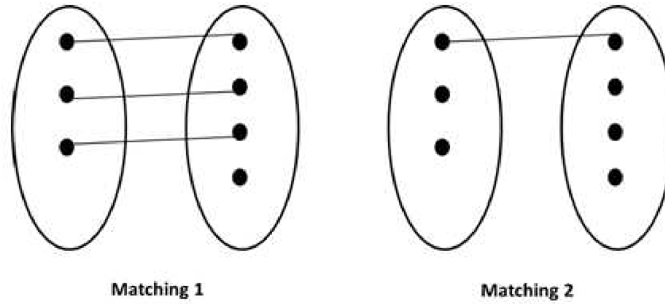


Figure 2.1: Matching examples

To select the optimal matching, Bellur and Kulkarni [7] propose the use of the Hungarian algorithm [48]. However, the application of Hungarian algorithm using the weighing system given in Table 2.2 is not not possible. This because the Hungarian algorithm minimizes the sum of weights while the optimal matching is the one that minimizes the maximum weight. Bellur and Kulkarni [7] proved that a modification in weights as in Table 2.3 permits the Hungarian algorithm to identify correctly the optimal matching.

Table 2.3: Improved weighting system

Degree of match	Weight of edge
`Exact`	$w_1 = 1$
`Plugin`	$w_2 = (w_1 * \|V_0\|) + 1$
`Subsume`	$w_3 = (w_2 * \|V_1\|) + 1$
`Fail`	$w_4 = w_3 * 100$

The computing of the similarity measure $\mu(\cdot, \cdot)$ as described above is formalized in Algorithm 3. The function **ComputeWeights** applies the weight changes as described

in Table 2.3. The function **HungarianAlg** implements the Hungarian Algorithm [48]. The $w(a, b)$ in Algorithm 3 is the weight associated to edge (a, b).

Algorithm 3: SimilarityMeasure (as in Bellur and Kulkarni [7])

```
   Input  : S^R.A_i, // attribute A_i in requested service.
            S^A.A_j, // attribute A_j in advertised service.
   Output: μ(S^R.A_i, S^A.A_j)// similarity measure between S^R.A_i and S^A.A_j.
 1 Graph G ← EmptyGraph(V_0 + V_1, E);
 2 V_0 ← concepts of attribute A_i in requested service;
 3 V_1 ← concepts of attribute A_j in advertised service;
 4 (w_1, w_2, w_3, w_4) ←ComputeWeights();
 5 for (each a ∈ V_0) do
 6     for (each b ∈ V_1) do
 7         SemanticDistance ← δ(a, b);
 8         if (SemanticDistance = Exact) then
 9             w(a, b) ← w_1
10         else
11             if (SemanticDistance = Plugin) then
12                 w(a, b) ← w_2
13             else
14                 if (SemanticDistance = Subsume) then
15                     w(a, b) ← w_3
16                 else
17                     w(a, b) ← w_4
18 M ← HungarianAlg(G);
19 Let (a', b') denotes the maximal weighted edge in M;
20 FinalSemanticDistance ← δ(a', b');
21 return FinalSemanticDistance;
```

The optimality criterion used in [7] is designed to minimize the false positives and the false negatives. In fact, minimizing the maximal weight would minimize the '`Fail`' labeled edges. However, the choice of $\max(w_i)$ as a final return value is restrictive and the risk of false negatives in the final result is higher. To avoid this problem, we propose to consider both $\max(w_i)$ and $\min(w_i)$ as pertinent values in the matching.

In the two next sections, we introduce two different algorithms for the computing the similarity measure. These algorithms are based on Bellur and Kulkarni [7]'s approach. The main difference concerns the computation of degree of match $\delta(\cdot, \cdot)$.

2.3.3 Efficient computation of the similarity measure

The semantic distance values are defined similarly to Bellur and Kulkarni [7] (see Table 2.1). The improvement concerns the degree of match which is now computed as in Algorithm 4 where:

- $\equiv$: Equivalence relationship;
- $\sqsubset_1$: Direct child/parent relationship;
- $\sqsupset_1$: Direct parent/child relationship.

In this version of the algorithm, only direct related concepts are considered for `Plugin` and `Subsume` semantic distance values. This change affects the precision of the algorithm since it uses a small set of possible concepts. However, it improves considerably its complexity. In fact, there is no need to use inference: only facts are parsed in the related ontology, which reduces the complexity of Algorithm 4 (as discussed in Section 2.3.5).This will necessarily improves the query response time. The proposed algorithm provides a balance between response time and precision, which is valuable in critical situations.

Algorithm 4: Degree of Match $\delta(\cdot,\cdot)$ for an efficient computation of $\mu(\cdot,\cdot)$

Input : CA_R, // first concept.
CA_A, // second concept.
Output: degree of match
1 **if** $(CA_R \equiv CA_A)$ **then**
2 | return `Exact`;
3 **else if** $(CA_R \sqsubset_1 CA_A)$ **then**
4 | return `Plugin` ;
5 **else if** $(CAS_R \sqsupset_1 CA_A)$ **then**
6 | return `Subsumes`;
7 **else**
8 | return `Fail` ;
9 **endif**

The computation of the similarity measure is the same as in Bellur and Kulkarni [7] (see Algorithm 3) but the interpretation of the semantic distances is as given in the beginning of this section.

2.3.4 Accurate computation of the similarity measure

In this case, we define six semantic distance values, which are given in Table 2.4. The basic idea of this second improvement is that the precision of the algorithm increases with the number of granular values.

Table 2.4: Elements of the semantic distance values set V

V	Semantic distance value
v_1	`Exact`
v_2	`Plugin`
v_3	`Subsume`
v_4	`Extended-Plugin`
v_5	`Extended-Subsume`
v_6	`Fail`

In this version of the similarity measure, the improvement is also made over the computing of degree of match $\delta(\cdot,\cdot)$, which is now computed according to Algorithm 5 where:

- $\equiv$: Equivalence relationship (it does not need inference);

- $\sqsubset_1$: Direct child/parent relationship;
- $\sqsupset_1$: Direct parent/child relationship;
- $\sqsubset$: Indirect child/parent relationship;
- $\sqsupset$: Indirect parent/child relationship.

Then, we define an extended weighting system as Table 2.5. These weights verify the following constraints:

$$w_1 \leq w_2 \leq w_3 \leq w_4 \leq w_5 \leq w_6 \tag{2.2}$$

Table 2.5: Extended weighting system

Degree of match	Weight of edge
Exact	w_1
Plugin	w_2
Subsume	w_3
Extended-Plugin	w_4
Extended-Subsume	w_5
Fail	w_6

Algorithm 5: Degree of match $\delta(\cdot,\cdot)$ for the accurate computation of $\mu(\cdot,\cdot)$

Input : CA_R, // first concept.
CA_A, // second concept.
Output: degree of match//

1. **if** $(CA_R \equiv CA_A)$ **then**
2. | return `Exact` ;
3. **else if** $(CA_R \sqsubset_1 CA_A)$ **then**
4. | return `Plugin` ;
5. **else if** $(CA_R \sqsupset_1 CA_A)$ **then**
6. | return `Subsume` ;
7. **else if** $(CA_R \sqsubset CA_A)$ **then**
8. | return `Extended-Plugin` ;
9. **else if** $(CA_R \sqsupset CA_A)$ **then**
10. | return `Extended-Subsume` ;
11. **else**
12. | return `Fail`;
13. **endif**

In this algorithm, the consideration of indirect concepts is performed for both `Extended-Plugin` and `Extended-Subsume` semantic distance values. Following Definition (2.6), we have the following preference order:

`Exact` $\succ$ `Plugin` $\succ$ `Subsume` $\succ$ `Extended-plug-in` $\succ$ `Extended-subsume` $\succ$ `Fail`.

Direct relations (i.e. `Plugin` and `Subsume`) are preferred to indirect relations (i.e. `Extended-Plugin` and `Extended-Subsume`). To argument this choice, let consider the

Ontology fragment in Figure 2.2. A vehicle provider cannot offer every type of vehicles. This also true for the offered cars type. Hence, when the distance between the advertised concept and the requested concept increase, the match between the two concepts is less and less relevant.

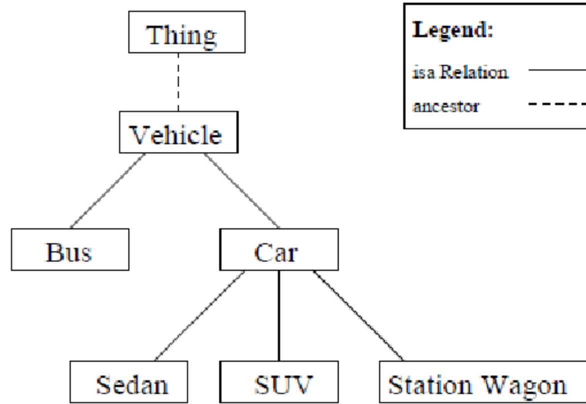


Figure 2.2: Fragment of the vehicle ontology

To apply the Hungarian Algorithm [48], we need to modify the weights as described in Table 2.6.

Table 2.6: Improved weighting system

Degree of match	Weight of edge
Exact	$w_1 = 1$
Plugin	$w_2 = (w_1 * \|V_0\|) + 1$
Subsume	$w_3 = (w_2 * \|V_1\|) + 1$
Extended-Plugin	$w_4 = (w_3 * \|V_1\|) + 1$
Extended-Subsume	$w_5 = (w_4 * \|V_1\|) + 1$
Fail	$w_6 = w_5 * 100$

The computation of the similarity measure is given in Algorithm 6. This algorithm extends the one proposed by Bellur and Kulkarni [7] (see Algorithm 3). The function **ComputeWeights** applies the weight changes as described in Table 2.6. The function **HungarianAlg** implements the Hungarian Algorithm [48]. The $w(a, b)$ in Algorithm 6 is the weight associated to edge (a, b).

2.3.5 Computational complexity

In this section we discuss the complexity of Algorithms 3 and 6. The most expensive operation is these algorithms is the computing of the degree of match. As underlined

Algorithm 6: SimilarityMeasure

```
Input  : S^R.A_i, // attribute A_i in requested service.
         S^A.A_j, // attribute A_j in advertised service.
Output: μ(S^R.A_i, S^A.A_j)// similarity measure between S^R.A_i and S^A.A_j.
1  Graph G ← EmptyGraph(V_0 + V_1, E);
2  V_0 ← concepts of attribute A_i in requested service;
3  V_1 ← concepts of attribute A_j in advertised service;
4  (w_1, w_2, w_3, w_4, w_5, w_6) ←ComputeWeights() ();
5  for (each  a ∈ V_0) do
6      for (each  b ∈ V_1) do
7          SemanticDistance ←δ(a,b);
8          if (SemanticDistance = Exact) then
9              w(a, b) ← w_1
10         else
11             if (SemanticDistance = Plugin) then
12                 w(a, b) ← w_2
13             else
14                 if (SemanticDistance = Subsume) then
15                     w(a, b) ← w_3
16                 else
17                     if (SemanticDistance = Extended-Plugin) then
18                         w(a, b) ← w_4
19                     else
20                         if (SemanticDistance = Extended-Subsume) then
21                             w(a, b) ← w_5
22                         else
23                             w(a, b) ← w_6
24 M ←HungarianAlg (G);
25 Let (a', b') denotes the maximal weighted edge in M;
26 FinalSemanticDistance ← δ(a',b');
27 return FinalSemanticDistance;
```

in [24], inferring degree of match by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines which use the fast Rete [28] pattern-matching algorithm leads to $O(|R||F||P|)$ where $|R|$ is the number of rules, $|F|$ is the number of facts, and $|P|$ is the average number of patterns in each rule. Accordingly, the complexity of Algorithms 2 and 5 is $O(|R||F||P|)$. However, the computing the degree of match according to Algorithm 4 is only $O(|F|)$ since there is not inference.

Let now discuss the algorithmic complexity of Bellur and Kulkarni [7]'s approach. Let m be an approximation of the number of concepts for the attributes to be compared. Then:

1. the computation of weights is an operation of $O(1)$ complexity;
2. the construction of the graph involves the comparison of every pair of concepts. It takes then $O(m^2)$ time complexity;
3. the Hungarian Algorithm has a time complexity of $O(m^3)$.

4. the degree of match in Algorithm 2 is either $O(|R||F||P|)$.

Generally, m is likely to take small values so we will consider it as a constant. The overall time complexity of Bellur and Kulkarni [7]'s algorithm is than $O(1 + m^2|F||R||P| + m^3) \simeq O(|F||R||P|)$.

Based on the above discussion, the complexity of Bellur and Kulkarni [7]'s algorithm will be $O(1 + m^2 \mid F \mid +m^3) \simeq O(\mid F \mid)$ when the degree of similarity is computed as in Algorithm 4.

The computation of the similarity measure according to Algorithm 6 is the same as [7]'s algorithm, i.e., $O(1 + m^2|F||R||P| + m^3) \simeq O(|F||R||P|)$.

2.4 Matching algorithms

In this section we provide three functional attribute-based conjunctive matching algorithms. These algorithms support different levels of customization. The first algorithm (Section 2.4.1) supports no customization. The second algorithm (Section 2.4.2) supports allows the user to specify the set of attributes to be used in the matching. The last algorithm (Section 2.4.3) allows the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed.

2.4.1 Trivial matching algorithm

In this first case, we assume that the user can specify only the functional specification of the desired service. Let S^R be the service that is requested, and S^A be the service that is advertised. A sufficient functional attribute-level conjunctive match between S^R and S^A can be defined as follows.

Definition 2.10 (Sufficient Functional Conjunctive Trivial Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. A sufficient conjunctive match exists between S^R and S^A if for* every *attribute in $S^R.A$ there exists an identical attribute of $S^A.A$ and the similarity between values of the attributes does not fail. Formally,*

$$\begin{array}{l}\forall_i \exists_j (S^R.A_i = S^A.A_j) \wedge \mu(S^R.A_i = S^A.A_j) \succ \textbf{Fail} \\ \Rightarrow \mathit{SuffFuncConjMatch}(S^R, S^A) \quad 1 \leq i \leq S^R.N. \diamond\end{array} \tag{2.3}$$

According to this definition, all the attributes of the requested S^R should be considered during the matching process. This is the default case with no support of customization.

The functional attribute-level conjunctive match is formalized in Algorithm 7. This algorithm follows directly from Sentence (2.3). Algorithm 7 proceeds as follows:

1. Loops over the set of attributes $S^R.A$ of the requested service S^R and for each one identify the corresponding attribute in advertised service S^A. The corresponding attributes are appended into two different lists `rAttrSet` (for requested service

S^R) and `aAttrSet` (for advisable service S^A). This operation is implemented by sentences 1 to 7 in Algorithm 7.

2. Loops over the corresponding attributes and for each one computes the similarity measure. The processes stops if the current compared pairs of attributes are not similar. This operation is implemented by sentences 8 to 11 in Algorithm 7.

The output of Algorithm 7 is either success (if all the attributes of the requested service S^R have similar attributes in the advertised service S^A) or fail (if the similarity for at least one attribute of the requested $S^R.A$ fails).

Algorithm 7: Trivial Matching

```
   Input  : S^R, // Requested service.
            S^A, // Advertised service.
   Output: Boolean, // fail/success.
 1 while (i ≤ S^R.N) do
 2 |  Append S^R.A_i to rAttrSet;
 3 |  while (k ≤ S^A.N) do
 4 |  |  if (S^A.A_k = S^R.A_i) then
 5 |  |  |_ Append S^A.A_k to aAttrSet;
 6 |  |_ k ⟵ k + 1;
 7 |_ i ⟵ i + 1;
 8 while (t ≤ S^R.N) do
 9 |  if (μ(rAttrSet[t], aAttrSet[t]) = Fail) then
10 |  |_ return fail;
11 |_ t ⟵ t + 1;
12 return success;
```

The function $\mu(\cdot, \cdot)$ in Algorithm 7 permits to compute the similarity degrees using one of the following algorithms:

- Bellur and Kulkarni [7]'s algorithm (see Algorithm 3) where the degrees of match are computed by Algorithm 2.
- Bellur and Kulkarni [7]'s algorithm (see Algorithm 3) where the degrees of match are computed by Algorithm 4.
- Algorithm 6 where the degrees of match are computed by Algorithm 5.

The complexity of Algorithm 7 is discussed in Section 2.4.4 and the extension to other types of functional matching is presented in Section 2.5.

2.4.2 Partially parameterized matching algorithm

In this case the user can specify the list of attributes to consider during the information. In order to allow this, we use the concept of Attributes List that serves as a parameter to the matching process.

Definition 2.11 (Attributes List) *An Attributes List, L, is a relation consisting of one attribute, L.A that describes the service attribute to be compared. Let $L.A_i$ denotes the service attribute value in the ith tuple of the relation. L.N denotes the total number of tuples in L.*$\diamond$

Example 2.5 *Table 2.7 shows a Attributes List example.*$\diamond$

Table 2.7: An example Attributes List

$L.A$
input
output
precondition
postcondition

Let S^R be the service that is requested, and S^A be the service that is advertised. A sufficient functional attribute-level conjunctive match between services can now be defined as follows.

Definition 2.12 (Sufficient Functional Conjunctive Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. Let L be a Attributes List. A sufficient* conjunctive *match exists between S^R and S^A if for* every *attribute in L.A there exists an identical attribute of S^R and S^A and the similarity between values of the attributes does not fail. Formally,*

$$\begin{aligned} &\forall_i \exists_{j,k} (L.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succ \textbf{Fail} \\ &\Rightarrow \mathit{SuffFuncConjMatch}(S^R, S^A) \quad 1 \leq i \leq L.N. \diamond \end{aligned} \tag{2.4}$$

According to this definition, only the attributes specified by the user in the Attributes List are considered during the matching process.

The partially parameterized functional attribute-level conjunctive match is formalized in Algorithm 8. This algorithm follows directly from Sentence (2.4). Algorithm 8 proceeds as follows:

1. Loops over the set of attributes in $L.A$ and for each one identify the corresponding attribute in requested S^R advertised S^A services. The corresponding attributes are appended into two different lists `rAttrSet` (for requested service S^R) and `aAttrSet` (for advisable service S^A). This operation is implemented by sentences 1 to 10 in Algorithm 8.
2. Loops over the corresponding attributes list and for each one computes the similarity measure. The processes stops if the current compared pairs of attributes are not similar. This operation is implemented by sentences 11 to 14 in Algorithm 8.

The output of Algorithm 8 is either success (if for every attribute in the Attributes List L there similar attributes in the advertised service S^A) or fail (if the similarity for at least one attribute in the Attributes List L fails).

Algorithm 8: Partially Parameterized Matching

```
   Input  : S^R, // Requested service.
            S^A, // Advertised service.
            L, // Criteria List.
   Output: Boolean, // fail/success.
1  while (i ≤ L.N) do
2      while (j ≤ S^R.N) do
3          if (S^R.A_j = L.A_i) then
4              Append S^R.A_j to rAttrSet;
5          j ⟵ j + 1;
6      while (k ≤ S^A.N) do
7          if (S^A.A_k = L.A_i) then
8              Append S^A.A_k to aAttrSet;
9          k ⟵ k + 1;
10     i ⟵ i + 1;
11 while (t ≤ L.N) do
12     if (μ(rAttrSet[t], aAttrSet[t]) = Fail) then
13         return fail;
14     t ⟵ t + 1;
15 return success;
```

The remark given after Algorithm 7 and which concerns the computing of the similarity measure $\mu(\cdot,\cdot)$ still hold for Algorithm 8.

The complexity of Algorithm 8 is discussed in Section 2.4.4 and the extension to other types of functional matching is presented in Section 2.5.

2.4.3 Fully parameterized matching algorithm

In this case the user can control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed. Three customization are taken into account in this case:

1. A first customization consists in allowing the user to specify the list of attributes to consider;
2. A second customization consists in allowing the user to specify order in which the attributes are considered;
3. A third customization is to allow the user to specify a desired similarity measure for each attribute.

In order to support all these customizations, we use the concept of Criteria Table, introduced by [24], that serves as a parameter to the matching process.

Definition 2.13 (Criteria Table) *A Criteria Table, C, is a relation consisting of two attributes, $C.A$ and $C.M$. $C.A$ describes the service attribute to be compared, and $C.M$ gives the least preferred similarity measure for that attribute. Let $C.A_i$ and $C.M_i$ denote the service attribute value and the desired measure in the ith tuple of the relation. $C.N$ denotes the total number of tuples in C.* $\diamond$

Example 2.6 *Table 2.8 shows a Criteria Table example.*

Table 2.8: An example Criteria Table

C.A	*C.M*
input	Exact
output	Exact
service category	Subsumes

Let S^R be the service that is requested, and S^A be the service that is advertised. Based on the concept of Criteria Table, a sufficient functional attribute-level conjunctive match between services is defined as follows.

Definition 2.14 (Sufficient Fully Parameterized Match) *PartiallyParam Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be a criteria table. A sufficient conjunctive match exists between S^R and S^A if for every attribute in $C.A$ there exists an identical attribute of S^R and S^A and the values of the attributes satisfy the desired similarity measure as specified in $C.M$. Formally,*

$$\begin{aligned} &\forall_i \exists_{j,k} (C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \\ &\Rightarrow \mathit{SuffFuncConjMatch}(S^R, S^A) \quad 1 \leq i \leq C.N. \diamond \end{aligned} \tag{2.5}$$

According to this definition, only the attributes specified by the user in the Criteria Table are considered during the matching process.

The fully parameterized functional attribute-level conjunctive match is formalized in Algorithm 9. The matching algorithm follows directly from Sentence (2.5). Algorithm 9 proceeds as follows:

1. Loops over the Criteria Table and for each attribute it identifies the corresponding attribute in the requested service S^R and the potentially advisable service under consideration S^A. The corresponding attributes are appended into two different lists `rAttrSet` (for requested service S^A) and `aAttrSet` (for advisable service S^A). This operation is implemented by sentences 1 to 10 in Algorithm 9.

2. Loops over the Criteria Table and for each attribute it computes the similarity degree between the corresponding attributes in `aAttrSet` and `aAttrSet`. This operation is implemented by sentences 11 to 14 in Algorithm 9.

The output of Algorithm 8 is either success (if for every attribute in the Criteria Table C there are similar attributes in the advertised service S^A) or fail (if the similarity for at least one attribute in the Criteria Table C fails).

The remark given after Algorithm 7 and which concerns the computing of the similarity measure $\mu(\cdot, \cdot)$ still hold for Algorithm 9.

Algorithm 9: Fully Parameterized Matching

```
   Input  : S^R, // Requested service.
            S^A, // Advertised service.
            C, // Criteria Table.
   Output: Boolean, // fail/success.
 1 while (i ≤ C.N) do
 2     while (j ≤ S^R.N) do
 3         if (S^R.A_j = C.A_i) then
 4             Append S^R.A_j to rAttrSet;
 5         j ⟵ j + 1;
 6     while (k ≤ S^A.N) do
 7         if (S^A.A_k = C.A_i) then
 8             Append S^A.A_k to aAttrSet;
 9         k ⟵ k + 1;
10     i ⟵ i + 1;
11 while (t ≤ C.N) do
12     if (μ(rAttrSet[t], aAttrSet[t]) ≺ C.M_t) then
13         return fail;
14     t ⟵ t + 1;
15 return success;
```

The complexity of Algorithm 9 is discussed in Section 2.4.4 and the extension to other types of functional matching is presented in Section 2.5.

Finally, we remark that the matching algorithms presented permits to compute the similarly between a requested service S^R and advertised service S^A. In practice, however, matching process should consider all the Web services available in the registry. The matching algorithm that we proposed in [30] take into account this fact.

2.4.4 Computational complexity

In this section we discuss the complexity of the three matching algorithms introduced in Sections 2.4.1, 2.4.2 and 2.4.3. Let us first focus on the complexity of computing $\mu(\cdot, \cdot)$, which is common to the three algorithms. As discussed in Section 2.3.5, complexity of computing $\mu(\cdot, \cdot)$ depends on the the computing of the degree of match which may or not require to use the inference. According to our discussion in Section 2.3.5, the complexity of computing $\mu(\cdot, \cdot)$ is $O(|R||F||P|)$ if Algorithm 2 or Algorithm 5 are used to compute the degree of match, or is only $O(|F|)$ if Algorithm 4 is used to compute the degree of match. In what follows, we denoted by α the complexity of computing $\mu(\cdot, \cdot)$.

Let first consider Algorithm 7. Generally, we have $S^A.N \gg S^R.N$. Then, the complexity of the first outer *while* loop in Algorithm 7 is $O(S^A.N)$. Then, the worst case complexity of Algorithm 7 is $O(S^A.N) + \alpha$. Furthermore, we observe, as in [24], that the process of computing $\mu(\cdot, \cdot)$ is the most 'expensive' step of the algorithms. Hence, the complexity of Algorithm 7 is $O(S^A.N) + \alpha \asymp \alpha$.

Let now focalize on Algorithms 8 and 9. Generally, we have $S^A.N \gg S^R.N$. Hence the complexity of the first outer *while* loop in Algorithm 8 is $O(L.N \times S^A.N)$. Then, the worst case complexity of Algorithm 8 is $O(L.N \times S^A.N) + \alpha$. Based on the remark

of [24], the complexity of Algorithm 8 is $O(L.N \times S^A.N) + \alpha \asymp \alpha$. In similar way, the worst case complexity of Algorithm 9 is $O(C.N \times S^A.N) + \alpha \asymp \alpha$.

2.5 Extension of matching algorithm

The this section, we extend the proposed matching algorithms to other types of matching, namely, functional attribute-level disjunctive matching (Section 2.5.1), functional attribute-level generic matchmaking (Section 2.5.2) and functional service-level matching (Section 2.5.3).

The discussion that follow assumes a fully parameterized matching as in Section 2.4.3. The extension to trivial matching (see Section 2.4.1) or partially parameterized matching (see Section 2.4.2) is similar to fully parameterized matching (the difference concerns mainly the list of parameters to be used). Finally, we indicate that some parts of the rest of this section are reproduced from [15][16] with minor modifications.

2.5.1 Functional attribute-level disjunctive matching

A less restrictive definition of sufficiency consists in using a disjunctive rule on the individual matching measures. The attribute-level disjunctive matching is defined as follows.

Definition 2.15 (Sufficient Functional Disjunctive Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be a criteria table. A sufficient disjunctive match exists between S^R and S^A if for at* least one *attribute in $C.A$ it exists an identical attribute of S^R and S^A and the values of the attributes satisfy the desired similarity measure as specified in $C.M$. Formally,*

$$\begin{aligned} &\exists_{i,j,k}(C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i \\ &\Rightarrow \mathit{SuffFunctionalDisjunctiveMatch}(S^R, S^A).\diamond \end{aligned} \tag{2.6}$$

The functional attribute-level disjunctive match is formalized in similar way to Algorithm 9. The main change concerns the last *while* loop in 9 which should be replaced by the following:

```
while (t < C.N) do
    if (μ(rAttrSet[t], aAttrSet[t]) ⪰ C.M_t) then
        return success;
    Assign t ⟵ t + 1;
```

2.5.2 Functional attribute-level generic matchmaking

In this section we extend the algorithms proposed above to generic binary connectors by allowing the user specify the conditional relationships between the different capability and property attributes. In order to define the sufficient functional attribute-level generic match, we need to introduce the concept of sufficient single attribute match.

The sufficient single attribute match is defined as follows.

Definition 2.16 (Sufficient Single Attribute Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be a criteria table. A sufficient match exists between S^R and S^A in respect to attribute $S^R.A_i$ if there exists an identical attribute of S^A and the values of the attributes satisfy the desired similarity measure as specified in $C.M_i$. Formally, then:*

$$\begin{aligned}&\exists_{j,k}(C.A_i = S^R.A_j = S^A.A_k) \wedge \mu(S^R.A_j, S^A.A_k) \succeq C.M_i)\\&\Rightarrow \text{SuffSingleAttrMatch}(S^R, S^A, A_i).\diamond\end{aligned} \tag{2.7}$$

The single attribute matching is formalized in Algorithm 10. This algorithm follows directly from Sentence (2.7).

Algorithm 10: SuffSingleAttrMatching

Input : S^R, // requested service.
S^A, // advertised requested.
C, // criteria table.
i, // service attribute index.
Output: Boolean// success: true; fail: false.

```
1  while (j ≤ S^R.N)) do
2      if (S^R.A_j = C.A_i) then
3          Append S^R.A_j to rAttrSet; ;
4      Assign j ⟵ j + 1; ;
5  while (k ≤ S^A.N) do
6      if (S^A.A_k = C.A_i) then
7          Append S^A.A_k to aAttrSet; ;
8      Assign k ⟵ k + 1; ;
9  if (μ(rAttrSet[i], aAttrSet[i]) ⪰ C.M_i) then
10     return success; ;
11 return fail; ;
```

The sufficient functional generic match can then be defined as follows.

Definition 2.17 (Sufficient Functional Generic Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be the criteria table. Let $\mathcal{T}$ be a complex logical clause where operands are the attributes related by logical operators (e.g. or, and, not). A sufficient functional generic match between S^R and S^A holds if and only the logical clause $\mathcal{T}$ holds. Formally,*

$$\begin{aligned}&\textbf{\textit{Parse}}(\mathcal{T}) \wedge \textbf{\textit{Evaluate}}(\mathcal{T})\\&\Rightarrow \text{SuffAttrGenericMatch}(S^R, S^A).\diamond\end{aligned} \tag{2.8}$$

*where **Parse** and **Evaluate** are functions devoted respectively to parse and evaluate the logical expression $\mathcal{T}$.$\diamond$*

Example 2.7 *The following are three examples of logical expressions:*

- $\mathcal{T} = A_1$ *and* $\cdots$ *and* A_η
- $\mathcal{T} = A_1$ *or* $\cdots$ *or* A_η

- $\mathcal{T} = A_5$ *or* $(A_2$ *and* $A_3)$

It is easy to see that the first and second expressions correspond to conjunctive and disjunctive attribute-based matching. The third expression is more complex. The matching holds when either (i) the matching in respect to attribute A_5 holds, or (ii) the matching in respect to attribute A_2 and the matching in respect to attribute A_3 hold jointly.$\diamond$

The generic matchmaking is formalized in Algorithm 11. This algorithm follows directly from Sentence (2.8).

Algorithm 11: FunctionalGenericMatching

```
   Input  : S^R, // requested service.
            S^A, // advertised requested.
            C, // criteria table.
            T, // logical expression.
   Output: Boolean// success: true; fail: false.
 1 if (NOT(Parse(T)) then
 2   return fail; ;
 3 T' <-- T; ;
 4 Z <-- ∅; ;
 5 for (each  A_l ∈ T') do
 6   if (A_l ∉ Z)) then
 7     t <-- false; ;
 8     t <-- SuffSingleAttrMatch(S^R, S^A, A_l); ;
 9     replace all A_l ∈ T by the value of t; ;
10     Z <-- Z ∪ {A_l}; ;
11 if (Evaluate(T)) then
12   return success; ;
13 return fail; ;
```

2.5.3 Functional service-level matching

The service-level matching concerns both attributes and service levels. It allows the client to use two types of desired similarity: (i) desired similarity values associated with each attribute in the criteria table, and (ii) a global desired similarity that applies to the service as a whole. In order to define the sufficient functional service-level match, we need to introduce the concept of aggregation rule.

The computing of service-level similarity measure requires the definition of an aggregation rule to combine the similarity measures associated with attributes into a single measure relative to the service.

Definition 2.18 (Aggregation Rule) *An aggregation rule ζ is a mean to combine the similarity measures into a single similarly measure. Mathematically,*

$$\zeta : \mathcal{F}_1 \times \cdots \times \mathcal{F}_N \rightarrow \quad V$$

where $\mathcal{F}_j = V$ $(j = 1, \cdots, N)$ is the set of possible semantic distance measures as defined in 2.3; and N is the number of attributes included in the Criteria Table.$\diamond$

The preference amongst similarity measures is governed by the same strict total order given in Definition 2.6.

Since the similarity measures are defined on an ordinal scale, there are only a few possible aggregation rules that can be used to combine similarity measures:

- **Minimum**: picks out the minimum similarity measure,
- **Maximum**: picks out the maximum similarity measure,
- **Median**: picks out the similarity measure corresponding to the median (in terms of order).
- **Floor**: picks out the similarity measure corresponding to the floor of the median values.
- **Ceil**: picks out the similarity measure corresponding to the ceil of the median values.

The Floor and Ceil rules apply only when there is an even number of similarity measures (which leads to two median values).

The service-level similarity measure quantifies the semantic distance between the requested service and the advertised service entities participating in the match by taking into account both attribute-level and service-level desired similarity measures.

Definition 2.19 (Sufficient Functional Service-Level Match) *Let S^R be the service that is requested, and S^A be the service that is advertised. Let C be a criteria table. Let β be the service-level desired similarity measure. A sufficient service-level match exists between S^R and S^A if (i) for every attribute in $C.A$ there exists an identical attribute of S^R and S^A and the values of the attributes satisfy the desired similarity measure as specified in $C.M$, and (ii) the value of overall similarity measure satisfies the desired overall similarity measure β. Mathematically,*

$$\begin{aligned}&[\forall i \quad (\mathit{SuffSingleAttrMatch}(S^R, S^A, A_i)) \quad 1 \le i \le C.\eta] \wedge \\ &\quad [\exists j_1, \cdots, j_i, \cdots, j_\eta \quad (\zeta(s_{1,j_1}, \cdots, s_{i,j_i}, \cdots, s_{\eta,j_\eta}) \succeq \beta)] \\ &\Rightarrow \mathit{SuffFunctionalServiceLevelMatch}(S^R, S^A),\end{aligned} \tag{2.9}$$

where ζ is an aggregation rule; and for $i = 1, \cdots, N$, N is the number of attributes included in the Criteria Table, and $j_i \in \{j_1, \cdots, j_N\}$:

$$s_{i,j_i} = \mu(S^R.A_i, S^A.A_{j_i}).\diamond$$

The service-level matching in formalized Algorithm 12. This algorithm follows directly from Sentence (2.9).

Algorithm 12: FunctionalServiceLevelMatching

```
   Input  : S^R, // requested service.
            S^A, // advertised requested.
            C, // criteria table.
            ζ,// aggregation rule.
            β,// overall similarity measure.
   Output: Boolean// success: true; fail: false.
1  while (i ≤ C.η) do
2      if (NOT(SuffSingleAttrMatch(S^R, S^A, A_i))) then
3          return fail
4  while (i ≤ C.η) do
5      while (j ≤ S^R.N) do
6          if (S^R.A_j = C.A_i) then
7              Append S^R.A_j to rAttrSet; ;
8          Assign j ⟵ j + 1; ;
9      while (k ≤ S^A.N) do
10         if (S^A.A_k = C.A_i) then
11             Append S^A.A_k to aAttrSet; ;
12         Assign k ⟵ k + 1; ;
13     Assign i ⟵ i + 1; ;
14 if (ζ(μ(rAttrSet[1], aAttrSet[1]), ··· , μ(rAttrSet[C.η], aAttrSet[C.η]))) ⪰ β) then
15     return success; ;
16 return fail; ;
```

2.5.4 Computational complexity

The complexity of the two first *while* loops in Algorithm 10 is equal to $O(S^R.N) + O(S^A.N)$. Since we have generally $S^A.N \gg S^R.N$, hence the complexity of the two first *while* loops is equal to $O(S^A.N)$. Then, based on the discussion given earlier, we conclude that the worst case complexity of Algorithm 10 is $O(S^A.N) + O(|R||F||P|) \asymp O(|R||F||P|)$, where R, F and P have the same definition given earlier.

The complexity of Algorithm 11 depends on the complexity of functions **Parse** and **Evaluate**. The complexity of these functions depends on the data structure used to represent the logical expression $\mathcal{T}$ (graph, truth tables, etc.). Clearly the complexity of **Evaluate** function is largely greater than the complexity of **Parse** function. Accordingly and based on the discussed above, the complexity of Algorithm 11 is $O(|R||F||P|) + O(\gamma)$ where $O(\gamma)$ is the complexity of **Evaluate** function.

The complexity of the first *while* loop in Algorithm 12 is equal to $O(C.N \times (|R||F||P|))$ and the complexity of the second *while* loop is equal to $O(C.N \times S^A.N)$. Finding the minimum, maximum, median, floor and ceil of a list of values are $O(n)$ worst-case linear time selection algorithms. Then, the overall complexity of Algorithm 12 is equal to $O(C.N \times (|R||F||P|)) + O(C.N \times S^A.N) + O(n)$. Then, the worst case complexity of Algorithm 12 is $O(|R||F||P|) + O(n)$.

2.6 Comparative study

2.6.1 Computing of similarity measure

In this section, we compare some proposals in respect to similarity measure computing strategy. The comparison, which is summarized in Table 2.9, consider the following comparison criteria: the definition of the semantic distance set V; the computing of degree of match $\delta(\cdot, \cdot)$; the modelling technique of the matching problem; the level of precision; and the level of complexity.

Table 2.9 compares our efficient and accurate algorithms to the following ones: Doshi et *al* [24], Paollucci et *al* [64], Bellur and Kulkarni [7], Chakhar [15] and Chakhar et *al* [16].

Table 2.9: Comparison of similarity measure computing algorithms

Approach	Semantic distance	Degree of match (c_1, c_2)	Modelling technique	Precision	Complexity
[24]	v_1=`Exact` v_2=`Plug-in` v_3=`Subsumption` v_4=`Container` v_5=`Part-of` v_6=`Disjoint`	`Exact`: $c_1 \equiv c_2$ `Plug-in`: $c_1 \sqsubset c_2$ `Subsumption`: $c_1 \sqsupset c_2$ `Container`: $c_1 \succeq c_2$ `Part-of`: $c_1 \preceq {}^1 c_2$ `Disjoint`: $c_1 \; disj^2 \; c_2$	Unspecified by the authors	High	Moderate
[64]	v_1=`Exact` v_2=`Plugin` v_3=`Subsume` v_4=`Fail`	`Exact`: $c_1 \sqsubseteq_1 c_2$ `Plugin`: $c_1 \sqsubset c_2$ `Subsume`: $c_1 \sqsupset c_2$ `Fail`: $c_1 \; disj \; c_2$	Greedy Algorithm	Low	High
[7]	v_1=`Exact` v_2=`Plugin` v_3=`Subsume` v_4=`Fail`	`Exact`: $c_1 \equiv c_2$ `Plugin`: $c_1 \sqsubset c_2$ `Subsume`: $c_1 \sqsupset c_2$ `Fail`: $c_1 \; disj \; c_2$	Bipartite Graph	High	Moderate
Efficient Algorithm	v_1=`Exact` v_2=`Plugin` v_3=`Subsume` v_4=`Fail`	`Exact`: $c_1 \equiv c_2$ `Plugin`: $c_1 \sqsubset_1 c_2$ `Subsume`: $c_1 \sqsupset_1 c_2$ `Fail`: $c_1 \; disj \; c_2$	Bipartite Graph	Moderate	Low
Accurate Algorithm	v_1=`Exact` v_2=`Plugin` v_3=`Subsume` v_4=`Extended-Plugin` v_5=`Extended-Subsume` v_6=`Fail`	`Exact`: $c_1 \equiv c_2$ `Plugin`: $c_1 \sqsubset_1 c_2$ `Subsume`: $c_1 \sqsupset_1 c_2$ `Extended-Plugin`: $c_1 \sqsubset c_2$ `Extended-Subsume`: $c_1 \sqsupset c_2$ `Fail`: $c_1 \; disj \; c_2$	Bipartite Graph	High	Moderate

2.6.2 Matching algorithms

Table 2.10 compares our matching algorithms to several other matching algorithms in respect to several criteria including the matching type, matching attributes, customization support and complexity.

The first and traditional matchmaking frameworks, such as Jini [4], Konark [50] and Salutation [62], are based on strict syntactic matching. Such syntactic matching approaches only performs service discovery and service matching based on particular interface or keyword queries from the user, which generally leads to low recall and low precision of the retrieved services [54].

Table 2.10: Comparison of matching algorithms

Matchmaker	Matching type	Matched attributes	Customization support	Speed	Performance
Jini [4]	Syntactic	Capability attributes	No	Slow	Low
Konark [50]	Syntactic	Capability attributes	No	Slow	Low
Salutation [62]	Logic-based	Capability attributes	No		
MatchMaker [77]	Syntactic	Capability attributes	No		
RACER [51]	Syntactic	Capability attributes	No		
PSMF [24]	Logic-based	Capability attributes	Yes	Fast	Moderate
SPARQLent [72]	Logic-based	Capability attributes	No	Moderate	Moderate
iSeM-logic-based [43]	Logic-based	Capability attributes	No	Fast	Low
QoSeBoker[16]	Logic-based	Capability/QoS/ Property attributes	Yes	Moderate	Moderate
Algorithm 7	Logic-based	Capability/Property attributes	No	Fast	High
Algorithm 8	Logic-Based	Capability/Property attributes	Yes	Fast	High
Algorithm 9	Logic-based	Capability/Property attributes	Yes	Fast	High

Some advanced techniques and algorithms (e.g., genetic algorithmic as in [53], utility function as in [82][86]) have been used to overcome the problem of syntactic matching.

In order to overcome the limitation of strict syntactic matching, many authors propose to include the concept of semantics as in [7][9][29][34][42][51][64][73][77]. The use of ontology eliminates the limitations caused by syntactic difference between terms since matching is now possible on the basis of concepts of Ontologies used to describe input and output terms [8].

2.7 Conclusion

In this chapter we first first introduced two algorithms the improve the computing of the similarity proposed in [7]. The first algorithm affects the precision of [7]'s algorithm but improves considerably its complexity while the second algorithm enriches the semantic distance values used in [7]'s, ameliorate considerably the precision of [7]'s algorithm.

In the second part of this chapter we detailed thee functional conjunctive and attribute-based matching algorithms that support three levels of customization: (i) the trivial matching algorithm supports no customization (ii) partially parameterized matching algorithm allows the user to specify the set of attributes to be used in the matching, and (iii) fully parameterized matching algorithm allows the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed. These algorithms generalize and improve the proposals of [7][15][16][24]. The chapter also discusses the extension of the proposed matching algorithms to other types of matching and compare them some existing ones.

The proposed matching algorithms permits to solve the first and third shortcomings of semantic matchmaking frameworks. For the second shortcoming, the QoS-based semantic matching algorithms proposed in [16] can be used with very minor modifications. The next chapter addresses the problem of Web services ranking.

Chapter 3

Web Services Ranking

This chapter first presents a technique for computing Web services scores. Then, it details three approaches for ranking Web services. The first approach relies on the scores only. The second approach defines and uses a series of rules to rank Web services. It permits to solve the ties problem encountered by the first approach. The third approach is based on the use of a tree data structure. It permits to solve the problem of ties of the first approach. In addition, it is computationally better than the second approach. A series of algorithms are proposed for the different ranking approaches. This chapter also studies the algorithmic complexity of these algorithms and compares them to some existing ones.

3.1 Introduction

The number of Web services that satisfy the user request may be very high. In practice, however, it is more appropriate to provide the user with a manageable set of 'best' Web services from which s/he can select one Web service to deploy. A possible solution is to use some appropriate techniques to rank the Web services and then provide the first ranked one(s) to the user.

In this chapter, we present three approaches to rank Web services: score-based, rule-based and tree-based. The score-based approach relies on the use of the scores of Web services, which are computed based on the input data. Two different versions are proposed here. They differ only on the way the similarity degrees are computed.

The rule-based ranking approach uses a series of rules to rank the Web services. The four designed ranking rules can be applied successively to avoid the problem of ties encountered by the first approach. We note that a first version of the rule-based ranking approach is available in [30].

The basic idea of the tree-based approach is to use jointly the two different types of score computing. This approach is operationalized using a tree data structure, which permits to solve the problem of ties of the first approach. This approach is also computationally better than the second approach.

A series of algorithms are proposed for the different ranking approaches. The proposed ranking algorithms permit to solve the fourth shortcoming of the semantic matchmaking frameworks (see Section 1.7).

The chapter is organized as follows. Section 3.2 presents Web services scoring technique. Sections 3.3, 3.4 and 3.5 detail the score-based, the rule-based and tree-based ranking approaches, respectively. Section 3.6 compares the proposed ranking algorithms to some existing ones. Section 3.7 concludes the chapter.

3.2 Scoring Web services

3.2.1 Score definition

In this section, we propose a technique to compte the scores of the Web services based on the input data. First, we need to assign a numerical weight to every similarity degree as indicated in Table 3.1. In this report, we assume that the weights are computed as follows:

$$w_1 \geq 0, \quad (3.1)$$
$$w_i = (w_{i-1} \cdot N) + 1, \quad i = 2, \cdots, N; \quad (3.2)$$

where N is the number of attributes. This way of weights computation ensures that a single higher similarity degree will be greater than a set of N similarity degrees of lower weights taken together. Indeed, the weights values verify the following condition:

$$w_i > w_j \cdot N, \quad \forall i > j. \quad (3.3)$$

Table 3.1: Weights of similarity degrees

Similarity degree	Weight
Fail	w_1
Part-of	w_2
Container	w_3
Subsumption	w_4
Plug-in	w_5
Exact	w_6

Then, the initial score of an advertised Web service S^A is computed as followers:

$$\rho(S^A) = \sum_{i=1}^{i=N} w_i. \quad (3.4)$$

Example 3.1 Assume that the matching algorithm has identified the Web services shown in Table 3.2. Assume also that the Ontology inference system supports all the semantic relationships given in Table 3.1. Finally assume that $w_1 = 0$. The application of Equation (3.4) leads to the following scores: $\rho(S_1) = 147$, $\rho(S_2) = 174$,

$\rho(S_3) = 120$, $\rho(S_4) = 174$ and $\rho(S_5) = 120$. For example, by Equation (3.4) we obtain $\rho(S_2) = w_5 + w_6 + w_4$. Knowing that $w_1 = 0$, Equation (3.2) leads to $w_4 = 13$, $w_5 = 40$ and $w_6 = 121$. Finally we obtain: $\rho(S_2) = 13 + 40 + 121 = 174.\diamond$

Table 3.2: Web services identified by the matching algorithm

Web Service	**Input**	**Output**	**Service category**
S_1	Exact	Subsume	Subsume
S_2	plug-in	Exact	Subsume
S_3	Plug-in	Plug-in	Plug-in
S_4	Plug-in	Subsume	Exact
S_5	Subsume	Exact	Subsume

The scores as computed by Equation (3.4) are not in the range 0-1. Hence, we need to normalize these scores. The following equation can be used for this purpose:

$$\rho'(S^A) = \frac{\rho(S^A) - \min_K \rho(S^K)}{\max_K \rho(S^K) - \min_K \rho(S^K)}. \tag{3.5}$$

This technique assigns to each advertised Web service S^A the percentage of the extent of the similarity degrees scale, i.e., $\max_K \rho(S^K) - \min_K \rho(S^K)$. We note that other normalization techniques may also apply. The advantage of this technique is its ability to ensure that the scores cover all the range [0,1]. In other words, the lowest score will be equal to 0 and the highest score will be equal to 1.

Example 3.2 The normalized scores of the Web services shown in Table 3.2 are as follows: $\rho'(S_1) = 0.5$, $\rho'(S_2) = 1$, $\rho'(S_3) = 0$, $\rho'(S_4) = 1$ and $\rho'(S_5) = 0.5$. For example, $\rho'(S_2)$ is computed as follows:

$$\begin{aligned}
\rho'(S_2) &= \frac{\rho(S_2) - \min\{\rho(S_1), \rho(S_2,)\rho(S_3)\}}{\max\{\rho(S_1), \rho(S_2,)\rho(S_3)\} - \min\{\rho(S_1), \rho(S_2,)\rho(S_3)\}} \\
&= \frac{174 - \min\{147, 174, 120\}}{\max\{147, 174, 120\} - \min\{147, 174, 120\}} \\
&= 1.\diamond
\end{aligned}$$

3.2.2 Score computing algorithms

The computing of the normalized scores is operationalized by Algorithm 13. This algorithm takes as a input a list `mServices` of Web services each is described by a set of N similarity degrees where N is the number of attributes. The data structure `mServices` used as input assumed to defined as:

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \cdots, \mu(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes and $\mu(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j. At the output, Algorithm 13 provides as an updated version of `mServices` by adding to it the normalized scores of the Web services. The new version of `mServices` is defined as follows:

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \cdots, \mu(S_i^A.A_N, S^R.A_N), \rho'(S_i^A)),$$

where $\rho'(S_i^A)$ is the normalized score of Web service S_i^A computed based on Equation (3.5).

Algorithm 13: ComputeNormalizedScores

```
   Input  : mServices,// List of Web services.
            N,// Number of attributes.
   Output: mServices,// List of Web services with normalized scores.
1  r ⟵ length(mServices);
2  t ⟵ 1;
3  while (t ≤ r) do
4  |  row ⟵ the tth row in mServices;
5  |  s ⟵ ComputeInitialScore(row, N, w);
6  |_ mServices[t, N + 2] ⟵ s;
7  a ⟵ mServices[1, N + 2];
8  b ⟵ mServices[1, N + 2];
9  t ⟵ 1;
10 while (t < r) do
11 |  if (a > mServices[t + 1, N + 2])) then
12 |  |_ a ⟵ mServices[t + 1, N + 2];
13 |  if (b < mServices[t + 1, N + 2])) then
14 |_ |_ b ⟵ mServices[t + 1, N + 2];
15 t ⟵ 1;
16 while (t ≤ r) do
17 |  ns ⟵ (mServices[t, N + 2] − a)/(b − a);
18 |_ mServices[t, N + 2] ⟵ ns;
19 return mServices;
```

The function `ComputeInitialScore` in Algorithm 13 line is given in Algorithm 14. This function permits to compute the initial scores of Web services using Equation (3.4). Algorithm 14 takes a list `simDegrees` of similarity degrees for a given Web service and computes the initial score of this Web service based on Equation (3.4). The list `simDegrees` is assumed to be defined as follows:

$$(S_i^A, \mu(S_i^A.A_1, S^R.A_1), \cdots, \mu(S_i^A.A_N, S^R.A_N)),$$

where S_i^A, S^R, N and $\mu(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ as defined previously. It is easy to see that the list `simDegrees` is a row from the data structure `mServices` introduced earlier.

The function `ComputeWeight` used in Algorithm 14 permits to compute the weights based on Equations (3.1) and (3.2). This function is given in Algorithm 15.

Algorithm 14: ComputeInitialScore

```
   Input  : simDegrees,// List of similarity degrees.
            N,// Number of attributes.
   Output: Number// Score.
1  s ⟵ 0;
2  t ⟵ 1;
3  while (t ≤ N) do
4      sd ⟵ simDegrees[t + 1];
5      switch sd do
6          case 'Exact'
7              w ← ComputeWeight(1, N);
8          case Plug-in
9              w ← ComputeWeight(2, N);
10         case Subsumption
11             w ← ComputeWeight(3, N);
12         case Container
13             w ← ComputeWeight(4, N);
14         case Part-of
15             w ← ComputeWeight(5, N);
16         case Fail
17             w ← ComputeWeight(6, N);
18         s ⟵ s + w;
19 return s;
```

Algorithm 15: ComputeWeight

```
   Input  : i, // Order of similarity degree.
            N, // Number of attributes.
   Output: Number// weight.
1  w_c ⟵ 1 ;
2  if (i = 1) then
3      return w_c;
4  w ⟵ 0 ;
5  j ⟵ 1 ;
6  while (j < i) do
7      w ⟵ (w_c × N) + 1;
8      w_c ⟵ w;
9      j ⟵ j + 1;
10 return w;
```

3.2.3 Algorithmic complexity

Algorithm 15 runs in $O(N)$ where N is the number of attributes. Then, the complexity of Algorithm 14 is $O(N^2)$. The complexity of the first *while* in Algorithm 13 is $O(rN^2)$ where r is the number of Web services in `mServices`. The complexity of the second and third *while* loops in Algorithm 13 is $O(r)$ each. Hence, the complexity of Algorithm 13 is equal to $O(r(2 + N^2))$.

3.3 Score-based ranking of Web services

In this section, we present an algorithm for ranking Web services by using the scores.

3.3.1 Score-based ranking algorithm

The Web services can be ranked based on their scores computed as detailed previously. This idea is implemented by Algorithm 16. The input of this algorithm are a list `mServices` of matching Web services and the number of attributes N. The function `ComputeNormalizedScores` in Algorithm 16 is given in Algorithm 13. The score-based ranking algorithm uses a *merge sort procedure* (implemented by lines 3-11 in Algorithm 16) to rank the Web services based on their normalized scores.

Algorithm 16: Score-Based Ranking

```
Input  : mServices,// List of matching Web services.
         N,// Number of attributes.
Output: mServices,// Ranked list of ranked Web services.
1  mServices ← ComputeNormalizedScores(mServices,N);
2  r ← length(mServices );
3  while (i ≤ r) do
4      Let row_i be the ith row in mServices;
5      while (j ≤ r) do
6          Let row_j be the jth row in mServices;
7          if (mServices[i, N + 2] > mServices[j, N + 2])) then
8              tmp ⟵ row_j;
9              row_j ⟵ row_i;
10             row_i ⟵ tmp;
11             update mServices;
12 return mServices;
```

Two versions can be distinguished for the definition of the list `mServices`, along with the way the similarity degrees are computed. The first version of `mServices` is as follows:

$$(S_i^A, \mu_{\max}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\max}(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes and $\mu_{\max}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j. In this case, the similarity measure is computed by selecting the edge with the **maximum weight** in the matching graph (See Section 2.3).

The second version of `mServices` is as follows:

$$(S_i^A, \mu_{\min}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\min}(S_i^A.A_N, S^R.A_N)),$$

where S_i^A, S^R and N are as defined above and $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j. In this case, the similarity measure is computed by selecting the edge with the **minimum weight** in the matching graph (See Section 2.3).

To obtain the final rank, we need to use Algorithm 13 separately with each of these versions and then combine the obtained rankings. However, a problem of ties may occur since several Web services may have the same scores with both versions. This will deteriorate the precision of the ranking algorithm. Indeed, the implementation and testing of Algorithm 13 with the two versions of `mServices` show that in some cases the version based on maximum weight outperforms the version ranking based on minimum weight and in other cases is the inverse that happen. Table 3.3 shows the precision rate of Algorithm 16 with the two versions of `mServices` and using two different data sets from the OWL-TC.

Table 3.3: Precision of score-based ranking

Data set	**Version of `mServices`**	**Precision rate**
Government-Missile-Funding-Service	Maximum weight	0.677
	Minimum weight	0.819
Shoppingmall-Camera-Price-Service	Maximum weight	0.339
	Minimum weight	0.749

To avoid the problem of ties, we propose in Section 3.4 a more advanced ranking algorithm which combines the scores and some preferences information to design a set of ranking rules permitting to obtain a strict order on the Web services.

3.3.2 Algorithmic complexity

The complexity of function `ComputeNormalizedScores` in Algorithm 16 is $O(r(2+N^2))$ where r is the number of Web services and N is the number of attributes (see Section 3.2.3). The `length` in line 2 is assumed to be a built-in function and its complexity is not considered here. The sentences in lines 3-11 in Algorithm 16 implement a merge sort procedure, which at best has a time complexity of $O(r \log r)$ and in worst case, it makes $O(r^2)$. Hence, the overall complexity of Algorithm 16 in best case is $O(r(2 + N^2)) + O(r \log r)$ and in worst case is $O(r(2 + N^2)) + O(r^2)$.

3.4 Rule-based ranking of Web services

The score-based ranking algorithm presented earlier may lead to the problem of ties. To avoid this problem, we propose in this section a more advanced ranking algorithm that combines the scores and some preferences information to obtain a strict order on the Web services and hence avoids the ties problem.

3.4.1 Ranking rules

We design in section a series of rules that can be used sequentiality to rank the Web services. The first rule is based on the scores of Web services while the other rules use some user preference information extracted from the Criteria Table in order to define a strict order on the Web services and hence avoids the problem of ties. Indeed, the

Criteria Table contains three types of preference information: (i) the attribute that should be used for matching, (ii) the minimal similarity degree for each attribute, and (iii) the order of attributes.

A first ranking rule is defined as follows:

RR1:

`if` $\rho'(S_i^A) > \rho'(S_j^A)$ `then` $S_i^A \succ S_j^A$

where S_i^A and S_j^A are two advertised Web service and $a \succ b$ means that a is better ranked that b.

Example 3.3 The normalized scores obtained in Example 3.2 are as follows: $\rho'(S_1) = 0.5$, $\rho'(S_2) = 1$, $\rho'(S_3) = 0$, $\rho'(S_4) = 1$ and $\rho'(S_5) = 0.5$. The application of the first ranking rule RR1 leads to the following rank: $S_2 = S_4 \succ S_1 = S_5 \succ S_3$.$\diamond$

As shown in the this example, the first version of the ranking rule may face the problem of ties when two different services have exactly the same score. A second version of the ranking rule can be defined by using the difference between the desired similarities specified in the Criteria Table and the actual similarity degrees of Web services. Let s be a similarity degree and denote by $Ord(s)$ the ordinal rank of s in respect to the other predefined similarity degrees. The definition of $Ord(\cdot)$ function for all similarity degrees are given in Table 3.4. Let S^A be an advertised service and let $(s_1, \cdots, s_k, \cdots, s_N)$ be its similarity degrees for attributes $A_1, \cdots, A_k, \cdots, A_N$; where N is the number of attributes in the Criteria Table. Then, we define the function $Diff(\cdot, \cdot)$ as follows:

$$Diff(S^A, C) = \sum_{k=1}^{k=N} \{Ord(s_k) - Ord(C.M_k)\} \quad (3.6)$$

Table 3.4: Ordinal rank of similarity degrees

Similarity degree s	$Ord(s)$
Fail	1
Part-of	2
Container	3
Subsumption	4
Plug-in	5
Exact	6

A second version of ranking rule can than be defined textually as follows: use the score-based ranking rule and if there is a tie, select the service with the largest difference between the desired similarities specified in the Criteria Table and the actual similarity degrees of the Web services. Formally,

RR2:

`if` $\rho'(S_i^A) > \rho'(S_j^A)$ `then` $S_i^A \succ S_j^A$

`else if` $\rho'(S_i^A) = \rho'(S_j^A)$ `then`

`if` $Diff(S_i^A, C) > Diff(S_j^A, C)$ `then` $S_i^A \succ S_j^A$

Example 3.4 The ranking obtained in Example (3.5) contains two ties. Hence, we need to apply the second ranking rule. Assume that the similarity degrees in respect to Input, Output and Service category attributes in the Criteria Table are equal to Subsumes for all of the three attributes. The application of Equation (3.6) leads to $Diff(S_2, C) = Diff(S_4, C) = 3$. Similarly, we obtain $Diff(S_1, C) = Diff(S_5, C) = 2$. Accordingly, in this particular example, the second ranking rule RR2 cannot differentiate between Web services S_2 and S_4 nor between Web services S_1 and S_5.$\diamond$

A third possible solution consists to use the order of attributes in the Criteria Table. A new version of the ranking rule can than be defined textually as follows: use the score-based ranking rule and if there is a tie, apply the order-based rule; if another tie occurs, apply the *lexicographic rule*. Let first define the lexicographic rule. Let S_i^A and S_j^A be two Web services and let $(s_1, \cdots, s_k, \cdots, s_N)$ and $(s'_1, \cdots, s'_k, \cdots, s'_N)$ be their similarity degrees for attributes $A_1, \cdots, A_k, \cdots, A_N$, where N is the number of attributes. Then, the lexicographic rule is defined as follows:

$$Lex_C(S_i^A) > Lex_C(S_j^A) \Leftrightarrow (\exists l)(\forall r < l)(s_r = s'_r) \wedge (s_l > s'_l) \tag{3.7}$$

The third version of the ranking rule can than be formally defined as follows:

RR3:

if $\rho'(S_i^A) > \rho'(S_j^A)$ then $S_i^A \succ S_j^A$
  else if $\rho'(S_i^A) = \rho'(S_j^A)$ then
    if $Diff(S_i^A, C) > Diff(S_j^A, C)$ then $S_i^A \succ S_j^A$
    else if $Diff(S_i^A, C) = Diff(S_j^A, C)$ then
      if $Lex_C(S_i^A) > Lex_C(S_j^A)$ then $S_i^A \succ S_j^A$

Example 3.5 We should now apply the third ranking rule RR3 to differentiate between Web services S_2 and S_4 and Web services S_1 and S_5. Consider first Web services S_2 and S_4. We should compare successively S_2 and S_4 on attributes Input, Output and Service category and stop when one of the Web services ranks the other. In respect to Input, we have $Lex_C(S_2) = Lex_C(S_4)$ since both Web services have a similarity degree (which is equal to Plug-in). In respect to Output, we have $Lex_C(S_2)$=Exact and $Lex_C(S_4)$ =Subsume which means $S_2 \succ S_4$. The same process leads to $S_1 \succ S_5$ on the first attribute. Consequently, the final ranking is as follows: $S_2 \succ S_4 \succ S_1 \succ S_5 \succ S_3$.$\diamond$

In the previous example, all the ties problems have been resolved. In practice, however, the problem of ties may occur even with third version of the ranking rule. This holds when two or more Web services have the same similarity degrees on all the attributes. To solve this problem, we propose a fourth version of the ranking rule which is stated textually as follows: use the score-based ranking rule and if there is a tie, apply the order-based rule; if another tie occurs, apply the lexicographic rule; and if a further tie occurs, select the first service. Formally,

RR4:

```
if ρ'(S_i^A) > ρ'(S_j^A) then S_i^A ≻ S_j^A
   else if ρ'(S_i^A) = ρ'(S_j^A) then
           if Diff(S_i^A, C) > Diff(S_j^A, C) then S_i^A ≻ S_j^A
           else if Diff(S_i^A, C) = Diff(S_j^A, C) then
                   if Lex_C(S_i^A) > Lex_C(S_j^A) then S_i^A ≻ S_j^A
                   else if Lex_C(S_j^A) = Lex_C(S_i^A) then
                        else Select the first service
```

In the second and third rules, we assumed that the Criteria Table is used, which means that the Web services in input have been identified using the fully parameterized matching algorithm given in Section 2.4.3. However, these rules can also be applied when the Web services in the input have been identified with the partially parameterized matching algorithm (see Section 2.4.2) or even the trivial matching algorithm (see Section 2.4.1). In the first case, we need simply to transform the Criteria List into a Criteria Table by adding a new column and by setting the minimal similarly degrees to Fail for all the attributes. In the second case, we need simply to define a Criteria Table containing all attributes and by setting the minimal similarly degrees to Fail for all of these attributes.

3.4.2 Rule-based ranking algorithm

The rule-based ranking approach is operationalized through Algorithm 17. This algorithm uses the idea of comparison-based sorting to rank the Web services. Function `RR4` in Algorithm 17 corresponds to the fourth ranking rule introduced earlier. The function `ComputeNormalizedScores` in Algorithm 17 is given in Algorithm 13. It permits to compute the normalized scores, as explained previously.

The main input for this algorithm is list `mServices` which, as previously, can take one of the following forms:

$$(S_i^A, \mu_{\max}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\max}(S_i^A.A_N, S^R.A_N)),$$

or

$$(S_i^A, \mu_{\min}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\min}(S_i^A.A_N, S^R.A_N)),$$

where S_i^A, S^R, N, $\mu_{\max}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ and N $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ are as defined in Section 3.3.1

The output of Algorithm 17 is an ordered list `mServices` of matching Web services.

The rule-based ranking algorithm uses a *merge sort procedure* (implemented by lines 3-11 in Algorithm 17) to rank the Web services based rule **RR4**. It is easy to see that the score-based and rule-based ranking algorithms differ only at the level of the IF-THEN test in line 7. In the score-based ranking algorithm, this test relies on the normalized scores while with the rule-based ranking algorithm the test is based on the ranking rule **RR4**. It is also easy to see that the use of ranking rule **RR1** in Algorithm 17 will lead exactly to the same result as with the score-based algorithm.

Algorithm 17: Rule-Based Ranking

```
Input  : mServices,// List of matching Web services.
         C,// Criteria Table.
Output: mServices,// Ranked list of ranked Web services.
1  mServices ← ComputeNormalizedScores(mServices,C.N);
2  r ⟵ length(mServices);
3  while (i ≤ r) do
4      while (j ≤ r) do
5          row_i ⟵ row i in mServices ;
6          row_j ⟵ row j in mServices ;
7          if (RR4(row_i[1], row_j[1], C)) then
8              tmp ⟵ row_j;
9              row_j ⟵ row_i;
10             row_i ⟵ tmp;
11             update mServices;
12 return mServices;
```

Algorithm 18: RR4

```
Input  : S_i^A, // Web service.
         S_j^A, // Web service.
         C,// Criteria Table.
Output: Boolean//true: S_i^A rank better than S_j^A; false: otherwise.
1  if (ρ'(S_i^A) > ρ'(S_j^A)) then
2      return success;
3      else if (ρ'(S_i^A) = ρ'(S_j^A)) then
4          if (Diff(S_i^A, C) > Diff(S_j^A, C)) then
5              return success;
6              else if (Lex_C(S_i^A) > Lex_C(S_j^A)) then
7                  return success;
8  return false;
```

3.4.3 Algorithmic complexity

The complexity of function `ComputeNormalizedScores` in Algorithm 17 is $O(r(2+N^2))$ where r is the number of Web services and N is the number of attributes (see Section 3.2.3). The `length` in line 2 is assumed to be a built-in function and its complexity is not considered here. The complexity of function `RR4` in Algorithm 17 is equal to $O(2N)$ at worst. The sentences in lines 3-11 in Algorithm 16 implement a merge sort procedure, which at best has a time complexity of $O(r \log r)$ and in worst case, it makes $O(r^2)$. Hence, the overall complexity of Algorithm 17 in best case is $O(r(2+N^2))+O(2Nr \log r)$ and in worst case is $O(r(2 + N^2)) + O(2Nr^2)$.

3.5 Tree-based ranking of Web services

In this section, we propose a new algorithm for Web services ranking using a tree data structure. First, we present the basic idea of the tree-based ranking algorithm

in Section 3.5.1. Then, we propose the tree construction algorithm in Section 3.5.2. Then, we detail in Section 3.5.3 the tree-based ranking algorithm. Finally, we study the complexity of proposed ranking algorithm in Section. 3.5.4.

3.5.1 Principle

The basic idea of the tree-based ranking algorithm is to use jointly the trivial ranking algorithms based on $\max(w_i)$ and $\min(w_i)$ criteria. For computational performances, the proposed algorithm will rely on a four-level tree T data structure. The construction of the tree T where the root contains the initial list of Web and the leafs are individual Web services. The final ranking is obtained by identification the leafs from the left to the right. Figure 3.1 provides an example of tree constructed based on this idea.

Figure 3.1: Tree structure

In what follows, we assume that the initial list of Web services `mServices` is defined as follows:

$$(S_i^A, \mu_{\max}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\max}(S_i^A.A_N, S^R.A_N), \mu_{\min}(S_i^A.A_1, S^R.A_1), \cdots, \mu_{\min}(S_i^A.A_N, S^R.A_N)),$$

where: S_i^A is an advertised Web service, S^R is the requested Web service, N the total number of attributes and

- $\mu_{\max}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **maximum weight** in the matching graph (See Section 2.3); and
- $\mu_{\min}(S_i^A.A_j, S^R.A_j)$ $(j = 1, \cdots, N)$ is the similarity measure between the requested Web service and the advertised Web service on the jth attribute A_j computed by selecting the edge with the **minimum weight** in the matching graph (See Section 2.3).

The tree-based ranking algorithm given later in Section 3.5.3 is composed of two main phases. The objective of the first phase is to construct a tree T as described earlier. The objective of the second phase is to identify the final ranking. The first phase is detailed in the next section.

3.5.2 Tree construction

The construction of the tree requires the use of some appropriate functions to split the nodes of each level. Before detailing the tree construction algorithm, we introduce the nodes splitting functions.

3.5.2.1 Node splitting functions

The construction of the tree T requires the use of some functions for splitting the nodes. The first node splitting function is formalized in Algorithm 19. This function receives in entry one node with a list of Web services and generates a ranked list of nodes each with one or more Web services. The function `SortScoresMax` and `SortScoresMin` in Algorithm 19 permit to sort Web services based either on the maximum edge value or the minimum edge value, respectively. Function `Split` permits to split the Web services in L_0 into a set of sublists, each with a subset of services having the same score. The instructions in lines 8-11 in Algorithm 19 permits to create a node for each sublist in `Sublists`. The algorithm outputs a list T of nodes ordered according to the scores of Web services in each node from the left to the right.

Algorithm 19: Node splitting

```
   Input  : Node,// A node.
            NodeSplittingType,// Node splitting type.
   Output: L,// List of ranked nodes.
1  L ← ∅;
2  L0 ← Node.mServices;
3  if (NodeSplittingType = 'Max') then
4      L0 ← SortScoresMax(L0);
5  else
6      L0 ← SortScoresMin(L0);
7  SubLists ← Split(L0);
8  for (for each elem in SubLists) do
9      create node n;
10     n.add(elem);
11     L.add(n);
12 return L;
```

The first node splitting function is formalized in Algorithm 20. This functions permits to split randomly a node into a set of nodes, each with one Web service.

Algorithm 20: Random Node Splitting

```
   Input  : Node,// A node.
   Output: L,// List of ranked nodes.
1  L ← ∅;
2  L0 ← Node.mServices;
3  for each elem in L0 do
4      create node n ;
5      n.add(elem);
6      L.add(n);
7  return L;
```

3.5.2.2 Tree construction algorithm

The construction process starts by initializing the root nodes with the initial list of Web services. Then, the root node is split into a set of nodes, each contains a set of Web services having the same value. The nodes that contains more then one Web services

are split as previously. The process is repeated until all the leaf nodes contain only one Web service.

The tree construction process contains four steps:

- ***Step 0***. *Construction of Level 0*: In this initialization step, we create a root node r containing the list of Web services to rank. At this level, the tree T contains only the root node.
  Step 1. *Construction of Level 1*: Here, we split the root node into a set of nodes by using the node splitting in Algorithm 19 with the desired sorting type (i.e. based either on the maximum edge value or the minimum edge value). At the end of this step, a new level is added to the tree T. The nodes of this level are ordered from left to right. Each node of this level will contain one or several Web services. The Web services of a given node will have the same score.

- ***Step 2***. *Construction of Level 2*: Here, we split the nodes of the previous level that have more than one Web services using by using the node splitting in Algorithm 19 with a different sorting type than the previous step. At the end of this step, a new level is added to the tree T. The nodes of this level are ordered from left to right. Each node of this level will contain one or several Web services. The Web services of a given node will have the same score.

- ***Step 3***. *Construction of Level 3*: Here, we apply the random splitting function given in Algorithm 20 to randomly split the nodes of the previous level that has more one Web service. At the end of this step, a new level is added to the tree T. All the nodes of this level contains only one Web service.

Depending on the order the node slipping types are used, we may distinguish two version for the tree construction. The first version, which is shown in Algorithm 21, uses the node splitting based on the Max to generate the nodes of the second level and then the node splitting based on the Min to generate the nodes of the third level. The second version, shown in Algorithm 22, uses an opposite order.

Algorithm 21: Tree Construction—Version Max-Min

```
   Input  : L,// Initial list of Web services.
   Output: T,// Tree.
1  create node Node;
2  Node.add(L);
3  add Node as root of the tree T;
4  FirstLevelNodes ← NodeSplitting(Node,'Max');
5  for each node n1 in FirstLevelNodes do
6      add n1 as a child of the root;
7      if n1 is not a leaf then
8          SecondLevelNodes ← NodeSplitting(Node,'Min');
9          for each node n2 in SecondLevelNodes do
10             add n2 as a child of n1;
11             if n2 is not a leaf then
12                 ThirdLevelNodes ← RandomNodeSplitting(n2);
13                 for each node n3 in ThirdLevelNodes do
14                     add n3 as a child of n2;
15 return T;
```

Algorithm 22: Tree Construction—Version Min-Max

```
   Input  : L,// Initial list of Web services.
   Output: T,// Tree.
1  create node Node;
2  Node.add(L);
3  add Node as root of the tree T;
4  FirstLevelNodes ← NodeSplitting(Node,'Min');
5  for each node n1 in FirstLevelNodes do
6      add n1 as a child of the root;
7      if n1 is not a leaf then
8          SecondLevelNodes ← NodeSplitting(Node,'Max');
9          for each node n2 in SecondLevelNodes do
10             add n2 as a child of n1;
11             if n2 is not a leaf then
12                 ThirdLevelNodes ← RandomNodeSplitting(n2);
13                 for each node n3 in ThirdLevelNodes do
14                     add n3 as a child of n2;
15 return T;
```

The construction process is illustrated graphically in Figure 3.2. For simplicity, the initial Web services list (in the root node) contains 8 Web services each with two scores.

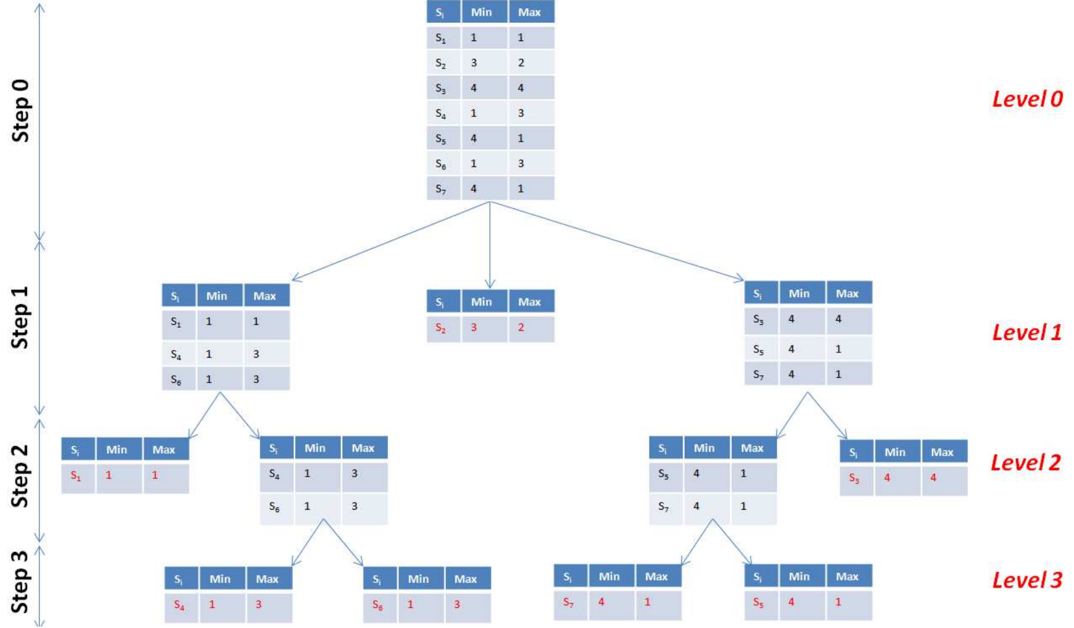


Figure 3.2: Illustration of tree construction process

3.5.3 Tree-based ranking algorithm

We propose here a new algorithm for implementing the solution proposed in Section 3.5.1. The idea of the algorithm is to construct first a tree T using one of the algorithms discussed in the previous section and then scanning through this tree in order identify the final ranking. The identification of the best and final ranking needs to apply a *tree traversal* (also known as *tree search*) on the tree T. The tree traversal refers to the process of visiting each node in a tree data structure, exactly once, in a systematic way. Different types of traversals are possible: pre-order, in-order and post-order. They differ by the order in which the nodes are visited. In this report, we will use the pre-order type.

The idea discussed in the previous paragraph is implemented by Algorithm 23. The main input for this algorithm is the initial list `mServices` of Web services. This list is assumed to have the same structure as indicated in Section 3.5.1. The output of Algorithm 23 is a list `FinalRanking` of ranked Web services.

The functions `ComputeNormalizedScoresMax` and `ComputeNormalizedScoresMin` are not given in this report. They are similar to function `ComputeNormalizedScores` introduced previously in Algorithm 17. The scores in function `ComputeNormalizedScoresMax` are computed by selecting the edge with the **maximum weight** in the matching graph while the scores in function `ComputeNormalizedScoresMin` are computed by selecting the edge with the **minimum weight** in the matching graph (See Section 2.3).

Algorithm 23 can be organized into two phases. The first phase concerns the construction of the tree T. This phase is implemented by the instructions in lines

1-7. According to the type of nodes splitting order (Max-Min or Min-Max), Algorithm 23 uses either function `ConstructTreeMaxMin` (for Max-Min order) or function `ConstructTreeMinMax` (for Min-Max order) to construe the tree. These functions are implemented by Algorithms 21 and 22, respectively.

Algorithm 23: Tree-Based Ranking

```
   Input  : mServices,// List of matching Web services.
            N,// Number of attributes.
            SplittingOrder, // Nodes splitting order.
   Output: FinalRanking,// Ranked list of Web services.
 1 T ← ∅;
 2 mServicesComputeNormalizedScoresMax(mServices,N);
 3 mServicesComputeNormalizedScoresMin(mServices,N);
 4 if (SplittingOrder = 'MaxMin') then
 5 |  T ← ConstructTreeMaxMin(mServices );
 6 else
 7 |  T ← ConstructTreeMinMax(mServices )
 8 FinalRanking ← TreeTraversal(T);
 9 return FinalRanking;
```

The second phase of Algorithm 23 concerns the identification of the best and final ranking by applying a pre-order tree traversal on the tree T. The pre-order tree traversal contains three main steps:

1. examine the root element (or current element);
2. traverse the left subtree by recursively calling the pre-order function;
3. traverse the right subtree by recursively calling the pre-order function.

The pre-order tree traversal is implemented by Algorithm 24. This algorithm takes in input a tree T and generated the final ranking list L. Algorithm 24 scans the tree T and picks out all leaf nodes of T ordered from the left to the right.

Algorithm 24: Tree Traversal

```
   Input  : T,// Tree.
   Output: L,//Final ranking.
 1 L ← ∅;
 2 CurrNode ← T.R;
 3 Traverse (CurrNode,L)is
 4 |  if CurrNode contains a single Web service then
 5 |  |  Let currService be the single Web service in CurrNode;
 6 |  |  L.Append(currService);
 7 |  for each child f of CurrNode do
 8 |  |  Traverse (f,CurrNode);
 9 return L;
```

Figure 3.3 illustrates graphically the pre-order tree traversal. The numbers in this figure indicate the order of nodes examining. The final ranking is given by the leaf nodes in Figure 3.3 ordered from left to right, i.e., 4→5→6→7→8.

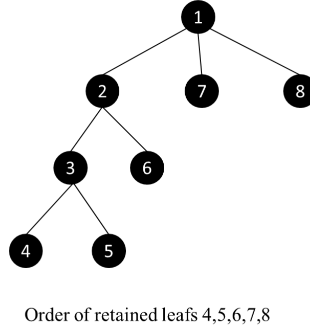


Figure 3.3: Illustration of tree traversal algorithm

3.5.4 Algorithmic complexity

In this section, we design by:

- r: the total number of Web services;
- r_n: the number of Web services in a given node n;
- $k_{n_j}^{l_i}$ the number of services of the node n_j of the level l_i of the tree. ;
- T_n: the total number of nodes in the tree T;
- L_{n_j}: the number of nodes in the jth level of the tree T.

Node splitting algorithms Let focus on Algorithms 19 and 20 of node splitting.

- Algorithm 19: The most expensive operation in Algorithm 19 is the sorting procedure (lines 4 or 6). In this algorithm, we assumed that a merge-sort procedure is in use, which makes the time complexity of Algorithm 19 is approximately $O(r_n \log r_n)$.
- Algorithm 20: The single loop in Algorithm 20 iterates over the Web services in the node received as in input. Consequently, the time complexity of Algorithm 20 is $O(r_n)$.

Tree construction algorithms Algorithms 21 and 22 have the same complexity. In order to compute it, we chose to consider each line apart. The algorithm contains three embedded loop and the number of iteration of the these loops can take any value, so considering the algorithms in a holistic way and discussing the complexity of each line was the most simple approach to take.

- Lines 1, 2 and 3 contains constant operations.

- Line 4 corresponds to the function `NodeSplitting`, which is implemented through Algorithm 19. The time complexity of this Algorithm is $O(r_n \log r_n)$. Since at this level r_n is equal to the total number of Web services, the complexity of the instruction in line 4 is equal to $O(r \log r)$.
- The first for Loop that we denote by fl spans form line 5 to 15. Operations within fl are repeated L_{n_1} times. Within fl :
  - Lines 5 to 7 contains constant operations, hence their time complexity is $O(L_{n_1})$.
  - Line 8 corresponds to the function `NodeSplitting`, which is implemented through Algorithm 19. The time complexity of this Algorithm is $O(\sum_{i=1}^{i=L_{n_1}} k_{n_i}^{l_1} \log k_{n_i}^{l_1});$
  - The second embedded loop el_1 that spans from line 9 to 14, is repeated L_{n_2} times and within el_1:
    * Lines 9 to 11 contains constant operations, hence have a time complexity $O(L_{n_2})$.
    * Line 12 corresponds function `RandomSplitting`, which is implemented by Algorithm 20. This function has complexity of $O(\sum_{i=1}^{i=L_{n_2}} k_{n_i}^{l_2});$
    * The last embedded loop el_2 spans from line 13 to 14 and contains constant operations which are repeated L_{n_3} times. Hence, its time complexity is $O(L_{n_3})$

The total time complexity is consequently $O(r \log r + L_{n_1} + \sum_{i=1}^{L_{n_1}} k_{n_i}^{l_1} \log k_{n_i}^{l_1} + L_{n_2} + \sum_{i=1}^{L_{n_2}} k_{n_i}^{l_2} + L_{n_3})$.

Nodes in the tree can be assimilated to a bag of services. At each phase of the tree construction, a traversal of the tree boundaries from left to right as illustrated in Figure 3.4 with a sequential record of services within the nodes, would produce a ranked list of services and exactly r services. This characteristic of the tree guarantees that the number of nodes and the total number of services at each level of the tree would never exceed r, hence we have the following equations:

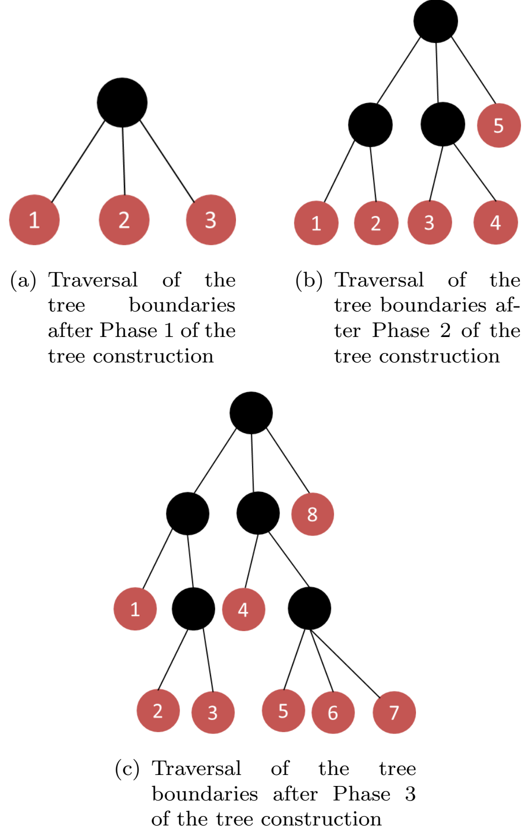
$$L_{n_1} \leq r \tag{3.8}$$

$$\sum_{i=1}^{L_{n_1}} k_{n_i}^{l_1} \leq r \tag{3.9}$$

$$L_{n_2} \leq r \tag{3.10}$$

$$\sum_{i=1}^{L_{n_2}} k_{n_i}^{l_2} \leq r \tag{3.11}$$

$$L_{n_3} \leq r \tag{3.12}$$



(a) Traversal of the tree boundaries after Phase 1 of the tree construction

(b) Traversal of the tree boundaries after Phase 2 of the tree construction

(c) Traversal of the tree boundaries after Phase 3 of the tree construction

Figure 3.4: Traversal of the tree boundaries after each phase of the tree construction

We approximate that $k^{l_1}_{n_1} = k^{l_1}_{n_2} = cdots = k^{l_1}_{n_i} = cdots = k^{l_1}_{n_{L_{n_1}}} = k^{l_1}$. In other words, the nodes of the first level will contain the same number of services. Consequently and according to equation 3.9 we have $k^{l_1}.L_{n_1} \leq r$. Hence, $\sum_{i=1}^{L_{n_1}} k^{l_1}_{n_i} \log k^{l_1}_{n_i} \leq r \log \frac{r}{L_{n_1}}$, consequently we have equation 3.13:

$$\sum_{i=1}^{L_{n_1}} k^{l_1}_{n_i} \log k^{l_1}_{n_i} \leq r \log r \tag{3.13}$$

From equations 3.8, 3.9, 3.10, 3.11, 3.12 and 3.13 we conclude that the time complexity of the tree construction algorithm is $O(r \log r)$.

Tree-based ranking and tree traversal algorithms Let focus on Algorithms 23 and 24 of tree-based ranking and tree traversal, respectively.

- Algorithm 23: Functions `ComputeNormalizedScoresMax` and `ComputeNormalized-ScoresMin` in Algorithm 23 are similar to function `ComputeNormalizedScores`. The latter is implemented by Algorithm 17 with a time complexity of $O(r(2+N^2))$ (see Section 3.2.3). Functions `ConstructTreeMinMax` and `ConstructTreeMinMax` are given in Algorithms 21 and 22, respectively. As shown earlier, these algorithms have a complexity of $O(r \log r)$. Function `TreeTraversal` is implemented by Algorithm 24 which has a complexity of $O(r)$, as discussed in the next item. Finally, Algorithm 23 has a time complexity of $O(r \log r)$.
- Algorithm 24: The tree traversal algorithm has a complexity of $O(T_n)$ where T_n is the total number of nodes in the tree T. Based on Equations (3.8), (3.10) and (3.12), we conclude that $T_n < 3r$. Hence, the time complexity of Algorithm 24 is $O(r)$.

3.6 Comparative study

In this section, we briefly compare the proposed algorithm to some existing ones in respect to several characteristics. The characteristics concerns:

1. Use of external information to the Web service description;
2. Use of the available information in the Web service description;
3. Use of the user preferences;

Table 3.5 summaries the characteristics of the proposed approaches and some existing ones (namely, [6][44][58][58][74][75]).

Table 3.5: Comparison of ranking approaches

Approach	Use of external information	Use of Web service description information	Use of user preferences
Beck and Freitag [6]		×	×
Kokash et al. [44]	×	×	
Manoharan et al. [58]		×	
Segev and Toch [74]	×	×	
Skoutas et al. [75]		×	
Score-based ranking (Algorithm 16)		×	
Rule-based ranking (Algorithm 17)		×	×
Tree-based ranking (Algorithm 23)		×	

3.7 Conclusion

In this chapter, we first presented a technique for computing Web services scores. The prossed technique permits to transform the similarity measures, which are defined on

an ordinal scale, into numerical values. Then, we detailed three Web services ranking approaches: score-based, rule-based and tree-based approaches. We presented a series of algorithms are proposed for these different ranking approaches.

In the next chapter, we present the developed prototype and then evaluate the performances of the proposed matching and ranking algorithms.

Chapter 4

Implementation and Performance Evaluation

We implemented a prototype called PMRF (Parameterized Matching-Ranking Framework). This chapter first presents the architecture of the developed system and discusses some implementation issues. Then, it provides the results of performance evaluation of the PMRF. It also compares PMRF to two exiting frameworks, namely iSeM-logic-based and SPARQLent. The different matching and ranking algorithms have been implemented and evaluated using the OWLS-TC4 datasets. The evaluation has been conducted employing the SME2 (Semantic Matchmaker Evaluation Environment) tool. The results show that the algorithms behave globally well in comparison to iSeM-logic-based and SPARQLent.

4.1 Introduction

In the second and third chapters, we unproduced a series of algorithms for matching and ranking Web services. We implemented a highly configurable framework called PMRF (Parameterized Matching-Ranking Framework) supporting the different proposed algorithms. We compared the performance of these algorithms by testing seven different configurations. We also compared the proposed framework to two well-known matchmakers, namely iSeM-logic-based [43] and SPARQLent [72][73]. The SME2 (Semantic Matchmaker Evaluation Environment) has been used to evaluate the performance of different algorithms. The SME2 is an open source tool for testing different semantic matchmakers in a consistent way.

The rest of the chapter is organized as follows. Section 4.2 presents the architecture of PMRF and some implementation issues. Section 4.3 introduces the performance evaluation framework and metrics. Section 4.4 provides the analysis of performance evaluation. Section 4.5 discusses the effect of the edge criteria order on the tree-based ranking algorithm. Section 4.6 compares the PMRF to other similar frameworks. Section 4.7 concludes the chapter

4.2 System architecture and implementation

In this section, we first present the conceptual (Section 4.2.1) and the functional (Section 4.2.2) architectures of the PMRF. Then, we discuss some implementation issues (Section 4.2.3).

4.2.1 System design and conceptual architecture

Figure 4.1 provides the conceptual architecture of the PMRF. The inputs of the system are: the Criteria Table/List, the published Web services repository, the user request and its corresponding Ontologies. The weights of similarity degrees and order functions are computed by the PMRF. The output of the PMRF is a ranked list of Web services.

The PMRF is composed of two layers. The role of the first layer is to parse the input data and parameters and then transfer it to the second layer, which represents the matching and ranking engine. The Matching Module filters Web service offers that match with the Criteria Table/List. The result is then passed to the Ranking Module. This module produces a ranked list of Web services. The assembler guarantees a coherent interaction between the different modules in the second layer.

The three main components of the second layer of PMRF are:

- **Matching Module**: This component contains the different matching algorithms introduced in Section 2.4:
  - Trivial matching algorithm (Section (Section 2.4.1),
  - Partially parameterized matching algorithm (Section 2.4.2),
  - Fully parameterized matching algorithm(Section 2.4.3).
- **Similarity Computing Module**: This component supports the different similarity measure computing approaches introduced in Section 2.3, and which are:
  - Efficient similarity with MinEdge (Section 2.3.3),
  - Accurate similarity with MinEdg (Section 2.3.4),
  - Accurate similarity with MaxEdge 2.3.4),
  - Accurate similarity with MaxMinEdge 2.3.4).
- **Ranking Module**: This component is the repository of the score computing technique and the different ranking algorithms proposed in the third chapter. It contains the following elements:
  - Score computing technique (Section 3.2),
  - Score-based ranking algorithm (Section 3.3),
  - Rule-based ranking algorithm (Section 3.4),
  - Tree-based ranking algorithm (Section 3.5).

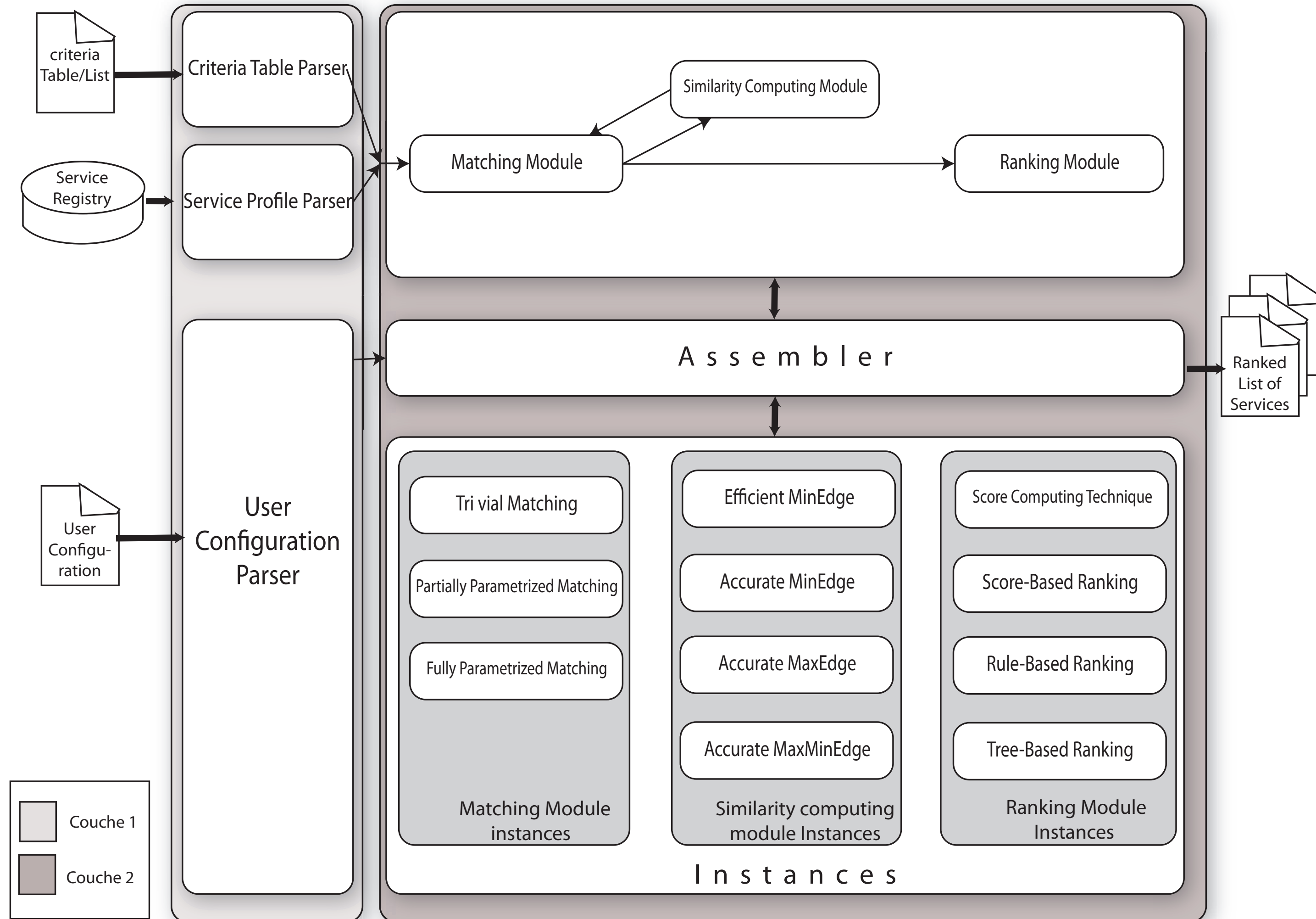


Figure 4.1: Conceptual architecture of the PMRF

4.2.2 Functional architecture

The functional architecture of the PMRF is given in Figure 4.2. It shows graphically the different steps from receiving the user query (query and the different parameters) until the delivery of the final results (ranked list of Web services matching the query) to the user. We can distinguish the following main operations:

- The PMRF receives (1) the user query including the desired Web services and the required parameters;
- The Matching Module scans (2) the Registry in order to identify the Web services matching the user query;
- During the matching process, the Matching Module uses (3) the Similarity Computing Module to calculate the similarity degrees;
- The Matching Module delivers (4) the Web services matching the user query;
- The Ranking Module receives (5) the matching Web services and processes them for ranking;
- During the ranking operation, the Ranking Module uses (6) the Scoring Module to compute the scores of the Web services;
- The Ranking Module delivers (7) a ranked list of Web services;
- The PMRF delivers (8) the ranked list of Web services to the user.

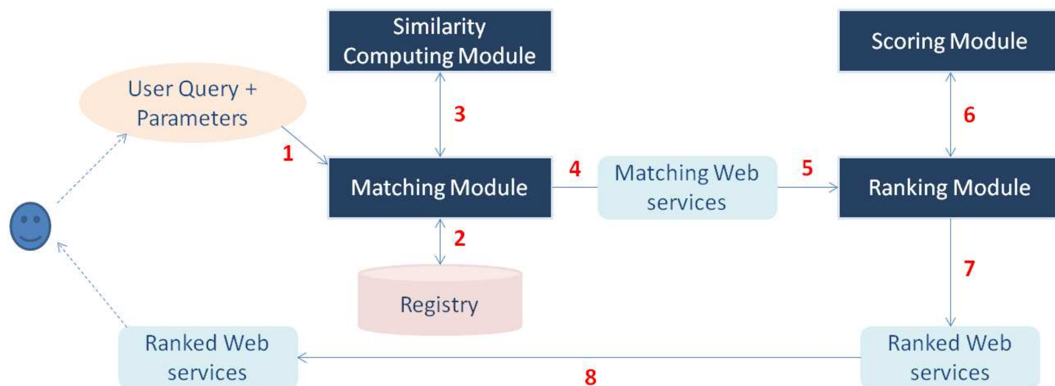


Figure 4.2: Functional architecture of the PMRF

4.2.3 Implementation

To develop the PMRF, we have used the following tools:

- OWLS-API (`http://on.cs.unibas.ch/owls-api/`) to parse the OWLS service descriptions;
- OWL-API (`http://owlapi.sourceforge.net/`) and the Pellet-reasoner (`http://clarkparsia.com/pellet/`) to perform the inference for computing the similarity measure;
- Eclipse IDE (`http://eclipse.org/ide/` as a developing tool.

The inference is one of the main issues encountered during the developing of the PMRF. We perform the following procedure in order to minimize resources consumption, especially memory:

1. A local ontology is created at the start of the matchmaking process. The incremental classifier class, taken from the Pellet reasoner library, is associated to this ontology.
2. The service parser based on the OWLs-API retrieves the URI of the attributes values of each service. The concepts related to these URIs are added incrementally to the local ontology and the classifier is updated accordingly.
3. In order to infer the semantic relations between concepts, the similarity measure module uses the knowledge base constructed by the incremental classifier.

Figure 4.3 an extract from the Matching Class. In this figure, we can see the input and output functions. The latter contains the call for the matching and ranking operations.

4.3 Performance evaluation framework and metrics

In this section, we introduce the performance evaluation framework (Section 4.3.1), specify the test collection used (Section 4.3.2) and the different evaluation metrics (Section 4.3.3).

4.3.1 Evaluation framework

To evaluate the performance of the PMRF, we used the Semantic Matchmaker Evaluation Environment (SME2). The SME2[1] is an open source tool for testing different semantic matchmakers in a consistent way. The SME2 uses OWLS-TC collections to provide the matchmakers with Web service descriptions, and to compare their answers to the relevance sets of the various queries.

[1]See here: `http://projects.semwebcentral.org/projects/sme2/`.

```
27 public class Matchmaker implements IMatchmakerPlugin {
28
29         PelletReasoner reasoner=new PelletReasoner();
30         ServiceTuple query;
31         ArrayList<ServiceTuple> offers=new ArrayList<ServiceTuple>();
32
33
34         public Matchmaker()
35         {}
36
37         @Override
38         public void input(URL arg0) {
39             try {
40                     ServiceTuple service=new ServiceTuple(arg0,reasoner);
41                     offers.add(service);
42                     System.out.println("helloooo");
43             } catch (Exception e) {
44                     e.printStackTrace();
45             }
46         }
47         @Override
48         public Hashtable<URL, Vector<URL>> query(URL arg0)
49         {
50         Hashtable<URL,Vector<URL>> finalOutput=new Hashtable<URL,Vector<URL>>();
51         try
52         {
53             query=new ServiceTuple(arg0,reasoner);
54             /*
55              * ***************************************************************************
56              * We first perform the matching
57              * ***************************************************************************
58              */
59             Group initialGroup=new Group();
60             for(ServiceTuple serviceAd:offers)
61             {
62                 match(query,serviceAd,reasoner);
63                 initialGroup.addAService(serviceAd);
64             }
65             /*
66              * **************************************************************************
67              * We secondly perform the ranking
68              * **************************************************************************
69              */
70             Node<Group>  root= new Node<Group>();
71             root.setData(initialGroup);
```

...

Figure 4.3: Extract from class Matching

A series of experimentations have been conducted on a Dell Inspiron 15 3735 Laptop with an Intel Core I5 processor (1.6 GHz) and 2 GB of memory.The result of these experimentations will be discussed later in Section 4.4.

4.3.2 Test collection

The test collection used is OWLS-TC4, which consists of 1083 Web service offers described in OWL-S 1.1 and 42 queries. For illustration, we provide in Figure 4.4 an Ontology example, concerning health insurance, which has been used for the experimentations.

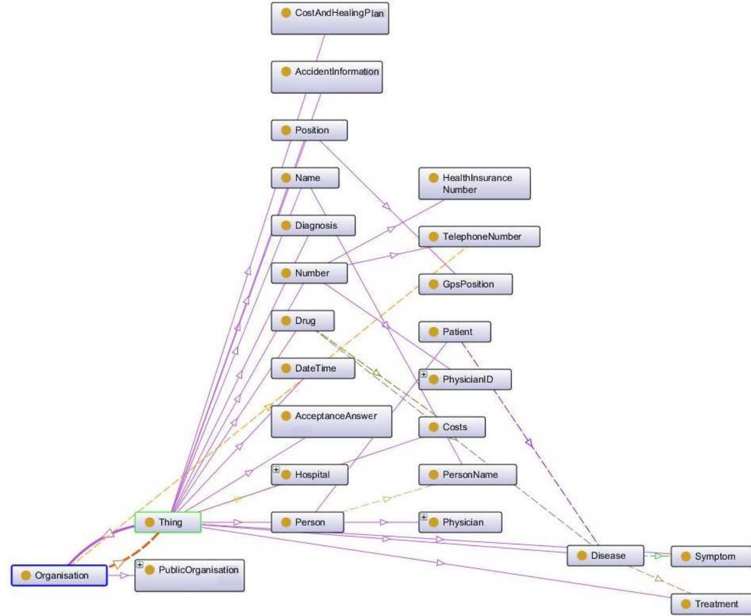


Figure 4.4: Ontology example about Health Insurance

4.3.3 Evaluation metrics

The SME2 provides several metrics to evaluate the performance and effectiveness of a Web service matchmaker. The metrics that have been considered in this chapter are: precision and recall (Section 4.3.3.1), average precision (Section 4.3.3.2), query response time (Section 4.3.3.3) and memory consumption (Section 4.3.3.4). The definition of these metrics are given in the rest of this section. They have reproduced from [40][49].

4.3.3.1 Precision and Recall

These are standard measures for evaluating the performance of information Retrieval systems. Given a query q, and a set of items D, let R_q be the relevance set of q (i.e, the set of relevant items for the query q), and let A_q be the computed answer set (i.e., a set of items returned as answer to q). Then, the Precision and Recall are defined as follows:

$$Pecision = \frac{|R_q \bigcap A_q|}{|A_q|} \tag{4.1}$$

$$Recall = \frac{|R_q \bigcap A_q|}{R_q} \tag{4.2}$$

Precision can then be measured as a function of Recall by scanning the output ranking from the top to the bottom and observing the Precision at different Recall levels.

4.3.3.2 Average Precision

If the performance of a system needs to be captured in a single measure, the common one is *Average Precision* over relevant items. The Average precision is the average of precisions computed at each point of the relevant documents is the returned ranked list. It is calculated as follows:

$$AvePrecision = \sum_{R \in RS} \frac{1}{|Rel_R|} \sum_{r=1}^{|L_R|} isrel(r) \frac{count(r)}{r} \tag{4.3}$$

where L_R is a ranked list of Web services retrieved from the request R,

$$count(r) = \sum_{i=1}^{r} isrel(i), \tag{4.4}$$

and

$$isrel(r) = \begin{cases} 1, & \text{if service in } L_R \text{ at rank } r \text{ is relevant,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

4.3.3.3 Query Response Time

This metric measures the the time required by the matchmaker to answer a single query. This metric does not take into account the time spent in the initialization phase for parsing Web service descriptions.

4.3.3.4 Memory Usage

This metric measures the about the memory usage during the total execution time.

4.4 Performance evaluation analysis

In order to study the performance of each instance of the modules supported by the PMRF and describe the difference between them, we implemented seven plugins to be used with the SME2 tool. Each of these plugins represents a different combination of the matching, similarity computing and ranking algorithms. The characteristics of these plugins are summarized in Table 4.1.

Table 4.1: Description of the evaluated configurations

Configuration	Similarity measure instance	Matching instance	Ranking instance
`Configuration 1`	Accurate MinEdge	Trivial Matching	Trivial Ranking
`Configuration 2`	Efficient MinEdge	Trivial Matching	Trivial Ranking
`Configuration 3`	Accurate MaxEdge	Trivial Matching	Trivial Ranking
`Configuration 4`	Accurate MinEdge	Fully Parameterized Matching	Trivial Ranking
`Configuration 5`	Accurate MaxMinEdge	Trivial Matching	RankMinMax
`Configuration 6`	Accurate MinEdge	Trivial Matching	Rule Based Ranking
`Configuration 7`	Efficient MinEdge	Trivial Matching	Rule Based Ranking

4.4.1 Comparison of configurations 1 and 2

The evaluation of configurations 1 and 2 yields to the results shown in Figures 4.5-4.7. The difference between the two configurations is the similarity measure module instance. Indeed, the first configuration employs the **Accurate MinEdge** instance while the second employs the **Efficient MinEdge** instance.

Figure 4.5 shows the average precision and figure 4.6 illustrates the recall/precision plot. We can see that configuration 1 outperforms configuration 2 for these two metrics, this is due to the use of logical inference, that obviously enhances the precision of the first configuration. In Figure 4.7, however, configuration 2 is shown to be remarkably faster than configuration 1. This is due to the inference process that consumes considerable resources.

The goal to offer a configurable, i.e., flexible, solution is fulfilled since the user can choose, according to his/her needs, a different similarity measure computing versions. In critical situations, the efficient version is preferable over the accurate one. However, if the user seeks for more precision, the accurate version is the best choice.

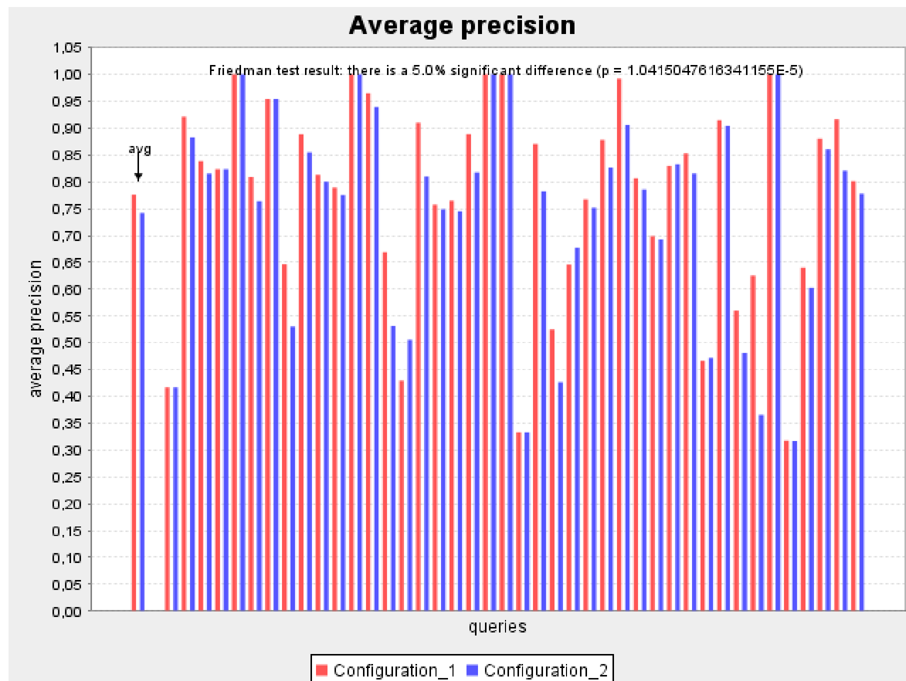


Figure 4.5: Configuration 1 vs Configuration 2: Average precision

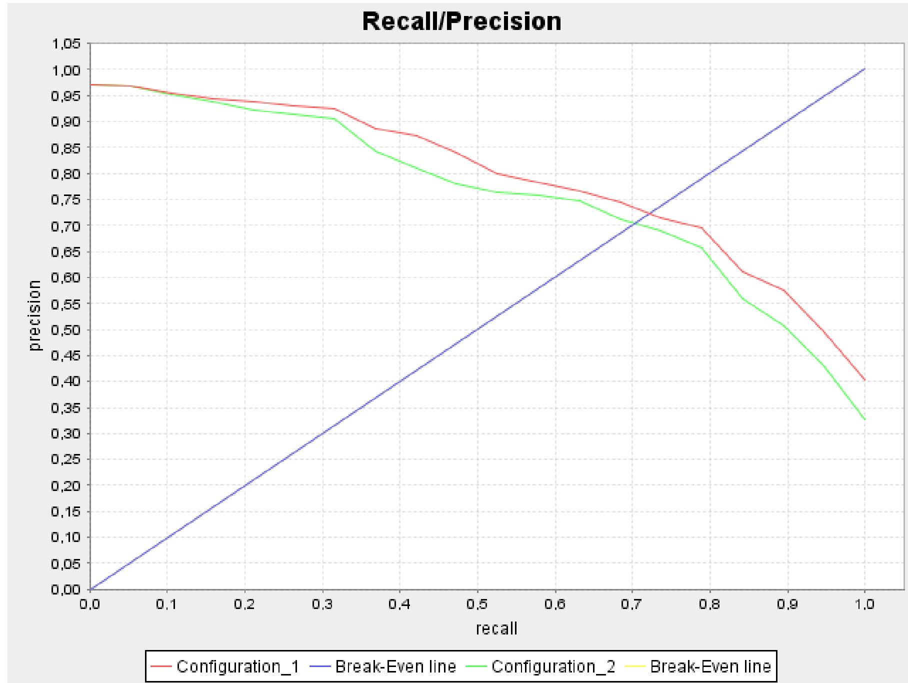


Figure 4.6: Configuration 1 vs Configuration 2: Recall/Precision

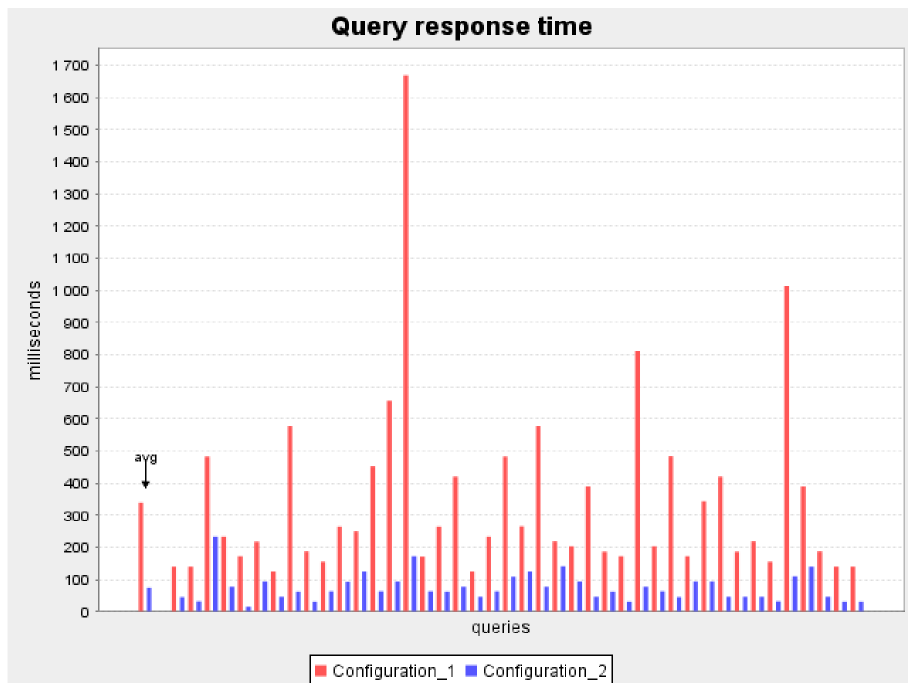


Figure 4.7: Configuration 1 vs Configuration 2: Query response time

4.4.2 Comparison of configurations 1 and 4

The results of comparison of configuration 1 and 4 are shown in Figures 4.8 and 4.9. The difference between these two configurations is the matching module instance. The first configuration is based on the trivial matching algorithm while the second uses the fully parameterized matching. Figure 4.8 shows the Recall/Precision metric results. It is easy to see that configuration 4 outperforms configuration 1. This is due to the fact that the Criteria Table restricts the results to the most relevant Web services, which will have the best ranking leading to a high Recall/Precision value. Figure 4.9 illustrates Recall/Precision plot. It shows that configuration 4 has a low recall rate. The overly restrictive Criteria Table explains these results, since it fails to return some relevant services. The user can choose which configuration fits his/her needs: a restrictive yet accurate Criteria Table or a loose matching returning the whole set of Web services.

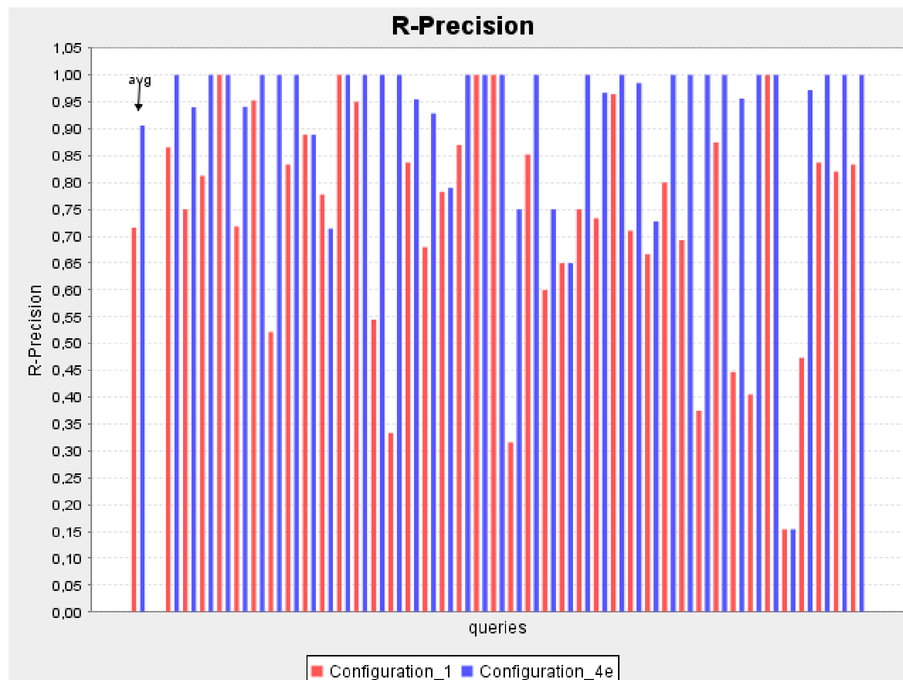


Figure 4.8: Configuration 1 vs Configuration 4: R-Precision

4.4.3 Comparison of configurations 5 and 6

Figures 4.10-4.11 show the evaluation results of configurations 5 and 6. The difference between these two configurations is the ranking module instance. The first uses the tree-based ranking algorithm while the second employs the rule-based ranking algorithm. Figure 4.10 shows that configuration 5 has a slightly better average precision than configuration 6. Figure 4.11 shows also that configuration 6 is obviously faster than configuration 5.

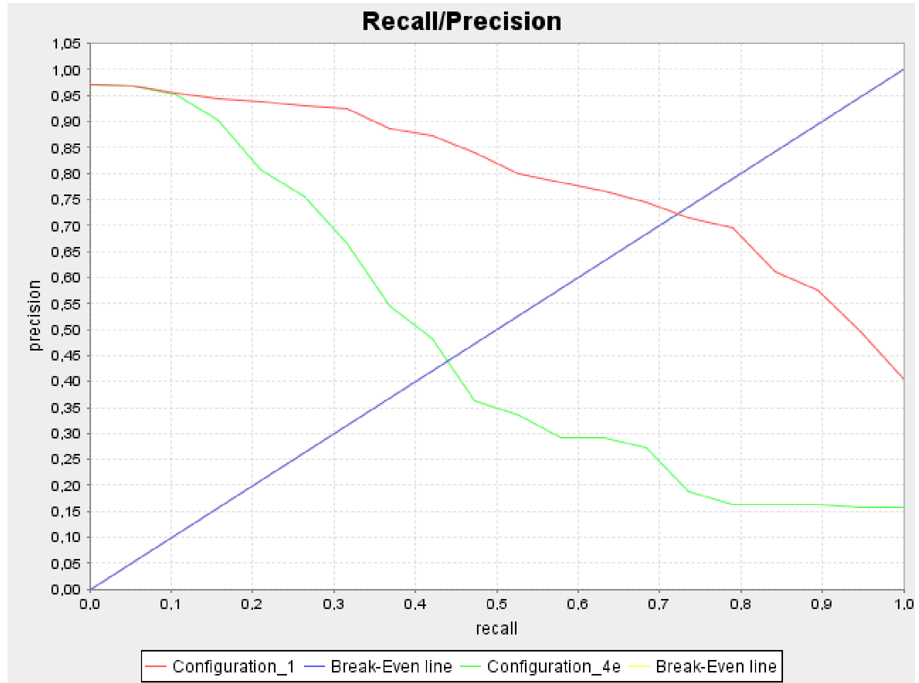


Figure 4.9: Configuration 1 vs Configuration 4: Recall/Precision

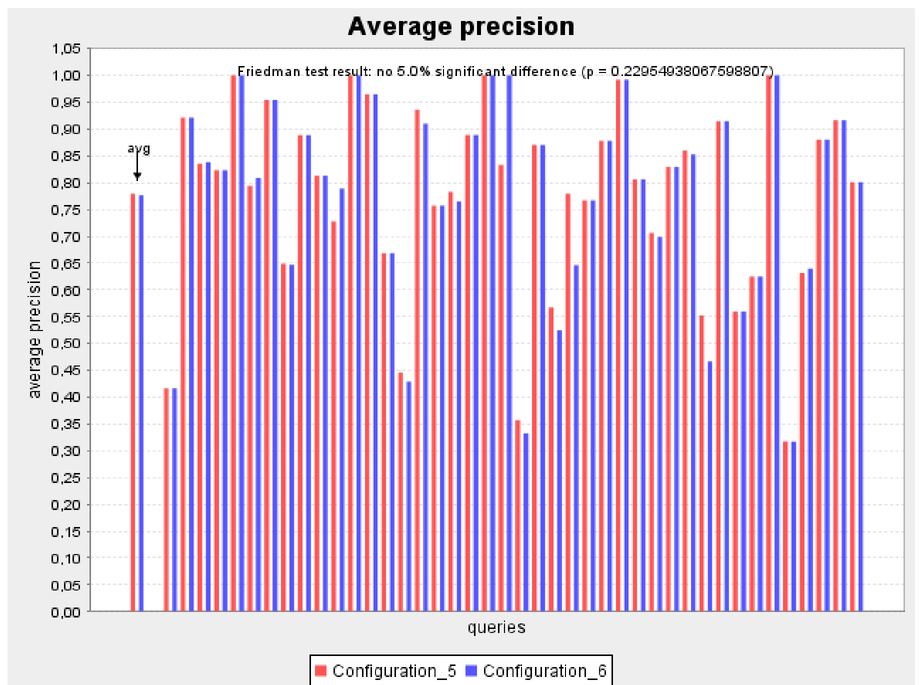


Figure 4.10: Configuration 5 vs Configuration 6: Average precision

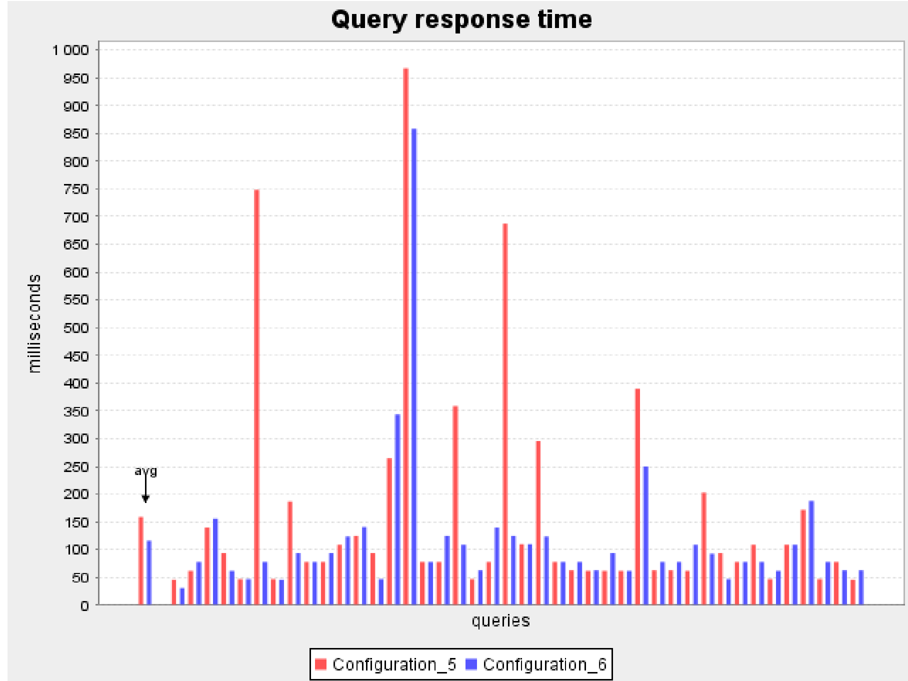


Figure 4.11: Configuration 5 vs Configuration 6: Query response time

4.5 Edge criteria order effect on the tree-based ranking

The tree-based ranking algorithm as clarified in Chapter 3 uses two criteria for the ranking. The first criteria c_{min} considers the MinEdge and the second c_{max} considers the MaxEdge. These two criteria are used sequentially during the construction of the tree and theoretically any order of their use is permitted. However, our experiments, over the OWL-TC test collection, showed that this order can affect the accuracy of the results. In detail, we implemented two algorithms each employing a different order of c_{max} and c_{min} as given in Table 4.2. We evaluated these two algorithms and measured the Average Precision and the Recall/Precision metrics.

Table 4.2: Order of the criteria considered

Order / Algorithms	First criteria	Second criteria
RankMinMax	c_{min}	c_{max}
RankMaxMin	c_{max}	c_{min}

The result of the comparison of the two situations are shown in Figures 4.12 and 4.13. According to these figures, we conclude that RankMinMax outperforms RankMaxMin. However, this final constatation should not be taken as a rule since it might depend on the considered test collection.

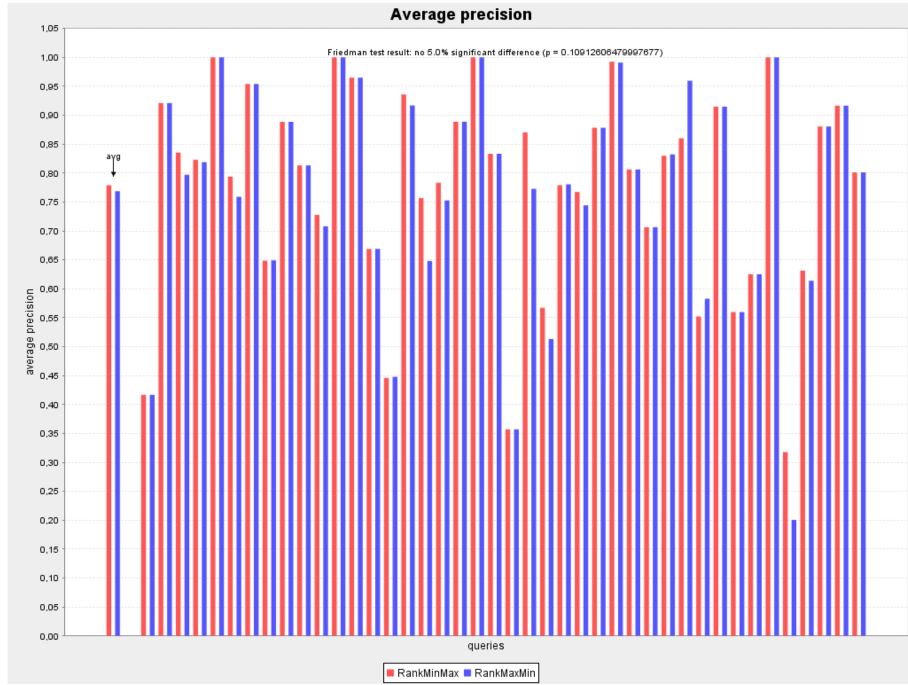


Figure 4.12: Effect of the criteria order: Average precision

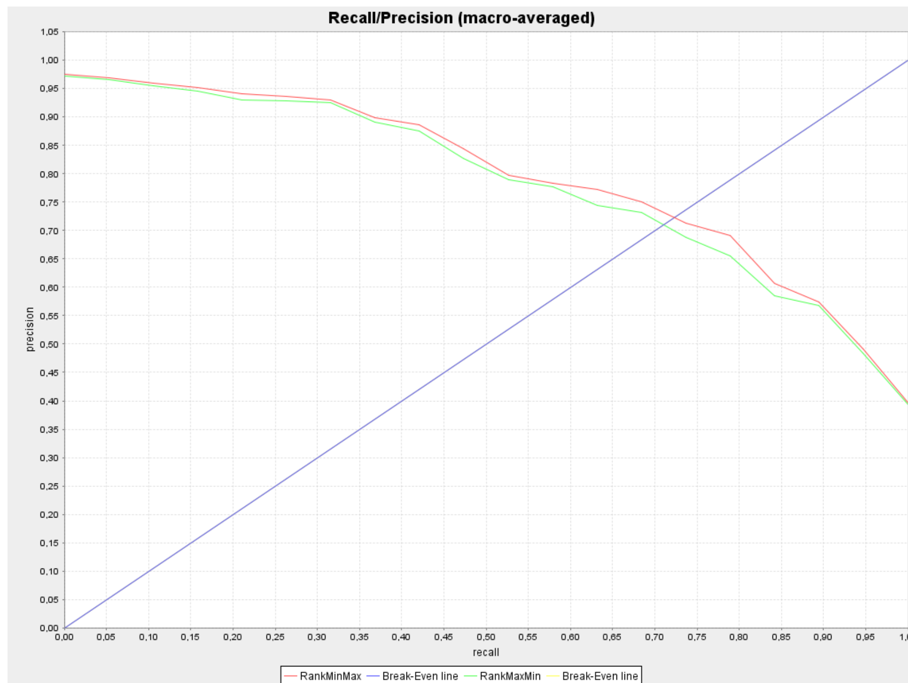


Figure 4.13: Effect of the criteria order: Recall/precision

4.6 Comparative study

We compared the results of the PMRF matchmaker with SPARQLent approach [72][73] and iSeM approach [43]. Configuration 7 was chosen to perform this evaluation. The SPARQLent is a logic-based matchmaker based on the OWL-DL reasoner Pellet (`http://pellet.owldl.com/`) to provide exact and relaxed service matchmaking. The iSeM is an hybrid matchmaker offering different filter matchings: logic-based, approximate reasoning based on logical concept abduction for matching Inputs and Outputs. We consider only the I-O logic-based matching for the comparison issue. We note that SPARQLent and iSeM consider preconditions and effects of Web services, which is not considered in our work.

4.6.1 Recall/Precision

Figure 4.14 presents the recall/precision of PMRF, iSeM logic-based and SPARQLent. This shows that PMRF recall is significantly better than both iSeM logic-based and SPARQLent. This means that our approach is able to reduce the amount of false positives.

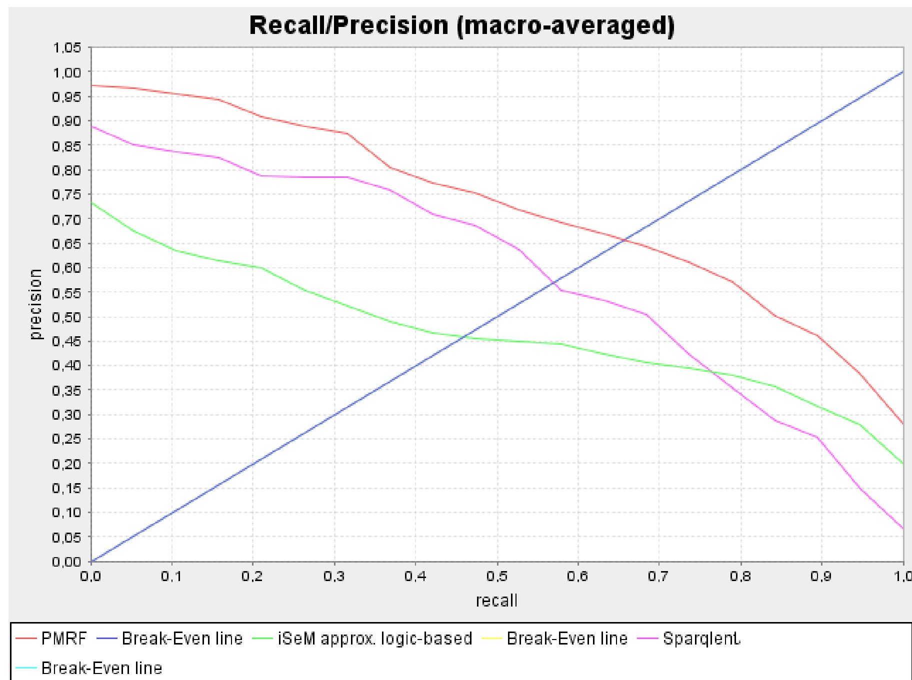


Figure 4.14: Comparative study: Recall/Precision

4.6.2 Average precision

The Average Precision is shown in Figure 4.15. This figure shows that PMRF has a more accurate precision than iSeM logic-based and SPARQLent. It is possible to conclude that PMRF has better ranking precision than the two other approaches. This is due to the score-based ranking that gives a more coarse evaluation than a degree aggregation. Indeed, SPARQLent and iSeM approaches adopt a subsumption-based ranking strategy as described in [64], which gives equal weights to all similarity degrees. In our approach, we used provides a scoring-based technique offering accurate ranking. In addition, the ranking generated is more fine-grained than SPARQLent and iSeM.

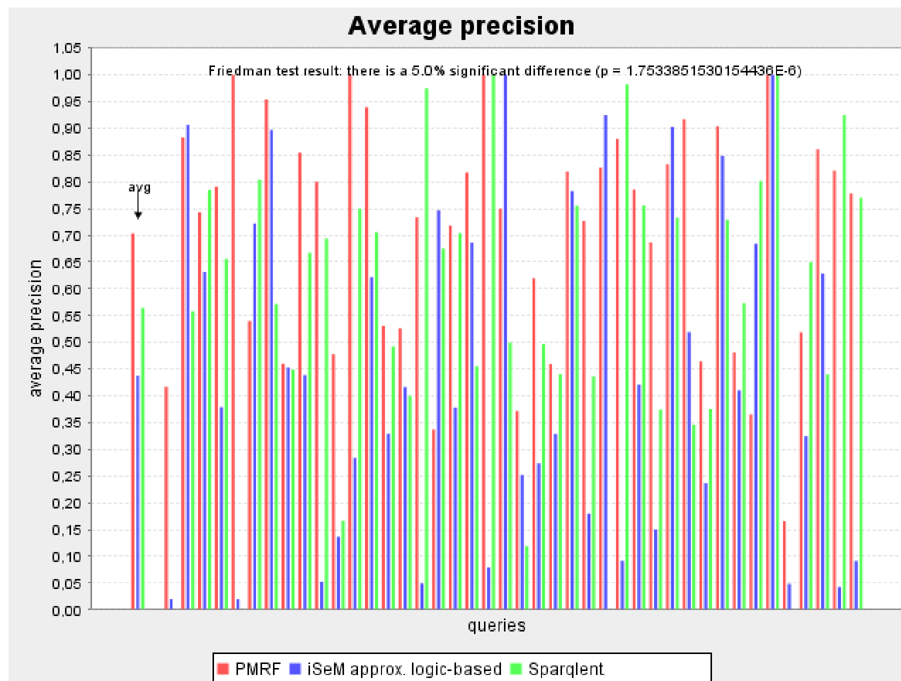


Figure 4.15: Comparative study: Average precision

4.6.3 Query response time

Figure 4.16 compares the Query Response Time of PMRF, logic-based iSeM and SPARLent. The first column (Avg) gives the average response time for the matchmakers. The experimental results show that the PMRF is faster than SPARQLent (760ms for SPARQLent versus 128ms for PMRF) and slightly less faster than logic-based iSeM (65ms for iSeM). We note that SPARQLent has especially high query response time if the query include preconditions/effects. The SPARQLent is also based on an OWL DL reasoner, which is an expensive processing. PMRF and iSeM have close query response time because both consider direct parent/child relations in a subsumption graph, which reduces significantly the query processing.

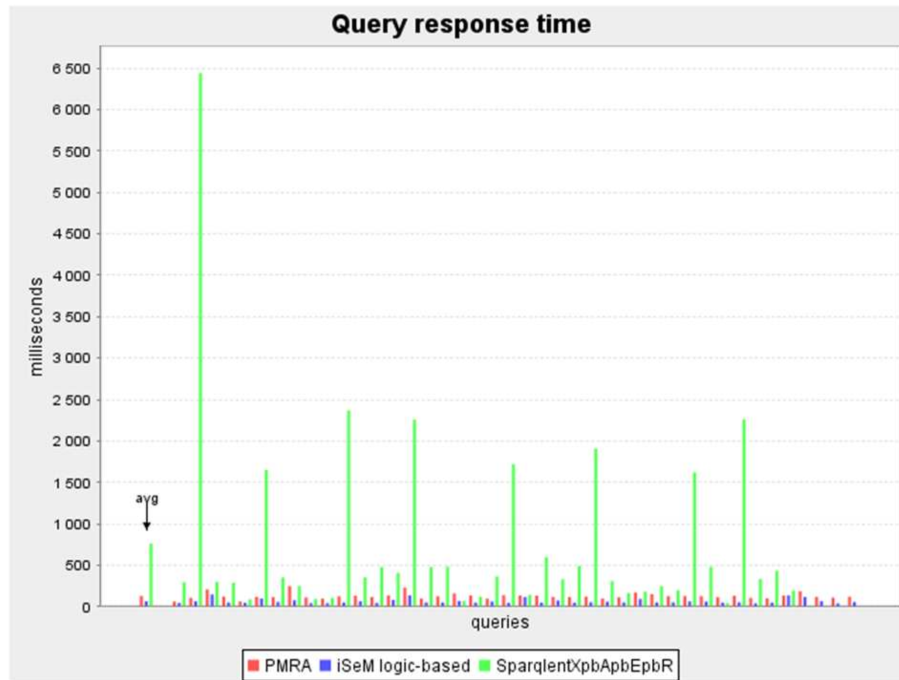


Figure 4.16: Comparative study: Query response time

4.6.4 Memory usage

Figure 4.17 shows the Memory Usage for PMRF, iSeM logic-based and SPARQLent. It is easy to see that PMRF consumes less memory than iSeM logic-based and SPARQLent. This can be explained by the fact that PMRF does not require a reasoner neither a SPARQL queries in order to compute similarities between concepts.

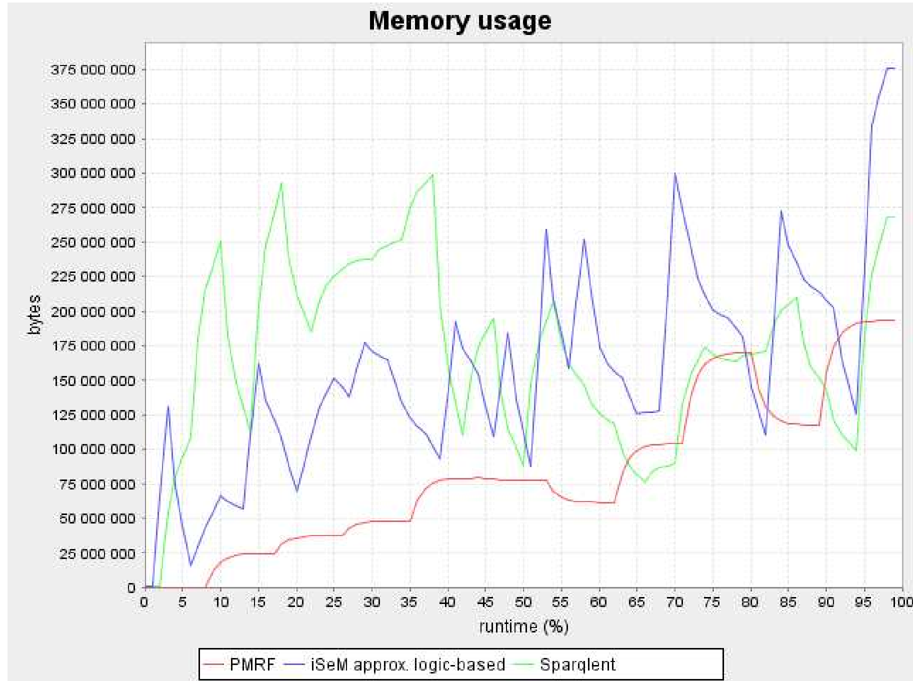


Figure 4.17: Comparative study: Memory usage

4.7 Conclusion

In this chapter, we first presented the architecture of the developed prototype, namely PMRF. Then, we provided the results of performance evaluation of the PMRF. The different algorithms have been implemented and evaluated using the OWLS-TC4 datasets. The SME2 has been used to evaluate the performance of these algorithms. Results show that the algorithms behave globally well in comparison to similar existing ones.

Conclusion

Summary

The matchmaking is a crucial operation in Web service composition. The objective of matchmaking is to discover and select the most appropriate (i.e., that responds better to the user request) Web service among the different available candidates. Several matchmaking frameworks are now available in the literature. However, most of these frameworks present at least one of the following shortcomings:

1. use of strict syntactic matching, which generally leads to low recall and low precision of the retrieved services;
2. use of capability-based matchmaking, which is proven to be inadequate in practice;
3. lack of customization support; and
4. lack of accurate ranking of matching Web services, especially within semantic-based matching.

In this research project, we proposed several conceptual and algorithmic solutions to jointly deal with these shortcomings. More precisely, we proposed:

- *Improved algorithms for similarity measure computing.* We proposed two algorithms to improve the computing of the similarity measure proposed in [7]. The first algorithm affects the precision of [7]'s algorithm but improves considerably its complexity, while the second algorithm enriches the semantic distance values used in [7]'s, which ameliorates considerably the precision of [7]'s algorithm.
- *A series of semantic matchmaking algorithms.* We proposed three matchmaking algorithms supporting different customization levels: (i) the trivial matching algorithm with no customization support, (ii) the partially parameterized matching algorithm that allows the user to specify the set of attributes to be used in the matching, and (iii) the fully parameterized matching algorithm that permits the user to control the matched attributes, the order in which attributes are compared, as well as the way the sufficiency is computed. These algorithms generalize and improve the proposals of [7][15][16][24]. We also discussed the extension of

the proposed matching algorithms to other types of matching. The proposed matching algorithms permit to solve the first and third shortcomings of semantic matchmaking frameworks.

- *A technique for scoring Web services.* We presented a technique for computing Web services scores based on the information provided by the user. The proposed technique permits to transform the similarity measures, which are defined on an ordinal scale, into numerical values. This technique assigns to each advertised Web service the percentage of the extent of the similarity degrees scale. The advantage of this technique is its ability to ensure that the scores cover all the range [0,1]. In other words, the lowest score will be equal to 0 and the highest score will be equal to 1.

- *Three algorithms for ranking Web services.* We proposed three approaches for ranking Web services: score-based, rule-based and tree-based. The score-based approach relies on the scores only. The rule-based approach defines and uses a series of rules to rank Web services. It permits to solve the ties problem encountered by the first approach. The tree-based approach relies on the use of a tree data structure. It permits to solve the problem of ties of the first approach. In addition, it is computationally better than the second approach. A series of algorithms are proposed for the different ranking approaches. These algorithms permit to solve the fourth shortcoming of semantic matchmaking frameworks.

- *A prototype.* A prototype called PMRF supporting all the proposed algorithms has been implemented. We used the Semantic Matchmaker Evaluation Environment (SME2) to compare PMRF to SPARQLent [73] and iSeM [43] frameworks in respect to precision and recall, average precision, query response time and memory time. Results show that PMRF's recall and average precision is significantly better than both iSeM logic-based and SPARQLent. This means that our approach is able to reduce the amount of false positive. The results show also that PMRF is more faster than SPARQLent and slightly less faster than logic-based iSeM. Finally, PMRF consumes less memory than iSeM logic-based and SPARQLent.

In this rapport, we also studied the computational complexity of all the matching and ranking algorithms. We also discussed and compared the proposed algorithms to existing ones.

The solutions proposed in this research project permit to fully overcome the first, third and fourth shortcomings of semantic matchmaking frameworks. The second shortcoming is partially addressed in this research project. It will be addressed in more depth in our future work (See paragraph B in the Future Work section).

Future work

In the rest of this section, we briefly discuss some topics for extending this research project.

A. QoS-aware semantic matchmaking and ranking of Web services

An important characteristic concerning matchmaking frameworks is their ability to support non-functional matching. In this respect, several existing approaches consider Quality of Service (QoS) attributes in the matching process. For instance, the author in [53] proposes two approaches to Web service selection based on QoS attributes. The authors in [71] discuss various techniques of QoS-based Web service selection. In the paper of [46], a QoS-based Web service selection based on a stochastic optimization is provided. The authors in [83] propose a QoS-aware Web service selection algorithm based on clustering. A Web service selection QoS broker by maximizing a utility function is proposed in [60].

The non-functional matching support have not been considered in this research report. In the future, we intend to enhance our framework to support QoS-aware semantic matchmaking and ranking of Web service. The work of [16] could be a start point.

B. Multicriteria-based matching and ranking of Web services

There are few proposals that explicitly use multicriteria evaluation to support matching and ranking of Web services (see, e.g., [23][38][59][60][88]). The authors in [88] use linear programming techniques to compute the optimal execution plans for Web services. The author in [59] considers two evaluation criteria (time and cost) and assigns to each one a weigh. The best composition of Web services is then decided on the basis of the optimum combined score and a service selection QoS broker by maximizing a utility function is provided by [60]. Most of these proposals use weighted-sum-like aggregation techniques. This type of methods have two main shortcomings: (i) they accept only numerical data, and (ii) may lead to the compensation problem since low values may be counterbalanced by high values.

In the future, we intend two use two well-known and more advanced multicriteria methods: DRSA [31][32][33] and ELECTRE TRI [3][27]. These two methods permit to take into account attributes that are defined on any scale type (binary, nominal, ordinal or numerical). Hence, they are particularity suitable for including the QoS attributes in the matching and ranking process. Furthermore, the DRSA and ELECTRE TRI can been seen as case-based reasoning methods [45], which minimizes the cognitive effort required from user.

C. Fuzzy semantic matchmaking and ranking of Web service

In this research project, we assumed that the data and user parameters are crisply defined. However, in practice these data and parameters can be imprecise and uncertain.

The fuzzy sets and logic theory, introduced by L.A. Zadeh [87] (see also, e.g., [25]), can be used to cope better with imprecision and uncertainty in Web services matching and ranking. Some recent proposals to dealt with the imprecision and uncertainty aspects of Web services matching and ranking have been proposed as, e.g., [20][26][36][84]. However, we think that despite of its importance in practice, the support of the imprecision and uncertainty in Web services matching and ranking have not received enough attention.

In the future, we intend to enhance this research project by conceiving and developing algorithms and tools that support of imprecision and uncertainty in Web services matching and ranking.

Bibliography

[1] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A system for end to end composition of Web services. *Journal of Web Semantics*, 3:311–339, 2005.

[2] M. Aiello, E. Khoury, A. Lazovik, and P. Ratelband. Optimal QoS-aware Web service composition. In *IEEE Conference on Commerce and Enterprise Computing*, pages 491–494, 2009.

[3] J. Almeida-Dias, J.R. Figueira, and B. Roy. Electre Tri-C: A multiple criteria sorting method based on characteristic reference actions. *European Journal of Operational Research*, 204(3):565–580, 2010.

[4] K. Arnold, B. O'Sullivan, R.W. Scheifler, J. Waldo, and A. Woolrath. *The Jini Specification*. Addison-Wesley, Reading, MA, 1999.

[5] P. Bartalos and M. Bielikova. Fast and scalable semantic Web service composition approach considering complex pre/postconditions. In *International Workshop on Web Service Composition and Adaptation*, pages 414–421, 2009.

[6] M. Beck and B. Freitag. Semantic matchmaking using ranked instance retrieval. In *Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval*, Seoul, South Korea, September, 11 2006.

[7] U. Bellur and R. Kulkarni. Improved matchmaking algorithm for semantic Web services based on bipartite graph matching. In *IEEE International Conference on Web Services*, pages 86–93, Salt Lake City, Utah, USA, 9-13 July 2007.

[8] U. Bellur, H. Vadodaria, and A. Gupta. Semantic matchmaking algorithms. In W. Bednorz, editor, *Advances in Greedy Algorithms*, pages 481–502. SInTech, Vienna, Austria, 2008. Available from: `http://www.intechopen.com/books/greedy_algorithms/semantic_matchmaking_algorithms`.

[9] S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient semantic service discovery in pervasive computing environments. In *ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 240–259, Melbourne, Australia, 27 November - 1 December 2006.

[10] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. http://www.scientificamerican.com/article/the-semantic-web/, 2011. Scientific American Magazine. Retrieved November 5, 2014.

[11] N. Bikakis, C. Tsinaraki, N. Gioldasis, I. Stavrakantonakis, and S. Christodoulakis. The XML and semantic Web worlds: Technologies, interoperability and integration. a survey of the state of the art. In I.E. Anagnostopoulos, M. Bielikova, P. Mylonas, and N. Tsapatsoulis, editors, *Semantic Hyper/Multimedia Adaptation*, volume 418 of *Studies in Computational Intelligence*, pages 319–360. Springer Berlin Heidelberg, 2013.

[12] BPEL. Business process execution language for Web services, version 1.1. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf`, May 2003.

[13] V. Cardellini, E. Casalicchio, V. Grassi, and F. Lo Presti. Flow-based service selection for Web service composition supporting multiple qos classes. In *IEEE International Conference on Web Services*, pages 743–750, 2007.

[14] S. Chakhar. QoS-enhanced broker for composite Web service selection. In *Proc. of the 8th International Conference on Signal Image Technology & Internet Based Systems*, pages 533–540, Sorrento-Naples, Italy, September 2012.

[15] S. Chakhar. Parameterized attribute and service levels semantic matchmaking framework for service composition. In *Fifth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2013)*, pages 159–165, Seville, Spain, 27 January - 1 February 2013.

[16] S. Chakhar, A. Ishizaka, and A. Labib. Qos-aware parameterized semantic matchmaking framework for Web service composition. In V. Monfort and K.-H. Krempels, editors, *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 1, Barcelona, Spain, 3-5 April, 2014*, pages 50–61, Barcelona, Spain, 2014. SciTePress.

[17] S. Chakhar, S. Youcef, V. Mousseau, L. Mokdad, and S. Haddad. Multicriteria evaluation-based conceptual framework for composite Web service selection. `http://www.lipn.univ-paris13.fr/youcef/BookQoS`, 2011.

[18] M. Champion, E. Newcomer, and D. Orchard. Web service architecture. W3C Draft, 2000.

[19] H. Chang and K. Lee. Quality-driven Web service composition methodology for ubiquitous services. *Journal of Information Science and Engineering*, 26(6):1957–1971, 2010.

[20] K.-M. Chao, M. Younas, C.-C. Lo, and T.-H. Tan. Fuzzy matchmaking for Web services. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 2, pages 721–726, March 2005.

[21] H. Che, Y. Li, A. Oberweis, and W. Stucky. Web service composition based on XML nets. In *International Conference on Systems*, pages 1–10, Hawaii, 2009.

[22] UDDI Consortium. Uddi executive white paper. `http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf`, November 2001.

[23] L. Cui, S. Kumara, and D. Lee. Scenario analysis of Web service composition based on multi-criteria mathematical goal programming. *Service Science*, 3(4):280–303, 2011.

[24] P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division, March 2004.

[25] D. Dubois, W. Ostasiewicz, and H. Prade. Fuzzy sets: History and basic notions. In D. Dubois and H. Prade, editors, *Fundamentals of Fuzzy Sets*, volume 7 of *The Handbooks of Fuzzy Sets Series*, pages 21–124. Springer, US, 2000.

[26] G. Fenza, V. Loia, and S. Senatore. A hybrid approach to semantic Web services matchmaking. *International Journal of Approximate Reasoning*, 48(3):808–828, 2008.

[27] J.R. Figueira, V. Mousseau, and B. Roy. Electre methods. In J.R. Figueira, S. Greco, and M. Ehrgott, editors, *Multiple criteria decision analysis: State of the art surveys*, pages 133–162. Springer-Verlag, New York, 2005.

[28] C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[29] P. Fu, S. Liu, H. Yang, and L. Gu. Matching algorithm of Web services based on semantic distance. In *International Workshop on Information Security and Application (IWISA 2009)*, pages 465–468, Qingdao, China, 21-22 November 2009.

[30] F.E. Gmati, N. Yacoubi-Ayadi, and S. Chakhar. Parameterized algorithms for matching and ranking Web services. In *Proceedings of the On the Move to Meaningful Internet Systems: OTM 2014 Conferences 2014*, volume 8841 of *Lecture Notes in Computer Science*, pages 784–791. Springer, 2014.

[31] S. Greco, B. Matarazzo, and R. Slowiński. Rough sets theory for multicriteria decision analysis. *European Journal of Operational Research*, 129(1):1–47, 2001.

[32] S. Greco, B. Matarazzo, and R. Slowiński. Rough approximation by dominance relations. *International Journal of Intelligent Systems*, 17(2):153–171, 2002.

[33] S. Greco, R. Slowiński, and Y. Yao. Bayesian decision theory for dominance-based rough set approach. In J.T. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N.J. Cercone, and D. Slezak, editors, *Rough Sets and Knowledge Technology*, volume 4481 of *Lecture Notes in Computer Science*, pages 134–141. Springer Berlin Heidelberg, 2007.

[34] R. Guo, J. Le, and X.L. Xiao. Capability matching of Web services based on OWL-S. In *Sixteenth International Workshop on Database and Expert Systems Applications*, pages 653–657, 22-26 August 2005.

[35] H. Hao, H. Haas, and D. Orchard. Web services architecture usage scenarios, 2004.

[36] C.-L Huang. A moderated fuzzy matchmaking for Web services. In *Proceedings of the Fifth International Conference on Computer and Information Technology (CIT 2005)*, pages 1116–1122, 2005.

[37] Y.Qiu J. Ge and S. Yin. Web services composition method based on owl. In *International Conference on Computer Science and Software Engineering*, pages 74–77, China, 2008.

[38] B. Jeong, H. Cho, B. Kulvatunyou, and A. Jones. A multi-criteria Web services composition problem. In *IEEE International Conference on Information Reuse and Integration(IRI 2007)*, pages 379–384, 2007.

[39] M. Klusch. Semantic Web service coordination. In *CASCOM: Intelligent Service Coordination in the Semantic Web Whitestein Series in Software Agent Technologies and Autonomic Computing*, pages 59–104. Springer Verlag, 2008.

[40] M. Klusch, M. Dudev, J. Misutka, P. Kapahnke, and M. Vasileski. *SME^2 Version 2.2. User Manual.* The German Research Center for Artificial Intelligence (DFKI), Germany, 2010.

[41] M. Klusch and A. Gerber. Evaluation of service composition planning with QWLS-XPlan. In *International Conference on Web Intelligence and Intelligent Agent Technology*, pages 117–120, 2006.

[42] M. Klusch and P. Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.

[43] M. Klusch and P. Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(0):1–14, 2012.

[44] N. Kokash, A. Birukou, and V. DAndrea. Web service discovery based on past user experience. In *Proceedings of the 10th International Conference on Business Information Systems*, pages 95–107, April 25-27 2007.

[45] J.L. Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34, 1992.

[46] R. Krithiga. Qos-aware Web service selection using SOMA. *Global Journal of Computer Science and Technology*, 12(10):46–51, 2012.

[47] J. Kuck and M. Gnasa. Context-sensitive service discovery meets information retrieval. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom'07)*, pages 601–605, March 2007.

[48] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[49] U. Küster and B. König-Ries. Measures for benchmarking semantic Web service matchmaking correctness. In *Proceedings of the 7th International Conference on The Semantic Web: Research and Applications - Volume Part II*, ESWC'10, pages 45–59, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] C. Lee, A. Helal, N. Desai, V. Verma, and B. Arslan. Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 33(6):682–696, 2003.

[51] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 331–339, New York, NY, USA, 2003. ACM.

[52] N. Lin, U. Kuter, and E. Sirin. Web service composition with user preferences. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *The Semantic Web: Research and Applications*, volume 5021 of *Lecture Notes in Computer Science*, pages 629–643. Springer Berlin Heidelberg, 2008.

[53] S.A. Ludwig. Memetic algorithm for Web service selection. In *Proceedings of the 3rd Workshop on Biologically Inspired Algorithms for Distributed Systems*, BADS '11, pages 1–8, New York, NY, USA, 2011. ACM.

[54] Q. Lv, J. Zhou, and Q. Cao. Service matching mechanisms in pervasive computing environments. In *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, pages 1–4, May 2009.

[55] Z. Maamar, S.K. Mostefaoui, and Q.H. Mahmoud. Context for personalized Web services. In *In Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*, pages 166b–166b, Jan 2005.

[56] M. Malaimalavathani and R. Gowri. A survey on semantic Web service discovery. In *International Conference on Information Communication and Embedded Systems (ICICES 2013)*, pages 222–225, February 2013.

[57] U.S. Manikrao and T.V.Prabhakar. Dynamic selection of Web services with recommendation system. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP 2005)*, pages 117–121, August 2005.

[58] R. Manoharan, A. Archana, and S.N. Cowla. Hybrid Web services ranking algorithm. *International Journal of Computer Science Issues*, 8(2):83–97, 2011.

[59] D.A. Menascé. Composing Web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90, 2004.

[60] D.A. Menascé and V. Dubey. Utility-based QoS brokering in service oriented architectures. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 422–430, 2007.

[61] H. Mili, G. Tremblay, A. Caillot, and R.B. Tamrout. Web service composition as a function cover problem. In *MCeTech Montreal Conference on eTechnologies*, pages 73–85, Canada, 2005.

[62] B.A. Miller and R.A. Pascoe. Salutation service discovery in pervasive computing environments,. White paper, IBM Pervasive Computing, February 2000.

[63] D. Mukhopadhyay and A. Chougule. A survey on Web service discovery approaches. In D.C. Wyld, J. Zizka, and D. Nagamalai, editors, *Advances in Computer Science, Engineering & Applications*, volume 166 of *Advances in Intelligent and Soft Computing*, pages 1001–1012. Springer Berlin Heidelberg, 2012.

[64] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, UK, 2002. Springer-Verlag.

[65] D. Petrova-Antonova and A. Dimov. Towards a taxonomy of Web service composition approaches. *Scalable Computing: Practice and Experience*, 12:377–384, 2011.

[66] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring Web service composition. *Lecture Notes in Computer Science*, 3192:106–115, 2004.

[67] G. Priyadharshini, R. Gunasri, and B.B. Saravana. A survey on semantic Web service discovery methods. *International Journal of Computer Applications*, 82(11):8–11, 2013.

[68] M. Qiao, F. Khendek, A. Serhani, R. Dsouli, and G. Roch. An architecture for automatic qos adaptation for composite Web services. *Journal of Web Services Practices*, 4(1):18–27, 2009.

[69] P.R. Reddy, A. Damodaram, and A.V.K. Prasad. Heterogeneous matchmaking approaches for semantic Web service discovery using owl-s. In SureshChandra Satapathy, P.S. Avadhani, and Ajith Abraham, editors, *Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012) held in Visakhapatnam, India, January 2012*, volume 132 of *Advances in Intelligent and Soft Computing*, pages 605–612. Springer Berlin Heidelberg, 2012.

[70] W. Rong, K. Liu, and L. Liang. Personalized Web service ranking via user group combining association rule. In *IEEE International Conference on Web Services (ICWS 20090*, pages 445–452, July 2009.

[71] M. Sathya, M. Swarnamugi, P. Dhavachelvan, and G. Sureshkumar. Evaluation of QoS based Web-service selection techniques for service composition. *International Journal of Software Engineering*, 1(5):73–90, 2011.

[72] M.L. Sbodio. SPARQLent: A SPARQL based intelligent agent performing service matchmaking. In B. Blake, L. Cabral, B. König-Ries, U. Küster, and D. Martin, editors, *Semantic Web Services*, pages 83–105. Springer Berlin Heidelberg, 2012.

[73] M.L. Sbodio, D. Martin, and C. Moulin. Discovering semantic Web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):310–328, 2010.

[74] A. Segev and E. Toch. Context based matching and ranking of Web services for composition. *IEEE Transactions on Services Computing*, 3(2):210–222, 2011.

[75] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering Web services using multicriteria dominance relationships. *IEEE Transactions on Services Computing*, 3(3):163–177, 2010.

[76] A. Sridhar, S.S. Prasad, and M. Ubale. Semantic Web service composition with quality of service. *International Journal of Advanced Research in Computer Science & Technology*, 1(1):73–81, 2013.

[77] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The retsina mas infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.

[78] Y. Syu, S. Ma JY. Kuo, and Yong-Yi FanJiang. A survey on automated service composition methods and related techniques. In *Proc. of the Ninth International Conference on Services Computing*, pages 290–297, Hawaii,USA, 24-29 June 2012.

[79] W3C. Web services architecture requirements. `http://www.w3.org/TR/wsa-reqs`, October 2002.

[80] W3C. Web services glossary. `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice`, February 2004.

[81] W3C. W3C semantic Web activity. `http://www.w3.org/2001/sw/`, November 7 2011. Retrieved November 3, 2014.

[82] R. Wang, C.-H. Chi, and J. Deng. A fast heuristic algorithm for the composite Web service selection. In *Proceedings of the Joint International Conferences on Advances in Data and Web Management*, APWeb/WAIM '09, pages 506–518, Berlin, Heidelberg, 2009. Springer-Verlag.

[83] Y. Xia, P. Chen, L. Bao, M. Wang, and J. Yang. A QoS-aware Web service selection algorithm based on clustering. In *IEEE International Conference on Web Services (ICWS)*, pages 428–435, 2011.

[84] B.-H. Xie, Y.-J. Zhang, and Y.-Y. Guo. A Web service matchmaker based on fuzzy logic and OWL-S. In *Computational Aspects of Social Networks (CASoN), 2010 International Conference on*, pages 590–599, 2010.

[85] H. Q. Yu and S. Reiff-Marganiec. Semantic Web services composition via planning as model checking. Technical report, University of Leicester, 2006.

[86] T. Yu and K.-J. Lin. Service selection algorithms for Web services with end-to-end qos constraints. In *Proceedings of the IEEE International Conference on e-Commerce Technology (CEC 2004)*, pages 129–136, July 2004.

[87] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[88] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. Quality driven Web services composition. In *12th International Conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.

Glossary

- **BPEL**: Business Process Execution Language.
- **HTTP**: HyperText Transfer Protocol.
- **HTTPS**: Hypertext Transfer Protocol Secure.
- **IOPE**: Inputs-Outputs-Preconditions-Effects.
- **OWL**: Web Ontology Language.
- **OWL-S**: Web Ontology Language for Services.
- **OWLS-TC**: Web Ontology Language Service retrieval Test Collection.
- **PMRF**: Parameterized Matching-Ranking Framework.
- **QoS**: Quality of Service.
- **SME2** Semantic Matchmaker Evaluation Environment.
- **RDF**: Resource Description Framework.
- **SOA**: Service-Oriented Architecture.
- **SOAP**: Simple Object Access Protocol.
- **UDDI**: Universal Description, Discovery, and Integration.
- **URI**: Uniform Resource Identifier.
- **W3C**: World Wide Web Consortium.
- **WSDL**: Web Service Description Language.
- **XML**: eXtensible Markup Language.

Résumé

Le matchmaking est une tâche d'importance cruciale au niveau de la composition des services Web. L'objectif du matchmaking est la découverte et la sélection des services Web les plus pertinents parmi les différents services publiés. Plusieurs frameworks de matchmaking sont disponibles dans la littérature, mais ces frameworks présentent au moins l'une de limites suivantes : (i) l'utilisation d'un filtre unique basé sur la similarité syntaxique, ce qui réduit la précision du matchmaker ; (ii) la considération des attributs fonctionnels uniquement au cours du matchmaking ; (iii) le manque de flexibilité du matchmaking ; et (iv) l'absence d'un ranking précis des services Web identifiés, en particulier quand le matchmaking est basé sur la sémantique. Dans ce projet de recherche, nous proposons plusieurs solutions conceptuelles et algorithmiques, pour remédier à l'ensemble de ces problèmes. Nous allons particulièrement traiter la première, troisième et quatrième limites. La deuxième limite est abordée partiellement. Un prototype appelé PMRF (**P**arametrized **M**atching and **R**anking **F**ramework) supportant les algorithmes proposés a été implémenté. L'analyse de performance de ce prototype a montré qu'il réalise des résultats satisfaisants en comparaison à des framework similaires.

Mots clés: Services Web, Composition des services Web, Similarité sémantique, Matchmaking, Ranking des services Web.

Abstract

The matchmaking is a crucial operation in Web service composition. The objective of matchmaking is to discover and select the most appropriate (i.e., that responds better to the user request) Web service among the different available candidates. Several matchmaking frameworks are now available in the literature. However, most of these frameworks present at least one of the following shortcomings: (i) use of strict syntactic matching, which generally leads to low recall and low precision of the retrieved services; (ii) use of capability-based matchmaking, which is proven to be inadequate in practice; (iii) lack of customization support; and (iv) lack of accurate ranking of matching Web services, especially within semantic-based matching. In this research project, we propose several conceptual and algorithmic solutions to jointly deal with these shortcomings. More precisely, the solutions proposed in this research project permit to fully overcome the first, third and fourth shortcomings of semantic matchmaking frameworks. The second shortcoming is partially addressed in this research project. A prototype called PMRF (**P**arametrized **M**atching and **R**anking **F**ramework) supporting the proposed algorithms has been implemented. The performance analysis shows that the algorithms behave globally well in comparison to similar existing ones.

Keywords: Web Service, Service Composition, Semantic Similarity, Matchmaking, Service Ranking.