Improving Semantic Web Services Discovery and Ranking

A lightweight, integrated approach

José María García

University of Seville

Ph.D. Dissertation

Supervised by

Prof. Dr. David Ruiz

Prof. Dr. Antonio Ruiz-Cortés

September, 2012

First published in September, 2012 by

Applied Software Engineering Research Group (ISA)
Department of Computer Languages and Systems
ETSI Informática
Av. Reina Mercedes, s/n
41012 Seville, Spain
http://www.isa.us.es/

Classification (ACM 1998): D.2.11 [Software Engineering]: Software Architectures – Service-oriented architecture (SOA); H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval – Information filtering, Retrieval models, Selection process; H.3.4 [Information Storage and Retrieval]: Systems and Software – Semantic Web; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods – Representation languages.

Support: This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT projects WEB-FACTORIES (TIN2006-00472) and SETI (TIN2009-07366), by the Andalusian Government under projects ISABEL (TIC-2533) and THEOS (TIC-5906), by the EU FP7 IST project 27867 SOA4All, and by the EC FP7 Network of Excellence 215483 S-CUBE.

University of Seville

The committee in charge of evaluating the dissertation presented by José María García in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering, hereby recommends _____ of this dissertation and awards the author the grade _____.

Miguel Toro Bonilla
Catedrático de Universidad
Univ. de Sevilla

Xavier Franch Gutiérrez
Titular de Universidad
Univ. Politécnica de Cataluña

Antonio Brogi
Full Professor
Univ. of Pisa

Carlos Pedrinaci Godoy
Research Fellow
The Open University

Manuel Lama Penín
Titular de Universidad
Univ. Santiago de Compostela

To put record where necessary, we sign minutes in _____, _____

Emma and Puri, discovering and ranking, by Cristina, aged seven.

To Puri and Emma,
without them there would have never existed
neither PURI nor EMMA.

Contents

Part IV Final Remarks

Appendices

List of Figures

List of Tables

List of Listings

ACKNOWLEDGMENTS

Now that this long journey comes to an end, it is time to acknowledge all the people who have been supporting me along this thesis work. First and foremost, I would like to thank my supervisors, David and Antonio, who has been essential to finally achieve what is written in this document. Antonio, thanks for your guidance through this path, your encouragement and open mind enlightened each and every corner of my research work (and even further). David, without your constancy and support there would not be any contribution worth to present here. Thank you for all those endless meetings where we saved the world, and for understanding my erratic schedule and my deadlines policy.

I could not forget to thank all my colleagues from the ISA research group and the LSI Department, who have give me insightful help during my research, and allowed me to unwind when it was necessary. The list of names is so long that I could not write it without forgetting anyone. You know that you are also part of this story. Furthermore, I would also like to thank the external reviewers that gave me many useful comments and insights that significantly improved this dissertation, namely Dr. Ioan Toma and Dr. Sudhir Agarwal, as well as many colleagues I have met during the development of this thesis, who also gave me invaluable feedback on my research, especially Dr. Carlos Pedrinaci.

Finally, I owe my most sincere gratitude to my family and friends for all this years of kind support, love, and understanding. All your words of motivation has inspired me throughout this hard path that you have also walked with me. Specially you, Puri, that had to deal with my mood during my most stressful working periods. The smile on your face and love when arriving late at home, as well as Emma's hugs, definitely made my day. Thank you girls for making this possible.

Abstract

Semantic Web Services (SWSs) have become a preeminent research area, where various underlying frameworks, *e.g.* WSMO or OWL-S, define Semantic Web ontologies to describe Web services, so that they can be automatically discovered, ranked, composed, and invoked according to user requirements and preferences. Specifically, several service discovery and ranking techniques have been envisioned, and related tools have been made available for the community. However, existing approaches offer a limited expressiveness to define preferences that are highly dependent on underlying techniques. Furthermore, discovery and ranking mechanisms usually suffer from performance, interoperability and integration issues that prevent a wide exploitation of semantically-enhanced techniques.

In order to address these issues, current research focus is on developing lightweight SWSs descriptions, which enable interoperability of existing approaches, and corresponding discovery and ranking solutions that offer a better performance with a contained loss on precision and recall. In this thesis dissertation, we address those challenges by proposing SOUP, a fully-fledged preference ontological model that serves as the foundations for the development of lightweight tools, namely EMMA and PURI, to both improve discovery performance and integrate current ranking proposals, correspondingly.

Our contributions have been thoroughly evaluated and validated with both synthetic and real-world scenarios. First, SOUP preference model expressiveness and independence has been validated by completely describing complex scenarios from the SWS Challenge. Moreover, we have carried out an experimental study of EMMA that shows a significant performance improvement while obtaining a negligible penalty on precision and recall. Finally, PURI has been applied within the EU FP7 project SOA4All, successfully integrating its three existing ranking mechanisms (objective, NFP-based, and fuzzy based) into an interoperable discovery and ranking solution.

Resumen

Los Servicios Web Semánticos (SWSs) se han convertido en un área de investigación muy activa, en la que diversos frameworks, como WSMO u OWL-S, definen ontologías de la Web Semántica para describir servicios Web, de forma que puedan descubrirse, ordenarse, componerse y ejecutarse automáticamente, de acuerdo a los requisitos y preferencias del usuario. En concreto, diversas técnicas de descubrimiento y ranking han sido propuestas, incluyendo herramientas relacionadas. Sin embargo, las propuestas existentes ofrecen una expresividad limitada para definir preferencias, siendo muy dependientes de los formalismos subyacentes. Además, las técnicas semánticas actuales sufren de problemas de rendimiento, interoperabilidad e integración, impidiendo su explotación efectiva.

Para solucionar estos problemas, la investigación se centra actualmente en el desarrollo de descripciones de SWSs ligeras, las cuales faciliten la interoperabilidad de las propuestas actuales, así como en soluciones para el descubrimiento y el ranking que proporcionen mejor rendimiento con una disminución acotada en su precisión. En esta memoria de tesis, resolvemos esos desafíos mediante la propuesta de SOUP, un completo modelo ontológico de preferencias sobre el que hemos desarrollado unas herramientas ligeras, llamadas EMMA y PURI, que mejoran el rendimiento del descubrimiento e integran las propuestas actuales de ranking, respectivamente.

Nuestras contribuciones han sido evaluadas aplicándolas a escenarios sintéticos y reales. En primer lugar, la expresividad e independencia del modelo de preferencias SOUP se ha validado mediante la descripción completa de escenarios complejos del SWS Challenge. Además, hemos llevado a cabo un estudio experimental de EMMA que muestra una mejora de rendimiento significativa, obteniendo una penalización en la precisión insignificante. Por último, hemos aplicado PURI dentro del proyecto SOA4All, integrando sus tres mecanismos de ranking (objetivo, basado en NFP, y en lógicas difusas) en una solución de descubrimiento y ranking interoperable.

PART I

INTRODUCTION AND BACKGROUND

INTRODUCTION

A science only advances with certainty, when the plan of inquiry and the object of our researches have been clearly defined; otherwise a small number of truths are loosely laid hold of, without their connexion being perceived, and numerous errors, without being enabled to detect their fallacy.

Jean-Baptiste Say (1767–1832)
French economist

This thesis dissertation presents our results on Improving Semantic Web Services Discovery and Ranking *processes, proposing* a lightweight, integrated approach *based on a novel preference model that enables the optimization, interoperability and integration of discovery and ranking mechanisms. In this chapter we present an overview on the contributions discussed in this dissertation, firstly introducing our research context in §1.1 so that the summary of contributions discussed in §1.2 can be properly understood. Furthermore, §1.3 contextualizes this thesis within a number of projects under the actual research work has been done. Finally, §1.4 describes the structure of this document.*

1.1 Research Context

Our research work is contextualized in the current trend of semantically enriching information published on the Web, in order to improve the user experience in tasks related to finding, extracting and combining that information. Concerning Web Services, this has lead to the adoption of the so-called Semantic Web Services (SWSs), which apply Semantic Web technologies in order to automate common Web service tasks, such as discovery, ranking, composition and execution [85]. In the following we further introduce this context, focusing on SWS discovery, ranking and user preferences.

1.1.1 Semantic Web and Service Web

The current Web is aimed at providing information and services that are directly consumed by human beings. Although most of the content is stored in databases, the information is presented without the structural information found in databases. From a machine point of view, it is very difficult to process all this information, so the Web can not be automatically manipulated by computers [6].

The Semantic Web (SW) vision [14] constitutes an extension to current Web, where information has associated semantics that can be processed and *understood* by machines. Thus, using Semantic Web (SW) technologies, all this information can be automatically processed, extracting knowledge to feed algorithms, in order to allow autonomous interaction between computers, and to improve the cooperation between people and computers. Unfortunately, this vision has not yet succeeded because "[current web applications] have little ability to interact with heterogeneous data and information types" [77].

That original vision of the SW has been refocused in a Web of Data, which aims to use the Web as a large "traditional" database [71]. Linked Data (LD) is a successful initiative within the Web of Data that consists of a number of principles for publishing, connecting and querying data, relying on SW technologies [45], such as Resource Description Framework (RDF), RDF Schema (RDFS) and OWL ontology languages, and the SPARQL query language [6]. If the published data is freely available using an open license it is

commonly identified as Linked *Open* Data (LOD). Datasets that offer their information using LOD principles increased their number from 12 to 203 between 2007 and 2010 [45], and, currently, there exist 328 LOD datasets[1]. Furthermore, the triples that they offer also increased from 500 million to 26.9 billion in the same period [45].

Another development to the current static, syntactic Web that focuses on providing a more dynamic interaction is offered by Web Services technologies. According to the World Wide Web Consortium (W3C), "a Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [17].

Web Services (WSs) are the building blocks to provide a world of distributed computing on the Web, envisioning a Service Web where a large amount of stakeholders publish and consume services in order to dynamically access a concrete functionality taking benefit of Web infrastructures and Service Oriented Architectures. This scenario involves the use of mechanisms that allow to discover published services according to user requirements, and to rank those discovered services in terms of user preferences.

However, current WSs technologies have an important flaw: usage and integration of WSs needs to be performed manually. Thus, discovery and ranking mechanisms are supported by syntactical information descriptions, so these processes cannot take benefit of the SW. In order to realize that Service Web vision, there is a need for semantically aware descriptions and related technologies that allow for an automatic and flexible discovery and ranking of services.

1.1.2 Semantic Web Services

As stated before, current WSs technologies do not allow the automation of common processes such as discovery and ranking, which

[1] According to `http://thedatahub.org/group/lodcloud`; last accessed: Aug., 2012.

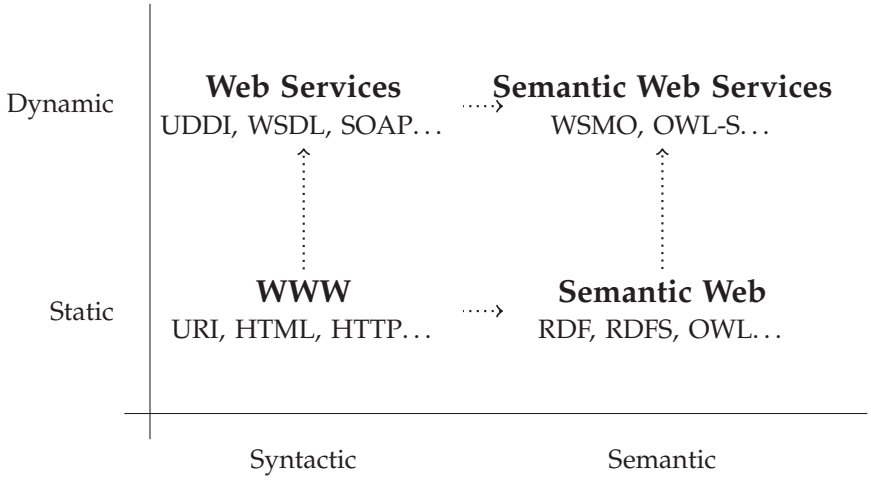


FIGURE 1.1: The Semantic Web Services vision.

are necessary to develop Service-Oriented Computing, among others like execution and composition. Furthermore, SW standards can be applied to markup information on the Web, adding semantics so machines are able to process and understand that information. In this scenario, a semantic markup of WSs definitions naturally comes up as the solution to perform automatic service discovery, execution, composition and interoperation [64]. Thus, a Semantic Web Service (SWS) can be simply defined as a WS whose description is in a language that has well-defined semantics [85].

This vision of joining together both WS and SW technologies to develop SWSs is usually shown as in Figure 1.1. We start from a static and syntactic World Wide Web (WWW), mainly focused on providing billions of information pages, where users have to interact directly with the Web to gather the required information. Because of the inherent characteristics of the current Web, this information is difficult to find, extract and interpret by computers, so the SW appears as a powerful solution, giving semantics to this static information. Furthermore, to improve new forms of interaction and to develop processes between computers connected to the Web, WSs technologies are used within this dynamic environment. Finally, SWSs transform the current Web from a static collection of infor-

mation into a distributed device of computation, using the SW as its foundation, so this information becomes processable and interpretable by a computer [19].

A SWS framework should concentrate on three key aspects, in order to enable such SWS vision. Firstly, it has to define exhaustive description frameworks for semantically describing WSs and related aspects. Secondly, it has to support ontologies to describe WSs, using them as its underlying data model, so machines are able to interpret all that information. Finally, a complete SWS framework has to define semantically driven technologies that support the automation of WS usage processes.

The most important proposals concerning SWS frameworks are the Web Service Modeling Ontology (WSMO) [73] and the OWL Ontology of Services (OWL-S) [62]. They support the previously presented aspects of a SWS framework, and provide tools to actually put into practice the SWS vision. Furthermore, there are other framework proposals that take other approaches, such as METEOR-S project that aims at extending current Web standards [68], and Semantic Web Services Framework (SWSF) [9], which is another W3C member submission to standardize SWS.

Recently, lightweight approaches have been being proposed in order to ease the adoption of SWS by simplifying semantic descriptions of services and user requests. Thus, lightweight service ontologies, such as WSMO-Lite [89], MicroWSMO [60], and the Minimal Service Model (MSM) [69], simplify the semantic annotation of web services, integrating these annotations within traditional, non-semantic service descriptions using either Semantic Annotations for WSDL and XML Schema (SAWSDL) [27] or Semantic Annotations for RESTful Services (SAREST) [78].

1.1.3 Discovery and Ranking

Once a service description using any SWS framework has been published, it is normally made available from a repository, where potential users fetch for desired services. This fetching involves two separate processes (sketched in Figure 1.2), that are referenced together as *service retrieval*, as well as service procurement [74]:

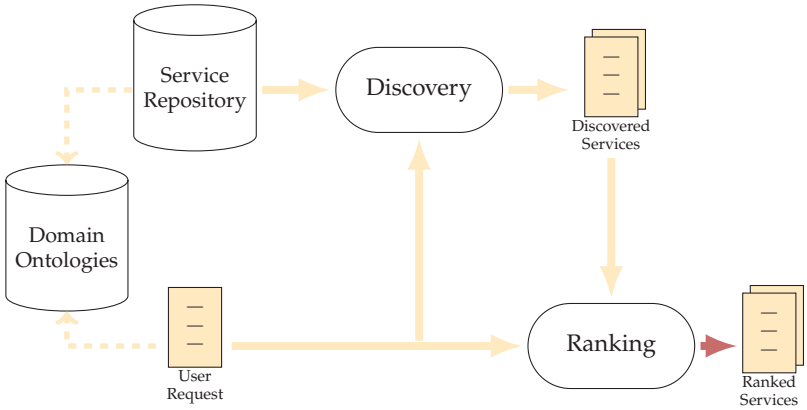


FIGURE 1.2: SWS retrieval activities.

1. **Discovery**, where candidate services, which fulfills the user requirements, are obtained from a repository.

2. **Ranking**, where the previous set of candidate services are ordered with regards to user preferences.

Therefore, a user request contains the semantic description of both requirements and preferences that are analyzed by discovery and ranking mechanisms, correspondingly. On the one hand, user requirements are considered as hard constraints that have to be met in order to consider a service published in the repository as a candidate. On the other hand, user preferences are often interpreted as soft constraints, whose degree of fulfillment determines to what extent one previously discovered service is preferred over another.

With respect to SWS discovery, McIlraith et al. [64] define it as the process of "automatically locating Web services that provide a particular service and that adhere to requested properties". This definition is very agnostic about the type of properties that the requester can use. Additionally, it does not consider the common scenario where discovery processes return not only one but a set of candidate services. The most common scenario in discovery process results in a search within the published services using functionality requirements to obtain a set of compatible services. SWS discov-

ery is usually performed by Description Logic reasoners, because semantic definitions are commonly based on this logic formalism.

The next step on SWS retrieval is to rank services in order to actually select the best one that fulfills the user request. In opposition to discovery, ranking processes focus on user preferences, which often state an order based on Quality of Service (QoS) or non-functional propertys (NFPs). Using these preferences, the set of discovered services are ranked so the best service can be chosen [76]. Besides, user preferences transform the ranking process into an optimization problem, so plain discovery techniques cannot be directly applied in this case, as they are mostly based on Description Logic formalisms [44].

1.1.4 Preferences

In the SWS retrieval scenario, user requests describe both requirements and preferences on services. Concerning requirements, they state the needed functionality a user is looking for, so they are linked to the discovery process, as it fetches candidate services from the repository that offer that requested functionality. In turn, preferences are interpreted solely by ranking mechanisms to order discovered services. While user requirements are usually directly supported by SWS frameworks, so discovery proposals rely on their models, preference models are often defined by concrete ranking mechanisms using them.

User preferences define the optimality criterion that is applied when selecting a service among a set of candidates. These preferences usually refer to QoS properties relevant to the user [11], though a service provider can also define preferences on service properties and prospective users. In any case, preference descriptions are evaluated in order to rank candidate services previously discovered. This ranking process can be interpreted as an optimization problem that obtains the best service according to the stated preferences [32].

There are several formalisms that can be used to represent preferences. Thus, preferences modeled as utility functions have been widely used in economics [29, 49] and web systems [5, 34, 92]. Another formalism based on partial orders were proposed in database

systems field [22, 52]. The main difference between these two formalisms is that the former constitutes a quantitative approach while the latter is qualitative.

Although quantitative approaches are more general because most preference relations can be defined in terms of utility functions, there are some intuitive preferences that cannot be captured by these functions [22] (see also §5.1 for further discussion). On the other hand, qualitative approaches have higher expressiveness and are more intuitive and user-friendly, though they are not directly applicable to a SWS scenario because they do not take into account that properties may be expressed using different abstraction levels depending on the stakeholder.

Preference queries from database systems can be also applied to the SWS scenario with ease. Furthermore, utility functions and quantitative models are usually applied when defining preferences to rank SWS. Moreover, recommender systems are also supported by preferences that model information extracted from the user interaction to offer relevant recommendations.

1.2 Summary of Contributions

The main objective of our research work is to improve SWS discovery and ranking processes, focusing not only on conceptual aspects enabling user preferences modeling and interoperability, but also on implementation level improvements, regarding performance, scalability, and integrability of discovery and ranking mechanisms.

Concerning preference modeling, we propose a comprehensive model to define service requests including user preferences, independently of the underlying formalisms used for describing, discovering and ranking services. This model effectively decouples the conceptual definition of preferences from the discovery and ranking implementation to be used, providing users with a higher reusability, flexibility and expressiveness for their preference definitions.

Our proposed preference model serves as the foundations for the rest of our research work. Using this model as a vocabulary and an upper ontology to abstractly define both service descriptions and user requests, we designed a generic optimization framework that

analyzes service requests to automatically generate filters that are applied to service repositories before actually performing discovery and ranking processes. As a result, both processes are significantly improved, offering better performance and scalability at a small cost on precision and recall. Moreover, the abstract descriptions provided by our model enable the application of our optimizations to any available SWS framework.

Furthermore, the inherently better interoperability provided by our preference model enables the integration of different discovery and ranking mechanisms. Consequently, we devised an integrated architecture for SWS discovery and ranking that further improve the flexibility to define and evaluate user preferences, taking a hybrid approach that integrates corresponding ranking mechanisms. This architecture offers a unique, lightweight façade to the whole discovery and ranking process.

This dissertation presents our contributions on preference modeling, discovery optimization, and flexible, integrated ranking, tackling specifically identified challenges on those areas. Note that challenges on preference modeling (*C1* and *C2*) are closely related to those on service discovery performance (*C3*) and interoperable, integrated ranking (*C4* and *C5*), because preference models constitute the foundation for descriptions that drive SWS discovery and ranking processes [76].

(*C1*) *Expressiveness* of preference models

Challenge Description: Most of the current discovery and ranking mechanisms offer a restricted number of facilities to define user preferences, consequently constraining preference expressiveness. Moreover, current approaches mostly rely on quantitative preferences. Therefore, there is a need for a generic, highly expressive preference model that offers users a comprehensive set of facilities to define and combine both quantitative and qualitative preferences, which are more intuitive and user-friendly.

Contributions: We propose in [38] a highly expressive ontological model, based on a strict partial order interpretation of user preferences, that offers several facilities to define qualitative

and quantitative preferences and combine them. We validated our proposal using use case scenarios from different domains, including the most complex scenario from the SWS Challenge. An early approach that models quantitative preferences as utility functions was also explored in [30].

(*C2*) *Independence* of user request models from underlying discovery and ranking mechanisms

Challenge Description: Since each mechanism proposed in the literature provides its own inherent model, both discovery and ranking mechanisms are implemented taking into consideration that corresponding model. As a result, there is a high dependency between models and implementations, constraining the expressiveness provided by each proposal as it depends on the formalisms used by its discovery and ranking mechanisms. Moreover, neither preference models nor heterogeneous mechanisms can be integrated together in a hybrid solution because of the inherent coupling found in current proposals.

Contributions: Precisely, we developed a hybrid solution for SWS discovery and ranking that separates preference models from the actual mechanisms that rank services [33]. In this approach, we split user requests depending on the preference facilities used, and each part is routed to an appropriate discovery or ranking mechanism. Concerning ranking, we also presented an interpretation of this process as an optimization problem, effectively decoupling preference modeling from the ranking algorithm by introducing transformations from preference statements to a Constraint Satisfaction Optimization Problem (CSOP) [32].

(*C3*) *Performance* of service discovery process

Challenge Description: Most proposals on SWS discovery do not scale well, because they are based on inefficient logic formalisms [44]. As a consequence, performance issues show up in several scenarios. There is a need for optimized techniques that allow a more performing discovery process, and some proposals are emerging in this field, providing caching, query

rewriting, query execution planning or repository optimizations, among other solutions. However, they are usually coupled with a particular discovery mechanism and/or a preference model, making difficult to apply this techniques in different scenarios.

Contributions: In [36] we discussed the requirements that any query-based discovery mechanism should fulfill in order to offer an optimized, decoupled experience to the user. Taking decoupling requirements into account, we developed a filtering approach that analyzes user requests in order to minimize the amount of data from the SWS repository to be accessed by any discovery mechanism, improving the scalability and performance of the process, regardless of the actual implementation used to discover services [41].

(*C4*) *Interoperability* between discovery and ranking mechanisms

Challenge Description: In a typical scenario, users are presented with a restricted set of facilities to define their preferences, depending on the concrete discovery and ranking mechanisms to be used. Due to the existing inter-dependence, if a user wants to define their preference using other facilities, corresponding mechanisms have to be used instead of the original ones. However, interoperability issues may arise when users need to combine preference facilities from different mechanisms based on heterogeneous underlying formalisms. Consequently, an upper, common preference model is necessary in order to allow users to flexibly choose different mechanisms depending on their particular needs – *e.g.* some users may find highly expressive mechanisms more useful despite their low performance, but others may prefer a more performing solution.

Contributions: We proposed an extension to a logic rule based ranking mechanism [87] that extended their preference model allowing the definition of utility functions in addition to tendencies [34, 35]. This proposal provided users with a higher interoperability and flexibility, as they could choose between

both facilities, without knowing the underlying ranking mechanism that needs to be executed [37].

(C5) *Integrability* and adaptability of different mechanisms

Challenge Description: Once interoperability issues at conceptual level are solved by the introduction of an expressive, common preference model, mechanisms should be also integrated at implementation level. In order to combine preferences from different ranking mechanisms, they have to be integrated in a seamless, efficient way, so that a single entry point for the hybrid ranking process is presented to the user.

Contributions: An early approach on integrating discovery and ranking mechanisms using a hybrid architecture is presented in [31]. Using that hybrid architecture, we successfully designed a solution to integrate several ranking mechanisms, that were validated in the context of EU FP7 SOA4All project [40].

1.3 THESIS CONTEXT

This thesis work has been developed in the context of the Applied Software Engineering (Ingeniería del Software Aplicada – ISA) research group[2] at the University of Seville. Starting from the deep background on Service Oriented Computing of the research group, we began a research line on Semantic Web Services, focusing on discovery and ranking processes. Concretely, our work has been developed during our participation in the following research projects and networks:

- WEB-FACTORIES: Fábricas Software para Sistemas con Arquitectura Orientada a Servicios Web. CICYT project referenced as TIN 2006-00472. We started our thesis during this project, developing current approaches and applying semantics to classical discovery and ranking processes.

- ISABEL: Ingeniería de Sistemas Abiertos Basada en LínEas de productos. Excellence project of the Andalusian Government,

[2] http://www.isa.us.es

referenced as TIC-2533. Our participation in this project allowed us to obtain our first contributions on preference modeling and integrated discovery and ranking architectures.

- SETI: reSearching on intElligent Tools for the Internet of services. CYCIT project referenced as TIN2009-07366. In this project we focused on preference modeling and improvement of current discovery techniques by filtering repositories.
- SOA4All: Service-Oriented Architectures For All. EU FP7 IST project, referenced as 27867. Our participation in the latter stage of this project led to a complete validation scenario for our thesis work, especially on preference modeling and integrated ranking.
- S-CUBE: Software Services And Systems Network of Excellence. Funded by the EU FP7, referenced as 215483. This project allowed us to validate and discuss our proposals with the research community focused on our topics.
- THEOS: Tecnologías Habilitadoras para EcOsistemas Software. Project funded by the Andalusian Government referenced as TIC-5906. Finally, in this ongoing project we started to apply our developed proposals to different domains and scenarios.

1.4 Structure of this Dissertation

This dissertation is divided in four parts that describe our thesis work using the following structure:

Part I: Introduction and Background introduces the research background of our thesis work. Chapter 1 describe the research context and summarizes our contributions with respect to several identified challenges in that context. Then, Chapter 2 analyzes the related work in preference modeling, discovery and ranking, comparing different proposals with respect to the challenges.

Part II: Improving SWS Discovery and Ranking presents all three contributions developed during our thesis work. Chapter 3

showcases SOUP, a highly expressive preference model that serves as the foundations for the rest of our proposal. Thus, Chapter 4 discusses EMMA, which is a filtering approach to improve SWS discovery processes, while Chapter 5 describes our PURI framework, which provides an infrastructure to integrate different ranking mechanisms using our common preference model.

Part III: Evaluation of Results thoroughly analyzes and evaluate our contributions. First, we validate our preference model using use case scenarios in Chapter 6. Second, a comprehensive evaluation of EMMA is discussed in Chapter 7, applying our filtering approach to various SWS matchmakers. Finally, Chapter 8 presents the application of PURI framework to a concrete use case scenario, showing the benefits of an integrated ranking solution.

Part IV: Final Remarks concludes our dissertation in Chapter 9 by summing up our contribution with respect to the degree of fulfillment of the identified challenges, also including future work that has been identified to continue our research work from the obtained results.

Additionally, we include three appendices that complement this dissertation: specifying the thesis work performed in the context of the EU project SOA4All (Appendix A); showcasing complete evaluation results of EMMA (Appendix B); and discussing an additional evaluation of EMMA applied to a WSMO scenario (Appendix C).

BACKGROUND

Nothing has such power to broaden the mind as the ability to investigate systematically and truly all that comes under thy observation in life.

Marcus Aurelius (121-180)
Roman emperor

After introducing the research context of this dissertation and the concrete challenges that we identified, this chapter describes the state-of-the-art on preference modeling for the discovery and ranking processes (§2.2), optimizations to improve service discovery performance (§2.3), and integration approaches for discovery and ranking mechanisms (§2.4). We analyze current proposals on those subjects in terms of a comparison framework defined in §2.1 which is based on the challenges identified in §1.2. As summarized in §2.5, the conclusions obtained from this literature review serve the purpose of further supporting our thesis and motivating our research contributions.

2.1 INTRODUCTION

During the development of our thesis, we have identified a number of issues that have not been successfully and completely addressed yet. In §1.2 we equate those issues with five challenges ahead on SWS discovery and ranking. In order to analyze to what extent those challenges are actually present in state-of-the-art proposals, we devise a comparison framework that defines a normalized scale to measure each challenge separately. As using quantitative measures for each challenge is out of the scope of this comparative study, we cluster current proposals using three levels of fulfillment (low, medium and high), presenting a qualitative comparison that can be analyzed jointly with other challenges.

Using this comparison framework, we have systematically analyzed several proposals on discovery and ranking. The conclusions of our literature review are discussed in the following research, where each section groups works depending on whether their main contribution is aligned with preference modeling, discovery optimization, or integrated ranking challenges, discussing our interpretation for each corresponding level. However, some proposals present approaches that may address challenges from more than one topic. Consequently, they may be analyzed in corresponding sections, though they are only presented in one of the sections, for the sake of clarity.

2.2 PREFERENCE MODELING

As discussed in §1.1.4, user requests containing preference definitions are necessary to perform SWS discovery and, especially, ranking. However, early approaches on SWS discovery and ranking, as in Benatallah et al. [13], Li and Horrocks [59], or Sycara et al. [86], do not rely on a separate preference model, but on functionality-based descriptions that specify user requirements as desired features of service offers, described using a SWS framework, such as OWL-S or WSMO. In consequence, ranking is performed using matching degrees obtained from a logics-based discovery, limiting both performance (*cf.* §2.3) and the available facilities to define preferences.

An extension to DAML Ontology of Services (DAML-S) to include QoS profiles is proposed by Zhou et al. [93]. Their extension is divided in three layers with different roles. Thus, the QoS profile layer provides support for the different roles of the provisioning process: service provider, user requests (*inquiries*), and templates. This layer provides three common superclasses for matchmaking. The constraints that can be defined in each profile can be described by properties definitions and cardinality. The QoS property definition layer allows to specify the domain of the QoS properties, *i.e.* core, input, output, preconditions and effects. Finally, the QoS metrics layer defines how each QoS property is measured, and who is the organization that guarantees that measurement. Zhou et al. provide a basic profile with commonly used QoS properties defined, such as cost, response time, reliability and throughput. Their proposal only allows order constraint between QoS parameters, so it performs discovery and ranking using Description Logics and matching degrees.

Another DARPA Agent Markup Language (DAML)-based proposal is also presented by Bilgin and Singh [15], where they provide a DAML-based query language, instead of just extending OWL-S. Using this language, they advertise QoS attributes and perform the ranking. They provide a simple ontology for service categories and QoS attributes associated to these categories. The main drawback of this approach is the limitation on the expressiveness of queries, due to the use of DAML as its foundation. Thus, user preferences cannot be expressed in those queries, and are inherent to their ranking algorithm, as in [93].

Maximilien and Singh [63] present a framework and a QoS ontology for dynamic selection. They provide a complete, agent-based architecture to perform service ranking, that uses a layered QoS ontology, including a QoS upper ontology, which defines basic concepts associated with a QoS parameter, such as measurement, relationships, and aggregate support. Using that layer, the QoS middle ontology defines the most frequent QoS parameters and metrics encountered in distributed systems, such as availability, cost, performance, etc. Finally, a user-defined lower ontology defines concepts from the domain of each service. This framework constitutes a very comprehensive solution to describe QoS in SWS, and it is referenced by other authors [25, 57, 67]. However, user preferences based on

QoS parameters from their ontology have to be defined externally to the ontology itself, in a QoS policy description that the agent-based architecture applies to perform the ranking process.

Another extension to OWL-S is QoSOnt, proposed by Dobson et al. [25], which is an ontology that describes QoS attributes and metrics. Their approach is similar to the one from Maximilien and Singh, separating QoSOnt ontology in different smaller ontologies. Thus, the base ontology contains basic QoS concepts, such as *Attribute* and *Metric*. Then, concepts specific to some attribute can be built into separate ontologies on top of the base concepts. For instance, a separate ontology for *availability* can be specified, which contains concepts and metrics that can be only applied to *availability* specification. Additionally, generic metrics concepts can be defined in another separate ontology, for reuse. These metrics also define the user preferences applied to them, using the *acceptability* direction, that is the preferred tendency of metric values (*e.g.* the higher the best). However, Dobson et al. do not explicitly show how to perform selection, and their proposal suffers from Ontology Web Language (OWL) limitations, so they have to use an *ad hoc* eXtensible Markup Language (XML) language to allow custom data ranges.

A different semantic framework for service discovery is presented by Pathak et al. [67], which models mappings between ontologies. They propose to use domain specific ontologies to define QoS properties among user preferences and service descriptions. In their work, ranking is done using matching degrees at a first stage, such as the degrees defined in [59]. Then, values from QoS parameters are collected in a *quality matrix* and normalized, in order to calculate a fixed, weighted utility function for each candidate service. Then, candidates whose utility function is above a given threshold, are passed to the final step of the process, where they are ranked by one QoS parameter that is applied to a ranking function in order to obtain the optimal service with regard to user preferences. These user preferences are specified in two different stages: (1) weights for each QoS parameter used in the computation of the utility function, and (2) ranking attribute and ranking function used at the final stage.

Wang et al. [91] also define a QoS-aware ranking model for SWS that makes use of QoS matrices. They provide an extension to WSMO ontology that enables handling QoS parameters. Within

this extension, users may define expressions that provide the actual value of a measured QoS parameter, and also the associated preference. This preference is defined in terms of relative weights and preferred tendency, namely smaller, larger or closer to a given value. Wang et al. define a QoS preference model and an algorithm based on a QoS matrix that contains values of QoS parameters. Before, in a previous step, discovery is performed by matching functional properties. The ranking algorithm first normalize matrix values in order to homogenize them. Then, a uniformity analysis is done applying different formulas depending on the tendency preference of each QoS parameter. Finally, for each service (represented by a row in the normalized matrix), the evaluation result can be computed by adding the resulting values for each parameter applying the corresponding weights.

Siberski et al. [79] propose an extension to SPARQL so that preferences are described directly using the query language, without being based on existing preferences and non-functional properties ontologies, as in other semantic ranking approaches [34, 38, 91]. They provide a `PREFERRING` clause that states preferences among values of variables, similar to `FILTER` expressions. The preference model is based on a qualitative approach that define preferences as partial orders [52]. However, this approach does not have the flexibility and reasoning facilities that provides a solution based on an external ontology, and it uses non-standard SPARQL extensions without providing an implementation.

Other proposals to model preferences on SWS are more focused on ranking mechanisms, so their preference model are specifically tailored towards their implementations. Toma et al. [87] presents a multi-criteria approach based on logic rules, modeling preferences by including simple annotations to WSMO goals. These annotations can define weights and tendencies as in [91], using a logic programming rules approach to evaluate QoS parameters and rank services.

In turn, Lamparter et al. [58] provide a more complex ontology to represent service offers and requests that conforms the foundations for a discovery and ranking process performed using rules in SWRL [46] and SPARQL queries. These queries include predicates that have to be evaluated at run-time, so they include an extension to SPARQL that is implemented using different proposed algorithms.

TABLE 2.1: Preference modeling in discussed proposals.

Proposal	(*C1*)	(*C2*)
Early approaches (2003-04) [13, 59, 86]	Low	Low
Zhou et al. (2004) [93]	Low	Low
Bilgin and Singh (2004) [15]	Low	Low
Maximilien and Singh (2004) [63]	Low	Medium
Dobson et al. (2005) [25]	Low	Medium
Pathak et al. (2005) [67]	Low	Low
Wang et al. (2006) [91]	Medium	Low
Siberski et al. (2006) [79]	High	Low
Toma et al. (2007) [87]	Medium	Low
Lamparter et al. (2007) [58]	High	Medium
Palmonari et al. (2009) [66]	High	Medium

Thus, a query for a user request is provided, though this query depends on rules that change the matchmaking policy, *e.g.* allowing matching degrees as in [86]. The provided preference model is based on utility functions that are evaluated within query execution.

Finally, Palmonari et al. [66] propose the definition of policies, though they provide more facilities to express relative weights and different offered policies. Thus, each service may offer several policies that can be requested depending on the user preferences. The ranking process is performed using a hybrid approach that evaluates properties and then combine them for each defined policy. The extensibility and generality of the allowed policies offer a highly expressive preference model.

In Table 2.1 discussed proposals are summarized in terms of both expressiveness (*C1*) and independence (*C2*) challenges identified in §1.2. The considered levels of fulfillment to compare each proposals are interpreted as:

(*C1*) Expressiveness. Each proposal offers a number of facilities to define preferences, providing a concrete degree of expressiveness that depends on the complexity and completeness of the underlying preference model:

Low: Approaches that do not provide facilities to let users de-

fine their own preferences are considered to provide a low expressiveness.

Medium: We consider a proposal to offer a medium expressiveness if it provides some simple facilities to define preferences, such as tendencies or weights.

High: In turn, a highly expressive solution allows users to define preferences using and possibly combining complex facilities, as utility functions or fuzzy rules.

(*C*2) Independence. In order to evaluate this challenge, we measure the independence degree between discovery and ranking mechanisms and their corresponding preference model, using the following scale:

Low: An implementation with low independence means that the preference model is tightly associated with the underlying mechanisms used to perform discovery and ranking, resulting in a system that cannot be integrated, adapted or extended to support other preference facilities.

Medium: When proposals offer separate and/or external ontological models to define preferences, they are considered to offer a medium independence, as those models only include facilities that can be managed by the corresponding implementation.

High: If the preference model is generic enough and not tied to the underlying formalism used to effectively discover and rank services, the corresponding proposal may present a high independence.

On the one hand, classical approaches do not tend to allow the definition of user preferences at all [13, 59, 86]. Other approaches only focus on QoS modeling but do not offer facilities to define preference over those QoS attributes [15, 93], or rely on externally user-defined ontologies [25, 63]. More recent discovery and ranking solutions offer a wide variety of preference models, increasing the expressiveness from weight and tendencies [87, 91] to utility functions [58]. However, only two of them offer both qualitative and

quantitative preference constructs that allow for a higher expressiveness [66, 79].

On the other hand, *C2* analysis shows that preference models often present a low independence degree with the associated discovery and / or ranking mechanism, making difficult to reuse these models, as they are defined in an *ad hoc* manner depending on the underlying mechanism. Nevertheless, models defined using external, extensible ontologies offer a medium degree of independence that ease their reuse in other scenarios.

2.3 Service Discovery Optimization

In order to improve discovery engines, Stollberg et al. [84] provides a caching mechanism that reduces the search space and minimizes matchmaking operations. The proposed cache uses a graph that stores relationships between user requests described as WSMO goal templates, and their related services. Thus, goal instances are compared with cached templates in terms of semantic similarity, and if there is a match, only the related services stored in the graph are used for the subsequent discovery.

Carenini et al. [20] propose a customizable hybrid architecture for SWS discovery and ranking named GLUE2. GLUE2 offers a set of specialized discovery components, such as functional discovery, dynamic discovery, non-functional discovery, and ranking, among other additional components [66]. Based on WSMO, GLUE2 enables the configuration of the discovery workflow on a case by case basis, optimizing each scenario using the most appropriate components.

An hybrid matchmaker called iMatcher that uses information retrieval techniques to improve the discovery process is presented by Kiefer and Bernstein [50]. In this proposal, authors use a SPARQL extension (iSPARQL [51]) that enables the introduction of similarity operators into query elements. Thus, different similarity strategies are combined with logic-based discovery in order to improve precision and recall of the matchmaking process. Additionally, machine-learning can also be applied to choose the most appropriate strategy to be included in the hybrid matchmaking, for each case.

Concerning the need for an improved discovery process which

TABLE 2.2: Performance of discovery approaches.

Proposal	(*C3*)
Pref. model. approaches (§2.2)	Low
Stollberg et al. (2007) [84]	Medium
Carenini et al. (2008) [20]	Medium
Kiefer and Bernstein (2008) [50]	High
Agarwal et al. (2009) [3]	Medium
Klusch et al. (2009) [54]	Medium
Sbodio et al. (2010) [75]	High

tackles scalability issues, Agarwal et al. [3] discuss a hybrid approach that uses different discovery mechanisms together, in order to improve discovery performance. They also propose a simple filtering stage based on an efficient classification-based discovery. However, this filter rely on a less expressive user request. Thus, the preference model also suffers from the issues identified in previous sections.

A different approach is taken by Klusch et al. [54] in OWLS-MX, where they present a hybrid matchmaker that combines information retrieval techniques, such as syntactic similarity, with classical DL-based discovery, in order to improve OWL-S service matchmaking. Their comprehensive evaluation proves that hybrid approaches present a better performance than classical ones. Moreover, similar solutions have been also proposed by the authors for WSMO [53] and SAWSDL [55] service matchmaking.

Finally, Sbodio et al. [75] introduce SPARQL queries to describe OWL-S service pre- and post-conditions, and user requests, providing a matchmaker implementation based on agents called SPARQ-Lent. They discuss a complete discovery solution that uses SPARQL queries to modify and ask the agent's knowledge base, evaluating their proposal against OWLS-MX using Semantic Web Service Matchmaking Evaluation Environment (SME^2). They provide some optimizations to their discovery algorithm, that performs better than some OWLS-MX variants.

The three levels defined in our comparison framework to compare proposals focused on service discovery optimization are the

following:

(*C3*) Performance. Concerning the optimization of service discovery, we compare the scalability and performance of each proposal, ranking them in the following terms:

Low: Proposals offering a low performance do not scale well in general, mainly because they only use logic-based techniques to perform discovery.

Medium: Most approaches try to improve performance by using different strategies, such as caching, filtering, or query rewriting, presenting better results. Consequently, we rate these solutions as a medium performance, as they focus on a single strategy.

High: Higher performing discovery techniques usually apply a mixture of strategies that provides a better user experience in this scenario.

As shown in Table 2.2, discovery and ranking approaches presented in the previous Section suffer from scalability and performance issues (*C3*), mainly because of the logics-based and *ad hoc* formalisms used. However, proposals discussed in this Section provide some sort of optimization techniques that improve performance considerably, though they may still present scalability issues. The most common approach to optimize discovery and ranking is to use hybrid solutions that can take benefit from several mechanisms at the same time. Particularly, [50] offers an intelligent hybrid matchmaking that is able to improve the process considerably by choosing the most appropriate techniques for each case.

Note that these proposals cannot be analyzed with regard to §2.2 challenges, because they are mostly focused on SWS discovery and do not provide separate preference models. However, ranking mechanisms can be easily incorporated into the process in hybrid approaches, as in the case of [20] that is actually integrated with Palmonari et al. [66] ranking proposal.

2.4 Interoperable and Integrated Ranking

Concerning hybrid proposals, a semantically-enabled matching using CSOPs is presented by Kritikos and Plexousakis [57], based on [74]. They propose an ontology similar to the proposed by Maximilien and Singh [63], mixing requests and QoS-aware service descriptions within OWL-S. Moreover, they present a matching algorithm to infer equivalences between differently named QoS parameters that are semantically equivalent, although it is generally undecidable. Their extension ontology, OWLQ, is separated in several facets that concentrates on a particular part of their QoS WS description, such as connection with OWL-S instances, the actual QoS descriptions, metrics facets that provide classes to formally define QoS metrics, units, etc. They use CSOPs to perform the matchmaking of compatible provided services, and then select the best service by means of a weighted composition of utility functions, which balance the worst and best scenarios to compute the utility value.

Another approach that merges discovery and ranking algorithms execution is presented by Vu et al. [90]. They provide an extension to WSMO ontology in order to support QoS properties. Their extension model is based on axioms from the underlying WSMO ontology. They show a QoS-aware discovery framework that takes QoS values of WSs based on user feedback and perform the discovery process, ranking the services in terms of QoS compliance. Actually, the ranking of services is obtained with regard to user preferences, defined by relative weights. Additionally, they sketch both a centralized architecture and a scalable one, that can be deployed into a peer-to-peer network.

Concerning the integration of SWS frameworks, there are some proposals in the literature that address this issue to tackle interoperability. Chabeb et al. [21] discuss a systematic approach to generate mappings between OWL-S, WSMO and plain Web Service Description Language (WSDL) services, matching concepts from the different SWS ontologies using similarity techniques that validate the inferred correspondences. Their resulting ontology merges concepts from different SWS frameworks, as opposite to the following two approaches, that only capture part of those SWS ontologies, offering a more concise approach.

TABLE 2.3: Interoperability and integration analysis.

Proposal	(*C4*)	(*C5*)
Pref. model. approaches (§2.2)	Low / Med	Low / Med
Disco. optim. approaches (§2.3)	Medium	Med / High
Kritikos and Plexousakis (2006) [57]	Medium	Low
Vu et al. (2006) [90]	Medium	Medium
Chabeb et al. (2009) [21]	High	Low
Norton et al. (2010) [65]	High	Low
Pedrinaci and Domingue (2010) [69]	High	Medium

Thus, Norton et al. [65] present a similar proposal, where the authors take a 'union' approach to integrate OWL-S, WSMO, and WSMO-Lite descriptions. They present several SPARQL `CONSTRUCT` queries that transform SWS descriptions to and from the Semantic SOA Reference Ontology, a standard proposed by OASIS.

Another ontological model of integration is proposed by Pedrinaci and Domingue [69], who present a service repository called iServe that exposes service descriptions as linked data in terms of a Minimal Service Model (MSM). This model integrates not only OWL-S, WSMO, SAWSDL and WSMO-Lite services, but also MicroWSMO [60] or SAREST [78] descriptions of Web APIs. However, its minimal nature constraints the expressiveness of service definitions and user preferences, but provides a lightweight solution to discover services using SPARQL endpoints.

Table 2.3 sums up how the previously discussed proposals cope with interoperability (*C4*) and integrability (*C5*) research challenges, where each level of fulfillment corresponds with the following interpretation:

(*C4*) Interoperability. At conceptual level, we analyzed to what degree proposals provide interoperable preference models, so that users can flexibly choose and combine facilities from different ranking mechanisms depending on their needs.

Low: Proposals offering a low interoperability do not provide a separate semantic model to define user preferences, so they

cannot be combined and reused with models proposed by the rest of the proposals.

Medium: Most analyzed approaches present a medium interoperability, as they semantically define their models using separate ontologies or extensions to existing ones, enabling their interoperability with other proposals.

High: Solutions that are designed to be highly interoperable provide several preference constructs that can be extended and composed together with external preference models.

(*C5*) Integrability. Finally, integrability measures to what extent discovery and ranking mechanisms can be easily integrated, adapted or extended at implementation level, *e.g.* using their interfaces and provided hooks.

Low: Proprietary, difficult to extend proposals are rated to have a low integrability, because their underlying ranking mechanisms are very different, particularized for a concrete scenario.

Medium: Proposals that provide components that are easier to integrate, because, for instance, they offer hybrid architectures or SPARQL endpoints are rated at a medium degree of integrability.

High: Hybrid architectures that can be also extended with new mechanisms to customize discovery and ranking processes are considered highly integrable.

Proposals already presented in §2.2 offer a low to medium degree for both challenges. On the one hand, early approaches [13, 59, 86] do not provide a separate semantic model to define user preferences, so they cannot be combined and reused with models proposed by the rest of the proposals, producing a low interoperability and a corresponding low integrability. However, other proposals are rated at a medium interoperability, as they semantically define external models that can be easier extended and adapted,

In turn, discovery optimization proposals discussed in §2.3 present a medium to high rating for both *C4* and *C5*. Most proposals allows for customization and adaptation of their mechanisms

to different SWS frameworks and use case scenarios. Furthermore, some proposed architectures provide a higher integrability because of their hybrid nature, that allows to configure different components to work together in a discovery and ranking scenario.

Integration solutions described in this section offer a high grade of interoperability (*C4*) because they are highly interoperable by design [21, 65, 69]. Nevertheless, [57, 90] provide external ontologies that can be adapted to different scenarios, offering a medium interoperability in comparison. With respect to integrability (*C5*), only [69, 90] present some facilities to integrate them using different architectures, obtaining a medium rank for this challenge.

2.5 SUMMARY

From the results of our literature review, we obtain several conclusions that motivate our thesis work presented in this dissertation. Concerning preference modeling, though there are some proposals that provide highly expressive facilities to define preferences, they all present low independence between models and implementations, making difficult to extend or adapt their models to other scenarios. Therefore, there is a need for a highly expressive and generic preference model that has to be designed independently from underlying discovery and ranking mechanisms to be used to evaluate those preferences. Chapter 3 discuss our proposal on this subject.

In turn, current service discovery optimization techniques offer fair solutions to improve the perceived performance of this process. However they are designed to work in particular scenarios using a specific discovery mechanism. In Chapter 4 we present a discovery optimization technique that can be integrated into every discovery and ranking scenario, because it is independent from the applied mechanisms. Moreover, our proposal can be also applied on top of other optimized discovery proposals, such as the analyzed ones.

Finally, a flexible discovery and ranking solution is not supported by current proposals, because their models are not completely interoperable and consequently their implementations cannot be easily integrated in order to offer the flexibility that power users need. Moreover, hybrid architectures offer the highest integra-

bility among current proposals. In order to provide a highly interoperable discovery and ranking solution that can combine several preference facilities that may be evaluated by different mechanisms, we propose in Chapter 5 an integrated ranking solution based on our preference model that takes an hybrid approach to seamlessly integrate ranking mechanisms.

Part II

Improving Semantic Web Service Discovery and Ranking

A Preference Model for Semantic Web Service Discovery and Ranking

When we program a computer to make choices intelligently after determining its options, examining their consequences, and deciding which is most favorable or most moral or whatever, we must program it to take an attitude towards its freedom of choice essentially isomorphic to that which a human must take to his own.

John McCarthy (1927–2011)
American computer scientist

Preference modeling constitutes an essential component for the execution of discovery and, especially, ranking mechanisms, providing facilities to define user requests and preferences. In this chapter we describe our proposed preference model, introducing in §3.1 our motivation and thesis. §3.2 presents an abstract upper ontology to define both services and user requests, that serves as the common model for the rest of our contributions. Then, §3.3 further describes our preference model and its facilities to define preferences within a user request. We developed a concrete application to WSMO framework as shown in §3.4. Finally, we sum up the main characteristics of our solution to model preferences for discovery and ranking in §3.5, discussing its fulfillment degree with respect to our identified challenges on preference modeling.

3.1 INTRODUCTION

SWS definition frameworks provide comprehensive tools to describe services and their interactions. Although they offer facilities to also state user requests, preferences cannot be described at the same detail level, *i.e.* users cannot define complex desires for a concrete service request. For instance, WSMO user requests, denoted by goals [73], only support the description of requirements about a request in the form of capabilities and interfaces. In turn, preferences to rank services fulfilling those requirements cannot be directly expressed by using a standard WSMO goal definition, which only provides means to define non-functional properties / values pairs. In other words, preferences are not considered first-class citizens in WSMO, in comparison to service capabilities, whose definitions are more expressive. Other frameworks, such as OWL-S [61] or SAWSDL [27], do not even define a specific model to describe user requests at all.

Discovery and ranking proposals try to fill this gap, extending SWS frameworks to support preferences definition [91, 93], or just providing separate user preferences descriptions [58, 63], using different formalisms as discussed in §2.2. Consequently, these formalisms actually determine the level of expressiveness of each proposal (*C1*), while resulting in a high dependence between user preferences definition and its corresponding discovery and ranking implementations (*C2*).

In order to overcome these identified challenges in current proposals, we present a Semantic Ontology of User Preferences (SOUP), which is a highly expressive, intuitive model of user preferences. This proposal adapts a well-known model designed for database systems [52] that allows to define preferences constructively and user-friendly. Starting from an abstract model that defines both service, user requests and preferences description at the same semantic level, next sections describe SOUP in detail, also introducing elements that conform the foundations to our proposals on improving discovery (Chapter 4) and ranking integration (Chapter 5). Furthermore, we describe our model application to WSMO definitions, extending its *goal* element in order to allow the specification of preferences using SOUP. Additionally, Chapter 6 presents our proposal validation that consists on the complete definition of a discovery

scenario from the SWS Challenge[1]. Particularly, we validated SOUP using the Logistics Management scenario, that contains several service descriptions and user requests contextualized in a transportation and logistics domain. In the following we use concepts from these domains to illustrate the different facilities provided by our model to define preferences.

3.2 An Abstract Upper Ontology of Services

As discussed before, service descriptions, user requests and preferences should be semantically described at the same detail level. Therefore, there is a need for the definition of an ontological model that leverages preference descriptions as first-class citizens in the discovery and ranking scenario. Moreover, this model has to provide intuitive and user-friendly facilities to easily define both requirements and preferences, so that service descriptions can be matched with user requests. Furthermore, these facilities have to conform a sufficiently expressive model so that a user can fully describe any preference, without being limited by a concrete formalism or representation.

In order to specify a preference model, firstly we need to establish a clear separation between requirements that have to be met, and preferences that have to be taken into account once requirements have been fulfilled. Typically, requirements are hard constraints that are used to filter service repositories in the discovery process, while preferences are used to rank previously discovered services so that the most appropriate service can be selected after the ranking process. Therefore, preferences define a strict partial order in our model, providing a framework to compare and rank a set of services.

Figure 3.1 shows the upper ontology of SOUP, which is represented using a UML-based notation for OWL ontologies [18] that we also use throughout the rest of this dissertation. `UserRequest` and `ServiceDescription` are the root concepts in our proposal. On the one hand, a `ServiceDescription` describes features provided by the service itself, using the `hasFeature` object property to link

[1] `http://sws-challenge.org`

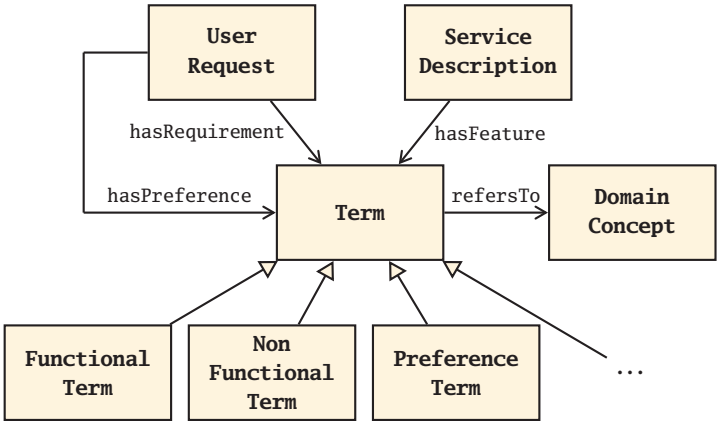


FIGURE 3.1: Upper ontology of services.

corresponding terms about functionality, NFP, input and output parameters, among others. Listing 3.1 shows an excerpt of an abstract service description from the logistics scenario using our upper ontology, where some of the functional and NFP terms of service `:ws1` are defined. Its graphical representation is also depicted in Figure 3.2, where namespaces are omitted for the sake of clarity.

LISTING 3.1: Example of an abstract service description.

```
:ws1 a soup:ServiceDescription;
  soup:hasFeature :transOrder, :basePrice. # among others...

:transOrder a soup:FunctionalTerm;
  soup:refersTo logistics:TransportOrder.
:basePrice a soup:NonFunctionalTerm;
  soup:refersTo logistics:BasePrice.
```

On the other hand, a `UserRequest` is the materialization of user desires with respect to a particular service request. These desires are described using requirements and preference terms, which are linked with the particular `UserRequest` instance using respectively `hasRequirement` and `hasPreference` object properties. Terms related with requirements state hard constraints that have to be fulfilled in order to consider a certain service as a matching candidate with respect to the user request. For instance, users searching for

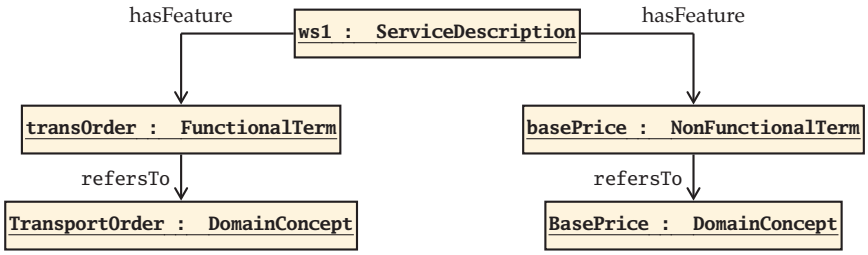


FIGURE 3.2: Graphical representation of the abstract service.

services usually interpret functionality, service classification terms, input and output parameters, among others, as requirements on their desired service. In turn, preferences can be considered as soft constraints whose degree of fulfillment determine to what extent a candidate service is preferred against other candidate services that also fulfill the user requirements. In other words, ranking mechanisms evaluate preference terms in order to obtain the best candidate service with respect to a user request.

An example of a user request `:goalD1` defined within the SWS Challenge logistics scenario is showcased in Listing 3.2, along with its graphical representation in Figure 3.3. This request comprises a complex functional requirement term, which may contain pickup and delivery time among other information regarding the transportation of clothes, and a preference term referring to the base price. Precisely, the concrete preference term that should be used when instantiating this request (a `LowestPreference` as defined in §3.3.2) is discussed in Chapter 6.

LISTING 3.2: Example of an abstract user request.

```
1 :goalD1 a soup:UserRequest;
2   soup:hasRequirement :clothesOrder;
3   soup:hasPreference :lowestPrice.
4
5 :clothesOrder a soup:FunctionalTerm;
6   soup:refersTo logistics:TransportOrder.
7 :lowestPrice a soup:PreferenceTerm;
8   soup:refersTo logistics:BasePrice.
```

Both requirements and preferences are related with one or more

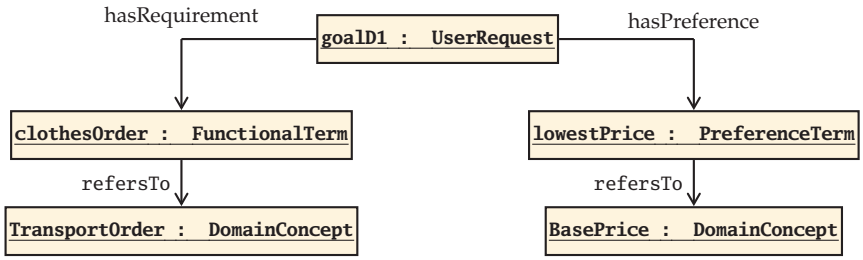


FIGURE 3.3: Graphical representation of the abstract user request.

`DomainConcept` classes, which are referred inside each term, and explicitly stated using the `refersTo` object property. Domain concepts usually represent service properties related to the domain-specific ontology used for service description, such as functional classification, input and output parameters types, process description, behavioral parameters, and non-functional properties, with the latter being specially important for preference terms definition. The above examples contains some logistics concepts such as a transport order and the base price for shipping.

Both functional and non-functional requirements specification has been widely discussed in the literature [74], and SWS frameworks provide sufficiently expressive facilities to define them, so in the following we will focus on preference modeling. Moreover, the validation scenario described in Chapter 6 consists on a series of user requests whose requirement terms can be simply considered as property/property value pairs, so it is not necessary to define a complex hierarchy of functional terms in order to validate the upper ontology. Nevertheless, concrete applications may extend our upper ontology adding specialized terms in order to achieve a better integration with their discovery and ranking mechanisms, as in the case of our filtered discovery solution presented in Chapter 4.

3.3 SOUP: DEFINING AN ONTOLOGY OF USER PREFERENCES

Concerning preference terms, Figure 3.4 presents the middle ontology of SOUP, where we differentiate atomic preferences from com-

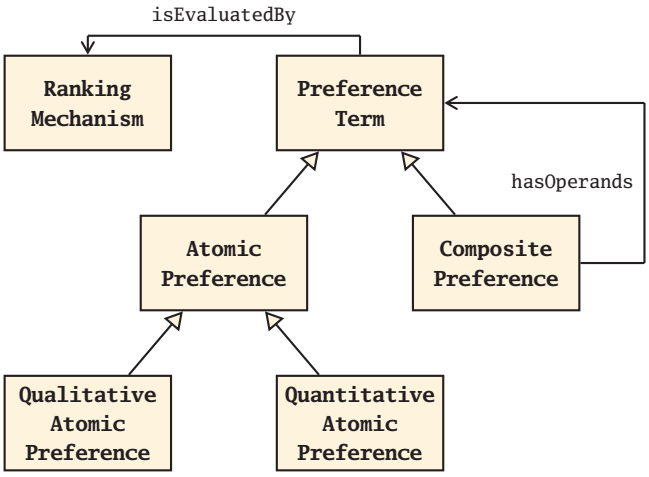


FIGURE 3.4: Middle ontology of preferences.

posite ones. Thus, a `PreferenceTerm` can be an `AtomicPreference`, or a composition of two preference terms by applying a `CompositePreference`. On the one hand, atomic preferences are those which refers to a single domain concept, and can describe either a qualitative or a quantitative preference that users may have with respect to the referred service concept. On the other hand, composite preferences relate different preferences between them, so that a complex preference can be described using the `hasOperands` to associate a composite preference with its components.

As a preference is always related to some domain concepts, it can be intuitively expressed as "I prefer y rather than x", where x and y are instances of those concepts. This relationship between concept instances can be mathematically interpreted as a strict partial order. Therefore, we define a preference in general as:

Definition 3.1 - Preference.
Let $\mathcal{C}$ be a non-empty set of domain concepts, and $\text{dom}(\mathcal{C})$ the set of all possible instances of those concepts. We define a preference as $\mathcal{P} = (\mathcal{C}, <^{\mathcal{P}})$, where $<^{\mathcal{P}} \subseteq \text{dom}(\mathcal{C}) \times \text{dom}(\mathcal{C})$ is a strict partial order (irreflexive, transitive and asymmetric), and if $x, y \in \text{dom}(\mathcal{C})$, then $x <^{\mathcal{P}} y$ is interpreted as "I prefer y rather than x".

If we consider a finite set of concept instance pairs $(x, y) \in <^{\mathcal{P}}$, $\mathcal{P}$ can be represented as a directed acyclic graph, also known as *Hasse diagrams* [23], where each node corresponds to a concept instance, and edges represent the preference relationship $<^{\mathcal{P}}$. This representation is used to return ranking results in PURI (see Chapter 5). Furthermore, each preference term instance defines its order depending on the concrete concepts referred ($\mathcal{C}$) and some operand values that determine the evaluation of the $<^{\mathcal{P}}$ relation.

At this level we also add information about which particular ranking implementation is able to analyze and evaluate a certain preference term, associating this term with an instance of a `RankingMechanism` via the `isEvaluatedBy` object property. Our integrated ranking solution discussed in Chapter 5 makes use of this information in order to dynamically instantiate relevant ranking mechanisms when evaluating a particular preference term. Consequently, in order to abstract our preference model definition from the ranking implementation, we intentionally omit this property in the following.

Each preference construct derived from the hierarchy shown in Figure 3.4 is defined both formally and intuitively in the following, including a motivating example described in natural language from the SWS Challenge scenario used to validate our proposed model in Chapter 6, where some of these constructs are applied to describe that scenario goals. A formal discussion of the algebra of the described preference terms can be found at [52], where the foundations of our model are thoroughly discussed.

3.3.1 Qualitative atomic preferences

The first group of preferences that we present in the following corresponds to the qualitative and atomic constructs, which means that every preference $\mathcal{P} = (\mathcal{C}, <^{\mathcal{P}})$ that belongs to this kind refers to a single domain concept that represents a non-numerical service property, *i.e.* $|\mathcal{C}| = 1$. Figure 3.5 shows the available hierarchy of preference terms belonging to this group. Note that different terms use specific object properties to associate their operands to values from the referred domain concept, depending on the semantics of each preference construct. In each example, *italics* text cor-

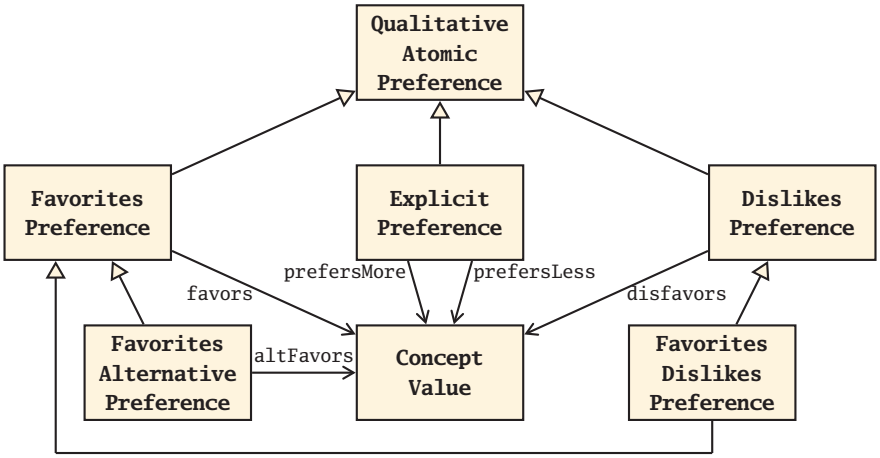


FIGURE 3.5: Qualitative atomic preference terms hierarchy.

respond to service property values or instances used as operands, while `typewriter` text are used to denote domain concept classes that represents those properties.

Definition 3.2 - FavoritesPreference.
Let $FAV \subseteq \text{dom}(\mathcal{C})$ be a non-empty, finite set of preferred values for property $\mathcal{C}$, and $x, y \in \text{dom}(\mathcal{C})$ property values from two services. $\mathcal{P}_{FAV} = (\mathcal{C}, <^{\mathcal{P}_{FAV}})$ is a `FavoritesPreference` iff

$$x <^{\mathcal{P}_{FAV}} y \iff x \notin FAV \wedge y \in FAV$$

A *favorites preference* defines a finite set of property values that constitute the desired values of the referred service property. Thus, services whose value for that property is a member of the *favorite set* are preferred to services that provide any other values from the property domain. An instance of this preference constructor has many operands as the cardinality of the favorite values set, associated using the `favors` object property.

Example: I prefer services that provide *carriageForward* as a possible `PaymentMethod`.

Definition 3.3 - `DislikesPreference`.
Let $DIS \subseteq \text{dom}(\mathcal{C})$ be a non-empty, finite set of disliked values for property $\mathcal{C}$, and $x, y \in \text{dom}(\mathcal{C})$ property values from two services. $\mathcal{P}_{DIS} = (\mathcal{C}, <^{\mathcal{P}_{DIS}})$ is a `DislikesPreference` iff

$$x <^{\mathcal{P}_{DIS}} y \iff y \notin DIS \wedge x \in DIS$$

As opposite to `FavoritesPreference`, a *dislikes preference* defines a set of property values that the service should not provide for the referred property in order to be preferred to another service whose property values coincide with any of the values in the associated *dislikes set*. In this case, operands are linked to the term via the `disfavors` object property.

Example: I prefer SWSs that do not offer *refundForDamage* as an available `Insurance` option.

Definition 3.4 - `FavoritesAlternativePreference`.
Let $FAV \subseteq \text{dom}(\mathcal{C})$ and $ALT \subseteq \text{dom}(\mathcal{C})$ be two non-empty, finite sets of preferred values for property $\mathcal{C}$, and $x, y \in \text{dom}(\mathcal{C})$ property values from two services. $\mathcal{P}_{FAV,ALT} = (\mathcal{C}, <^{\mathcal{P}_{FAV,ALT}})$ is a `FavoritesAlternative Preference` iff

$$\begin{aligned} x <^{\mathcal{P}_{FAV,ALT}} y \iff & (x \in ALT \wedge y \in FAV) \vee \\ & (x \notin FAV \wedge x \notin ALT \wedge y \in ALT) \vee \\ & (x \notin FAV \wedge x \notin ALT \wedge y \in FAV) \end{aligned}$$

A *favorites or alternative preference* is an extension of `Favorites Preference`, where there are two favorite sets. The second set is called *alternative set*, and links their values with the `altFavors` object property. In this case, services whose property values are in the favorite set are the most preferred. Otherwise their values should be on the alternative set. If this is not the case either, then the corresponding services will be undesirable, because their property values are not member of any of the two sets. Note that `favors` property is inherited because of the subclass relationship between `FavoritesPreference` and `FavoritesAlternativePreference`.

Example: I prefer SWSs whose `PaymentMethod` is *carriagePaid*, but if that is infeasible, then it should be *carriageForward*.

Definition 3.5 - `FavoritesDislikesPreference`.
Let $FAV \subseteq \text{dom}(\mathcal{C})$ and $DIS \subseteq \text{dom}(\mathcal{C})$ be two non-empty, finite sets of preferred and disliked values for property $\mathcal{C}$, and $x, y \in \text{dom}(\mathcal{C})$ property values from two services. $\mathcal{P}_{FAV,DIS} = (\mathcal{C}, <^{\mathcal{P}_{FAV,DIS}})$ is a `FavoritesDislikesPreference` iff

$$x <^{\mathcal{P}_{FAV,DIS}} y \iff (x \in DIS \wedge y \notin FAV) \vee (x \notin DIS \wedge x \notin FAV \wedge y \in FAV)$$

It is also possible to combine a `FavoritesPreference` with a `DislikesPreference` in the following form: a given service property should have a value on the defined favorite set. Otherwise, values should not belong to the dislikes set. If none of these two conditions hold, then the service will be less preferred than others fulfilling the first or the second condition. Again, subclass relationships bring both `favors` and `disfavors` object properties to this preference term.

Example: I prefer SWSs that provide *refundForLoss* as an option for `Insurance`, but if that is infeasible, then it should not be *refundForDamage*.

Definition 3.6 - `ExplicitPreference`.
Let $G = \{(v_1, v_2), \dots\}$ be a non-empty, finite directed acyclic graph that represents "better-than" relationships between its nodes $v_i \in \text{dom}(\mathcal{C})$ corresponding to values of property $\mathcal{C}$, and V be the set of nodes belonging to G. Then, a strict partial order $E = (V, <^E)$ is induced as follows:

$$\text{a) } (v_i, v_j) \in G \implies v_i <^E v_j$$
$$\text{b) } v_i <^E v_j \wedge v_j <^E v_k \implies v_i <^E v_k$$

Therefore, given $x, y \in \text{dom}(\mathcal{C})$ property values from two services, $\mathcal{P}_E = (\mathcal{C}, <^{\mathcal{P}_E})$ is an `ExplicitPreference` iff:

$$x <^{\mathcal{P}_E} y \iff x <^E y \vee (x \notin V \wedge y \in V)$$

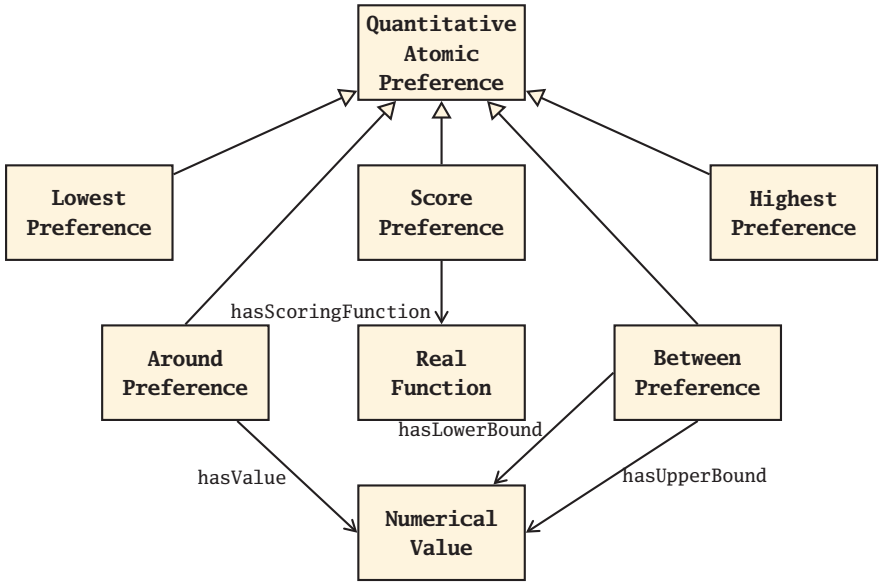


FIGURE 3.6: Quantitative atomic preference terms hierarchy.

An *explicit preference* can be used to explicitly represent the strict partial order between a pair of property values. Thus, a directed acyclic graph comprising better-than relationships can be defined using several *explicit preferences*. In this case, `prefersMore` denote the value that is considered better than the `prefersLess` value.

Example: SWSs that provide *carriageForward* as a possible value for the `PaymentMethod` are more preferred than those that provide the *carriagePaid* value.

3.3.2 Quantitative atomic preferences

When the referred domain concept of an atomic preference is a numerical property, the quantitative constructs shown in Figure 3.6 may be used to express user preferences on that single property. Therefore, $\mathrm{dom}(\mathcal{C})$ values are numbers that support the total order operator $<$ and the subtraction $-$.

Definition 3.7 - `LowestPreference`.
Let $x, y \in \text{dom}(\mathcal{C})$ be values for property $\mathcal{C}$ from two services. $\mathcal{P}_L = (\mathcal{C}, <^{\mathcal{P}_L})$ is a `LowestPreference` iff:

$$x <^{\mathcal{P}_L} y \iff x > y$$

A *lowest preference* does not have any operand, but prefer services whose property values are as low as possible for the referred service property.

Example: I prefer SWSs that provide a `BasePrice` as low as possible.

Definition 3.8 - `HighestPreference`.
Let $x, y \in \text{dom}(\mathcal{C})$ be values for property $\mathcal{C}$ from two services. $\mathcal{P}_H = (\mathcal{C}, <^{\mathcal{P}_H})$ is a `HighestPreference` iff:

$$x <^{\mathcal{P}_H} y \iff x < y$$

In opposition to the last constructor, a *highest preference* is used when property values should be as high as possible.

Example: I prefer SWSs that provide a `PaymentDeadline` as long as possible.

Definition 3.9 - `AroundPreference`.
Let $z \in \text{dom}(\mathcal{C})$ be the most preferred value of $\mathcal{C}$. For all values $v \in \text{dom}(\mathcal{C})$ we define:

$$dist(v, z) = |v - z|$$

Then, given $x, y \in \text{dom}(\mathcal{C})$ property values from two services, $\mathcal{P}_z = (\mathcal{C}, <^{\mathcal{P}_z})$ is an `AroundPreference` iff:

$$x <^{\mathcal{P}_z} y \iff dist(x, z) > dist(y, z)$$

An *around preference* determines which property value is better by determining the distance of each values to a concrete value provided as an operand of this preference term using the `hasValue` object property. Thus, services which provide exactly that value are preferred to the rest of them. If this is infeasible, services with closer values to the operand are preferred.

Example: I prefer SWSs that provide a `BasePrice` closer to *180 Euros*.

Definition 3.10 - `BetweenPreference`.
Let $[low, up] \in \text{dom}(\mathcal{C}) \times \text{dom}(\mathcal{C})$ be the preferred values interval of $\mathcal{C}$. For all values $v \in \text{dom}(\mathcal{C})$ we define:

$$dist(v, [low, up]) = \begin{cases} 0 & \text{if } v \in [low, up] \\ low - v & \text{if } v < low \\ v - up & \text{if } v > up \end{cases}$$

In this case, given $x, y \in \text{dom}(\mathcal{C})$ property values from two services, $\mathcal{P}_{[low,up]} = (\mathcal{C}, <^{\mathcal{P}_{[low,up]}})$ is a `BetweenPreference` iff:

$$x <^{\mathcal{P}_{[low,up]}} y \iff dist(x, [low, up]) > dist(y, [low, up])$$

In this case, a service should have values for the referred property between a range that are defined as operands in the preference (using `hasLowerBound` and `hasUpperBound` to actually define range bounds). If this is not the case, *between preferences* prefer services closer to the interval boundaries, computing the distance as in around preferences.

Example: I prefer SWSs that provide a `PaymentDeadline` within the interval of $[45, 60]$ days.

Definition 3.11 - `ScorePreference`.
Let $f\colon \text{dom}(\mathcal{C}) \to \mathbb{R}$ be a scoring function and $<$ the usual less-than order in $\mathbb{R}$. $\mathcal{P}_f = (\mathcal{C}, <^{\mathcal{P}_f})$ is a `ScorePreference` iff for $x, y \in \text{dom}(\mathcal{C})$:

$$x <^{\mathcal{P}_f} y \iff f(x) < f(y)$$

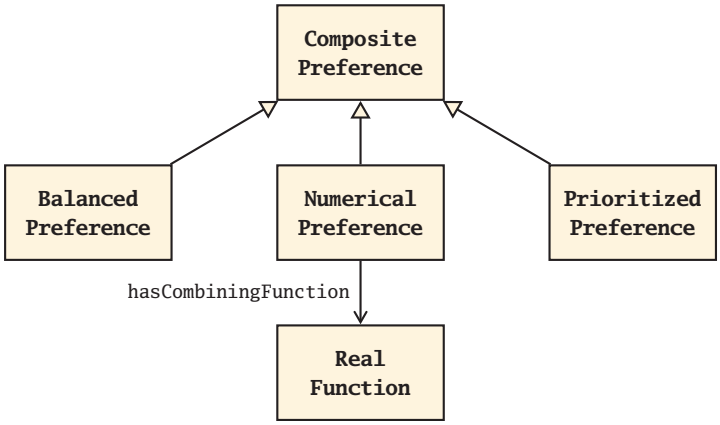


FIGURE 3.7: Composite preference terms hierarchy.

A *score preference* basically defines a scoring function (*i.e.* a utility function like in [34], linked via `hasScoringFunction`) that takes a property value as its argument and returns a real value that can be interpreted in the following form: the higher the value returned by the function is, the more preferred the property value entered as the argument. Note that this kind of preference is not as intuitive as the rest, but it is still useful when a user wants to express complex grades of preference, using for instance a piecewise function depending on the property values.

Example: I prefer SWSs with the highest score with respect to `Price PerKg`, where the scoring function is defined as:

$$f(pricePerKg) = \frac{-1}{50} pricePerKg + 1$$

3.3.3 Composite Preferences

The last group of preference constructs are used to compose two different preference terms by stating the preference relationship between each component term, which can be also a composite preference. Composite preferences `refersTo` property associate the preference with the union of the properties referred by component preferences.

These complex constructors are defined in the following for two preferences, though they can be trivially generalized to a greater number of preferences. Consequently, our model does not initially restrict the number of preference terms that can be composed using composite preferences.

Definition 3.12 - `BalancedPreference`.
Let $\mathcal{P}_1 = (\mathcal{C}_1, <^{\mathcal{P}_1})$ and $\mathcal{P}_2 = (\mathcal{C}_2, <^{\mathcal{P}_2})$ be two different preferences defined after $\mathcal{C}_1$ and $\mathcal{C}_2$ properties, and $x = (x_1, x_2), y = (y_1, y_2) \in \text{dom}(\mathcal{C}_1) \times \text{dom}(\mathcal{C}_2)$ be two value tuples for each property. $\mathcal{P} = (\mathcal{C}_1 \cup \mathcal{C}_2, <^{\mathcal{P}_1 \otimes \mathcal{P}_2})$ is a `BalancedPreference` iff:

$$x <^{\mathcal{P}_1 \otimes \mathcal{P}_2} y \iff (x_1 <^{\mathcal{P}_1} y_1 \wedge (x_2 <^{\mathcal{P}_2} y_2 \vee x_2 = y_2)) \vee (x_2 <^{\mathcal{P}_2} y_2 \wedge (x_1 <^{\mathcal{P}_1} y_1 \vee x_1 = y_1))$$

A *balanced preference* $\mathcal{P}$ combines two preference terms $\mathcal{P}_1$ and $\mathcal{P}_2$ using the *Pareto-optimality principle*, which considers that $\mathcal{P}_1$ and $\mathcal{P}_2$ are equally important preferences. Thus, a service SWS_1 is better than another service SWS_2 with respect to $\mathcal{P}$, if SWS_1 is better than SWS_2 with respect to $\mathcal{P}_1$ and SWS_1 is not worse than SWS_2 with respect to $\mathcal{P}_2$, and *vice versa*. Intuitively, this preference balance the fulfillment of each preference component, so that the composite preference is the average degree of preference taking both components into account.

Example: I prefer SWSs that best fit (with an average satisfaction) the following three (atomic) preferences: the lowest `BasePrice`, the `PaymentDeadline` within the interval of $[45, 60]$ days, and provided `Insurance` options of *refundForLoss* or *refundForDamage*. See Figure 6.4 for a graphical instantiation of this example.

Definition 3.13 - `PrioritizedPreference`.
Let $\mathcal{P}_1 = (\mathcal{C}_1, <^{\mathcal{P}_1})$ and $\mathcal{P}_2 = (\mathcal{C}_2, <^{\mathcal{P}_2})$ be two different preferences defined after $\mathcal{C}_1$ and $\mathcal{C}_2$ properties, and $x = (x_1, x_2), y = (y_1, y_2) \in \text{dom}(\mathcal{C}_1) \times \text{dom}(\mathcal{C}_2)$ be two value tuples for each property. $\mathcal{P} = (\mathcal{C}_1 \cup \mathcal{C}_2, <^{\mathcal{P}_1 \& \mathcal{P}_2})$ is a `PrioritizedPreference` iff:

$$x <^{\mathcal{P}_1 \& \mathcal{P}_2} y \iff x_1 <^{\mathcal{P}_1} y_1 \vee (x_1 = y_1 \wedge x_2 <^{\mathcal{P}_2} y_2)$$

In the case of a *prioritized preference* $\mathcal{P}$ that compose two preference terms $\mathcal{P}_1$ and $\mathcal{P}_2$, $\mathcal{P}_1$ is considered more important than $\mathcal{P}_2$. Thus, $\mathcal{P}_2$ is evaluated only if $\mathcal{P}_1$ does not mind (*i.e.* service property values compared using $\mathcal{P}_1$ do not return enough information to rank those services). In this case, operands have to be evaluated in a specific order, so the `hasOperands` property should be properly specialized to account for operands ordering. For instance, the range of the property could be defined as an RDF list.

Example: I prefer SWSs that provide *carriageForward* as a possible `PaymentMethod`. In the case of equal satisfaction degree on that preference, I prefer SWSs whose `BasePrice` are closer to *180 Euros*. See Figure 6.2 for a graphical representation of this prioritized preference.

Definition 3.14 - `NumericalPreference`.
Let f and g be two scoring functions that define score preferences $\mathcal{P}_f = (\mathcal{C}_1, <^{\mathcal{P}_f})$ and $\mathcal{P}_g = (\mathcal{C}_2, <^{\mathcal{P}_g})$, respectively, and $F\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a combining function. For $x = (x_1, x_2), y = (y_1, y_2) \in \mathrm{dom}(\mathcal{C}_1) \times \mathrm{dom}(\mathcal{C}_2)$, $\mathcal{P} = (\mathcal{C}_1 \cup \mathcal{C}_2, <^{\mathrm{rank}_F(\mathcal{P}_f, \mathcal{P}_g)})$ is a `NumericalPreference` iff:

$$x <^{\mathrm{rank}_F(\mathcal{P}_f, \mathcal{P}_g)} y \iff F(f(x_1), g(x_2)) < F(f(y_1), g(y_2))$$

Finally, a *numerical preference* is the combination of a number of score preferences using a function that takes the values returned by the score preferences as its arguments and returns another real number that gives information about the global preference, considering all the properties referred by concrete score preferences. Notice that component preferences must be score preferences in order to properly compose them using a combining function, which is associated with this term using the `hasCombiningFunction` object property.

Example: Provided that $f(basePrice)$ and $g(pricePerKg)$ are already defined and they range within the interval $[0, 1]$, I prefer SWSs that have a higher combined score, where the combining function is defined as:

$$F(basePrice, pricePerKg) = 0.8 * f(basePrice) + 0.4 * g(pricePerKg)$$

3.4 Application to a WSMO Scenario

Our defined preference model can be applied to different scenarios, because it offers a comprehensive set of facilities to define complex user preferences. However, particular use cases may need an extension and/or adaptation of the provided model in order to better reflect service requests defined in these cases. In order to apply our model to these scenarios, preference terms can be specialized, creating a lower ontology that extends the hierarchy of available terms according to the concrete scenario needs. For instance, §8.3 describes an extension of different preference facilities to take benefit of specific ranking mechanisms that provides monitoring properties and fuzzy logic based preferences.

Furthermore, as the proposed preference model is general and independent from the formalism, it can be applied as an extension to current SWS frameworks, such as WSMO, OWL-S, or even SAWSDL, so that these frameworks can support user preference modeling. Concerning WSMO, our proposed model can be implemented as an extension of its meta-model. Thus, user requests from our model corresponds to WSMO *goals*. Moreover, functionality terms are already supported by WSMO capabilities and interfaces, so that user requirements described in our model can be easily translated into them. However, preference terms have to be added to the specification of goals. Therefore, in order to apply our preference model to WSMO, we define a new meta-model class in Listing 3.3, `preferenceGoal`, which is a subclass of `goal` that adds a `hasPreference` property where preference terms can be linked with a user goal.

Listing 3.3: WSMO goal extended with preferences.

```
Class preferenceGoal sub-Class goal
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type {ooMediator, ggMediator}
  requestsCapability type capability
    multiplicity = single-valued
  requestsInterface type interface
  hasPreference type PreferenceTerm
    multiplicity = single-valued
```

This implementation allows a seamless integration of preference information in WSMO, without actually modifying the goal metamodel, because it is only refined. Thus, current WSMO discovery and ranking proposals could be still applied to extended goals transparently. A different approach can be found in [34], where preferences are included within `nonFunctionalProperties` section by using logic programming rules, although it is only applied to preferences defined as utility functions.

Listing 3.4 shows an example of how to describe a WSMO goal using our proposed implementation to include an instance of our preference model. Thus, goal D1 from §6.2 can be described in WSMO easily. The domain ontology for the Logistics Management scenario is supposed to be properly defined in `Logistics.wsml`.

LISTING 3.4: Extended goal description with preferences from D1.

```
namespace {_"GoalD1.wsml#", lm _"Logistics.wsml#",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax/",
  pref _"http://www.isa.us.es/ontologies/PreferenceModel.wsml#"}

preferenceGoal GoalD1

 capability D1requestedCapability
 preference D1preference

ontology preferenceOntology

instance D1preference memberOf pref#LowestPreference
  pref#refersTo hasValue lm#BasePrice
```

From the above example, one concludes that transforming user requests modeled using our proposed ontology for preferences to a WSMO goal is a straightforward process, provided that the ontological model is expressed in the Web Service Modeling Language (WSML) [81]. Also notice that the WSML variant used in Listing 3.4 includes new keywords to link specialized goals to preference terms which are described in a separate ontology.

3.5 SUMMARY

In this chapter, a highly expressive preference model for SWS discovery and ranking named SOUP is presented. This model, specified as an ontology, represents a novel approach that leverages preference descriptions so that they become a first-class citizen in SWS frameworks. Additionally, SOUP has been validated using a complex discovery scenario from the SWS Challenge in order to prove the applicability of our solution to an actual discovery and ranking scenario (see Chapter 6). The main benefits of our proposed model can be summarized as follows:

- **Expressiveness**. The model is sufficiently expressive to describe complex user desires about requested services, providing a comprehensive hierarchy of preference terms.
- **Intuitive semantics**. Based on a strict partial order interpretation of preferences, the model is user-friendly and machine-readable, so preferences may be automatically processed and inferred.
- **Qualitative and Quantitative**. Available constructs allow to express both qualitative and quantitative preferences, and even combine them in a general preference term.
- **Independence**. Our proposal is not coupled with a concrete SWS solution, neither with a discovery nor ranking mechanism, so it is not limited by the formalisms used to implement these mechanisms.
- **Extensibility**. Because the model is presented as an ontology, it can be further extended with new preference constructs with ease.
- **Applicability**. Our model can be implemented within any SWS framework, extending current proposals to leverage preference descriptions.

An implementation of our model that extends WSMO goals is also discussed. This actual application consists in a seamless exten-

sion of WSMO constructs to allow the definition of complex preferences, that can be used within any discovery and ranking solution, provided that it supports or adapts the proposed preference ontological model.

Regarding our thesis on preference modeling, our proposed preference model achieves a high expressiveness because of the high number of preference facilities offered to the user. Consequently, challenge *C1* is completely fulfilled, providing a step forward on state of the art on SWS preference modeling. Furthermore, our proposal obtains a high fulfillment of challenge *C2*, allowing the application and adaptation to any SWS framework and enabling the seamless integration of discovery and ranking mechanisms.

We have published our contributions on preference modeling in several conferences. First, we proposed an initial approach to model user preferences using utility functions in [30]. Concerning model independence, we presented a hybrid, independent discovery and ranking solution in [33], while transformations mechanisms from models to ranking implementation was proposed in [32]. Finally, we published the definition and validation of SOUP in [38].

4

Optimizing Discovery and Ranking Processes

Simplicity is prerequisite for reliability.

Edsger W. Dijkstra (1930–2002)
Dutch computer scientist

Lightweight solutions to discover and rank services have been preeminently gaining interest within the community, as more services are going to be accessible in the near future. In this chapter, we present an optimization to discovery and ranking processes, which takes a novel approach to offer a more lightweight SWS discovery, consisting on an additional filtering stage. The motivation of our work is further discussed in §4.1. Then, §4.2 presents our proposal to optimize service discovery processes. We apply it to a concrete OWL-S scenario, discussing our Enhanced MatchMaking Addon (EMMA) implementation of devised filters in §4.3. Finally, we outline our thesis contribution with respect to the relevant challenges in this area in §4.4.

4.1 INTRODUCTION

The number of currently available services in public repositories[1] is expected to explode in the future, so that billions of services will be able to be consumed in the Web [24]. Furthermore, currently available semantic descriptions, in terms of SWS classical ontologies such as OWL-S or WSMO, present a high complexity for defining and processing them. Both issues lead to a scenario where discovery mechanisms based on different logic formalisms have scalability issues. Consequently, current research efforts focus on providing improvements and optimizations of those mechanisms, using lightweight semantic technologies, in order to enhance the usability of SWS [26, 28].

In order to alleviate the scalability problem on semantic discovery mechanisms, there are some proposals that provide different techniques to improve the discovery performance, such as indexing or caching descriptions [84], using several matchmaking stages [3], and hybrid approaches that include non-semantic techniques [54]. Our proposed Enhanced MatchMaking Addon (EMMA) takes a novel approach of reducing the input for discovery mechanisms, so that the resulting process is more streamlined, only reasoning about services which actually matter with respect to the user request. Thus, services that can be discarded *a priori*, because they are not related at all with requirements and preferences stated by the user, are filtered so that the search space is considerably reduced prior to actual discovery.

For example, consider the following scenario: a semantic service repository contains thousands of services related to logistics and transportation domains, such as couriers, warehousing, truck rentals, and packaging. If a user looks for a service that is able to compute the time needed to deliver some goods to a particular city, it is not necessary to process the whole repository to discover candidate services for the user request, but only consider the portion of services that are specifically related to couriers domain concepts that appear on the request, in this case. Thus, using lightweight

[1] At the moment of writing, *seekda!* service crawler has indexed 28,606 services, *ProgrammableWeb* has registered 6,990 web APIs, and *iServe* repository contains 2,193 SWS descriptions.

technologies to preprocess the repository, the search space can be reduced in order to save computational resources and improve discovery performance.

For the proposed preprocessing, the user request is analyzed in order to extract the concepts that are being used in its semantic definition (in the above example, some of them could be *Goods*, *City* or *Time*, for instance). Then, the repository is filtered so that only services that use those concepts or related ones are selected to become the input for the subsequent discovery process (*e.g.* services whose definitions refer to *Goods*, *City* and/or *Time* concepts, in the latter case).

Filtering is performed in our approach by two different SPARQL [72] queries, namely $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$. The former returns only those services whose definitions contain *all* the concepts referred by a user request, assuming that services have to fulfill every term of the request in order to be useful for the user. In turn, the latter query selects service definitions that refer to *some* (at least one) of the concepts referred by a user request, assuming that those services may satisfy its requirements and/or preferences to some extent, despite the missing information.

Our solution does not pretend to provide yet another discovery mechanism, but to introduce a preprocessing filtering stage, based on an accepted standard, that yields a notable improvement on heavyweight semantic processes, such as matchmaking of services. Furthermore, our proposed filtering does not add a noticeable amount of execution time with respect to matchmaking, because SPARQL queries used present a linear complexity on the size of the dataset and graph patterns included [70].

To the best of our knowledge, there is no proposals on filtering semantically-enhanced service repositories, but it is acknowledged that some sort of preprocessing can alleviate discovery and ranking tasks performed on those repositories [3]. Furthermore, our analysis of current approaches discussed in §2.3 corroborates this hypothesis. Consequently, in this chapter we thoroughly describe our thesis contribution with respect to the identified challenge on discovery optimization (*C3*). Our solution not only improves SWS discovery performance, but is applicable on top of any current discovery mechanism, allowing interoperability and integration with

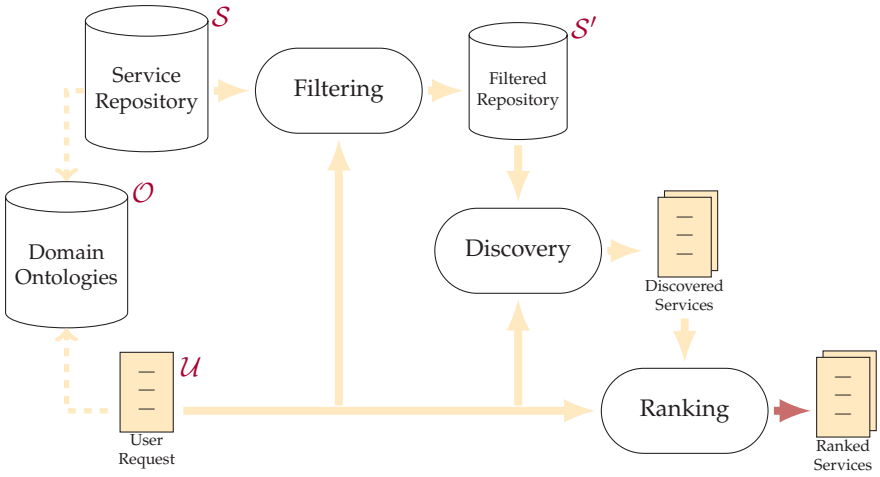


FIGURE 4.1: Service retrieval architecture including a filtering stage.

existing service repositories. An actual application of EMMA on top of OWLS-MX hybrid matchmaker is used to illustrate this point in our carried out experimental study, which is showcased in Chapter 7.

4.2 EMMA: PREPROCESSING REPOSITORIES USING SPARQL

As discussed before, current SWS discovery and ranking tend to be complex, heavyweight processes. In this thesis we propose the introduction of a preprocessing stage that improves the performance of those processes without changing the underlying mechanisms, by filtering service candidates from a repository with respect to the user requirements before actual discovery. In the following we present our abstract proposal and how it can be implemented using standard, automatically generated SPARQL 1.0 queries.

4.2.1 Filtering a Service Repository

Once services and user requests are well established, the concrete optimizations that can be used to improve discovery and ranking

processes has to be defined. Our proposal adds a new preprocessing stage previous to the discovery process, where the service repository is filtered, using SPARQL queries as described in §4.2.2. In Figure 4.1 the proposed architecture is showcased. The aim of the filtering stage is to obtain $\mathcal{S}'$ services from the original repository $\mathcal{S}$ that may be possibly matched with the user request $\mathcal{U}$ in the discovery process, discarding those ones that cannot fulfill that request at all.

In a general scenario, our proposed filtering stage discriminates service descriptions depending on whether the concepts referenced within their terms are present in the user request or not. To this extent, two different filters can be applied, offering different filtering levels. On the one hand, one of the filters ($\mathcal{Q}_{all}$) only returns service descriptions that refer to the whole set of related concepts described in the user request. On the other hand, a more relaxed filter ($\mathcal{Q}_{some}$) returns those service descriptions that refer to some (at least one) of the concepts that are also referred by the user request. In turn, services whose features do not refer to any of the related concepts referred in the requirements of the user request are discarded by both filters, because in that case it can be inferred that they are not related to the service the user is searching for.

Using the upper ontology discussed in §3.2 as an abstract vocabulary to help our discussion, we can describe our generic proposal as follows. Let $\mathcal{D} = (\mathcal{O}, \mathcal{S}, \mathcal{U})$ be a 3-tuple that represent a discovery scenario as outlined in Figure 4.1, where each element of the tuple is defined in the following.

Definition 4.1 - Domain ontologies ($\mathcal{O}$).
Let $\mathcal{O}_i$ be a certain domain ontology whose concepts can be referred by the user request and service descriptions from a certain discovery scenario $\mathcal{D}$. We define the set of *domain ontologies* $\mathcal{O}$ as the set of ontologies that can be used to define the rest of the elements from that scenario, *i.e.* the user request and service descriptions.

$$\mathcal{O} = \mathcal{O}_1 \cup \cdots \cup \mathcal{O}_n$$

The set of domain ontologies, in addition to SWS models, are used to formulate service descriptions that are stored in a concrete service repository, which defined as follows:

Definition 4.2 - Service repository ($\mathcal{S}$).
Let $\mathcal{O}_{\mathcal{S}_i}$ be a subset of $\mathcal{O}$. A *service repository* $\mathcal{S}$ is a set of service descriptions $\mathcal{S}_i$ that are defined by several terms t_{ij}. Each term refer to a set of concepts $\mathcal{C}_{ij}$ defined in the ontology $\mathcal{O}_{\mathcal{S}_i}$. Therefore, each $\mathcal{S}_i$ is represented as a set of tuples that relate terms with their corresponding set of referred concepts:

$$\mathcal{S}_i = \{(t_{i1}, \mathcal{C}_{i1}), \ldots, (t_{in}, \mathcal{C}_{in}) : \mathcal{C}_{i1} \cup \cdots \cup \mathcal{C}_{in} \subseteq \mathcal{O}_{\mathcal{S}_i}\}$$

Similarly, a user request contains requirements in the form of terms that refer to some subset of concepts from a domain ontology

Definition 4.3 - User request ($\mathcal{U}$).
Assuming $\mathcal{O}_{\mathcal{U}} \subseteq \mathcal{O}$, we define a *user request* $\mathcal{U}$ as:

$$\mathcal{U} = \{(t_1, \mathcal{C}_1), \ldots, (t_n, \mathcal{C}_n) : \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_n \subseteq \mathcal{O}_{\mathcal{U}}\}$$

In order to better illustrate previous definitions, consider an scenario where a user is searching for a courier service like the described in Listing 4.1. The corresponding user request $\mathcal{U}$ is defined as follows:

$$\begin{aligned}\mathcal{U} = \{&(inputTerm_{u1}, \{\text{logi:Goods}\}),\\ &(inputTerm_{u2}, \{\text{geo:City}\}),\\ &(outputTerm_{u1}, \{\text{logi:Time}\})\}\end{aligned}$$

This user is going to search for services described in a repository $\mathcal{S}$ that contains three services related to travel domains, such that:

$$\begin{aligned}\mathcal{S}_1 = \{&(inputTerm_{11}, \{\text{geo:City}\}),\\ &(outputTerm_{11}, \{\text{logi:Perishable}\})\}\\ \mathcal{S}_2 = \{&(inputTerm_{21}, \{\text{logi:Goods}\}),\\ &(inputTerm_{22}, \{\text{geo:City}\}),\\ &(outputTerm_{21}, \{\text{logi:Time}\})\}\\ \mathcal{S}_3 = \{&(inputTerm_{31}, \{\text{logi:Message}\}),\\ &(outputTerm_{31}, \{\text{logi:POBox}\})\}\end{aligned}$$

Finally, the global domain ontology in this example could be simply considered as the set of concepts involved in previous descriptions, *i.e.* $\mathcal{O} = \{\text{geo:City}, \text{logi:Perishable}, \text{logi:Goods}, \text{logi:Time}, \text{logi:Message}, \text{logi:POBox}\}$.

Once the elements that conform the discovery scenario $\mathcal{D} = (\mathcal{O}, \mathcal{S}, \mathcal{U})$ are properly defined, the two previously introduced filters can be used alternatively to obtain a $\mathcal{S}' \subseteq \mathcal{S}$ so that the subsequent discovery process defined by $\mathcal{D}' = (\mathcal{O}, \mathcal{S}', \mathcal{U})$ can perform better.

In order to simplify both filters definitions, we denote with $\mathcal{C}_{\mathcal{S}_i}$ the subset of concepts from $\mathcal{O}_{\mathcal{S}_i}$ that are actually referred in the terms featured in $\mathcal{S}_i$. Equivalently, $\mathcal{C}_{\mathcal{U}}$ is the subset of referred concepts in $\mathcal{U}$.

$$\mathcal{C}_{\mathcal{S}_i} = \{c \in \mathcal{O}_{\mathcal{S}_i} : \exists (t_{ij}, \mathcal{C}_{ij}) \in \mathcal{S}_i | c \in \mathcal{C}_{ij}\}$$
$$\mathcal{C}_{\mathcal{U}} = \{c \in \mathcal{O}_{\mathcal{U}} : \exists (t_j, \mathcal{C}_j) \in \mathcal{U} | c \in \mathcal{C}_j\}$$

Consequently, in the example described before, the corresponding concepts subsets of $\mathcal{O}$ for the service descriptions in $\mathcal{S}$ and the user request $\mathcal{U}$ are the following:

$$\mathcal{C}_{\mathcal{S}_1} = \{\text{geo:City}, \text{logi:Perishable}\}$$
$$\mathcal{C}_{\mathcal{S}_2} = \{\text{logi:Goods}, \text{geo:City}, \text{logi:Time}\}$$
$$\mathcal{C}_{\mathcal{S}_3} = \{\text{logi:Message}, \text{logi:POBox}\}$$
$$\mathcal{C}_{\mathcal{U}} = \{\text{logi:Goods}, \text{geo:City}, \text{logi:Time}\}$$

The application of both filters to a service repository $\mathcal{S}$ return a subset $\mathcal{S}'$ depending on the corresponding filter applied. In the case that $\mathcal{S}' = \mathcal{Q}_{all}(\mathcal{S}, \mathcal{U})$, the application of the filter returns a subset of $\mathcal{S}$ only containing services whose referred concepts are a superset of those referred by a user request $\mathcal{U}$, *i.e.* all concepts referred by the user request are referred by returned service descriptions.

Definition 4.4 - $\mathcal{Q}_{all}$ filter.

$$\mathcal{Q}_{all}(\mathcal{S}, \mathcal{U}) = \{\mathcal{S}_i \in \mathcal{S} : \mathcal{C}_{\mathcal{U}} \subseteq \mathcal{C}_{\mathcal{S}_i}\}$$

In turn, if we identify $\mathcal{S}' = \mathcal{Q}_{some}(\mathcal{S},\mathcal{U})$, the filter selects those services from $\mathcal{S}$ that share at least one referred concept with the user request $\mathcal{U}$, so the intersection of corresponding referred concepts sets cannot be empty.

Definition 4.5 - $\mathcal{Q}_{some}$ filter.

$$\mathcal{Q}_{some}(\mathcal{S},\mathcal{U}) = \{\mathcal{S}_i \in \mathcal{S} : \mathcal{C}_{\mathcal{U}} \cap \mathcal{C}_{\mathcal{S}_i} \neq \emptyset\}$$

Results of applying both filters to the described example are, in the first proposed filter case: $\mathcal{Q}_{all}(\mathcal{S},\mathcal{U}) = \{\mathcal{S}_2\}$, and in the second case: $\mathcal{Q}_{some}(\mathcal{S},\mathcal{U}) = \{\mathcal{S}_1,\mathcal{S}_2\}$.

Although $\mathcal{Q}_{all}$ effectively reduces the discovery search space (*i.e.* $\mathcal{Q}_{all}(\mathcal{S},\mathcal{U}) \subseteq \mathcal{S}$) and, consequently, processing time, it may excessively restrict the candidate services to be considered for the subsequent discovery process, whose resultant precision and/or recall may be affected, as we corroborate in our experiments in Chapter 7. Thus, the proposed $\mathcal{Q}_{some}$ filter relaxes the former one by considering each concept referenced in the user request as a matching alternative within the set of concepts referred by service description terms. In this case, service descriptions that do not refer to any concept used in the user request are discarded for the following discovery stage. In general, $\mathcal{Q}_{all}(\mathcal{S},\mathcal{U}) \subseteq \mathcal{Q}_{some}(\mathcal{S},\mathcal{U}) \subseteq \mathcal{S}$, so filtering repositories using $\mathcal{Q}_{some}$, the amount of services that are considered for discovery (and ranking) is reduced less than in the $\mathcal{Q}_{all}$ scenario. However, the overall performance improvement is also high, while it slightly affects the process precision/recall relation, as analyzed in Section Chapter 7.

4.2.2 A SPARQL Implementation

The abstract description of our proposed filters $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ introduced previously can be implemented in any existing SWS discovery scenario using SPARQL SELECT queries. Given a concrete user request defined using an existing SWS framework, both EMMA filters can be instantiated as SPARQL queries that select corresponding services from an RDF-based repository, which contains descrip-

tions based on the same SWS framework. In this case, generated queries have to be also based on graph patterns ranging over that SWS framework RDF representation. Nevertheless, to better account for interoperability some proposals that integrate SWS framework definitions [21, 69] can also apply our proposed filters.

Queries need to be instantiated for each user request $\mathcal{U}$, because they depend on the structure of that request. In order to compose $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ filters, some analysis has to be done, because concrete concepts referred by the user request are used in the corresponding SPARQL query. Specifically, query generation depends not only on the structure of the ontology our proposal is being applied to, but also on the concrete instance $\mathcal{U}$ of the user request itself, especially on the concepts referred by its terms ($\mathcal{C}_{\mathcal{U}}$). As a consequence, queries have to be tailored depending on the corresponding instances managed by each discovery process. However, the generation of $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ SPARQL queries can be done automatically, maintaining the transparency for the user of our proposed filtering stage within the discovery process.

On the one hand, $\mathcal{Q}_{all}$ filter is implemented as a query that searches for services whose featured terms refer to every concept referred in the user request. Thus, for each term and its corresponding concepts, $\mathcal{Q}_{all}$ query contains a triple pattern that matches service definition triples that contains those concepts, depending on the structure of the underlying SWS ontology. On the other hand, $\mathcal{Q}_{some}$ query is generated similarly, but each triple pattern matching a user request referred concept is grouped with the rest as alternative patterns, *i.e.* using the `UNION` keyword, because $\mathcal{Q}_{some}$ searches for services whose terms refer to at least one concept referred by the user request. An application of both queries to OWL-S is presented in the following section.

4.3 Application to Existing SWS frameworks

Our proposed preprocessing stage can be easily adapted to any SWS framework, such as WSMO, OWL-S, SAWSDL or WSMO-Lite, so that it can be virtually included within any discovery process. In order to do so, elements from the filter definition discussed in §4.2.1

have to be identified with corresponding user requests and service descriptions expressed using a specific SWS framework. Therefore, the SPARQL implementation of both filters contains triple patterns that refers to services ($\mathcal{S}$), requests ($\mathcal{U}$), terms and domain concepts ($\mathcal{C}_{S_i}$ and $\mathcal{C}_{\mathcal{U}}$) on the target SWS framework. In the following, EMMA, which is a concrete OWL-S implementation of our filters, is presented, but Appendix C also discusses another early implementation to WSMO services.

4.3.1 Applying EMMA to OWL-S

In order to implement an application of our proposed filtering stage that relies on OWL-S descriptions, they have to be published in a triple store and queries have to be defined in terms of OWL-S constructs. Basically, both service descriptions ($\mathcal{S}$) and user requests ($\mathcal{U}$) are modeled as `ServiceProfiles`. A service profile may contain several terms that further define features of an OWL-S service functionality, such as `Inputs`, `Outputs`, `Preconditions`, and `Results`. Listing 4.1 presents an example OWL-S service profile RDF description.

LISTING 4.1: OWL-S service profile example.

```
1  :GoodsCityTimeProfile a profile:Profile;
2      profile:hasInput  :GoodsInput;
3      profile:hasInput  :CityInput;
4      profile:hasOutput :TimeOutput.
5
6  :GoodsInput a process:Input;
7      process:parameterType logi:Goods.
8  :CityInput a process:Input;
9      process:parameterType geo:City.
10 :TimeOutput a process:Output;
11     process:parameterType logi:Time.
```

This service profile example corresponds to the user request used in examples from §4.2.1. We only consider input and output terms from OWL-S user requests, though preconditions and results may also be analyzed (*cf.* Appendix C). Although both inputs and outputs can be related to the corresponding service profile by using the abstract `hasParameter` OWL-S property, we explicitly relate profiles to inputs and outputs with `hasInput` and `hasOutput` properties. In

consequence, our filters are refined to take into account the stated difference between inputs and outputs terms in OWL-S descriptions, so that more accurate results can be obtained.

LISTING 4.2: $\mathcal{Q}_{all}$ SPARQL query applied to OWL-S.

```
1 SELECT DISTINCT ?service
2 WHERE {
3   ?service a service:Service;
4            service:presents ?profile.
5   # ?profile has at least two inputs and an output...
6   ?profile profile:hasInput ?inputTerm1.
7   ?profile profile:hasInput ?inputTerm2.
8   ?profile profile:hasOutput ?outputTerm1.
9   # ...and referred input concepts are Goods...
10  {?inputTerm1 process:parameterType logi:Goods}
11  # ...City...
12  {?inputTerm2 process:parameterType geo:City}
13  # ...and the output concept is Time
14  {?outputTerm1 process:parameterType logi:Time}
15 }
```

Listing 4.2 and Listing 4.3 presents our proposed $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ filter queries as issued by EMMA, respectively. The identified correspondences between the elements of our abstract filtering proposal and OWL-S constructs are introduced for both SPARQL queries, as described in §4.2.2, so that EMMA can directly use them to filter an OWL-S repository. In this example, both queries have been generated from the user request $\mathcal{U} = \{(inputTerm1, \{\text{logi:Goods}\}), (inputTerm2, \{\text{geo:City}\}), (outputTerm1, \{\text{logi:Time}\})\}$, where their referred concepts to be matched against service descriptions are:

$$\mathcal{C}_{\mathcal{U}} = \{\text{logi:Goods}, \text{geo:City}, \text{logi:Time}\}$$

Note that the presented OWL-S application refines the filters proposed in §4.2, taking into account that each type of term in $\mathcal{U}$ should be matched with the corresponding terms from service descriptions $\mathcal{S}_i$. In consequence, $\mathcal{C}_{\mathcal{U}}$ and $\mathcal{C}_{\mathcal{S}_i}$ sets of concepts are split in two subsets each, depending on the type of term (input or output), and compared with the corresponding one to obtain both filters results.

LISTING 4.3: $\mathcal{Q}_{some}$ SPARQL query applied to OWL-S.

```
1 SELECT DISTINCT ?service
2 WHERE {
3   ?service a service:Service;
4            service:presents ?profile.
5   # match all inputs and outputs of the profile...
6   ?profile profile:hasInput ?inputTerms.
7   ?profile profile:hasOutput ?outputTerms.
8   # ...that refer to some concepts of the user request
9   {?inputTerms process:parameterType logi:Goods}
10  UNION {?inputTerms process:parameterType geo:City}
11  UNION {?outputTerms process:parameterType logi:Time}
12 }
```

In principle, if RDFS entailment regime were applied to the RDF dataset of the service repository, making the inferred knowledge explicit, $\mathcal{Q}_{some}$ could have been written using a more concise and general approach that does not need to process the user request instance in order to explicitly reflect its referred concepts. Thus, lines 9 to 11 in Listing 4.3 could be substituted by the following excerpt, with `:reqProfile` being the concrete `ServiceProfile` instance that is used to look for requested services. However, if we have to account for inference as considered in §4.3.3 because a basic entailment is the only available in our querying system, then $\mathcal{Q}_{some}$ as defined in Listing 4.3 is more convenient.

```
?inputTerms process:parameterType ?inputConcepts.
?outputTerms process:parameterType ?outputConcepts.
:reqProfile rdf:type service:Service.
:reqProfile profile:hasInput ?reqInputTerms.
:reqProfile profile:hasOutput ?reqOutputTerms.
?reqInputTerms process:parameterType ?inputConcepts.
?reqOutputTerms process:parameterType ?outputConcepts.
```

4.3.2 Automatic Generation of Filter Queries

Right before the filtering is executed, EMMA has to generate corresponding SPARQL queries using OWL-S user requests. Consequently, generation algorithms needs to be applied to the OWL-S ontology, as discussed in §4.2.2. Essentially, the user request $\mathcal{U}$ (defined as a service profile like in Listing 4.1) has to be analyzed to obtain the concepts that are referred by each description term ($\mathcal{C}_{\mathcal{U}}$).

The automatic generation of queries can also differentiate terms in order to get better results with basic entailment regimes.

For the evaluation discussed in Chapter 7, EMMA filtering queries generated from OWLS-TC user requests only take inputs and outputs into account, though service profiles may contain more information terms that could be also analyzed to obtain more referred concepts from the corresponding domain ontology [39]. Therefore, for each OWLS-TC user request, its service profile is traversed identifying each input and output, and adding a triple pattern to the corresponding query to match services with the same referred parameter types.

4.3.3 Dealing with SPARQL Entailment

If the RDF dataset does not contain subclassing knowledge as explicit triples, there are two different approaches to deal with the SPARQL basic entailment regime issues. On the one hand, the implicit knowledge concerning subclasses can be retrieved using a DL reasoner [43, 80], so that corresponding RDF triples can be added to the RDF dataset, providing RDFS entailment. As this inferencing process is time-consuming, it may be executed periodically on the whole repository to properly update the dataset, in order to minimize its impact on query execution. However, this approach does not account for the fact that at the moment a query is executed, the RDF dataset may not contain all the corresponding inferred triples.

On the other hand, queries can be rewritten, explicitly including subclasses of the concepts referenced in user requests. Thus, a DL reasoner is executed when generating SPARQL queries for both $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ filters to obtain the related subclasses for each concept referred in the user request. As a consequence, service descriptions whose referred concepts are subclasses of user request concepts can also be returned by our filtering stage, improving the accuracy of the results.

For instance, the chosen reasoner (Pellet [80] in EMMA implementation) may infer that `Capital` instances are also `City` instances, because there is a subclass relationship between these classes. Then both of them can be considered as valid alternatives for a referred concept in a service description, if the user is looking for a service

that features a `City` concept as its input. Thus, an additional pattern alternative where `?inputTerms` refers to a `Capital` concept have to be included in line 9 of Listing 4.3. Similarly, $\mathcal{Q}_{all}$ queries can also be modified to take concept subclasses into account. In this case, line 10 of Listing 4.2 have to be modified to the same patterns used in the $\mathcal{Q}_{some}$ case for `City` concept, *i.e.*:

```
{?inputTerms process:parameterType geo:City}
UNION {?inputTerms process:parameterType geo:Capital}
```

4.4 SUMMARY

Although Semantic Web query languages are not widely used for SWS discovery and ranking, they can certainly play a role in these processes. As discussed in Chapter 2, some authors extend SPARQL query language to directly support these stages, but our proposal sticks to the recommendation at the time of writing (1.0), providing two different filter queries that may be used before actual discovery process in order to reduce the set of available services from the initial repository. Consequently, the reduced search space further improves scalability and performance in discovery and ranking stages, decreasing the total execution time and memory consumption of these processes, with a contained penalty on precision, recall and fallout.

In this chapter we only presented EMMA an its application to OWL-S by means of a prototype implementation. Nevertheless, Chapter 7 discusses comprehensive evaluation tests that we have also run, analyzing the actual reduction of the search space. The conclusions obtained are mainly that our proposal effectively reduce the search space, obtaining a better performance at a contained loss on precision. Furthermore, it conforms a generic solution, adaptable to any SWS framework that a potential user may want to use.

Our proposal of including a (possibly multiple) filtering stage before the discovery and ranking processes has several benefits in addition to the already discussed optimization of discovery and ranking processes by reducing the search space. These additional benefits are enumerated in the following:

- Proposed filters are generic, so they can be used no matter what kind of user request and service descriptions are defined for each concrete scenario. Corresponding SPARQL queries can be generated automatically from a given user request.
- Our proposal does not distinguish between types of concepts, *i.e.* both functional and non-functional concepts can be used to filter the repository. In consequence, concepts being used for both discovery and ranking stages can be considered.
- Filters can be applied to any SWS framework because they are based only on domain concepts referred by service descriptions and user requests. An application to the OWL-S framework is implemented in EMMA prototype.
- Our filtering stage can be applied to improve any currently available matchmaking implementation. The actual improvement on the overall discovery performance depends on the nature of the matchmaker, as discussed in Chapter 7.
- Our proposal is based on the current standard query language for the Semantic Web, *i.e.* SPARQL 1.0. Nevertheless, our proposed queries do not use any extension to the standard, so they are compatible with most SPARQL implementations.

In conclusion, our proposal follows the current research trend on developing lightweight, scalable applications and extensions that effectively enable the adoption of Semantic Web technologies, by improving current discovery mechanisms in terms of scalability and performance, while offering a contained penalty on precision with respect to classical, heavyweight approaches to SWS matchmaking. As a consequence, challenge *C3* can be completely fulfilled by applying our filters proposed in this thesis.

EMMA proposal along with the thorough evaluation discussed in Chapter 7 has been published in [41]. Previously, we had also published an initial analysis of the requirements of query-based, optimized discovery mechanisms in [36], and a technical report describing the application of EMMA to WSMO [39].

5

Integrating Ranking Mechanisms

Any inaccuracies in this index may be explained by the fact that it has been sorted with the help of a computer.

Donald Knuth (1938–)
American computer scientist

Our third and last contribution proposed in this thesis improves the interoperability and integrability of ranking processes. In this chapter we discuss our proposal on integrating different ranking mechanisms using our devised preference model to enable the needed interoperability. We motivate our approach with respect to relevant challenges in §5.1. Our solution architecture is described in §5.2. Then, §5.3 thoroughly discusses the Preference-based Universal Ranking Integration (PURI) framework implementation to integrate ranking mechanisms. Finally, §5.4 sums up our proposal and analyzes the fulfillment of the challenges.

5.1 INTRODUCTION

In the current service retrieval scenario, where service repositories are being actively developed [69, 83] in order to foster a growth in the number of services, ranking mechanisms have been long-acknowledged to be required for the selection of the best retrieved offerings with respect to certain user-defined preferences.

Each ranking mechanism usually provides an *ad hoc* preference model that constrains the expressiveness of user preferences, which are tightly coupled with the underlying ranking mechanism applied. However, in order to allow the expression of complex preferences for end users, they should be provided with more flexibility to define preferences, so a service retrieval and ranking system may integrate several ranking mechanisms, providing a higher number of facilities to state user preferences. Nevertheless, interoperability issues between preference models may appear, as they cannot be easily combined, and potential synergies may remain unexploited. For instance, consider a sample user request informally defined as:

> *"I want to look for services that can deliver some goods to a city, preferring the cheaper ones though the deadline for the payment of the service should also be fair enough."*

In this request, the user not only states the desired functionality (to deliver goods), but their preferences about some service properties. Concretely, the user is looking for services with the *lowest* possible base price, but also considering that the payment deadline should be *fair*. After retrieving the compliant services with respect to functionality, a ranking process have to be performed in order to rank the result list in terms of the user preferences, simplifying the selection of the best service for the user. These preferences have to be expressed in terms of some model of a particular ranking mechanism that has to be chosen to perform that process.

On the one hand, the lowest price preference can be modeled using ranking approaches that allow to define the desired tendency of a given attribute, usually a NFP of a service, such as base price in the example. Thus, retrieved services should be ranked according to the price value in ascending order. NFP-based simple ordering proposals [25] or multi-criteria ranking approaches [87, 91] can be directly

applied to evaluate this kind of preference, offering simpler preference modeling and more efficient ranking mechanisms. In turn, more expressive approaches, such as those based on utility functions [34, 58] or fuzzy logics [4, 48], are less suitable because they present more complex preference modeling facilities, in addition to a lower ranking performance, in general.

On the other hand, in order to model the preference on the payment deadline, we need to define what is considered to be a *fair* time for the user. For instance, a user may specify that a fair deadline is a value between 45 and 60 days. In this case, a desired tendency definition (as in the price preference discussed before) cannot be used because services are preferred if the NFP value is around the desired interval, instead of preferring a minimum or maximum value. In turn, a fuzzy based ranking mechanism offers means to express this more complex preference, provided that a fuzzy membership function is defined such that it determines to which extent a NFP value can be considered to be *fair*. Furthermore, mechanisms based on utility functions can be also applied, as a fuzzy membership function can be considered as a particular case of a utility function.

Desirably, both preferences should be defined and combined using a unique preference model, so a single ranking mechanism may be chosen to help the user to select the best service. However, if a multi-criteria, tendency based ranking mechanism were chosen, the second preference on payment deadline could not be properly described. In turn, a utility function or fuzzy based ranking approach could allow to define both preferences, but the first one concerning price would be more difficult to describe by the user using their provided facilities, and the global performance would be lower, compared to a tendency based approach. From the user's perspective, it would be more valuable if they could flexibly choose between expressiveness and performance for each preference description.

Consequently, in order to perform service ranking with respect to a combination of both preferences, different ranking approaches could be used correspondingly for price and payment deadline preferences (*e.g.* a tendency based and a fuzzy based approach), though the user should then define each part of their preference using a different model. However, as preference models cannot be directly combined at the conceptual level, results from each ranking mech-

anism have to be manually analyzed so that the user can come up with a global rank. In consequence, there exist challenges on how to combine several preference models (*C4*) so that ranking results obtained from corresponding ranking mechanisms can be automatically integrated, transparently returning a global rank to the user (*C5*).

In this chapter, we present a preference-based ranking integration framework named Preference-based Universal Ranking Integration (PURI), which provides a solution to these remaining challenges, using the SOUP preference model discussed in Chapter 3 as its foundations. Our integrated ranking solution gives the user control on how the ranking process should be performed, because user-specified preferences can combine every facility that the integrated ranking mechanisms provide, seamlessly integrating them and making the most of each ranking approach, according to the user's particular needs.

Additionally, a use case application is discussed in Chapter 8 in order to evaluate PURI applicability. The chosen scenario involves three different ranking mechanisms, which were developed within the SOA4All EU FP7 project. These mechanisms are integrated using PURI framework, adapting our preference model to the particular needs of this retrieval scenario. Our evaluation shows that PURI framework allows to fulfill both challenges *C4* and *C5*.

5.2 Integrated Discovery and Ranking Architecture

As discussed before, there are several interoperability issues between ranking mechanisms that constrain the usability and flexibility of semantic service ranking systems, which tend to be designed eclectically, *i.e.* allowing the application of a single ranking mechanism that provides a limited set of facilities to define preferences. In the following we present an integrated solution to semantic service discovery and ranking that uses our interoperable, highly expressive preference model (see Chapter 3) that allows to integrate several ranking mechanisms into the service discovery and ranking system.

Applying our interpretation of a user request to the service dis-

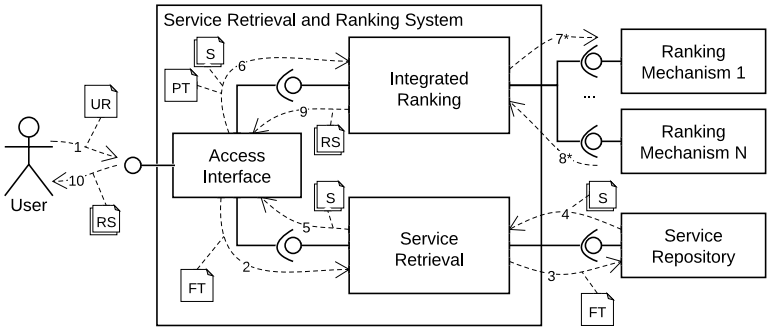


FIGURE 5.1: Integrated SWS discovery and ranking system architecture.

covery and ranking scenario, we can separate terms depending on their belonging class, so that functionality terms can be forwarded to the service discovery component, while preference terms will be solely used by the integrated ranking [31]. Figure 5.1 showcases the hybrid architecture and workflow of our proposed service discovery and ranking system.

Firstly, a user sends a request (UR) to the system (1). The access interface analyzes the request and differentiates functionality (FT) and preference terms (PT) depending on their belonging class. Then, the discovery component searches in a service repository in order to retrieve matching services (S) with respect to functionality-related terms (2,3,4). This set of retrieved services are routed by the access interface component (5) to the preference-based integrated ranking component (6) that analyzes preference terms previously identified so that needed ranking mechanisms are executed (7) and combined (8). Finally, the combined results from integrated ranking mechanisms are returned as the response to the user (9,10), who obtains a ranking of services (RS) that provide the desired functionality ordered according to the requested preferences.

Note that in this proposed architecture, EMMA can be also applied to seamlessly improve the discovery component, filtering the repository before executing the actual discovery (see Chapter 4). As the access interface component simply routes relevant terms from the user request to both discovery (functionality terms) and rank-

ing (preference terms) components, the following section focuses on describing how the PURI framework can be applied to develop an integrated ranking component.

5.3 PURI: A Framework to Integrate Ranking Mechanisms

Our integrated preference based ranking solution uses preferences defined in terms of the SOUP model in order to rank a set of matching services. As discussed in §3.3, each preference term is handled by a particular ranking mechanism. In order to correctly call each mechanism, compose the results, and manage in general the integrated ranking process, we propose the use of the PURI framework, which is described in the following.

PURI framework provides facilities to integrate several ranking mechanisms by using our common preference model. Essentially, its integrated ranking solution takes a set of discovered services and a user preference, which is analyzed in order to obtain the needed combination of ranking mechanisms that have to be invoked such that a given set of services is ordered according to the preferences. The returned service ranking is interpreted as a strict partial order, because some services may not be able to be compared to each other.

Consequently, PURI is able to manage the whole integrated ranking process, with minimal developing effort, ranking a set of services with respect to a user preference, based on our proposed model. Its execution process is as follows:

1. The user preference is analyzed in order to identify the corresponding ranking mechanism(s), denoted by its `hasRankingMechanism` relation. In the case that various ranking mechanisms can be used to evaluate the same preference term, PURI chooses the most suitable according to the user preference structure, maximizing the execution performance.

2. Appropriate ranking mechanism(s) are dynamically instantiated using an abstract factory. The same mechanism may be reused if it needs to be instantiated by several terms in a composite preference.

3. The associated ranking algorithm is executed, which in turn may need other ranking mechanisms to be instantiated and executed if the analyzed preference is a composite one. In this latter case, PURI applies two different workflows depending on the concrete composite preference being evaluated, in order to optimize execution times:

    a) `Balanced` and `Numerical` preference terms fork the ranking execution in order to evaluate their compound preferences in parallel. After all the corresponding ranking mechanisms finish their execution, results are properly composed to obtain a strict partial order.

    b) `Prioritized` preference terms sequentially evaluate each compound term in order, so that the execution is terminated as soon as the evaluation of the corresponding compound term returns an ordering of the retrieved services.

4. The computed ranking is finally returned as a strict partial order, that can be used to select the best service according to the original user preference.

There are some key features that the framework offers for developers to extend and adapt PURI to their particular needs. First and foremost, ranking mechanisms can be dynamically registered into an abstract factory that is utilized to transparently instantiate them when needed to evaluate specific preferences. Secondly, each ranking implementation can be adapted to handle several preference terms from the upper model, which in turn can be also extended to fulfill each scenario particular requirements. Thirdly, composite preferences default implementation handles atomic preferences aggregation automatically, whether they are quantitative or qualitative ones, because every preference is interpreted as a strict partial order [38]. Finally, the returned ranking can be adapted to any desired implementation that successfully represents a strict partial order, though a default implementation, which resembles a directed acyclic graph, is also provided by PURI.

Consequently, a PURI adaptation that already provides an integrated ranking system can be also extended, integrating additional

ranking mechanisms to support other preference facilities or provide higher performance. Although the use case application described in Chapter 8 focuses on integrating three different mechanisms, another ranking implementation may be added seamlessly, provided that its preference model is mapped to our common model, and a corresponding adapter is implemented, so that PURI can properly access and integrate that mechanism with the existing ones.

5.4 SUMMARY

Current service discovery systems have to perform a subsequent ranking so that they can return an ordered list of services in terms of defined preferences, allowing the user to obtain the best service that fulfills the request. However, ranking mechanisms are coupled with *ad hoc* preference models that constrain the expressiveness of user preferences. Furthermore, these models are not interoperable in general, so a service retrieval system cannot combine several ranking mechanisms to provide more flexible and expressive facilities to define preferences.

Our proposal solves those identified issues of SWS discovery and ranking scenario by applying our highly expressive semantic preference model that allows the integration of different ranking mechanisms adapting the PURI framework, which is presented in this chapter. Consequently, our contribution offers a series of features that can be summed up as follows:

- **Flexibility**. The integration of every available ranking mechanisms using a common preference model allows the user to choose which preference facilities need for each request, without knowing the underlying ranking mechanisms that will be required to actually rank retrieved services.

- **Ease of use**. Final users do not need to access each ranking mechanism separately if they want to combine their results. A single entry point is provided in our solution for users to define their preferences and process them to rank the retrieved services.

- **Efficiency**. PURI provides a lightweight integration solution that does not add any noticeable performance penalty to the ranking process performed by each mechanism alone.

Furthermore, we performed a validation of our proposal contextualized in the SOA4All European R+D project, in addition to other validation scenarios where PURI have been successfully applied, such as public administration service infrastructures and proposed scenarios from the SWS Challenge. Particularly, in SOA4All, we have integrated three different ranking mechanisms, namely objective multi-valued ranking, NFP-based multi-criteria ranking, and fuzzy based ranking. Furthermore, our solution to this scenario provides a single user interface to define requirements and preferences, simplifying their definition and offering a unique entry point for the whole service discovery and ranking system, no matter what ranking mechanisms will be needed in the process. Chapter 8 further describes this application scenario.

Using our preference model and adapting it to a concrete service retrieval scenario enables interoperability between ranking mechanisms, fulfilling the challenge *C4*. Moreover, the system integration features provided by the application of our PURI framework solve the integrability issues that we identified as challenge *C5*.

The contributions described in this chapter has been published in several research conferences and journals. An early approach to the hybrid architecture discussed in §5.2 was presented in [31]. Then, several ranking prototypes that integrate logic programming rules mechanisms with constraint programming techniques were implemented and showcased in [34, 35, 37]. One of these prototypes, UPSranker, is available at `http://www.isa.us.es/upsranker`, along with more information and detailed documentation. Finally, design and implementation of the PURI framework discussed in §5.3 has been also presented in [40].

Part III

Evaluation of Results

VALIDATING THE PREFERENCE MODEL

. . . no matter how many instances of white swans we may have observed, this does not justify the conclusion that all swans are white.

Karl Popper (1902–1994)
Austrian philosopher

When evaluating to what extent an ontological model allows the representation of the knowledge from a certain domain, that model should be validated using widely adopted use scenarios. In this chapter, we present the validation of SOUP preference model, using a scenario proposed by the research community in the SWS Challenge, so that we can evaluate if our proposal meets the requirements identified to describe preferences in user requests. Thus, §6.1 describes the performed experiments to evaluate SOUP. Then, §6.2 showcases the validation scenario, proving that our preference model is suitable to define highly expressive and complex user preferences. Furthermore, we discuss additional scenarios that have been used to further validate our model in §6.3. Finally, §6.4 sums up the chapter discussing the conclusions of our carried out experiments.

6.1 INTRODUCTION

Our preference model, described in detail in Chapter 3, has to be validated in order to determine its applicability and usability in real use case scenarios. We perform this validation using one of the discovery scenarios from the SWS Challenge[1]. This Challenge is an initiative backed by the SWS research community, including mayor EU funded projects, institutions and enterprises, that provides a series of use case scenarios to validate and compare SWS mediation, discovery, ranking and composition mechanisms.

Concerning discovery and ranking use cases, there are three different scenarios on shipment, hardware purchasing, and logistics domain. Concretely, the chosen one has been the Logistics Management scenario, because of its higher complexity and the inclusion of preference descriptions. It consists on seven logistics service offers, described in natural language in terms of different properties, such as price, covered geographical areas, operating hours and truck fleets, among others. Additionally, several service requests (*i.e.* user goals) applicable to this scenario are defined, which contain both hard requirements and user preferences (they are referred as *soft constraints* in the scenario) that can be used to choose the most appropriate service (*i.e.* the best one in terms of preferences) among those which fulfill hard requirements.

In the chosen scenario, goals B1, C1, D1 and E1 define a variety of preferences against different service properties, in addition to describe how preferences should be combined within each goal. In order to validate SOUP preference model using this scenario, we provide in the following equivalent instantiations for each of these goals using the proposed ontology model. Thus, textual descriptions of goals directly extracted from the scenario description are shown alongside their equivalent representation using the preference ontology presented in Chapter 3. For the sake of simplicity, service properties are included as instances inside the same default namespace as the goal, though a domain ontology should be externally defined, covering all the existing properties in the Logistics Management domain.

[1] `http://sws-challenge.org/`

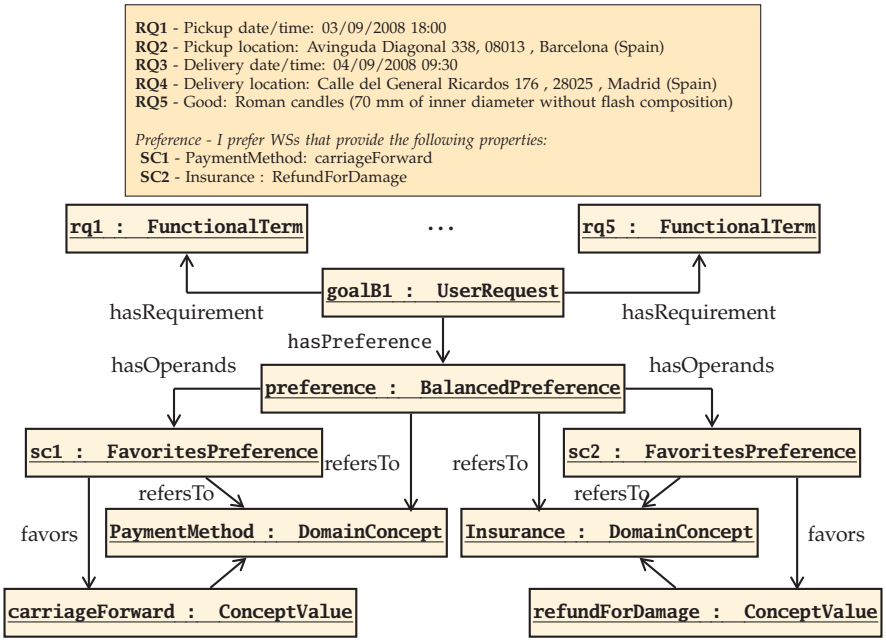


FIGURE 6.1: Goal B1 description excerpt and its SOUP instantiation.

6.2 VALIDATION WITH THE LOGISTICS SCENARIO

Figure 6.1 presents the instantiation of the goal B1 from the scenario. The goal as a whole is modeled with an instance of `UserRequest`, while each term is instantiated depending on its nature. Thus, requirements about pickup, delivery and transported goods are represented at the top of the figure. This representation is shown simplified, because requirements in every goal from the Logistics Management scenario are pairs between domain concepts and their required values. Consequently, in the following instantiated goals, requirements are omitted from the representation, though they can be easily described using functionality terms interpreted as property-value pairs.

Concerning the preference modeling, goal B1 states that the user prefers two properties, namely `PaymentMethod` and `Insurance`, to contain certain values, *carriageForward* and *refundForDamage*, respec-

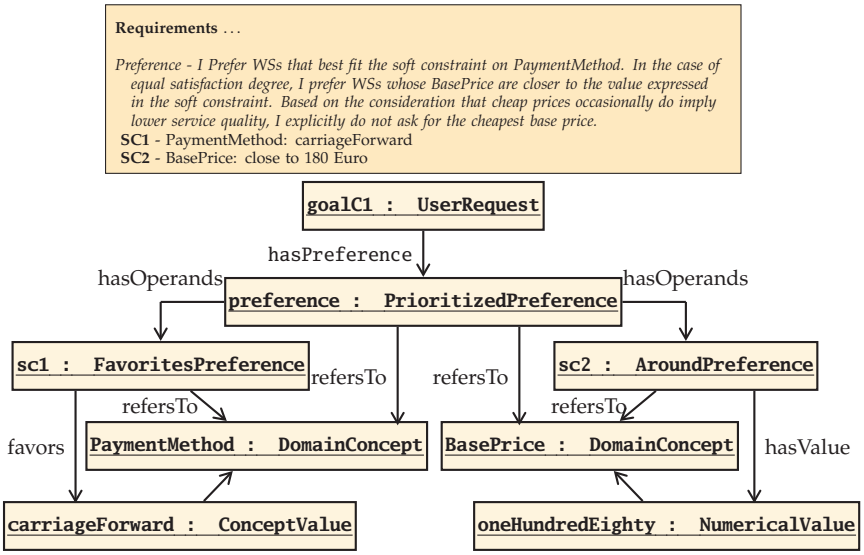


FIGURE 6.2: Goal C1 description excerpt and its SOUP instantiation.

tively[2]. Both of these soft constraints are considered qualitative preferences that define the favorite values for each property. However, the preference description do not explicitly express how to compose those two atomic preferences, so it can be inferred that a *balanced preference* can be applied to relate each one, because both atomic preferences can be considered equally important for the user.

The next goal used to validate our model is shown in Figure 6.2. In this case, the atomic preferences defined in C1 are instantiated as a *favorites preference* and an *around preference*. Moreover, the preference description gives more importance to the favorites preference on the `PaymentMethod` property, taking into consideration the preference about the `BasePrice` only if services have an equal satisfaction degree when evaluating the first preference. Thus, the most appropriate composite preference is a *prioritized preference*, because its semantics are exactly what the user is looking for in this goal. Note that, for the sake of clarity, we deliberately omitted the order of operands for the prioritized preference, though it should be taken

[2]For each example description, *italics* text correspond to service property values or instances used as operands, while `typewriter` text are used to denote domain concept classes that represents those properties, as in Chapter 3

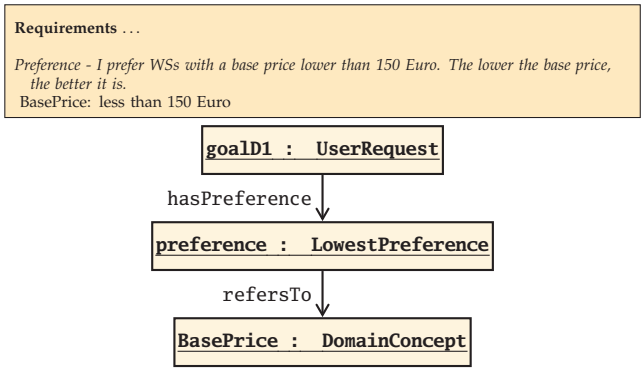


FIGURE 6.3: Goal D1 description excerpt and its SOUP instantiation.

into account using RDF lists, for instance.

Goal D1, which is represented in Figure 6.3, is the most simple goal of the scenario. There is no composition of atomic preferences, because it only states that the `BasePrice` should be as low as possible. The limit for the price that is included in the scenario is not necessary in our solution, because the semantics of the *lowest preference* is sufficient in this case to properly rank services with respect to the stated user preferences. Nevertheless, it is possible to take that price limit into account by modeling the user preference as a *prioritized preference*, where P1 is a *between preference* on `BasePrice` with the interval $[0, 150]$, and P2 is the *lowest preference* shown in Figure 6.3.

Finally, the most complex goal of the scenario is shown in Figure 6.4, where some of the `refersTo` relations are omitted for the sake of clarity[3]. The different atomic preferences are composed using *balanced preferences*, because the goal E1 description explicitly states that the user wants an average satisfaction degree among the atomic preferences. Notice that SC3 is a balanced preference decomposed into two *favorites preferences*, because it was interpreted that `Insurance` property should have both values. If SC3 were modeled using only one *favorites preference* with the two values in the favorite set, then services that supports only one type of insurance would be

[3]Actually, this relation can be inferred from the type of the operands involved in each preference.

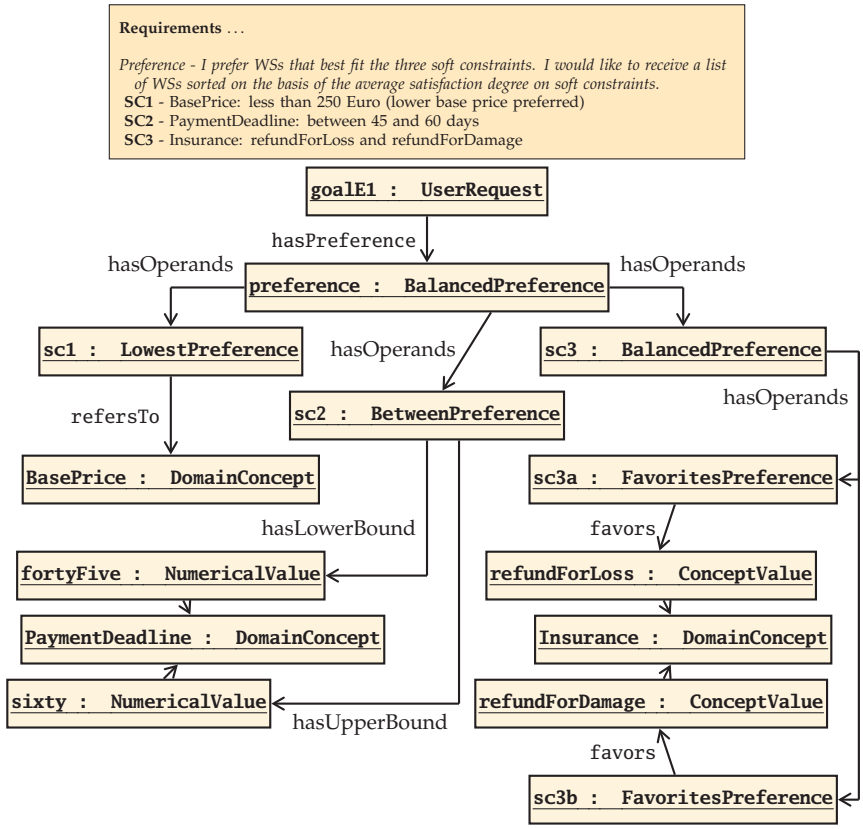


FIGURE 6.4: Goal E1 description excerpt and its SOUP instantiation.

considered equally preferred than those supporting both insurance values.

6.3 ADDITIONAL VALIDATION SCENARIOS

Ontology model validation may be performed either formally or using more practical approaches [42]. On the one hand, formal validation is based on model checking techniques and evaluation of ontology properties. On the other hand, practical approaches apply use cases, where ontologies are applied to modeling scenarios to check if they suffice to express all concepts involved in the use case scenario. These scenarios may be defined synthetically, where the

research community reach a consensus on a use case definition, or realistically, using actual use cases from applied projects, mainly.

SOUP validation has chiefly been performed using use cases, though a formal validation of an analogous preference model is discussed in [52]. In addition to the complex, synthetical discovery scenario previously discussed, we have applied our model to different real scenarios. Particularly, we successfully adapted our model to the PLATINA-FAST service trading system that is being implemented for the Regional Administration in Andalusia, Spain. In this scenario, users should be able to define some preferences concerning dynamic NFP of public services of the Administration. Consequently, our model were successfully applied to allow the definition of quantitative and composite preferences on those properties.

Moreover, we also apply our SOUP preference model as the foundations to define both EMMA and PURI proposals. Although EMMA only applies the upper ontology described in §3.2 to define the filters, PURI framework allows the adaptation of the whole preference model to a concrete ranking mechanisms integration scenario. Chapter 8 discusses a particular adaptation of the preference model to the SOA4All scenario, which also serves the purpose of further validating our proposal. Concretely, this application scenario extends some preference facilities to support three different ranking mechanisms, providing a common preference model to seamlessly integrate them.

6.4 SUMMARY

In conclusion, the presented validation using a relatively complex discovery and ranking scenario from the SWS Challenge proves that SOUP is sufficiently expressive and intuitive, obtaining a high degree of fulfillment of challenge *C1*. Furthermore, it allows to describe any kind of user preferences directly, user-friendly, and independently of the discovery and ranking technique to apply at a later stage, resulting in a low coupling degree (*C2*).

Additionally, the actual evaluation of the described preferences lead to the expected ranking results that are described in the scenario. This evaluation can be performed applying formal definitions

of the equivalent preference constructs from [52]. Further validation may be performed using other scenarios and test cases, such as the shipment discovery scenario used in [34], or the application to PLATINA-FAST project. Actually, Chapter 8 discuss another ranking scenario where the system integration is based on the adaptation of our preference model, serving this application as another real use case scenario to evaluate the utility of our proposed model.

APPLYING EMMA TO OPTIMIZE OWL-S MATCHMAKERS

For when one's proofs are aptly chosen,
Four are as valid as four dozen.

Matthew Prior (1664–1721)
English poet

Our proposed filters have to be thoroughly analyzed, using experimental results, in order to corroborate their soundness and expected benefits of EMMA. Each filter has been tested in different situations using SME², measuring several indicators to determine the actual improvements of our proposed preprocessing stage applied to actual OWL-S matchmakers. In this chapter, performed experimental evaluation is described in §7.1, while the obtained results are analyzed in §7.2. Then, §7.3 presents our interpretation and discussion of these results, which validates EMMA. Finally, in §7.4 we sum up the evaluation of EMMA and its success on dealing with our identified challenges. An additional evaluation using synthetic WSMO-based scenarios is presented in Appendix C.

7.1 INTRODUCTION

In order to experimentally test the suitability and performance of our filtering proposal, a proper test collection has to be used. There are some publicly available collections to evaluate service discovery algorithms for OWL-S and SAWSDL services. Particularly, we evaluate EMMA with respect to the OWL-S Services Retrieval Test Collection (OWLS-TC v3[1]). This collection contains 1007 OWL-S service descriptions from different domains, in addition to 29 user requests (referred as *queries*) and their corresponding sets of relevant services, so that, for each OWL-S query, the performance and effectiveness of matchmakers can be evaluated by checking whether returned services are relevant to the corresponding query or not.

In our experimental prototype of EMMA, SPARQL query execution was implemented in Java using the Jena Semantic Web Framework. First of all, input OWL-S service files are parsed and processed by Jena, which is able to execute SPARQL queries over them. Then, the results from the query execution are used to filter the list of services that take part in the subsequent discovery process.

Nevertheless, our proposal cannot be evaluated on its own, because it does not perform service discovery, but includes a preprocessing stage that filters repositories so that the subsequent service matchmaking can be improved. Thus, in order to evaluate the actual impact of proposed filters using OWLS-TC, they have to be tested on top of an OWL-S service matchmaker, so that the differences between using filters or directly performing the discovery process can be analyzed.

The actual evaluation of our prefiltering proposal has been conducted using the SME[2] v2.1[2]. SME[2] is an open source tool that can be used to test and compare several SWS matchmakers using the same test collection (OWLS-TC v3 in our case) as the input for each matchmaker. The variables measured by SME[2] that are used in our work to compare matchmakers are the following:

- **Precision.** The proportion of returned services that are actually relevant for the corresponding query. The more precision

[1] http://projects.semwebcentral.org/projects/owls-tc/
[2] http://projects.semwebcentral.org/projects/sme2/

a query execution presents, the more accurate the answer is.

- **Recall.** The proportion of the relevance set that is returned by a query. The more recall a query answer has, the more relevant services are returned by the corresponding query.

- **Fallout.** The proportion of non-relevant services retrieved by a query. In other words, it measures the amount of false positives returned by the corresponding query with respect to the complete answer set.

- **Query response time.** For each query, it measures the time a concrete matchmaker spends on evaluating that query and returning the corresponding results, without the initialization time needed for registering service descriptions.

- **Memory usage.** Measured samples of the amount of memory a matchmaker uses during its whole execution time.

Precision, recall and fallout are standard, well-known measures for evaluating information retrieval techniques [7]. In particular, SME^2 computes precision and fallout using a macro-averaged approach that sums up the results from all query executions. Thus, for each query, precision and fallout are measured at equidistant standard recall values, and then the mean value for these measures is obtained at each recall level. Nevertheless, the well-known *average precision* measure is also computed for each single query, enabling performance evaluation regardless of the number of services returned by the matchmaker. The mean average precision is discussed, along with the others measures, in §7.2.

Our EMMA prototype implements the `IMatchmakerPlugin` interface so that it can be plugged into SME^2, though it has to be associated with another matchmaker that is called using the same interface to actually perform SWS discovery after prefiltering the input. For evaluation purposes we have chosen some variants of OWLS-MX, which is a hybrid SWS matchmaker that combines both logic-based approaches and information retrieval techniques for a high performance discovery [54]. Each chosen variant is firstly executed as is, and then with $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$ filters on top of it. Thus, the different

TABLE 7.1: Average query response times and precision.

Matchmaker	Avg query response	Avg query precision
OWLS-M0	57332 (±1592) ms	49.55 (±6.70) %
+ $\mathcal{Q}_{all}$	1283 (±57) ms	31.62 (±6.20) %
+ $\mathcal{Q}_{some}$	6333 (±3023) ms	68.13 (±7.49) %
OWLS-MX3 (M3)	58456 (±214) ms	82.96 (±4.50) %
+ $\mathcal{Q}_{all}$	1321 (±61) ms	31.45 (±6.15) %
+ $\mathcal{Q}_{some}$	5500 (±2810) ms	72.02 (±6.28) %

combinations of a OWLS-MX variant and (possibly) a corresponding filter are compared against each other in order to evaluate the performance of our proposal in different situations.

For the sake of brevity, in the following we only compare the performance results of two different OWLS-MX variants, namely *OWLS-M0* and *OWLS-MX3 (M3)*, because the other variants present similar results to the latter. OWLS-M0 variant is a simple, logic-based matchmaker that only uses reasoning techniques, while the OWLS-MX3 (M3) adds text similarity matching to avoid false positives and improve the precision of the results.

7.2 ANALYZING TESTS RESULTS

Firstly, we analyze the performance improvement obtained by using our proposed filters before service discovery. Table 7.1 presents a summary of the evaluation performed where both OWLS-M0 and OWLS-MX3 (M3) variants are compared in terms of their average execution time and mean average precision for all OWL-S queries of the test collection, along with confidence intervals calculated using a confidence level of 95%. Most query response times are highly improved when using any of the filters, though $\mathcal{Q}_{some}$ filter impact is lower because it returns more results as shown in Figure 7.1. Noteworthy, actual filtering time does not affect the overall OWL-S query response time, because our proposed SPARQL queries can be executed in polynomial time by SPARQL implementations [70].

Experimental results show that, on average, response time of OWLS-M0 is 44.7 times faster if applying $\mathcal{Q}_{all}$ filter, and about 9

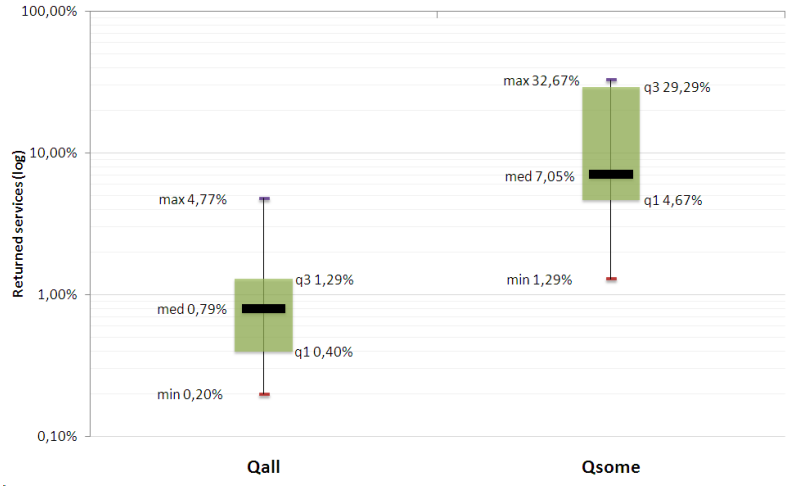


FIGURE 7.1: Returned results with respect to the original repository size.

times faster if $\mathcal{Q}_{some}$ filter is the applied one. OWLS-MX3 (M3) performance is similarly improved (44.3 times faster with $\mathcal{Q}_{all}$ and 10.6 times faster with $\mathcal{Q}_{some}$). Even though the confidence interval in $\mathcal{Q}_{some}$ cases is large, in the worst case scenario, the execution is at least 6.1 times faster when using OWLS-M0 matchmaker, and 7 times faster for OWLS-MX3 (M3).

Despite its high time performance, $\mathcal{Q}_{all}$ filtering shows worse performance in terms of average precision than the rest of the evaluated alternatives, providing an average value of about 31%. In turn, $\mathcal{Q}_{some}$ shows a better average precision on all the evaluation tests than $\mathcal{Q}_{all}$. Thus, for logic-based OWLS-M0 variant, $\mathcal{Q}_{some}$ filtering presents an improvement of about 19% on precision with respect to the execution of OWLS-M0 with no preprocessing. For OWLS-MX3 hybrid variant, average precision only drops by 11%, though response time is considerably faster. Note that average precision measures have a strong dependency on the concrete query and services registered in the repository.

Response time improvements are correlated to the degree of filtering each filter is able to provide. Figure 7.1 presents a logarithmically-scaled box plot that analyzes the proportion of services re-

turned for the 29 queries from OWLS-TC with respect to the initial repository of 1007 services. In general, $\mathcal{Q}_{all}$ filter returns a very low number of services (most queries returning between 0.4 and 1.29% of the original repository), greatly improving query response time as discussed before. On the other hand, $\mathcal{Q}_{some}$ filter results vary between a bigger range, with a median value of 7.05 % of the original repository, so the corresponding query response time for each matchmaker is slightly slower when using $\mathcal{Q}_{some}$ filter than when using $\mathcal{Q}_{all}$. In particular, some OWLS-TC queries present a lower filtering degree when using $\mathcal{Q}_{some}$, causing a noticeable variation on the response time that explains the larger $\mathcal{Q}_{some}$ confidence interval shown in Table 7.1. Additionally, the discovery process presents less initialization time because the number of services to be loaded by matchmakers is significantly low, especially when $\mathcal{Q}_{all}$ filter is applied.

Furthermore, the performance gain in terms of memory consumption is presented in Figure 7.2, where samples from the execution of OWLS-MX3 (M3) variant are only showcased, for the sake of clarity. Results show that filtering the repository leads to a lower memory usage, because less resources are needed to perform the actual matchmaking. On average, OWLS-MX3 (M3) needs 1.5 times less memory if $\mathcal{Q}_{some}$ filter is applied, and 2.8 times less if filtering with $\mathcal{Q}_{all}$. In conclusion, the use of our proposed filters substantially improves the overall performance of OWLS-MX matchmaker hybrid variants, both in terms of response time and memory consumption, though the impact on precision, recall and fallout has to be evaluated.

In order to analyze the penalty on precision and recall, Figure 7.3 compares the macro-averaged precision of the two discussed OWLS-MX variants when different filters are applied (*i.e.* using $\mathcal{Q}_{all}$, $\mathcal{Q}_{some}$, or no filter, respectively). It is observed that when prefiltering the repository using $\mathcal{Q}_{all}$, both OWLS-MX variants behave similarly. Precision in this case drops at a high pace as the recall level increases, performing much worse than the rest of the combinations, though at the highest recall levels $\mathcal{Q}_{all}$ filtering slightly improves precision over OWLS-M0 (Figure 7.3(a)) without filtering. The low number of results obtained when filtering repositories using $\mathcal{Q}_{all}$ query is the cause for this low precision.

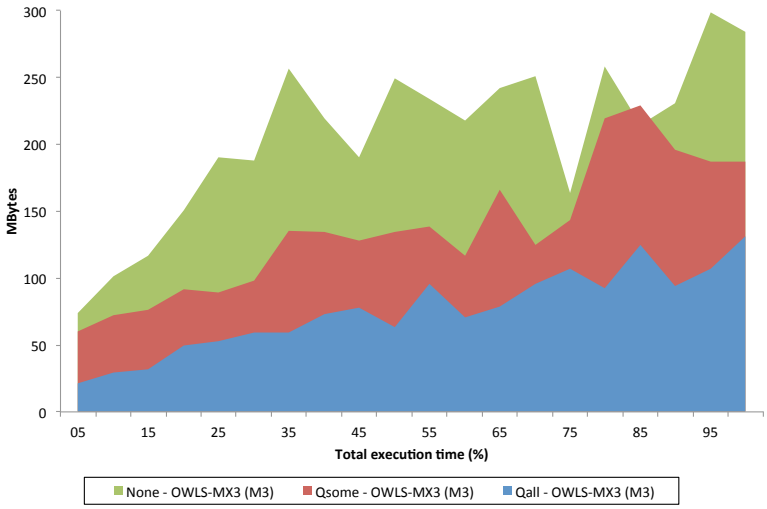
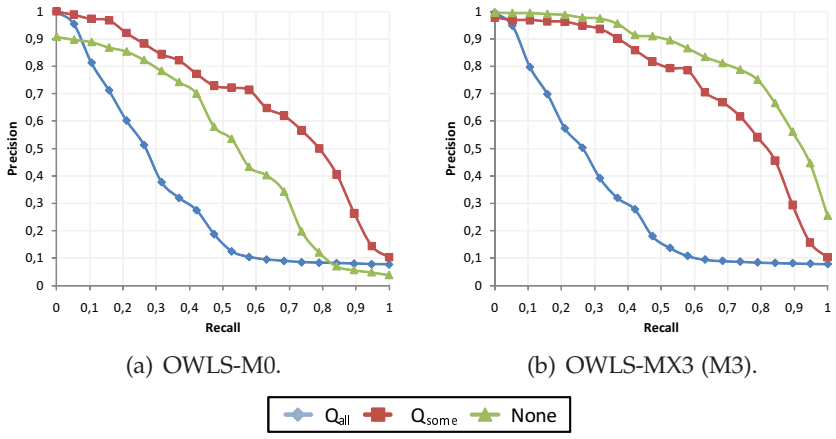


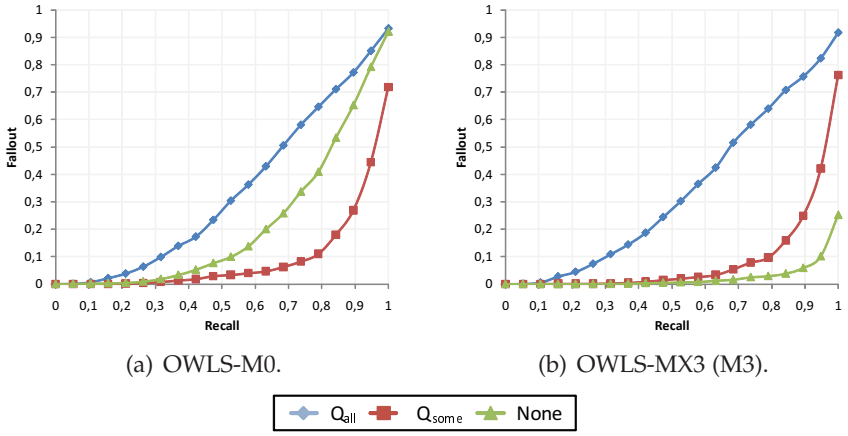
FIGURE 7.2: Memory consumption when filtering OWLS-MX3 (M3).



(a) OWLS-M0.

(b) OWLS-MX3 (M3).

FIGURE 7.3: Recall-Precision effect when filtering OWLS-MX variants.



(a) OWLS-M0. (b) OWLS-MX3 (M3).

FIGURE 7.4: Recall-Fallout effect when filtering OWLS-MX variants.

However, $\mathcal{Q}_{some}$ filtering performs reasonably well, with a loss in precision of at most 29% with respect to the precision obtained with OWLS-MX3 (M3) variant at high recall levels, as shown in Figure 7.3(b). Interestingly, the evaluation shows that applying $\mathcal{Q}_{some}$ filtering to OWLS-M0 variant improves the precision of the answered set (up to 38% of difference), especially with recall levels over 50%. Thus, the more accurate results obtained by $\mathcal{Q}_{some}$ filtering help purely logic-based formalisms to find more relevant services, while avoiding more false positives.

False positives returned by each compared variant, are represented in Figure 7.4 as fallout. $\mathcal{Q}_{some}$ filtering applied to OWLS-M0 again improves the results when compared to the results of OWLS-M0 without applying any filter, as shown in Figure 7.4(a). In the case of OWLS-MX3 (M3) (Figure 7.4(b)) fallout difference when applying $\mathcal{Q}_{some}$ filtering turns higher as recall level increases, especially from 70% on. As with precision, prefiltering repositories using $\mathcal{Q}_{all}$ query leads to much higher fallout levels, no matter the OWLS-MX variant used.

Obtained fallout performance results are a consequence of the prototype implementation of EMMA used to evaluate our proposal performance using SME^2, that requires each query result to be a ranked list of all the services that were registered in the system.

Thus, our prototype also includes those services that do not pass the corresponding filter at the end of the ranked list. Analyzing filtering results of both queries, if only filtered services are taken into account when evaluating the fallout for each case, fallout will drop to less than 7% for $\mathcal{Q}_{some}$, and 0.02% for $\mathcal{Q}_{all}$ filter. Thus, the amount of false positives in a generic discovery scenario is reduced by using our prefiltering proposal, in general.

7.3 Evaluation and Discussion

As a general conclusion from the performed evaluation, though the more restrictive $\mathcal{Q}_{all}$ filter may be better suited to filter because it reduces the size of the service repository to a greater extent, $\mathcal{Q}_{some}$ filter turns to be more suitable in general because the precision penalty is negligible while execution time is fairly improved, outperforming service matchmaking without applying any filter. In turn, $\mathcal{Q}_{all}$ filter scales well in every situation, though the greater loss of precision have to be considered, so it may only be applied in scenarios with really large repositories.

Both filters clearly improve the subsequent discovery stage by reducing the search space for matchmaking algorithms. However, there is a trade-off between precision, recall, and execution time that should be evaluated, depending on the concrete scenario, in order to choose the filter to use. Actually, the current trend in the literature and real-world applications is to achieve better performance and usability, by sacrificing precision, recall, or both [28], so our proposal provides a feasible and efficient solution in this direction.

The main feature of using our proposed filters is that not only total execution time is very low, but actual filtering is efficiently executed, providing a high scalability. Furthermore, the time needed for registering services for the matchmaking process is also reduced, because the number of candidate services are minimized after filter execution. Consequently, a hybrid architecture can be applied, where $\mathcal{Q}_{all}$ filter is executed in the first place. If after performing service matchmaking, the obtained results did not present sufficient quality, $\mathcal{Q}_{some}$ filter could be used in place, executing again the matchmaking process. Note that even in the worst case, *i.e.* applying both fil-

ters and the corresponding matchmaking for each filtered repository, the total query execution time is 7.5 times faster than the OWLS-M0 matchmaking process for the whole service repository, and 8.6 times faster than OWLS-MX3 (M3). This approach is similar to the Best-Matches-Only solution proposed in [52], where if the most accurate results are found (*i.e.* $\mathcal{Q}_{all}$ returns good enough results), they are used, but in other case fairly appropriate results (*i.e.* results from $\mathcal{Q}_{some}$) may also be useful.

Additionally, another mixed approach may be taken, where both filters are jointly used before discovery and ranking processes take part. Thus, $\mathcal{Q}_{all}$ may be used to filter services that refers to concepts from the hard requirements of the user request, *i.e.* terms that have to be fulfilled in order to consider the corresponding service as a candidate. Then, $\mathcal{Q}_{some}$ filter can be applied to obtain services that refers to some of the concepts used in preferences, *i.e.* terms that state how candidate services should be ranked after discovery. Consequently, both filters can be integrated into one that take into consideration the differences between requirements and preferences [38].

Concerning the user requests applied in our evaluation, OWLS-TC v3 only provides information about inputs and outputs. However, an OWL-S user request may also contain preconditions, results, functional classification, and non-functional properties, in general. Our proposal can be seamlessly applied to these different terms of an OWL-S profile description, or in general to any SWS user request, because they also refer to concepts from domain ontologies. For instance, conditional expressions can be simply analyzed in order to obtain which concepts appear inside them. An early prototype on filtering WSMO services described in [39] is able to obtain those referred concepts from conditions and rules described within a WSMO capability. The evaluation of that approach, which is included in Appendix C, presents similar results as the ones presented in this chapter, with respect to precision and improved performance of discovery when applying our proposed filters.

7.4 SUMMARY

In §1.2 we identified a series of challenges that should be taken into account to improve SWS discovery and ranking. In particular, *C3* accounts for optimizing service discovery mechanisms, improving their performance and scalability, but also independent solutions from concrete formalisms, user request models, and repositories *C2* (see Appendix C for another evaluation of EMMA applied to a WSMO-based scenario). The evaluation performed in this chapter shows that EMMA effectively optimizes any available discovery mechanism, providing a contained penalty on precision and recall that depends on the chosen filtering approach and the underlying matchmaking implementation that EMMA is applied to.

Although the evaluation of our proposal has been carried out using OWLS-MX variants as the underlying service matchmaker, EMMA can be easily adapted to any matchmaker that implements SME^2 interfaces. An evolution of the prototype implementation that allows to change the underlying service matchmaker was presented in the 4th International Semantic Service Selection (S3) Contest in 2010[3]. For this participation, EMMA was re-implemented as a configurable OWL-S matchmaking plug-in compatible with SME^2 2.1.1. Although the contest entry offers a similar precision as the EMMA prototype evaluated in this chapter, average query response time is worse than the prototype, because of the way SME^2 plug-ins register the available services. This issue is identified in the S3 Contest 2010 report, so the next version of SME^2 application (already available for this year's contest) allows the use of prefiltering techniques, such as our proposed solution.

[3] http://www-ags.dfki.uni-sb.de/~klusch/s3/s3c-2010-summary-report-v2.pdf

8

Applying PURI to Integrate Ranking Mechanisms

There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

Tony Hoare (1934–)
British computer scientist

In order to properly evaluate our solution to integrate ranking mechanisms, we adapt the PURI framework to a concrete scenario within the SOA4All research project, where three different ranking mechanisms are integrated. We further introduce our experimental evaluation scenario in §8.1. Then, §8.2 describes those three ranking mechanisms proposed in SOA4All, motivating the application of our proposal in this case. We discuss the required extension of our proposed preference model in §8.3. Implementation details on how to apply the PURI framework to this use case are introduced in §8.4, showcasing the prototype implementation of the SOA4All service discovery and integrated ranking. Finally, in §8.5 we evaluate the success of the application of PURI to the SOA4All use case, and §8.6 sums up our carried out evaluation and its results.

8.1 INTRODUCTION

In the SOA4All FP7 European project[1], a fully-fledged, semantically-enhanced infrastructure to describe, search, compose, and execute services is proposed to offer effective, scalable, and usable solutions in an envisioned world of billions of available services [24]. The service retrieval and ranking scenario proposed in SOA4All provides three different ranking approaches, namely objective, multi-criteria and fuzzy ranking mechanisms [88]. Each approach provides different user interfaces and preference expressiveness depending on the applied ranking mechanism. As a consequence, a SOA4All user cannot combine preferences from the three ranking mechanisms offered.

Comparing the three approaches with the challenges identified in §2.4, we conclude that, though they provide different levels of expressiveness, there exist interoperability issues between them that prevent their integration. Furthermore, users cannot choose which ranking mechanism (or combination of them) should be applied to different service requests, depending on the expressiveness and performance needed, for instance.

In order to take full advantage of the three developed ranking mechanisms in SOA4All, our PURI framework was applied to this scenario, so an integrated ranking was implemented using those mechanisms, adapting and extending the previously discussed preference model and developing a single user interface to perform the service retrieval and ranking scenario as a whole, allowing the definition of preferences based on the common adapted model. In the following we introduce SOA4All mechanisms and show how we have applied the PURI framework to integrate them into a single service discovery and ranking solution.

8.2 SOA4ALL RANKING MECHANISMS

In the subsequent paragraphs we give an overview of the three ranking methods developed in SOA4All. As we show at the end of this

[1] http://www.soa4all.eu

section, each approach has benefits and drawbacks in comparison to the other approaches.

8.2.1 Ontology-based Feature Aggregation for Multi-valued Ranking

The first approach ranks services based on objective features of Web services that can be automatically crawled and monitored [82]. For WSDL services, three independent ranking values are computed. The values are based on (i) crawl meta-data like the number of related documents, (ii) verboseness of WSDL documents (especially documenting parts), and (iii) monitoring data like availability and response time. These values are then combined with equal weights to one global rank. For Web APIs, a confidence score of a Web page describing a Web API is taken into account, only.

The global rank of services is independent of the user preferences and can be directly derived from the individual scores. Objective preferences can be applied for typical Web service meta-data, e.g., the related documents score, since it is mostly valid to prefer services with a high number of documents strongly related to that service over services with less related documents on the Web. Further, the WSDL metrics rank favors services with comments and descriptions in their WSDL service descriptions and the monitoring rank promotes services with high availability. The confidence score of a Web API denotes the likelihood of a resource to be a Web API. Obviously, services with higher values for this score are preferred. The global rank aggregates the individual values with equal weights. Then it is normalized to the interval $[0, 1]$, and is finally added as an additional service property to the service description.

8.2.2 Multi-criteria Ranking based on Non-Functional Properties

The second approach bases the service ranking on user defined preferences [87]. NFPs in offers and requests are specified by means of logical rules using terms of given NFP ontologies. The NFP model of descriptions is showcased in Listing 8.1.

LISTING 8.1: NFP model for the multi-criteria ranking.

```
Class nonFunctionalProperty
  hasAnnotations type annotation
  hasDefinition type axiom
```

Preferences of a request contain (i) the NFP of interest, (ii) its importance, (iii) the desired ordering (ascending, descending), and (iv) instance data of a desired service run. An ontology reasoner evaluates the rules during query time, e.g., IRIS[2] is used to evaluate WSML rules. The ranking scores for individual properties are normalized and aggregated to a global rank that determines the final ordering of services.

The novelties of the second ranking approach are the combined use of ontological representation of NFPs with multiple NFP dimensions and the possibility to justify the computed ranking by the provision of provenance information [88].

8.2.3 Fuzzy Logic Based Ranking Approach

The third service ranking mechanism advances the expressiveness of user preferences from the second approach [4]. User preferences and relationships between NFPs are expressed by a set of fuzzy if-then rules. The fuzzy logic based ranking mechanism features the following abilities: (i) express vagueness while formulating preferences using linguistic terms instead of crisp values, (ii) assign crisp property values to different categories by specifying overlapping fuzzy set membership functions that model these categories, and (iii) create complex preferences constructed by the combination of simple terms.

The value range of each property occurring in a preference must be categorized by a fuzzy set. Either given fuzzy sets are reused or the user customizes them according to their needs. For instance, in Figure 8.1 the property `PaymentDeadline` is modeled by the three membership functions "short", "fair", and "long", which are specified by three overlapping trapezoid-shaped fuzzy sets with varying payment deadline (PD) in horizontal and the degree of membership (d) between 0 and 1 in the vertical direction.

[2] `http://www.iris-reasoner.org`

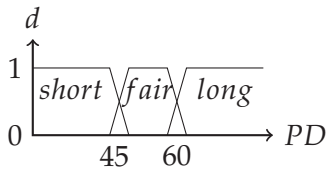


FIGURE 8.1: Example of membership functions.

The body of an if-then rule refers to a combination of property and fuzzy set pairs. Conjunctions, disjunctions, and negations are possible. The conclusion specifies the degree of acceptance that holds for the service if the condition in the rule body holds. For instance, "if PaymentDeadline=fair then acceptance=super" is a valid rule to favor services offering a fair payment deadline.

Service descriptions can be automatically classified in the fuzzy sets with a degree of membership information that can be computed and materialized in advance, independent from a particular request (fuzzification). Each fuzzy rule of a user preference of a request is processed in an inferring step and a degree of a rule's fulfillment is computed. In the aggregation step, chopped fuzzy sets in the conclusion of the rules are aggregated. The aggregated fuzzy set denotes the service rank as a fuzzy set, which is then defuzzified to a crisp value between 0 and 1 in order to obtain the actual rank [4].

8.2.4 Comparison of Ranking Methods

The first approach (objective) is clearly distinguished by its simplicity. It is similar to Google Web site ranking as the global rank can be computed off-line and independently from user preferences. Therefore, the ranking can be further exploited by other components like other ranking or retrieval mechanisms if top-k algorithms are applied. That is, the k most promising services (with a high global rank) are processed exclusively or privileged such that results can be delivered faster. The downside of the first mechanism is its limitation to a given set of properties that are observable by the crawler as well as the lacking ability of user customization, *i.e.* the expressiveness of available preferences is constrained.

The second ranking method (multi-criteria) overcomes the shortcoming of the previous one by providing a preference model and taking any ontologically defined NFPs into account. This method provides users simple means to express preferences on ascending and descending orderings with weighted aggregation into a global rank. However, this approach relies on the assumption of independent property preferences. That is, it cannot be expressed that a user accepts a higher price if a high quality is offered, for instance. The limited expressiveness of the preference model is therefore the motivation for the third method.

Fuzzy if-then preferences have a higher expressiveness. Dependencies between different desired properties as well as desired fuzzy value ranges can be specified. Further, users can express rather vague preferences by fuzzy sets. On the downside of this third approach is the increased computational effort that is required to compute a ranking, and the complexity of the preference definition.

As a conclusion, each SOA4All ranking mechanism serves a particular purpose depending on the level of expressiveness and flexibility the user needs when defining preferences for service ranking. For instance, in order to model the example discussed in §5.1, the multi-criteria ranking can be used to define and efficiently evaluate the price preference, whereas the fuzzy approach is useful to express the preference on the payment deadline. However, the combination of those two preferences is not possible in this use case, as it has already been identified in §1.2 as a challenge in SWS ranking. In this scenario our PURI framework can be applied to overcome the associated issues and effectively combine SOA4All ranking mechanisms, as described in the following.

8.3 Preference Model Adaptation

Before instantiating the PURI framework to provide an integrated ranking solution for the SOA4All use case, the different preference models offered by each ranking mechanism have to be integrated into our previously presented common preference model. Therefore, correspondences between our preference model facilities and those provided by SOA4All ranking mechanisms have to be identi-

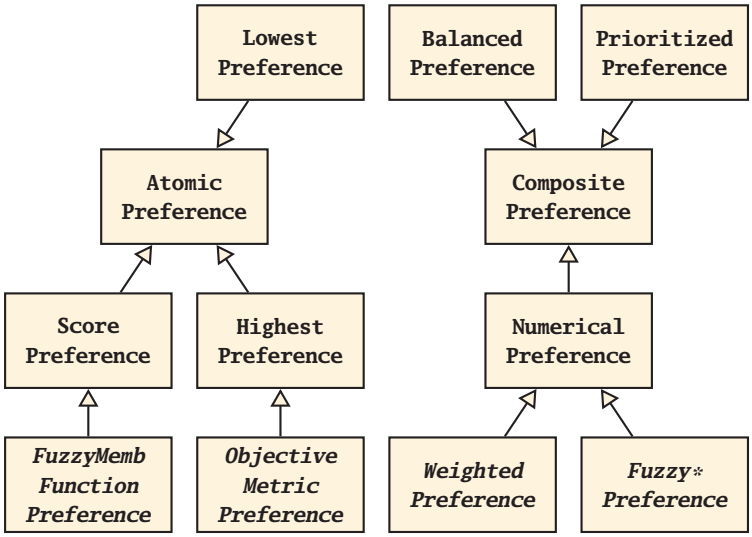


FIGURE 8.2: SOA4All adaptation of the preference model.

fied. Figure 8.2 shows the extended preference model that supports SOA4All facilities, where new preference terms with respect to the basic model described in Chapter 3 are depicted in italics.

In the first case, objective multi-valued ranking is based on metrics and derived ranks that can be used to order the retrieved services, where the resulting ranking should present at its top services with higher ranking values for the chosen metric. Consequently, we simply interpret a preference on a concrete `ObjectiveMetric` as a particular case of a `HighestPreference` that constraints the domain concept that can be referred to the available monitored metrics. For instance, a request may contain a preference where the user prefers services with higher global rank.

Concerning the NFP-based multi-criteria ranking mechanism, its own preference model allows the user to define the NFP of interest, which is identified as the `DomainConcept` that a preference refers to in our model as depicted in Figure 3.1. Depending on the desired ordering, the preference can be considered as a `Lowest` or a `Highest` one in the common model, which is extended by including an associated operand that can be used to define the relative importance as a float value. This importance value can be used itself to com-

pose several ascending or descending preferences, because it is used in the normalization and aggregation stage of this ranking mechanism. Therefore, we added a composite numerical preference called `WeightedPreference` that can combine several `Lowest` or `Highest` preferences provided that they define a corresponding importance value. Note that objective multi-valued ranking metrics can be also used as the referred NFP so that both ranking mechanisms can be easily combined using a `WeightedPreference`.

Finally, fuzzy logic based ranking mechanism provides fuzzy rules and membership functions as the basic constructs to define preferences. On the one hand, a fuzzy rule is interpreted as an specialization of a `NumericalPreference` whose combining function is defined by the fuzzy ranking algorithm. A `FuzzyRulePreference` contains a premise and a conclusion. Premises may contain fuzzy logic negations, disjunctions and conjunctions that are also considered specializations of `NumericalPreferences`[3], as they can combine different fuzzy membership functions. A conclusion also contains a fuzzy membership function that is interpreted as the fuzzy score of the rule. Furthermore, rules can also be combined in a `FuzzyGoalPreference`, that is interpreted as a particular `NumericalPreference`, too.

On the other hand, fuzzy membership functions are considered atomic preferences because they provide means to obtain a fuzzy score value depending on the value of a referred domain concept, such as price or payment deadline. In our common model, there exists a generic preference called `ScorePreference` that is defined after a real function that computes the score used to rank services (see [38]). Therefore, we model fuzzy membership functions as a particular case of a `ScorePreference` (denoted as `FuzzyMembFunctionPreference` in Figure 8.2), whose scoring function is precisely that membership function.

For instance, the example discussed in §5.1 can be modeled using the adapted common preference model as follows. In that example there are two atomic preferences that can be modeled using (1) a `LowestPreference` on the base price, as provided by the NFP-based

[3]In order to simplify Figure 8.2, all fuzzy preference composite constructors (rules, goals, negations, disjunctions and conjunctions) are denoted as `Fuzzy*Preference`.

multi-criteria ranking mechanism; and (2) a fuzzy membership function (a subtype of `ScorePreference`), which models the preference on the *fair* payment deadline, evaluated by the fuzzy logic based approach. Furthermore, both atomic preferences can be composed using a `BalancedPreference` (directly implemented by PURI), so that they are considered equally important for the user. A partial representation of this example described using Turtle [12] is shown in Listing 8.2.

LISTING 8.2: Excerpt of the adaptation of a user preference.

```
ex:userPreference
  rdf:type  soup:BalancedPreference;
  soup:hasOperands  ex:pricePreference,
                    ex:paymentDeadlinePreference.
ex:pricePreference
  rdf:type  soup:LowestPreference;
  soup:refersTo  logistics:BasePrice.
ex:paymentDeadlinePreference
  rdf:type  soa4all:FuzzyMembFunctionPreference;
  soup:refersTo  logistics:PaymentDeadline;
  soup:hasScoringFunction  ex:fairMembershipFunction.
```

8.4 APPLYING PURI TO SOA4ALL INTEGRATED RANKING IMPLEMENTATION

Starting from the model adaptation discussed in the previous section, the actual implementation of the SOA4All integrated ranking approach involved the application and extension of the PURI framework to develop a holistic solution to service retrieval that allows the combination of several ranking mechanisms, exploiting synergies and providing a better user experience by offering a single, unified user interface to the SOA4All service retrieval scenario.

First of all, each ranking mechanism interface was adapted to PURI ranking API. Essentially, each adapter supports all the corresponding preference terms that are handled by each ranking mechanism. Using a dynamic instantiation, PURI is able to identify which adapters have to be used to rank a set of retrieved services in terms of a user provided preference. Furthermore, PURI is also responsi-

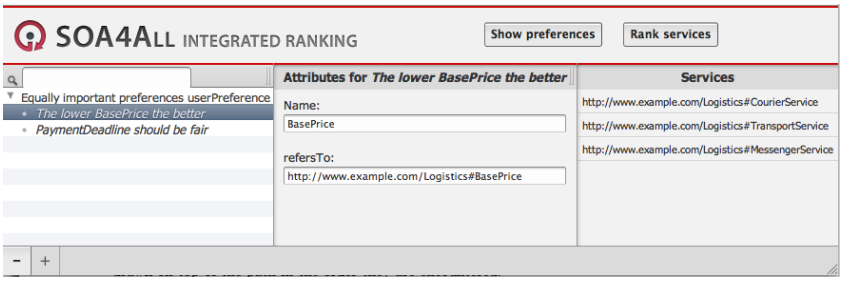


FIGURE 8.3: Screenshot of the preference definition user interface.

ble to orchestrate those adapters in order to combine ranking results from different ranking mechanisms in the event that composite preferences are specified by the user.

The developed SOA4All integrated ranking was deployed as a Web service itself, so that it cannot only be easily integrated within the global SOA4All service retrieval and ranking solution, but also be used as a standalone component to define preferences based on the discussed common model for the three ranking mechanisms proposed in SOA4All.

Finally, in order to apply our holistic solution to the SOA4All use case, it is necessary to put together both service discovery and integrated ranking implementations, integrating both components using a common user interface to the global SOA4All service retrieval system. A user can first enters criteria in order to filter the result set. A set of functionality classes from a tree-structured hierarchy can be selected. Multiple selection are interpreted such that desired services are member of all selected classes. Furthermore, the user may refine the desired service functionality with logic expressions describing inputs, outputs, pre-conditions, and effects. The desired values of NFPs can be constrained, too. Based on these requirements, the SOA4All retrieval component is able to determine the set of matching service descriptions.

In a second step, the user may specify preferences in order to rank services. Therefore, as depicted in Figure 8.3, the Web-based interface guides the user in expressing preferences with minimal knowledge about the syntax. The preference type (see Figure 8.2) is chosen from a predefined list and the referred NFP concept as well

as operands are entered in dedicated text fields. Finally a name is assigned to the preference that allows to construct composite preference structures more conveniently. Upon submit, the services are presented to the user in the ranked order.

8.5 Evaluation and Discussion

In order to evaluate if the application of PURI to integrate SOA4All ranking mechanisms addresses the challenges discussed in §2.4, we analyze the available ranking mechanisms in terms of those challenges, as presented in Table 8.1. Furthermore, we also analyzed the expressiveness of each proposal (*C1*) in order to evaluate the model adaptation, which is an additional validation scenario for our proposed preference model. The integrated ranking approach using PURI allows the user to define preferences using any of the facilities provided by each ranking mechanism, consequently offering a higher expressiveness even when compared to the fuzzy based approach, because of the possibility to combine other mechanisms (see §8.2 for a comparison of their associated models that justifies their corresponding expressiveness degrees).

Precisely, this combination leads to a completely interoperable ranking system based on the common preference model discussed. Although each ranking mechanism presented semantic models to define their preferences, the lack of an upper model made difficult their interoperation at the description level, giving a medium interoperability (*C4*). Using our common preference model, facilities from any ranking mechanism can be composed together, exploiting its synergies and providing the user with more control over the service retrieval and ranking process.

At implementation level, both multi-criteria and fuzzy based ranking mechanisms cannot be easily integrated because of the different underlying formalisms (*i.e.* they offer a low integrability degree), though the objective ranking can be transparently integrated with the service retrieval component, retrieving services already ordered by the computed global rank, that can be further processed by any of the other ranking mechanisms. However, the integrated ranking approach implemented using PURI is not only able to integrate

TABLE 8.1: Comparison between SOA4All ranking approaches.

Ranking Approach	*C1*	*C4*	*C5*
Objective Ranking	Low	Medium	Medium
Multi-criteria Ranking	Medium	Medium	Low
Fuzzy based Ranking	High	Medium	Low
Integrated Ranking	High	High	High

the three available ranking mechanisms in a unique service retrieval and ranking system, but also to orchestrate the ranking execution in terms of the concrete preferences defined by the user, providing a high integrability (*C5*).

Furthermore, as the integrated ranking approach developed using PURI is based on the three presented SOA4All ranking mechanisms, the integrated ranking performance depends on those mechanisms. The orchestration provided by the framework does not add a significant performance penalty, because it redirects ranking requests to relevant mechanisms, only including a post-processing to combine results from different ranking mechanisms that simply iterates over those results to obtain the ranking list. As a consequence, precision and recall of our solution is not affected by the combination that PURI performs, showing the same results as the obtained by the execution of single mechanisms separately.

Nevertheless, our proposal presents two particular limitations. On the one hand, in order to extend the integrated ranking system adding other ranking mechanisms, a proper adapter has to be implemented, possibly extending the preference model so that facilities provided by new mechanisms are integrated into the common model. On the other hand, if several ranking mechanisms can evaluate the same preference term, the user cannot specifically state which concrete mechanism should be used to rank with respect to that term. While the former issue can be solved in design time by solution developers, the latter can be considered a particular instance of a service ranking. Using this interpretation, the different ranking mechanisms should be described as candidate services, so that they could be ranked according to the user preferences on them. For instance, a user may prefer to rank services using more expressive

ranking mechanisms instead of more performing ones.

8.6 SUMMARY

The SOA4All service retrieval scenario serves our proposal as its main validation, because the adaptation of PURI solved the interoperability problem that ranking mechanisms offered by the project originally presented, fulfilling challenge *C4*. Initially, the user had to choose a specific model to define preferences, depending on the concrete mechanism to be used to rank services. However, the developed integrated ranking allows the effective combination of the three available mechanisms expressiveness by means of a common preference model, which serves as the foundations of a unique user interface to define user preferences for ranking in the service retrieval system. Consequently, the integrability challenge (*C5*) is also fulfilled by the presented application of PURI framework.

The prototype implementation of the integrated ranking component was developed within the SOA4All project, as an extension to the originally devised ranking approaches [4]. Moreover, the evaluation and development of PURI and its SOA4All adaptation is thoroughly discussed in [40], while source code and a published demo version can be reached at `http://www.isa.us.es/soa4all-integrated-ranking/`.

Part IV

Final Remarks

9

Conclusions and Future Work

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing (1912–1954)
British mathematician

Discovery and ranking have been considered key processes to achieve the vision of SWS, which aims at providing semantically-aware tools to properly manage the large amount of available knowledge in service repositories. As a consequence, great research effort has been put into SWS discovery and ranking solutions that effectively retrieve the best services regarding a user request. However, there are several challenges ahead to improve these processes and allow them to manage large knowledge repositories. Throughout this thesis dissertation we described three interrelated proposals that improve both discovery and ranking processes, providing a lightweight, integrated solution to fulfill those identified challenges. In this final chapter, §9.1 sums up main conclusions, while §9.2 discuss the publications obtained from our thesis work. Finally, §9.3 identifies remaining research challenges that are going to be tackled in our future research work.

9.1 Conclusions

Current discovery and ranking techniques present a number of issues that render them unusable or difficult to integrate in some scenarios where performance, scalability and interoperability are key points. On the one hand, preference models have to be treated as first-class citizens in SWS frameworks, and they have to remain independent from discovery and ranking techniques, in order to allow them to be interoperable. On the other hand, lightweight approaches and optimization techniques need to be applied in order to achieve the Future Internet vision of billions of services that would be able to be discovered and ranked from distributed service repositories. The following sentence summarizes our contribution to tackle these problems:

> *We provide an independent, lightweight preference model that serves as the foundation for an optimized and integrated solution to SWS discovery and ranking.*

In this dissertation, we first presented SOUP, a highly expressive preference ontological model that offers a series of facilities to define user preferences, independently of the discovery and ranking mechanisms to be used within a service retrieval scenario. Its intuitive semantics, based on strict partial orders, ease the definition of preferences by users, while providing complex facilities that allows the combination of atomic preferences. We thoroughly evaluated its applicability and extensibility, as it is not only validated against several use case scenarios, but also conforms the foundations of the rest of our thesis contributions.

The second contribution discussed consists on an improvement of discovery processes that filters service repositories before discovery mechanisms are executed, reducing the amount of service descriptions that the matchmaker has to compare with the user request. We developed a prototype, namely EMMA, that has been

successfully applied to OWL-S matchmakers using a comprehensive test collection. Our results show that EMMA effectively reduces discovery execution time, improving performance and scalability of discovery mechanisms while offering a contained penalty on precision.

Furthermore, this dissertation showcased another contribution named PURI, which is an integrated solution to service ranking, based on our preference model, that enables interoperability and integration of several ranking mechanisms into a single service retrieval system. PURI framework has been applied to a concrete scenario within the SOA4All EU FP7 project in order to evaluate its benefits. This application allows the seamless combination of three different ranking mechanisms, providing a single entry point to the whole service retrieval system.

9.2 Publications

We also want to remark that we have published several technical papers that cover all the contributions discussed in this dissertation. Figure 9.1 showcases these publications, that includes 10 technical papers in both international and national conferences and workshops, two research reports including a SOA4All project deliverable [4], as well as two contributions to top international journals, namely Journal of Web Semantics [41], which is in press, and Knowledge-Based Systems [40], which has been conditionally accepted with minor revisions at the time of writing. These results support the feasibility and quality of our thesis work. Furthermore, we have received more than thirty citations to our publications from other authors, further supporting the relevance of our research. .

9.3 Future Work

As we have separated our thesis contributions in three major areas, we have also identified remaining issues and future work for these three contributions. First, our preference model should be validated using additional scenarios outside the scope of SWS, in order to generalize the definition of user preferences in other application

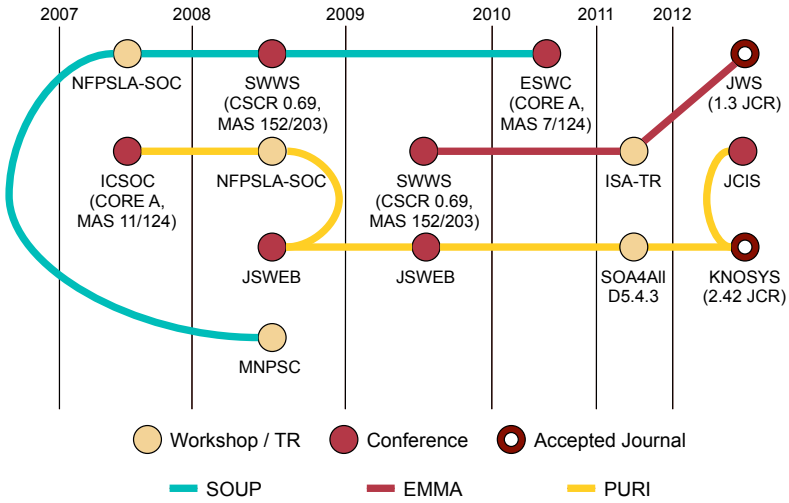


FIGURE 9.1: Publications derived from this thesis.

domain. For instance, Software Product Lines, configuration problems, and service level agreements are additional scenarios where there is a need for tools to model user preferences. Furthermore, we plan to achieve this generalization by adapting and publishing our preference model following Linked Data principles.

Concerning EMMA, we have also identified its generalization as future work. In this case we are implementing a generic prototype that filters large knowledge repositories, such as DBpedia [16], in order to improve query execution over those repositories. Preliminary results show a considerable improvement after an initial training period. Moreover, we plan to continue the development of EMMA as a SME^2 plug-in, so that a new version will be published for the next S3 Contest.

Finally, PURI can be also applied to integrate ranking mechanisms in different scenarios. Specifically, we plan to investigate how our approach can be applied to the more general ranking problem in the domain of information retrieval. Additional development should be done to adapt the available infrastructure after performing the generalization of the preference model that is also planned as future work.

APPENDICES

A

CONTRIBUTIONS TO THE SOA4ALL EU FP7 INTEGRATED PROJECT

Part of this thesis work was developed during our participation in the Service Oriented Architectures for All (SOA4All) EU FP7 Project, which is a Large-Scale Integrating Project funded by the European Seventh Framework Programme, under the Service and Software Architectures, Infrastructures and Engineering research area[1]. SOA4All aims at realizing a world where billions of parties are exposing and consuming services via advanced Web technology. Thus, the main objective of the project is to provide a comprehensive framework that integrates complementary and evolutionary technical advances (*i.e.*, SOA, context management, Web principles, Web 2.0 and semantic technologies) into a coherent and domain-independent service delivery platform [26].

For the last period of this project, our research group entered the consortium as a partner, in order to adapt our contributions described in Part II to the SOA4All scenario. We successfully applied SOUP preference model and PURI framework to the research performed within the Service Location work package. Chapter 8 describes that application, whereas SOA4All deliverable D.5.4.3 discuss more details on our contribution [4]. Moreover, the source code and implementation of our application has been made available in `http://www.isa.us.es/soa4all-integrated-ranking/`. Finally, we fairly improved our EMMA solution during our participation in this project, though it could not be applied because of budget limitations.

The validation of a significant part of our thesis under the umbrella of a major European Integrating Project remarks the soundness and quality of our research work, according to the received

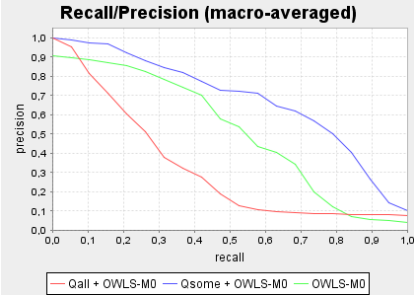
[1] `http://www.soa4all.eu`

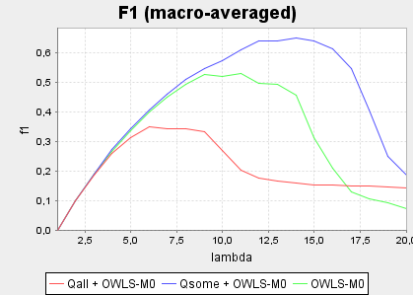
feedback of partners and the EU Commission committee that evaluated the whole project. Furthermore, our participation has led to several international contacts and opportunities for further collaborations that put the foundations for our future work after obtaining the doctorate degree.

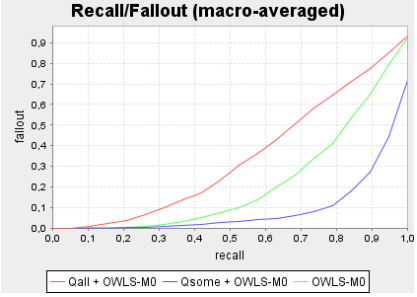
SME^2 Evaluation Report of EMMA

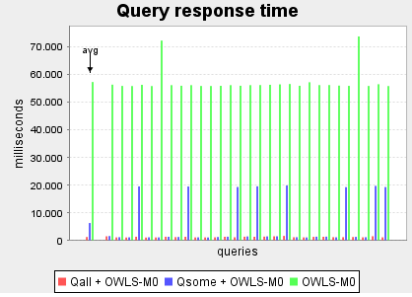
In order to complement the evaluation discussion already presented in Chapter 7, we reproduce in the following the output of SME^2 tests ran using five available variants of OWLS-MX. This evaluation report shows the effects of applying both EMMA filters to each OWLS-MX variant.

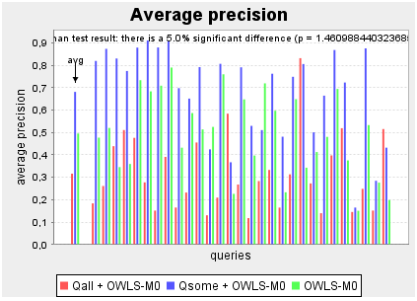
B.1 OWLS-M0

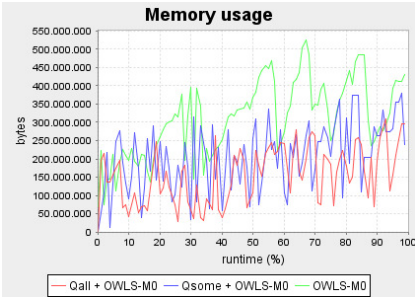




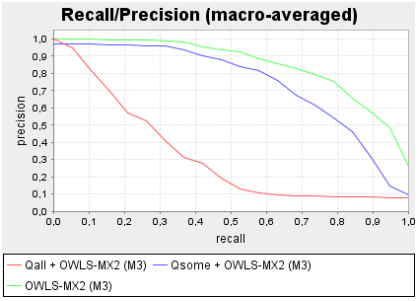


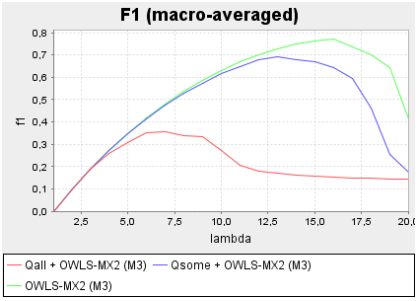


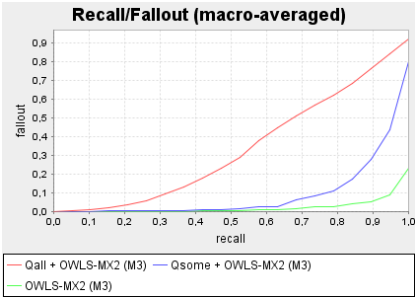


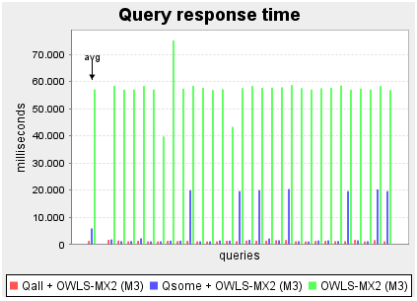


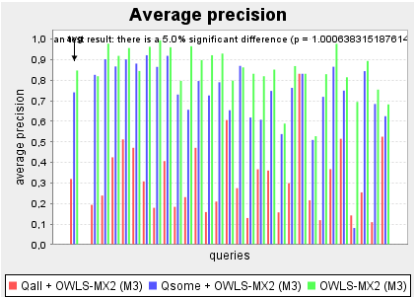
B.2 OWLS-MX2 (M3)

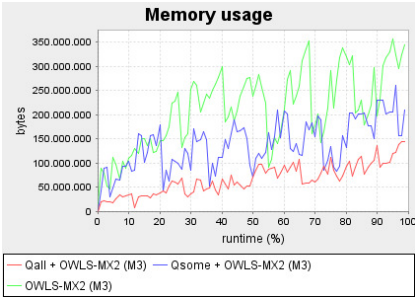




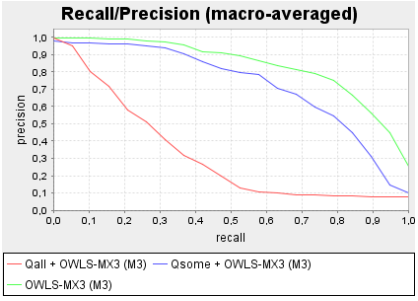


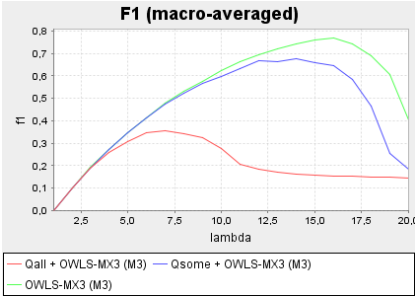


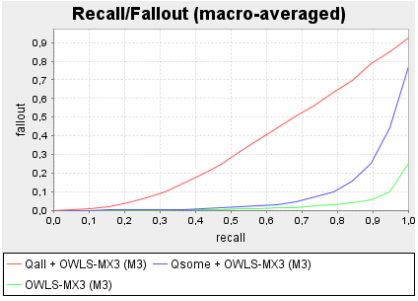


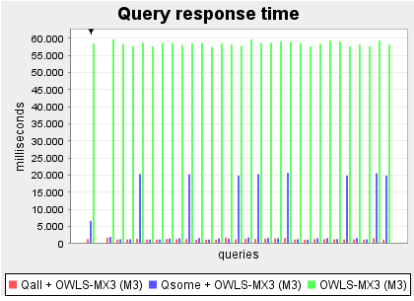


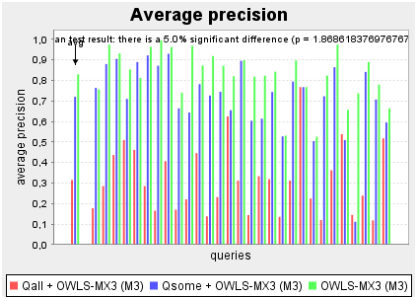
B.3 OWLS-MX3 (M3)

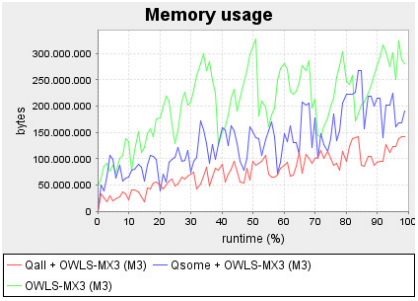




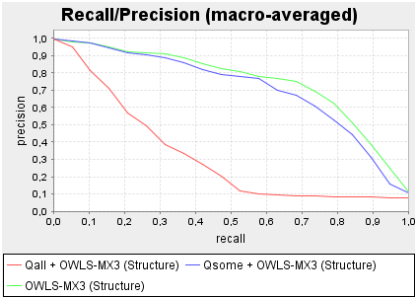


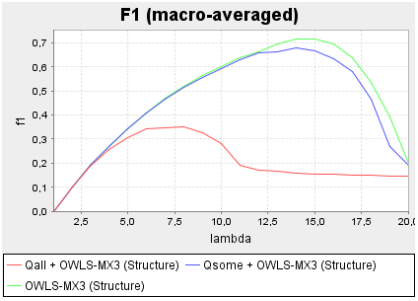


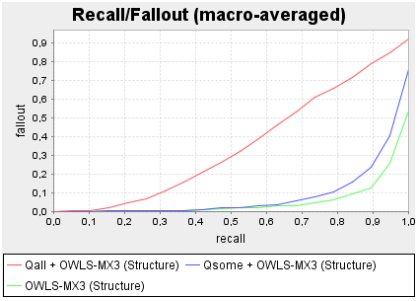


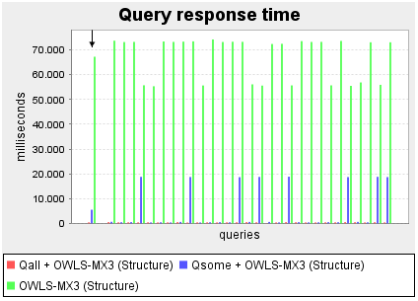


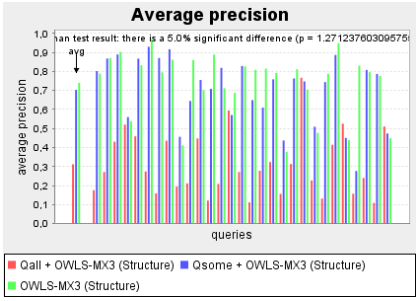
B.4 OWLS-MX3 (STRUCTURE)

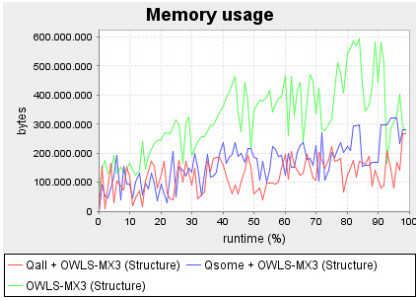




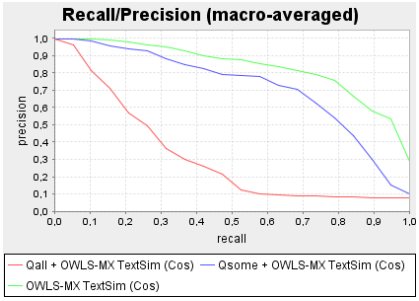


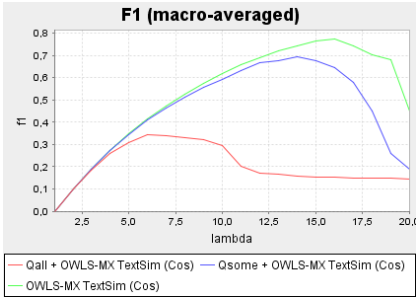


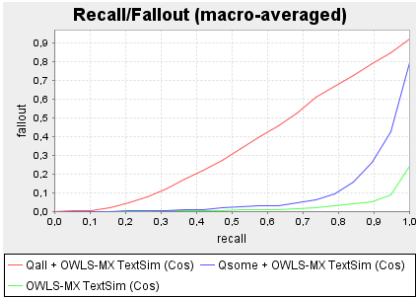


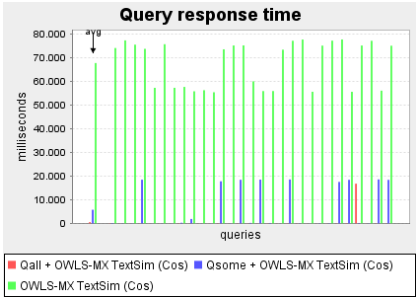


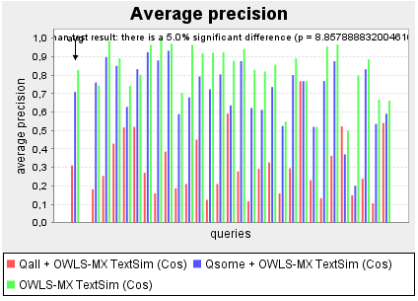
B.5 OWLS-MX TEXTSIM (COS)

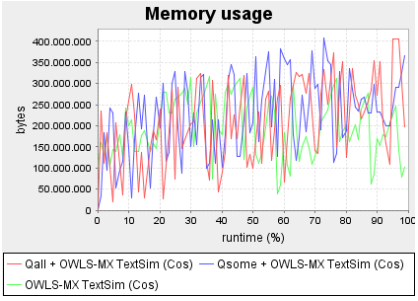









Average precision
average precision
queries
Qall + OWLS-MX TextSim (Cos)
Qsome + OWLS-MX TextSim (Cos)
OWLS-MX TextSim (Cos)


Memory usage
bytes
runtime (%)
Qall + OWLS-MX TextSim (Cos)
Qsome + OWLS-MX TextSim (Cos)
OWLS-MX TextSim (Cos)

C

SPARQL Filtering for Improving WSMO-based Discovery

An early version of our EMMA filtering solution to improve SWS discovery were developed using WSMO as the underlying framework. However, our final prototype were implemented as an OWL-S matchmaker because of the lack of a proper WSMO test collection and the features provided by SME^2 tool.

In this appendix we reproduce our evaluation results that prove that EMMA can be also applied to different SWS frameworks and discovery mechanisms than the chosen in Chapter 7. [39] discusses the WSMO-based solution in detail, including the specific filter definitions and mappings from our upper ontology of user requests.

C.1 Defining the Experiments

In order to test the suitability and performance of our WSMO-based proposal, there is a need for a test collection that can be used with the developed tools. Experiments were conducted within a WSMO discovery scenario, so services and user requests have to be described using WSML. However, a suitable, complex enough test collection of WSML descriptions is not available, so we developed a method to generate parametrized test collections, which could be used for performance tests. Thus, several service repositories and related ontologies were created for the experiments. In order to populate these repositories, each service description is generated using concepts from a Descriptions Logic ontology which contains a simple hierarchy of disjoint properties. Then, a goal is similarly generated.

Figure C.1 presents the already discussed discovery scenario that our experiments are contextualized in. The identified parameters,

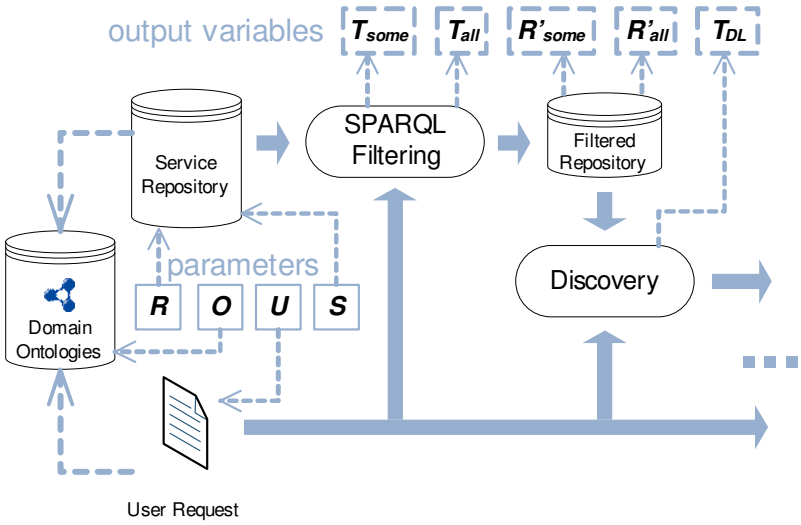


FIGURE C.1: Parameters and output variables of experiments.

shown using boxes, allow to test different situations varying their values. Each parameter have an influence in one of the input artifacts for the studied scenario, namely the service repository, the domain ontology and the user request, represented in Figure C.1 by the dashed arrows. Brief definitions and parameter ranges are enumerated in the following:

- **Repository size ($\mathcal{R}$).** The number of services stored in the repository is a parameter that ranged from 100 to 1,000, with a step of 100, for a total of 10 different values in the conducted experiments.

- **Domain ontology concepts ($\mathcal{O}$).** The number of the domain ontology concepts, which are doubled as instances of `Domain Concept` class from Figure 3.1, can be also parametrized. In our experiments, this number ranged from 20 to 100 concepts, incrementing by 20 for each step.

- **User request properties ($\mathcal{U}$).** Another parameter that varies in our tests is the proportion of properties referred by the user request (*i.e.* goals), with respect to the previous parameter, *i.e.*

the number of available properties defined in the domain ontology. Five different values were selected for this parameter, ranging from 5 percent to 25 percent. Higher values were not tested because it is unlikely that users define their requests using a high number of concepts, especially as the domain ontology size increases.

- **Service properties ($\mathcal{S}$).** Similarly, the proportion of properties referred by service descriptions is also parametrized, ranging as user request properties from 5% to 25%. As in the previous case, it is unlikely that services manage a lot of concepts, so it is not necessary to test higher values for this parameter.

The ontology representing the domain managed by services is populated with a concrete number of simple concepts ($\mathcal{O}$), depending on each generated repository ($\mathcal{R}$). These concepts are mutually disjoint, with the exception of a simple hierarchy that is randomly created within the ontology: a super-concept is chosen among all the concepts, and then a random number of concepts are declared as sub-concepts of the former. A scenario with a larger ontology represents a repository which could contain more heterogeneous services, *i.e.* described services offer many different functionalities. Moreover, a larger ontology also means a lower discovery performance, because its complexity increases.

Goals and services are created after the ontology, by selecting one concept that is going to be part of a simple post-condition of the corresponding capability, and, with some additional concepts (up to $\mathcal{U}$ and $\mathcal{S}$, respectively), they are directly included in a `refersTo` non-functional property of the element described. Note that concepts from the domain ontology are treated as functional or non-functional property depending on the case, because our proposal does not make any distinction between the nature of referred properties.

Additionally, properties referred by each service and the goal are selected using two different distributions, each one representing a different scenario. On the one hand, in the case of an uniform distribution of service properties, each property defined within the domain ontology has the same probability to appear in service de-

scriptions. Thus, this kind of repository reflects a situation where services may offer many different functionalities. Potentially, each concept from the domain ontology will be referred by the same number of services, *i.e.* the number of different functionalities among services in the repository will be approximately the number of concepts of the ontology.

On the other hand, a power-law distribution is also used to select which properties are referred by service definitions, so most of these definitions refers to a few common properties. Concretely, a Zipf distribution is used because it can be applied to our tested scenarios [2]. This distribution is based on the Zipf's law [94], which interprets that the frequency of any concept is inversely proportional to its rank in the frequency table, *i.e.* the most referred concept will occur twice as often as the second most referred one, which occurs twice as often as the following most frequent concept, and so on. In this case, repositories are fairly homogeneous, *i.e.* they contain many services with the same functionality, and there are few different functionalities. This scenario may be closer to real-world repositories than uniformly-distributed repositories, because in general service repositories are focused on a particular domain. However, larger and more general repositories may fall in between a uniform and a Zipf distribution of properties referred by their service descriptions, so it is worth to test both extremes.

For each experiment, several output variables have been measured. In Figure C.1, the following variables are showcased using dashed boxes, which are connected with dashed arrows to the measured artifacts and processes:

- **Filtering execution times ($\mathcal{T}_{some}$ and $\mathcal{T}_{all}$).** The SPARQL filtering stage execution time is measured for each corresponding query, so $\mathcal{T}_{some}$ contains the execution time in milliseconds of $\mathcal{Q}_{some}$, and $\mathcal{T}_{all}$ measures the same for $\mathcal{Q}_{all}$. These times actually includes both the repository serialization to WSML/RDF files and the query execution itself.

- **Filtered repository size ($\mathcal{R}'_{some}$ and $\mathcal{R}'_{all}$).** The filtered repository size is also measured for each query, correspondingly stored in $\mathcal{R}'_{some}$ and $\mathcal{R}'_{all}$ variables. These variables can be com-

pared to $\mathcal{R}$ to analyze to what extent the queries have filtered the original repository.

- **Discovery execution time ($\mathcal{T}_{DL}$).** After filtering, the discovery process is performed and its execution time is stored in the $\mathcal{T}_{DL}$ variable. In our experiments, this variable is measured in three different situations: (1) without filtering, (2) filtering with $\mathcal{Q}_{some}$, and (3) filtering with $\mathcal{Q}_{all}$, so that time improvement can be analyzed for each kind of filter.

C.2 Analyzing Tests Results

The implemented testing environment is able to generate several test collections and perform corresponding benchmarking tests at once. These tests were executed in a machine with Windows XP Professional SP3, Java 6, 2.4 GHz CPU and 2 GB of RAM. Furthermore, in order to thoroughly study the benefits of both queries in different situations, tests were conducted varying the four different parameters as described before.

Each combination of parameter values were used to generate two test repositories: one using a uniform distribution to pick up service properties, and the other using a Zipf distribution with 1.0 as its exponent. The whole generation, filtering and discovery process were executed 10 times for each parameter combination and distribution, for a total of 25,000 conducted experiments, measuring each output variable discussed in §C.1. Experimental results are detailed, analyzed, and discussed in the following.

C.2.1 Execution Time

Figure C.2 shows $\mathcal{T}_{some}$ and $\mathcal{T}_{all}$ varying $\mathcal{R}$, $\mathcal{O}$, and $\mathcal{S}$ parameters. [1] As $\mathcal{R}$ grows, total execution time of queries linearly increases, while it shows a greater slope as more concepts are present in service descriptions (higher values for $\mathcal{S}$ also confirm that behavior). $\mathcal{O}$ also affects the slope, in addition to a general increase in execution time as the number of concepts in the ontology rises. The more

[1] For the sake of clearness, intermediate values for some parameters are omitted in figures throughout this section.

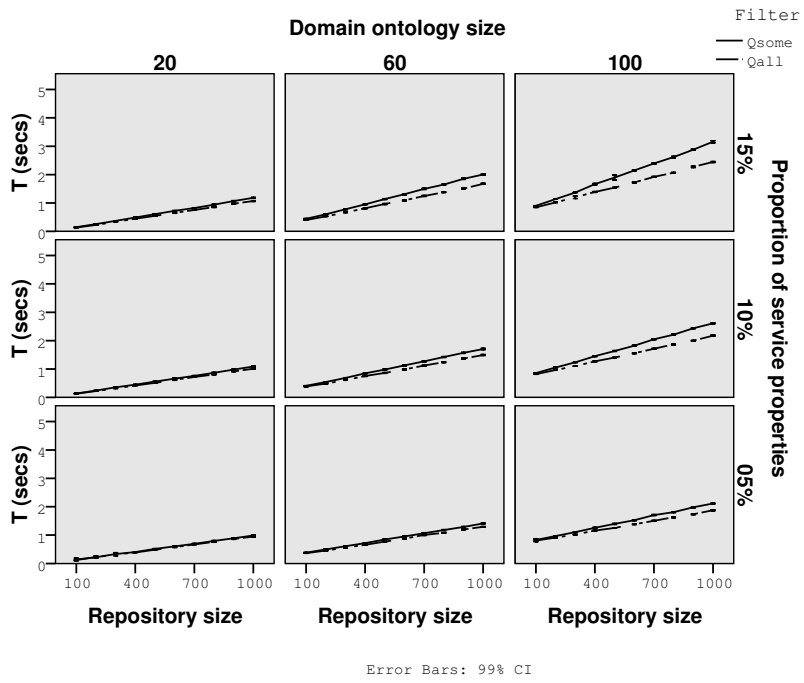


FIGURE C.2: Execution time results.

complex relations that can arise by using larger ontologies are the main reason for that behavior. Furthermore, query execution time significantly rises with higher values for both parameters.

In every test case, $\mathcal{T}_{some}$ (represented in Figure C.2 with a continuous line) is longer than $\mathcal{T}_{all}$ (the dash/dot line in the figure). Furthermore, the difference become larger with higher values for all the three parameters. However, with lower values, both queries tend to have a similar execution time. Note that for execution time, $\mathcal{U}$ does not affect at all, because they are not directly referred on any query, so they do not contribute to the complexity of each query execution (see §C.3 correlations discussion). Moreover, the distribution used to pick up properties for service descriptions does not affect query neither $\mathcal{T}_{some}$ nor $\mathcal{T}_{all}$.

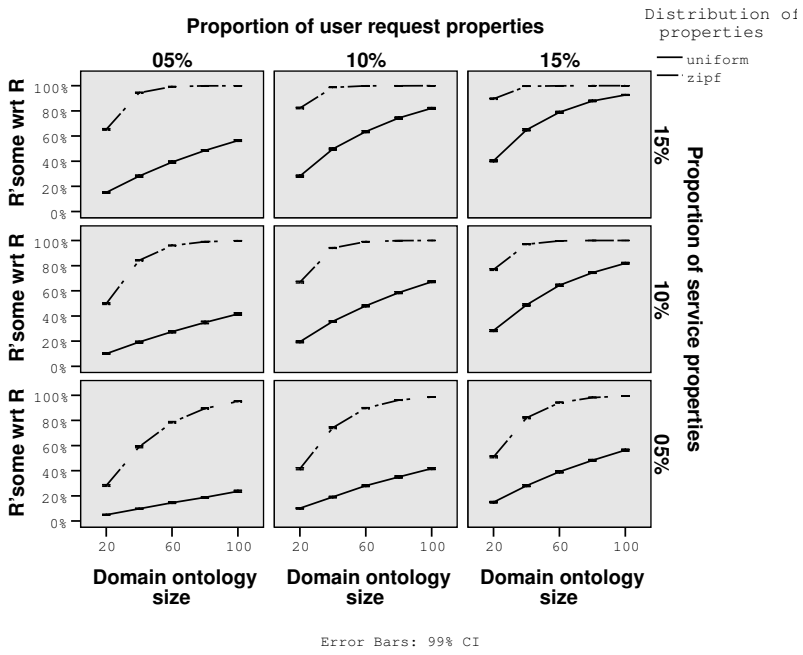


FIGURE C.3: Filtering results for $\mathcal{Q}_{some}$.

C.2.2 $\mathcal{Q}_{some}$ Results

In order to evaluate how $\mathcal{Q}_{some}$ performs, we measured the proportion of services returned by that query execution with respect to the $\mathcal{R}$ value for each experiment. Figure C.3 shows how $\mathcal{R}'_{some}$ behaves depending on $\mathcal{S}$, $\mathcal{U}$, $\mathcal{O}$, and the distribution of properties used to create each repository. As we are showing resulting repository proportions instead of number of services returned, $\mathcal{R}$ does not affect to $\mathcal{Q}_{some}$ performance evaluation, as expected.

In general, $\mathcal{Q}_{some}$ filters at a higher degree when the repository follows a uniform distribution of properties. In contrast, Zipf-distributed repositories are only filtered to some extent when the proportion of service properties has a low value (5%). As $\mathcal{U}$ and $\mathcal{S}$ increases, $\mathcal{R}'_{some}$ approaches the number of services in the original repository (100%). Furthermore, a higher number of $\mathcal{U}$ affects more to the performance decrease of $\mathcal{Q}_{some}$, because the more properties are referred by the user, the more services are likely to use one of them in their descriptions.

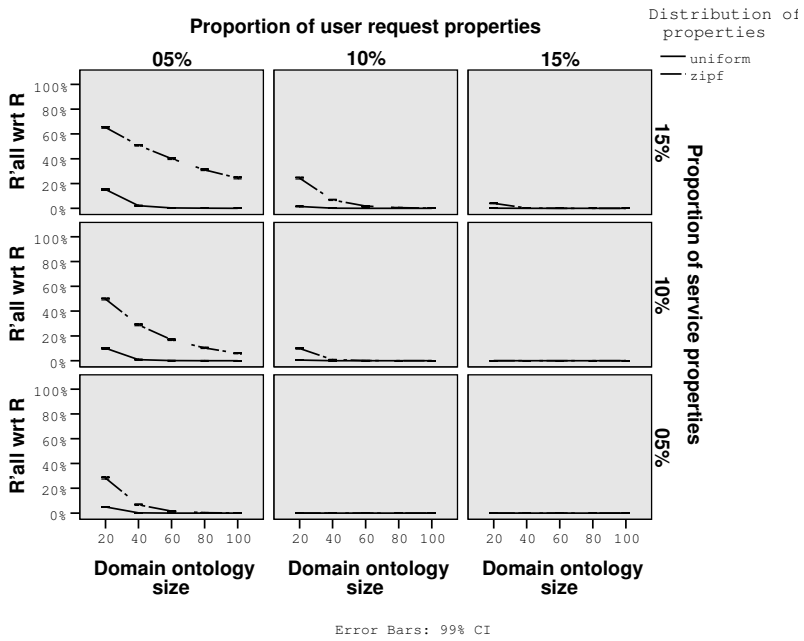


FIGURE C.4: Filtering results for $\mathcal{Q}_{all}$.

Increasing $\mathcal{O}$ mainly produces a higher $\mathcal{R}'_{some}$, which means that very large domain ontologies reduce filter performance. In conclusion, best scenarios for using $\mathcal{Q}_{some}$ are those where $\mathcal{U}$ is low, and there are not many concepts in the domain ontology (a low $\mathcal{O}$) but they are uniformly distributed among service descriptions.

C.2.3 $\mathcal{Q}_{all}$ Results

As in the previous case, $\mathcal{R}'_{all}$ with respect to $\mathcal{R}$ was measured for each test collection generated. Results are shown in Figure C.4, where a completely different behavior from $\mathcal{Q}_{some}$ is depicted. In this case, as $\mathcal{U}$ increases, $\mathcal{Q}_{all}$ filters more services (*i.e.* $\mathcal{R}'_{all}$ decreases). However, a higher $\mathcal{S}$ shows a less strict filter. As the nature of $\mathcal{Q}_{all}$ is inclusive, *i.e.* it looks for services that refer to *all* the properties of the user request, more concepts referred by service definitions cause that it is more likely that a service refers to all the properties of the user request.

Although $\mathcal{Q}_{all}$ filters many more services than $\mathcal{Q}_{some}$, higher val-

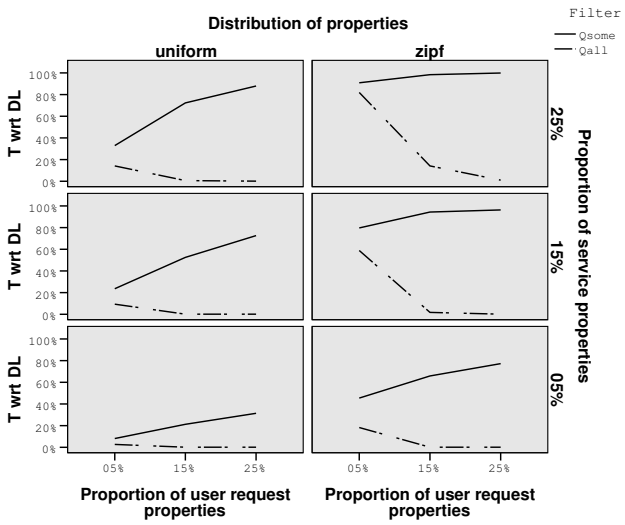


FIGURE C.5: Performance evaluation of discovery mechanisms.

ues of $\mathcal{O}$ or $\mathcal{U}$ actually make $\mathcal{Q}_{all}$ to return no results. This is even more noticeable with uniformly-distributed repositories. In this case, there might be some services in the original repository that would fulfill user requests to some extent. However, these service descriptions are not likely to contain every property of the user request, so $\mathcal{R}'_{all}$ tends to 0. Thus, the best situations for filtering repositories using $\mathcal{Q}_{all}$ query are those where both $\mathcal{U}$ and $\mathcal{O}$ have low values.

C.2.4 Discovery Improvement

The actual benefits of using the proposed filters in a discovery scenario is shown in Figure C.5. In this figure, $\mathcal{T}_{DL}$ values obtained when performing discovery after $\mathcal{Q}_{some}$ and $\mathcal{Q}_{all}$ filtering are compared with the case where no filtering stage is performed. Due to the DL discovery implementation used in experiments (Web Service eXecution Environment (WSMX) lightweight DL discovery), some parameters ($\mathcal{R}$ and $\mathcal{O}$) were fixed in order to get results in a reasonable time: a repository size of 600 service descriptions, and 40 ontology concepts were chosen. Furthermore, $\mathcal{T}_{DL}$ without any filter applied is constant, so the improvement can be measured depend-

TABLE C.1: Mean and std. deviation, and CI of evaluated variables.

	Uniform			Zipf		
	μ	σ	CI	μ	σ	CI
$\mathcal{R}'_{some}$	63.31%	27.90%	±0.64%	93.99%	12.95%	±0.30%
$\mathcal{R}'_{all}$	0.82%	3.37%	±0.08%	10.85%	21.40%	±0.49%
$\mathcal{T}_{some}$	1.27 s	0.77 s	±0.02 s	1.29 s	0.77 s	±0.02 s
$\mathcal{T}_{all}$	1.06 s	0.57 s	±0.01 s	1.08 s	0.58 s	±0.01 s

ing on the rest of the parameters (namely $\mathcal{U}$ and $\mathcal{S}$). Actually, $\mathcal{T}_{DL}$ increases linearly by $\mathcal{R}$, but exponentially by $\mathcal{O}$ [44].

When the service retrieval scenario includes $\mathcal{Q}_{some}$ as the choice for filtering the repository (continuous line), the $\mathcal{T}_{DL}$ improvement is noticeable, especially with lower values for $\mathcal{U}$ and $\mathcal{S}$, though in Zipf-distributed repositories the improve is not so accused. An increase on both $\mathcal{U}$ and $\mathcal{S}$ produces a higher $\mathcal{T}_{DL}$ in this case.

Finally, a filtering stage that uses $\mathcal{Q}_{all}$ before the DL discovery (dash/point line) shows a great improvement with respect to plain discovery. However, as shown in Figure C.4, with uniformly-distributed repositories, a low execution time may appear because $\mathcal{Q}_{all}$ returns no results, so the afterwards discovery does. However, with lower proportion of service and user request properties, the $\mathcal{T}_{DL}$ for this discovery mechanism is only, on average, a 14% of the plain discovery mechanism applied on a uniformly-distributed repository, and a 56% on a Zipf-distributed one.

C.3 STATISTICAL ANALYSIS

A complete statistical analysis have been performed on test runs in order to corroborate our expected results, and to further support the conclusions obtained from figures shown in this section. A summary of this analysis is presented in the following.

Main statistical descriptors are shown in Table C.1, for each measured variable in our experiments, with the exception made for $\mathcal{T}_{DL}$ because it has not been comprehensively tested. The analysis of these values shows that $\mathcal{Q}_{some}$ returns 63.31% of the services on average if their properties are distributed uniformly, though its high

TABLE C.2: Pearson correlations between evaluated parameters.

	Uniform				Zipf			
	$\mathcal{R}$	$\mathcal{O}$	$\mathcal{U}$	$\mathcal{S}$	$\mathcal{R}$	$\mathcal{O}$	$\mathcal{U}$	$\mathcal{S}$
$\mathcal{R}'_{some}$	0.000	**0.524**	**0.560**	**0.561**	0.000	**0.511**	**0.241**	**0.415**
$\mathcal{R}'_{all}$	-0.003	**-0.313**	**-0.321**	**0.141**	0.000	**-0.202**	**-0.618**	**0.345**
$\mathcal{T}_{some}$	**0.661**	**0.620**	0.019	**0.295**	**0.670**	**0.616**	0.019	**0.291**
$\mathcal{T}_{all}$	**0.692**	**0.642**	0.002	**0.229**	**0.700**	**0.630**	0.006	**0.231**

standard deviation is caused because of $\mathcal{Q}_{some}$ performance depends a lot on the repository parameters. The higher mean value of a Zipf-distributed repository shows that $\mathcal{Q}_{some}$ does not filter so much in that case. On the other hand, $\mathcal{Q}_{all}$ performs better, meaning that it filters on average more than $\mathcal{Q}_{some}$. Additionally, $\mathcal{Q}_{all}$ also filters more when service properties are uniformly distributed (it returns 0.82% of the original repository), in contrast to the case of a Zipf distribution (10.85%), though it could still be considered a good enough result.

Concerning execution time, values are very similar among the cases presented in Table C.1, with $\mathcal{Q}_{some}$ lasting about 0.21 seconds more than $\mathcal{Q}_{all}$. In this case, we can conclude with a high confidence that, on average, $\mathcal{Q}_{some}$ execution has a penalty time of at most 2.04 seconds ($\mu + \sigma$). However, note that this time includes the RDF serialization needed to use SPARQL with our test repository, so if the repository used allowed to be directly accessed in RDF, that penalty time could be significantly shorter.

Very narrow confidence intervals (CI), computed using a 99% confidence level, shows that, for every variable, mean values can be considered to be robust enough, so they can be used to summarize our experimental results. Thus, if our proposed filters were applied to real scenarios modeled like our test collections, the performance could be predicted by our presented results.

Table C.2 shows the two-tailed Pearson-coefficient values calculated between the evaluated variables and the parameters of each test scenario, as described in §C.1. Values written in bold face mark those *p-values* that give a correlation with a 99% significance. Thus, in the first two rows, $\mathcal{R}'_{some}$ and $\mathcal{R}'_{all}$ are correlated with $\mathcal{O}$, $\mathcal{U}$ and

$\mathcal{S}$. However, $\mathcal{U}$ is more important (*i.e.* has a higher correlation) for $\mathcal{Q}_{some}$ performance in a uniformly-distributed repository than for $\mathcal{Q}_{all}$, though it is the other way around in a Zipf-distributed repository. Execution times for both queries ($\mathcal{T}_{some}$ and $\mathcal{T}_{all}$) depends on $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{S}$, according to the *p-values* shown in Table C.2.

C.4 Discussion

As a general conclusion from the performed tests, the more specific query ($\mathcal{Q}_{all}$) is better suited to filter and reduce the size of the service repository, so it clearly improve the subsequent discovery stage by reducing the search space for matchmaking algorithms. Furthermore, it scales well in every situation, providing even better precision in proportion when the service repository contains a higher number of services.

However, in certain scenarios, where flexibility and soft matching are a concern, the more generic query ($\mathcal{Q}_{some}$) may be more suitable. The higher time penalty must be taken into consideration, both in the filtering and the discovery stage, because of the less reduced search space, though it still improves the performance of discovery and ranking processes. Thus, there is a trade-off between precision or recall that should be evaluated depending on the concrete scenario. Actually, the current trend in the literature and real-world applications is to achieve better performance and usability, by sacrificing precision, recall, or both[28], so our proposal provides a feasible and efficient solution in this direction.

The main feature of using our proposed queries is that the time penalty is very low, so a hybrid approach may be taken, where both queries are used successively before discovery and ranking processes take part. Firstly, $\mathcal{Q}_{all}$ may be executed, and if some results are returned, they are directly injected into the discovery and ranking process. However, if no results are returned by $\mathcal{Q}_{all}$, the more generic $\mathcal{Q}_{some}$ query is executed and its results are used in the subsequent discovery and ranking process. This approach is similar to the Best-Matches-Only solution proposed in [52], where if the most accurate results are found (*i.e.* $\mathcal{Q}_{all}$ returns results), they are used, but in other case fairly appropriate results (*i.e.* results from $\mathcal{Q}_{some}$)

can be useful.

Finally, if the execution time of the filtering stage is analyzed, actual query execution time is significantly lower than the WSM-L/RDF serialization (approximately 1 millisecond on average), an it is not so affected by the variation of the parameters defined in the experiment. Consequently, our proposed filtering stage could be further optimized by serializing repositories to RDF before performing that filtering.

Acronyms

CSOP Constraint Satisfaction Optimization Problem. 12, 27

DAML DARPA Agent Markup Language. 19

DAML-S DAML Ontology of Services. 19, *see* DAML

EMMA Enhanced MatchMaking Addon. xix, xxi, 16, 57, 58, 60, 64, 66–71, 77, 91, 93–95, 100, 103, 122–124, 127, 129, 135

HTTP HyperText Transfer Protocol. 5

LD Linked Data. 4

LOD Linked *Open* Data. 5, *see* LD

MicroWSMO MicroWSMO. 7, 28, *see* WSMO & WSMO-Lite

MSM Minimal Service Model. 7, 28

NFP non-functional property. xix, xxi, 9, 38, 74, 75, 81, 91, 107, 108, 110–112, 114

OWL Ontology Web Language. 4, 20, 37

OWL-S OWL Ontology of Services. xix, xxi, 7, 18–20, 25, 27, 28, 36, 52, 57, 58, 65–68, 70, 71, 93, 94, 96, 102, 103, 123, 135, *see* OWL

PURI Preference-based Universal Ranking Integration. xix, xxi, 16, 42, 73, 76, 78–81, 91, 105, 106, 110, 113, 115–117, 123, 124, 127

QoS Quality of Service. 9, 19–21, 23, 27

RDF Resource Description Framework. 4, 51, 64–66, 68, 69, 89

RDFS RDF Schema. 4, 68, 69, *see* RDF

SAREST Semantic Annotations for RESTful Services. 7, 28

SAWSDL Semantic Annotations for WSDL and XML Schema. 7, 25, 28, 36, 52, 65, 94, *see* WSDL & XSD

SME^2 Semantic Web Service Matchmaking Evaluation Environment. 25, 93–95, 100, 103, 124, 129, 135

SOAP Simple Object Access Protocol. 5

SOUP Semantic Ontology of User Preferences. xix, xxi, 16, 36, 37, 40, 54, 55, 76, 78, 85, 86, 91, 122, 127

SW Semantic Web. xix, 4–7

SWS Semantic Web Service. xix, xxi, 4, 6–14, 16, 18, 19, 21, 26–28, 30, 36, 37, 39, 40, 42, 44–52, 54, 55, 57–61, 64–66, 70, 71, 80, 81, 85, 86, 91, 94, 95, 102, 103, 110, 121–123, 135

SWSF Semantic Web Services Framework. 7

UML Unified Modeling Language. 37

W3C World Wide Web Consortium. 5, 7

WS Web Service. 4–7, 27

WSDL Web Service Description Language. 5, 27, 107

WSML Web Service Modeling Language. 53, 108, 135, 138, 147

WSMO Web Service Modeling Ontology. xix, xxi, 7, 16, 18, 20, 21, 24, 25, 27, 28, 35, 36, 52–55, 58, 65, 66, 71, 93, 102, 103, 135

WSMO-Lite WSMO-Lite. 7, 28, 65, *see* WSMO & SAWSDL

WSMX Web Service eXecution Environment. 143

WWW World Wide Web. 6

XML eXtensible Markup Language. 5, 20

XSD XML Schema Definition Language. *see* XML

BIBLIOGRAPHY

[1] Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors. *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, 2007. Springer. ISBN 978-3-540-76297-3. 157, 161

[2] Lada A. Adamic and Bernardo A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3(1):143–150, 2002. 138

[3] Sudhir Agarwal, Martin Junghans, Olivier Fabre, Ioan Toma, and Jean-Pierre Lorre. D5.3.1 First Service Discovery Prototype. Deliverable D5.3.1, SOA4All, 2009. 25, 58, 59

[4] Sudhir Agarwal, Martin Junghans, Barry Norton, and José María García. D5.4.3 Second Service Ranking Prototype. Deliverable D5.4.3, SOA4All, 2011. 75, 108, 109, 117, 123, 127

[5] Rakesh Agrawal and Edward L. Wimmers. A framework for expressing and combining preferences. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 297–306. ACM, 2000. ISBN 1-58113-218-2. 9

[6] Grigoris Antoniou and Frank van Harmelen. *A semantic web primer*. MIT Press, 2nd edition, 2008. ISBN 978-0-262-01210-2. 4

[7] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. ISBN 0-201-39829-X. 95

[8] Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors. *Service-Oriented Computing, 7th International Joint Conference, ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009. Pro-*

ceedings, volume 5900 of *Lecture Notes in Computer Science*, 2009. ISBN 978-3-642-10382-7. 159, 161

[9] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. Mcilraith, D. Mcguinness, J. Su, and S. Tabet. Semantic web services framework (SWSF) overview. Technical report, World Wide Web Consortium, September 2005. 7

[10] Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors. *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in Computer Science*, 2008. Springer. ISBN 978-3-540-68233-2. 157, 161

[11] Christian Becker, Kurt Geihs, and Jan Gramberg. Representation of Quality of Service Preferences by Contract Hierarchies. In *Elecktronische Dienstleistungswiltchatt und Financial Engineering*, 1999. 9

[12] David Beckett and Tim Berners-Lee. Turtle - terse rdf triple language. Team submission, W3C, 2011. 113

[13] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey, and Farouk Toumani. Semantic reasoning for web services discovery. In *WWW Workshop on E-Services and the Semantic Web*, 2003. 18, 22, 23, 29

[14] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001. 4

[15] A. Soydan Bilgin and Munindar P. Singh. A daml-based repository for qos-aware semantic web service selection. In *ICWS* IEE [47], pages 368–375. 19, 22, 23

[16] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3): 154–165, 2009. 124

[17] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. Working group note, World Wide Web Consortium, February 2004. 5

[18] Saartje Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual modeling of owl dl ontologies using uml. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2004. ISBN 3-540-23798-4. 37

[19] Christoph Bussler, Dieter Fensel, and Alexander Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Record*, 31(4):24–29, 2002. 7

[20] Alessio Carenini, Dario Cerizza, Marco Comerio, Emanuele Della Valle, Flavio De Paoli, Andrea Maurino, Matteo Palmonari, and Andrea Turati. Glue2: A web service discovery engine with non-functional properties. In Claus Pahl, Siobhán Clarke, and Rik Eshuis, editors, *ECOWS*, pages 21–30. IEEE Computer Society, 2008. ISBN 978-0-7695-3399-5. 24, 25, 26

[21] Yassin Chabeb, Samir Tata, and Djamel Belaïd. Toward an integrated ontology for web services. In Mark Perry, Hideyasu Sasaki, Matthias Ehmann, Guadalupe Ortiz Bellot, and Oana Dini, editors, *ICIW*, pages 462–467. IEEE Computer Society, 2009. 27, 28, 30, 65

[22] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003. 10

[23] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order (2. ed.)*. Cambridge University Press, 2002. ISBN 978-0-521-78451-1. 42

[24] John Davies, John Domingue, Carlos Pedrinaci, Dieter Fensel, Rafael González-Cabero, Morgan Potter, and Marc Richardson. Towars the open service web. *BT Technology Journal*, 26(2), feb 2009. 58, 106

[25] Glen Dobson, Russell Lock, and Ian Sommerville. Qosont: a qos ontology for service-centric systems. In *EUROMICRO-SEAA*, pages 80–87. IEEE Computer Society, 2005. ISBN 0-7695-2431-1. 19, 20, 22, 23, 74

[26] John Domingue, Dieter Fensel, and Rafael González-Cabero. Soa4all, enabling the soa revolution on a world wide scale. In *ICSC*, pages 530–537. IEEE Computer Society, 2008. 58, 127

[27] Joel Farrell and Holger Lausen. Semantic annotations for WSDL and XML Schema. Technical report, World Wide Web Consortium, August 2007. 7, 36

[28] Dieter Fensel. The potential and limitations of semantics applied to the future internet. In Joaquim Filipe and José Cordeiro, editors, *WEBIST*, pages 15–15. INSTICC Press, 2009. ISBN 978-989-8111-81-4. 58, 101, 146

[29] Peter C. Fishburn. *Utility theory for decision making*. Wiley, 1970. 9

[30] José María García, David Ruiz, and Antonio Ruiz-Cortés. On user preferences and utility functions in selection: A semantic approach. In Elisabetta Di Nitto and Matei Ripeanu, editors, *ICSOC Workshops*, volume 4907 of *Lecture Notes in Computer Science*, pages 105–114. Springer, 2007. ISBN 978-3-540-93850-7. 12, 55

[31] José María García, David Ruiz, Antonio Ruiz-Cortés, Octavio Martín-Díaz, and Manuel Resinas. An hybrid, qos-aware discovery of semantic web services using constraint programming. In Krämer et al. [56], pages 69–80. ISBN 978-3-540-74973-8. 14, 77, 81

[32] José María García, David Ruiz, Antonio Ruiz-Cortés, and José Antonio Parejo. Qos-aware semantic service selection: An optimization problem. In *SERVICES I*, pages 384–388. IEEE Computer Society, 2008. ISBN 978-0-7695-3286-8. 9, 12, 55

[33] José María García, David Ruiz, Antonio Ruiz-Cortés, and Manuel Resinas. Semantic discovery and selection: A qos-aware, hybrid model. In Hamid R. Arabnia and Andy Marsh,

editors, *SWWS*, pages 3–9. CSREA Press, 2008. ISBN 1-60132-089-2. 12, 55

[34] José María García, Ioan Toma, David Ruiz, and Antonio Ruiz-Cortés. A service ranker based on logic rules evaluation and constraint programming. In Flavio de Paoli, Ioan Toma, Andrea Maurino, Marcel Tilly, and Glen Dobson, editors, *NFPSLA-SOC'08*, volume 411 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. 9, 13, 21, 49, 53, 75, 81, 92

[35] José María García, Ioan Toma, David Ruiz, Antonio Ruiz-Cortés, Ying Ding, and Juan Miguel Gómez. Ranking semantic web services using rules evaluation and constraint programming. In *IV Jornadas Científico-Técnicas en Servicios Web y SOA (JSWEB)*, pages 111–119, Sevilla, Spain, October 2008. ISBN 978-84-691-6710-6. 13, 81

[36] José María García, Carlos R. Rivero, David Ruiz, and Antonio Ruiz-Cortés. On using semantic web query languages for semantic web services provisioning. In Hamid R. Arabnia and Andy Marsh, editors, *SWWS*, pages 67–71. CSREA Press, 2009. ISBN 1-60132-130-9. 13, 71

[37] José María García, David Ruiz, Pablo Fernandez, and Octavio Martín-Díaz. Upsranker: Integrando programacion con restricciones y evaluacion de reglas para el ranking de servicios web semanticos. In *V Jornadas Cientifico-Tecnicas en Servicios Web y SOA - JSWEB 09*, pages 275–277, 2009. 14, 81

[38] José María García, David Ruiz, and Antonio Ruiz-Cortés. A model of user preferences for semantic services discovery and ranking. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2010. ISBN 978-3-642-13488-3. 11, 21, 55, 79, 102, 112

[39] José María García, David Ruiz, and Antonio Ruiz-Cortés. A lightweight prototype implementation of sparql filters for wsmo-based discovery. Technical report ISA-11-TR-01, Applied

Software Engineering Research Group, University of Seville, May 2011. 69, 71, 102, 135

[40] José María García, Martin Junghans, David Ruiz, Sudhir Agarwal, and Antonio Ruiz-Cortés. Integrating semantic web services ranking mechanisms using a common preference model. *Knowledge-Based Systems*, 2012. Under Review. 14, 81, 117, 123

[41] José María García, David Ruiz, and Antonio Ruiz-Cortés. Improving semantic web services discovery using sparql-based repository filtering. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2012. In press. 13, 71, 123

[42] Asunción Gómez-Pérez. Ontology evaluation. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 251–274. Springer, 2004. ISBN 3-540-40834-7. 90

[43] Volker Haarslev and Ralf Möller. Racer system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083 of *Lecture Notes in Computer Science*, pages 701–706. Springer, 2001. ISBN 3-540-42254-4. 69

[44] Volker Haarslev and Ralf Möller. On the scalability of description logic instance retrieval. *J. Autom. Reasoning*, 41(2):99–142, 2008. 9, 12, 144

[45] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011. 4, 5

[46] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, World Wide Web Consortium, 2004. 21

[47] *Proceedings of the IEEE International Conference on Web Services (ICWS'04), June 6-9, 2004, San Diego, California, USA*, 2004. IEEE, IEEE Computer Society. 152, 162

[48] Hai Jin, Xiaomin Ning, Weijia Jia, Hao Wu, and Guilin Lu. Combining weights with fuzziness for intelligent semantic web search. *Knowledge-Based Systems*, 21(7):655 – 665, 2008. ISSN 0950-7051. 75

[49] Ralph L. Keeney and Howard Raiffa. *Decisions with multiple objectives: Preferences and value tradeoffs*. Cambridge Univ Press, 1993. 9

[50] Christoph Kiefer and Abraham Bernstein. The creation and evaluation of isparql strategies for matchmaking. In Bechhofer et al. [10], pages 463–477. ISBN 978-3-540-68233-2. 24, 25, 26

[51] Christoph Kiefer, Abraham Bernstein, and Markus Stocker. The fundamentals of isparql: A virtual triple approach for similarity-based semantic web tasks. In Aberer et al. [1], pages 295–309. ISBN 978-3-540-76297-3. 24

[52] Werner Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322. Morgan Kaufmann, 2002. 10, 21, 36, 42, 91, 92, 102, 146

[53] Matthias Klusch and Frank Kaufer. Wsmo-mx: A hybrid semantic web service matchmaker. *Web Intelligence and Agent Systems*, 7(1):23–42, 2009. 25

[54] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. Owls-mx: A hybrid semantic web service matchmaker for owl-s services. *J. Web Sem.*, 7(2):121–133, 2009. 25, 58, 95

[55] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. Sawsdl-mx2: A machine-learning approach for integrating semantic web service matchmaking variants. In *ICWS*, pages 335–342. IEEE, 2009. 25

[56] Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors. *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, volume 4749 of *Lecture Notes in Computer Science*, 2007. Springer. ISBN 978-3-540-74973-8. 154, 161

[57] Kyriakos Kritikos and Dimitris Plexousakis. Semantic qos metric matching. In *ECOWS*, pages 265–274. IEEE Computer Society, 2006. ISBN 0-7695-2737-X. 19, 27, 28, 30

[58] Steffen Lamparter, Anupriya Ankolekar, Rudi Studer, and Stephan Grimm. Preference-based selection of highly configurable web services. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 1013–1022. ACM, 2007. ISBN 978-1-59593-654-7. 21, 22, 23, 36, 75

[59] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW*, pages 331–339, 2003. 18, 20, 22, 23, 29

[60] Maria Maleshkova, Jacek Kopecký, and Carlos Pedrinaci. Adapting sawsdl for semantic annotations of restful services. In Robert Meersman, Pilar Herrero, and Tharam S. Dillon, editors, *OTM Workshops*, volume 5872 of *Lecture Notes in Computer Science*, pages 917–926. Springer, 2009. ISBN 978-3-642-05289-7. 7, 28

[61] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic markup for web services. Technical report, DAML, 2006. 36

[62] David Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with owl-s. *World Wide Web*, 10:243–277, 2007. ISSN 1386-145X. 10.1007/s11280-007-0033-x. 7

[63] E. Michael Maximilien and Munindar P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004. 19, 20, 22, 23, 27, 36

[64] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001. 6, 8

[65] Barry Norton, Mick Kerrigan, and Adrian Marte. On the use of transformation and linked data principles in a generic repository for semantic web services. In Mathieu d'Aquin, Alexander García Castro, Christoph Lange, and Kim Viljanen, editors, *ORES-2010*, volume 596 of *CEUR Workshop Proceedings*, pages 59–70. CEUR-WS.org, 2010. 28, 30

[66] Matteo Palmonari, Marco Comerio, and Flavio De Paoli. Effective and flexible nfp-based ranking of web services. In Baresi et al. [8], pages 546–560. ISBN 978-3-642-10382-7. 22, 24, 26

[67] Jyotishman Pathak, Neeraj Koul, Doina Caragea, and Vasant Honavar. A framework for semantic web services discovery. In Angela Bonifati and Dongwon Lee, editors, *WIDM*, pages 45–50. ACM, 2005. ISBN 1-59593-194-5. 19, 20, 22

[68] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW*, pages 553–562. ACM, 2004. ISBN 1-58113-844-X. 7

[69] Carlos Pedrinaci and John Domingue. Toward the next wave of services: Linked services for the web of data. *J. UCS*, 16(13): 1694–1719, 2010. 7, 28, 30, 65, 74

[70] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009. 59, 96

[71] Axel Polleres and David Huynh. Special issue: The web of data. *J. Web Sem.*, 7(3):135, 2009. 4

[72] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Recommendation, W3C, 2008. 59

[73] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005. 7, 36

[74] Antonio Ruiz-Cortés, Octavio Martín-Díaz, Amador Durán, and Miguel Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005. 7, 27, 40

[75] Marco Luca Sbodio, David Martin, and Claude 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. ISSN 1570-8268. 25

[76] Christian Schröpfer, Marten Schönherr, Philipp Offermann, and Maximilian Ahrens. A flexible approach to service management-related service description in soas. In Cesare Pautasso, Christoph Bussler, Marius Walliser, Stefan Brantschen, Monique Calisti, and Thomas Hempfling, editors, *Emerging Web Services Technology*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 47–64. Birkhäuser Basel, 2007. ISBN 978-3-7643-8448-7. 9, 11

[77] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006. 4

[78] Amit P. Sheth, Karthik Gomadam, and Jon Lathem. Sarest: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94, 2007. 7, 28

[79] Wolf Siberski, Jeff Z. Pan, and Uwe Thaden. Querying the semantic web with preferences. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 612–624. Springer, 2006. ISBN 3-540-49029-9. 21, 22, 24

[80] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007. 69

[81] Natalie Steinmetz and Ioan Toma. The Web Service Modeling Language WSML. Technical report, WSML, 2008. WSML Working Draft D16.1v0.3. 53

[82] Nathalie Steinmetz and Holger Lausen. Ontology-based feature aggregation for multi-valued ranking. In Asit Dan, Frederic Gittler, and Farouk Toumani, editors, *ICSOC/ServiceWave Workshops*, volume 6275 of *Lecture Notes in Computer Science*, pages 258–268, 2009. ISBN 978-3-642-16131-5. 107

[83] Nathalie Steinmetz, Holger Lausen, and Manuel Brunner. Web service search on large scale. In Baresi et al. [8], pages 437–444. ISBN 978-3-642-10382-7. 74

[84] Michael Stollberg, Martin Hepp, and Jörg Hoffmann. A caching mechanism for semantic web service discovery. In Aberer et al. [1], pages 480–493. ISBN 978-3-540-76297-3. 24, 25, 58

[85] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27–46, 2003. 4, 6

[86] Katia P. Sycara, Massimo Paolucci, Julien Soudry, and Naveen Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3): 66–73, 2004. 18, 22, 23, 29

[87] Ioan Toma, Dumitru Roman, Dieter Fensel, Brahmananda Sapkota, and Juan Miguel Gómez. A multi-criteria service ranking approach based on non-functional properties rules evaluation. In Krämer et al. [56], pages 435–441. ISBN 978-3-540-74973-8. 13, 21, 22, 23, 74, 107

[88] Ioan Toma, Natalie Steinmetz, Holger Lausen, Sudhir Agarwal, and Martin Junghans. D5.4.1 First Service Ranking Prototype. Deliverable D5.4.1, SOA4All, 2009. 106, 108

[89] Tomas Vitvar, Jacek Kopecký, Jana Viskova, and Dieter Fensel. Wsmo-lite annotations for web services. In Bechhofer et al. [10], pages 674–689. ISBN 978-3-540-68233-2. 7

[90] Lee-Hung Vu, Manfred Hauswirth, Fabio Porto, and Karl Aberer. A search engine for QoS-enabled discovery of semantic web services. *International Journal of Business Process Integration and Management*, 1(4):244–255, 2006. 27, 28, 30

[91] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qos-aware selection model for semantic web services. In Asit Dan and Winfried Lamersdorf, editors, *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 390–401. Springer, 2006. ISBN 3-540-68147-7. 20, 21, 22, 23, 36, 74

[92] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004. 9

[93] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee. Daml-qos ontology for web services. In *ICWS* IEE [47], pages 472–479. 19, 22, 23, 36

[94] George K. Zipf. *Selected studies of the principle of relative frequency in language*. Harvard University Press, 1932. 138