Semantic Web Service Filtering Strategy based on Categories, Attributes and Mediation

Samira Ghayekhloo

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Eastern Mediterranean University
December 2015
Gazimagusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Cem Tanova
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Assoc. Prof. Dr. Zeki Bayram
Supervisor

Examining Committee

1. Prof. Dr. Atilla Elçi
2. Prof. Dr. Can Özturan
3. Prof. Dr. Işık Aybay
4. Assoc. Prof. Dr. Alexander Chefranov
5. Assoc. Prof. Dr. Zeki Bayram

ABSTRACT

The primary contribution of this thesis is a novel logical discovery framework for semantic web services. The framework utilizes semantic description of Web services and user goals in the F-logic language. In this framework, Web service or goal pre-conditions and post-conditions can have embedded objects inside logical expressions. Full usage of conjunction, disjunction and negation operators are allowed in logical expressions occurring inside web service pre-conditions or goal post-conditions. This framework tackles the scalability problem and improves discovery performance by adding two pre-processing stages to the service matchmaking engine. The first stage eliminates web services that cannot satisfy the goal based on ontology comparison of user request and Web service categories. In the second stage, a novel algorithm that can deal with embedded objects inside logic expressions, concepts and attributes of objects that take part in the specification of the goal and Web service is used to analyse the goal and web service specifications and determine which web services will definitely fail in the subsequent logical matching phase. The result of the application of these two pre-filtering stages is that a much smaller pool of Web services need to be considered by the full-blown logic-based matcher against the client request, resulting in considerable gains in scalability of the discovery process. This effectiveness of this two-stage pre-filtering strategy has been verified using a new Web service repository, called WSMO-FL test collection.

The secondary contribution of this thesis is the creation of a novel framework called RFSWS consisting of rubric tables and a feature-based evaluation scheme for the

evaluation and comparison of Semantic Web service discovery and composition approaches, and its subsequent application to the evaluation of five recently introduced prominent Semantic Web services discovery and composition approaches. This is a novel application of rubrics, which have traditionally been used for grading student performance by teachers. Considering the shortcomings of existing Semantic Web services composition approaches that were discovered through the evaluation, an idealized dynamic Semantic Web service discovery and composition method, a yardstick by which all future Semantic Web services composition approaches can be evaluated, has been proposed as well.

Keywords: Rubric, Semantic Web service, Web service discovery, Web service composition, Feature, Evaluation, Pre-filtering, F-Logic, Test collection.

ÖZ

Bu tezin temel katkısı, anlamsal ağ hizmetleri için yeni bir mantıksal keşif çerçevesinin oluşturulmasıdır. Bu çerçeve, ağ hizmetlerinin ve kullanıcı isteklerinin anlamsal tanımı için F-logic dilini kullanır. Bu çerçevede, ağ hizmetleri veya kullanıcı isteklerinin ön-şartlarında ve arka-şartlarında mantıksal ifadeler içinde gömülü nesneler bulunabilir. Ağ himetleri önşartlarında ve kullanıcı isteği arkaşartlarında „ve", „veya" ve „değil" işlemleri kısıtlamasız bir şekilde kullanılabilir. Bu çerçeve, hizmet eşleştirme makinesinin önüne iki adet ön-işleme aşaması koyarak ölçekleme probleminin üstesinden gelip keşif performansını iyi bir seviyeye getirir. İlk aşamada, kullanıcı isteği ile ağ hizmetinin kategorileri ontolojiler kullanılarak mukayese edilir ve bu şekilde kullanıcı isteğine cevap veremeyecek olan ağ hizmetleri elenir. İkinci aşamada, kullanıcı isteği ve ağ hizmetinin tanımında yer alan mantık ifadeleri içinde gömülü nesnelerle, kavramlarla ve nesne özellikleri ile başa çıkabilen yeni bir algoritma kullanılarak kullanıcı isteği ve ağ hizmeti çözümlenir, ve bu şekilde kesinlikle bir sonraki mantıki eşleştirme aşamasında başarısız olacak ağ hizmetleri elenir. Bu iki ön-eleme aşamalarının uygulanması neticesinde, mantıksal eşleştirme işlemine tabi olacak ağ hizmeti sayısı önemli ölçüde azaltılmış olur ve böylelikle keşif sürecinin ölçeklenebilirliği konusunda önemli bir kazanım elde edilmiş olur. Bu iki aşamadan oluşan ön-eleme stratejisinin etkinliği, WSMO-FL adını verdiğimiz yeni bir test kolleksiyonu üzerinde doğrulanmıştır.

Bu tezin ikinci katkısı, rubric tabloları ve özellik tabanlı değerlendirmeye dayalı, anlamsal ağ hizmetleri keşif ve birleştirme yaklaşımlarını değerlendirme ve mukayese etme amaçlı, RFSWS adı verilen çerçevenin geliştirilmesi, ve bu

çerçevenin yakın geçmişte ilk kez kullanıma sunulan beş adet önemli anlam ağı hizmetleri keşif ve birleştirmesi yaklaşımlarını değerlendirmek için kullanılmasıdır. Bu, daha önceleri sadece öğrenci performans değerlendirmesi için kullanılan rubrikler için yeni bir uygulama alanıdır. Yapılan değerlendirme sonucu meydana çıkarılan anlamsal ağ hizmetleri keşif ve birleştirme yaklaşımlarının eksiklerinden yola çıkılarak, geleceğin anlamsal ağ hizmetleri keşif ve birleştirme yaklaşımlarının da mukayese yöntemi ile değerlendirilebileceği ideal dinamik anlamsal ağ hizmetleri keşif ve birleştirme yöntemi ilaveten önerilmiştir.

Anahtar kelimeler: Rubrik, Anlamsal ağ hizmeti, Ağ hizmeti keşfi, Ağ hizmeti birleşimi, Özellik, Değerlendirme, Ön-eleme, F-Logic, Test kolleksiyonu.

DEDICATION

To my family

And

Especially it is lovingly dedicated to my husband,
Hamed Khodadadi

For his support, encouragement, and his endless love during my life

ACKNOWLEDGMENT

First of all, I would like to explicitly express my heartfelt gratitude to my advisor Assoc. Prof. Dr. Zeki Bayram for the endless support of my Ph.D. thesis and related courses, for his patience, incentive, and excellent knowledge. His advice assisted me in all the steps of research of this dissertation. He was not only my supervisor but also like a father figure in his office, “Dünyanın En Tatlı Babası”. He really supported me like a father during my Ph.D. study in Cyprus. Many thanks for everything.

Besides, I am most grateful to the members of my thesis committee: Prof. Dr. Işık Aybay, chairman of department and Assoc. Prof. Dr. Alexander Chefranov, for agreeing to be on my committee. Their insightful comments and encouragement helped me greatly. Thanks for your helpful feedback.

Most importantly I must thank my wonderful supportive husband Hamed for putting up with me over the seven years. His aid, quiet patience, exhortation and endless love were certainly the reasons that I am in this place. He has shared this entire amazing trip with me, so it only seems right that I dedicate this thesis to him.

Last, but never least, I would like to thank my parents, my sisters and brothers, especially my twin sister, Sara and my parents-in-law for supporting me spiritually throughout study in Cyprus and my life in general.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ALGORITHMS

LIST OF FORMULAS

LIST OF ABBREVIATIONS

ACM	Abstract Composition Model
AI-planning	Artificial Intelligence planning
Cat_Filt	Category-based Filtering
Cap_Filt	Capability-based Filtering
CLM	Casual Link Matrix
DAG	Directed Acyclic Graph
DNF	Disjunctive Normal Form
DSD	DIANE Service Description
DynamiCoS	Dynamic Composition of Services
ECA	Event Condition Action
F-Logic	Frame Logic
FP	Functional Properties
HY	HYbrid method
NFP	Non-Functional Properties
NLOG	Non-Logic
OWL-S	Web Ontology Language for Services
QoS	Quality of Service
SAWSDL	Semantic Annotation for WSDL and XML schema
SDC	Semantic Discovery Caching
SOC	Service Oriented Computing
SWS	Semantic Web Service
SY	SYnonymity
Top-k ASC	Top-K Automatic Service Composition

TX	Taxonomy
WS	Web Service
WSC	Web Service Composition
WSDL	Web Service Description Language
WSDL-S	Web Service Description Language with Semantics
WSMO	Web Service Modelling Ontology

Chapter 1

INTRODUCTION

The major aim of this thesis is presenting a logical framework to improve scalability of automated Semantic Web Service (SWS) discovery through pre-filtering strategies. The secondary aim is the presentation of a novel framework for evaluating semantic web service discovery and composition approaches, which has been used to identify weaknesses of existing discovery methods, leading to the development of our discovery framework.

This chapter consists of three parts. In section 1.1, motivation of our work is given. The second part, section 1.2, explains our methodologies which have been employed in this dissertation context, and finally in section 1.3 the structure of this thesis is outlined.

1.1 Motivation

Service-Oriented Computing (SOC) is a computing paradigm that emerged at the end of 1990's as an evolution in design and delivery of software applications [63]. SOC employs service as an essential factor, which plays an important role in the development of distributed applications. Web Services (WSs) are web application components which can be described, published, and discovered through the Web. Also as an example of their features we can say that they are loosely coupled, platform independent, and are accessible through programming over the internet.

Although the advantages of traditional WSs are well established, there exist a number of challenges in service-oriented systems. WS descriptions are syntactic, so the tasks of discovery, selection, and composition have to be performed by the software developer (human). SWS technology, a combination of WSs and Semantic Web concepts, has been created to relieve to a large extent the programmer of the manual work required to use WSs. Semantic annotation of WSs makes them understandable by machines, and permits their automatic discovery and composition, with minimal user intervention.

Despite many efforts that have been performed in the context of SWS, there are still many open issues and active research is being carried out to address them [59]. In each step of our research on SWS discovery and composition methods, weaknesses of the available approaches motivated us to propose and implement new solutions. Figure 1 depicts the main procedure of our work which contains three steps. These three steps are: i) an evaluation of schemes for SWS discovery and composition approaches, ii) a logical framework which improves performance of SWS discovery process through pre-filtering strategies and iii) a proper test collection which covers main functional descriptions (inputs, outputs, preconditions and effects) of WSs and goals.

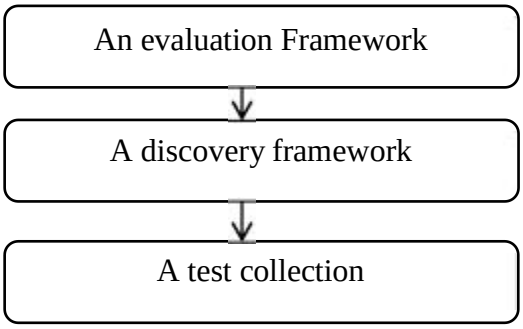


Figure 1. Procedures of our work

Steps of figure 1 are described below.

1.2 Overview of Achievements

1.2.1 New Evaluation Framework

Existing proposals regarding WS discovery and composition have been surveyed and the results presented in several review works in the last decade. These surveys have some important shortcomings, such as not stating clearly what requirements need to be met for an approach to successfully solve the problems of WSs discovery and composition, and lacking in the level of detail.

Such shortcomings prompted us to develop for the first time analytic rubrics, as well as a feature-based comparison scheme, for evaluating and comparing SWS discovery and composition approaches. We called our method *Rubric and Feature-based appraisal and comparison framework for Semantic Web Services* (RFSWS) [28].

RFSWS includes some of the key discovery and composition issues and requirements that were identified based on a comprehensive study of the literature on SWS discovery and composition methods. We then proceeded to actually evaluate five prominent approaches for SWS discovery and composition introduced since 2011 in order to determine their relative strengths and weaknesses, and finally come

up with an idealized approach for SWS discovery and composition. Our work is novel not only in its development and use of special rubrics (which have been traditionally used by teachers in grading the students‟ performance, Wolf & Stevens [75]) for the evaluation of SWS discovery and composition approaches, but also in the level of detail in which the evaluation has been made, and in the specification of an SWS discovery and composition approach which can be the yardstick with which future approaches can be compared to. Furthermore, it distinguishes itself from other recent surveys on WS discovery and composition approaches [69, 53, 6] in that it surveys and compares the current state-of-the-art SWS discovery and composition approaches.

1.2.2 New Logical Discovery Framework

In recent years, complexity of conceptual models (e.g. WSMO[1] and OWL-S[2]) for semantic description of WSs, as well as the increasing number of advertised services in repositories made the discovery processes of SWSs a difficult task [58]. In order to deal with the problems of scalability and complexity, researchers proposed various methods, such as indexing and caching mechanism [70], pre-processing strategies before actual matching [26, 34], and hybrid matchmakers that combine logic-based and non-logic based reasoning [44, 46]. However, we could not find any work that addresses the performance challenge of discovery process in a similar way to our work which is a logical discovery framework based on two new pre-filtering strategies which help to improve the speed of discovery processes of SWSs on large scale.

---

[1] Web Service Modeling Ontology
[2] Web Ontology Language for Services

Our discovery framework is based on the WSMO conceptual model for semantically describing user requests (goals), WSs and domain ontologies. During the discovery process, goal capability descriptions such as inputs, outputs, pre-conditions and post-conditions (effects) are compared with advertised WS capability descriptions in order to determine whether they match or not. Logical inference is utilized for matching, which guarantees that the capability requested by the goal is indeed satisfied by the capability of the WS and also that the WS has all it needs before its starts execution. Capability reasoning of goal and advertised services relies on ontologies which are used both to describe the services and goals and also to describe the common vocabulary needed by the services and goals.

Given a user request specified in the form of a goal, using the logical inference based matcher to match it against all WS specifications in the WS repository is an impractical approach as the number of WSs increases, and the need to narrow down the candidate WSs that will participate in the full matching phase arises. In order to deal with the problem, we use two pre-filtering stages to eliminate WSs that cannot possibly be successfully matched, and reduce the number of WSs which go through the full logic-based matching stage. Our first pre-filtering stage uses a categorization scheme of WSs and second stage uses a novel technique of extracting attributes and concepts of objects utilized in the goal and the WS specifications. Our technique can deal with objects that occur in a logical formula with full usage of the logical connectives conjunction, disjunction and negation, a first in literature. We also make use of ontology-based mediation between concepts and attributes, so that two syntactically different symbols may be declared to denote the same thing semantically.

1.2.3 New Test Collection

Test collections are yardsticks to evaluate the suitability and performance of service discovery frameworks. The majority of SWS approaches (such as [26, 46, 40, 56, 68, 2]) that proposed a solution to improve the discovery processes, evaluated efficiency and accuracy of their works based on OWLS-TC[3] version 3 test collection. Among all related works, although the authors of [17] evaluated their proposal based on last version of OWLS-TC test collection, only input and output parameters are considered for evaluation of their work.

Therefore, unavailability of an appropriate test collection motivated us to establish a new test collection which covers the main functional descriptions of WSs such as pre-conditions and post-conditions, as well as a categorization scheme of WSs. Our new own generated test collection of WS and goal specifications has been named *WSMO-FL*[4].

WSMO-FL covers main functional descriptions of WSs and goals such as pre-conditions and post-conditions, as well as a categorization scheme of WSs and goals. WSMO-FL contains three different domains, namely *transportation*, *food* and *education*, with 250 different F-Logic WS descriptions, 6 different F-Logic goals descriptions, 22 concepts, 3100 attributes and 1225 instances. We used it as an appropriate test collection to measure the gains in efficiency obtained by employing our proposed pre-filtering strategy.

---

[3] http://projects.semwebcentral.org/projects/owls-tc/
[4] http://cmpe.emu.edu.tr/samira/WSMO-FL.htm

1.3 Outline of this Dissertation

The remainder of this thesis is organized as follows.

In chapter 2 some background information about the concepts and terminologies which are used in this thesis, such as SWS, WSMO, SWS discovery, SWS composition, rubric, FLORA-2 etc. are introduced.

Chapter 3 comprises our novel RFSWS framework, and has five sub-parts. The first part presents our strategies in creation and implementation of RFSWS framework. The second part introduces approaches used different methodologies for discovery and composition of SWSs. Third part illustrates tabular evaluation and comparison of mentioned approaches in part two, part four demonstrates features and characteristics of an ideal SWS discovery and composition approach and closing part is related works.

Chapter 4 first introduces our new logical SWS discovery framework which is able to deal with all logical connectives inside the capability objects of WSs and goals. Then, it explains our enhancements over the original WSMO framework, such as optimization and categories. Finally, it presents some scenarios which are part of WSMO-FL test collection version 2 and have been used for the work reported in this thesis.

Chapter 5 describes our novel pre-filtering strategies which have been employed to improve the performance of the SWS discovery process. In this chapter both of the implemented filtering methods, Category-based (Cat_Filt) and Capability-based (Cap_Filt), are analysed in detail.

Chapter 6 is the implementation part of this dissertation which introduces main predicates used in our pre-filtering technique. All of these predicates are written in FLORA-2. Furthermore, in this chapter, step by step achieved results of each predicate during two filtering stages according to our predefined scenario in chapter 4 are displayed.

Chapter 7 contains the experimental results of our work along with explanation about related works. The graphical and tabular results which were obtained on our test collection after employing the pre-filtering steps are shown in this chapter.

Finally, chapter 8 is the conclusion and future work part, and summarizes and highlights the achievements that were explained in the previous chapters, and points out advantages of our work compared to prior works in this field.

Chapter 2

BACKGROUND

Chapter 2 introduces some background information about the concepts and technologies which were used in this thesis such as WS, SWS, WSMO, SWS discovery, SWS composition, rubric, FLORA-2 etc.

2.1 Web Service (WS)

WSs are web application components which can be described, published, and discovered through the Web. Two common types of WSs are SOAP-based WSs, which use WS Description language (WSDL) for their description, and RESTful WSs, which conform to the REST architectural principle [8]. From the information technology viewpoint, WSs are loosely coupled, platform independent, and are accessible through programming over the internet.

2.2 Semantic Web Service (SWS)

Semantic Web has been a popular topic of research since its introduction by Tim Berners-Lee in 2001 [10]. Automation of many tasks on the Internet is facilitated through the addition of machine understandable semantic information to Web resources. Applications of SWS technology include [25, 26], as well as others. SWS technology is the combination of WSs and Semantic Web concepts and allows for example automatic discovery of WSs based on their functionality or composition of WSs which cannot fulfil the user requests individually becomes possible [54].

2.3 Conceptual Models for Semantic Web Services

WSs are semantically described by providing a high level declarative specification of WS functionality and non-functional properties in order to facilitate automatic discovery, composition and invocation of WSs. Two prominent models in SWS descriptions are Web Service Modelling Ontology (WSMO) [65] and Web Ontology Language for Services (OWL-S) [53]. There also exist other special purpose languages for the semantic description of WSs, such as DIANE Service Description (DSD) language [50], WSMO-lite [76], and Semantic Annotation for WSDL and XML schema (SAWSDL) [47].

OWL-S: OWL-S has been developed in the years 2003-2006 by a mostly US-based consortium under the DAML program. It defines an upper ontology for semantically annotating Web services that consists of elements as shown in figure 2. Every description element is defined on the basis of domain ontology, and the current standard ontology language OWL is used as the specification language.

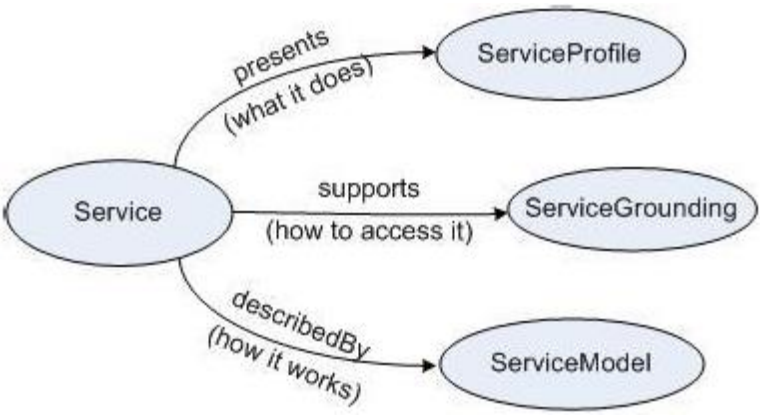


Figure 2: Overview of OWL-S

The Service Profile: The OWL-S profile specifies what functionality the service provides. The functionality description is split into the information transformation

performed by the service and the state change as a consequence of the service execution. The former is captured by defining the *inputs* and *outputs* of the service, and the latter is defined in terms of *preconditions* and *effects*. Inputs, outputs, preconditions and effects are normally referred to as IOPEs. In the last version of OWL-S, effects are defined as part of a result.

The Service Model: Describes how the Web service works. The service is conceived as a process, and the description model defines three types of processes (atomic, simple, and composite processes). These are described by IOPE along with a proprietary process language that defines basic control- and dataflow constructs.

The Service Grounding: Gives details of how to access the service which is realized as a mapping from the abstract descriptions to WSDL

WSMO: The Web Service Modeling Ontology WSMO is developed by a European initiative since 2004. WSMO comprises four core elements, namely ontology, goal, Web service, and mediator as it is shown in figure 3. The *Ontology* is defined as a formal, explicit specification of a shared conceptualization [31]. In the context of semantic WS, ontology provides a common vocabulary to denote the types in the form of classes or concepts, properties and interrelationships of concepts in a domain. A *Goal* describes what the requester can provide, and what it expects from a WS. A *Web service* description represents different functional and non-functional features of a deployed WS. Finally, *Mediator* handles heterogeneity problems that possibly arise between goals, WSs and ontologies.

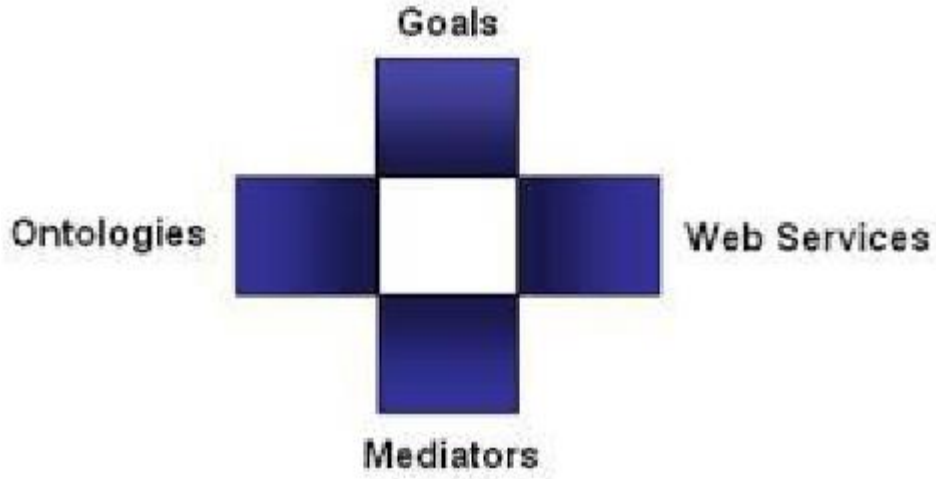


Figure 3: WSMO top level notions

Web services in WSMO are described by non-functional properties, a capability that specifies the provided functionality in terms of preconditions, assumptions, post-conditions, and effects. *Pre-condition* specifies the requested information before execution of the services. *Assumption* describes state of the word which is assumed before the execution of the services. *Post-condition* describes information that is guaranteed to be reached after successful execution of the service, and *Effect* describes the state of the word that is guaranteed to be reached after successful execution of the service. The information transformation performed by a service is described in WSMO by using pre-conditions and post-conditions of the service capability. State change is described in WSMO by using assumptions and effects. WSMO treats input and output (parameters in OWL-S) type description implicitly as part of its pre-condition and post-condition [32].

2.4 SWS Discovery

In general, WS discovery is the process of finding appropriate WSs with respect to the user request and ranking of discovered services based on user preference. Our discovery framework receives WSMO goal descriptions and WSMO WS descriptions, all coded in F-Logic, along with related mediators and ontologies as

input entities and for each goal returns an ordered list of WSs that can satisfy the needs of the goal.

2.5 SWS Composition

In recent years, due to the increasing the number of WSs and complexity of users' demands, traditional WSs have not been able to answer complex user requests adequately. In many instances, the user's request cannot be answered by just one service, and several services must be combined to produce the required result. This job must be done manually if traditional WSs are used. SWSs automate process of WS discovery, selection and composition through Semantic annotation of WSs and expressive definition of user desires in the form of goals.

2.6 FLORA-2 and F-Logic

F-Logic (frame logic) [41] is a powerful logic language with object modelling capabilities. It is used as a foundation for object-oriented logic programming and knowledge representation. Two popular reasoners of F-Logic are FLORA-2 [42] and OntoBroker [3]. Our proposed intelligent agent for semantic WS discovery uses the FLORA- 2 reasoning engine. FLORA-2 is considered as a comprehensive object-based knowledge representation and reasoning platform. The implementation of FLORA-2 is based on a set of run-time libraries and a compiler to translate a unified language of F-logic, HiLog [15], and Transaction Logic [12, 11] into tabled Prolog code [42]. Basically, FLORA-2 supports a programming language that is a dialect of F-logic including numerous extensions that involves a natural way to do meta-programming in the style of HiLog, logical updates in the style of Transaction Logic and a form of defeasible reasoning described in [74].

2.7 Rubrics

A rubric, in its traditional role, is a scoring and instructional tool used to assess student performance using a task-specific set of criteria, providing informative feedback to the instructor regarding the level of understanding on the part of the students, as well as informing students about the expectations of instructors from their work. To measure student performance a rubric contains the essential criteria for the task and levels of performance (i.e., from poor to excellent) for each criterion. The meaning of each level of performance for each criterion is defined explicitly to permit objective evaluation.

There exist two types of rubric, namely *holistic* and *analytic*. In the former, the teacher scores the overall process or product as a whole, without taking into account the component parts singly [60]. In the latter, first the teacher lists all parts of product or process, then considers a score for each part, and at the end sums the individual scores to obtain a total score [57, 60].

Rubrics generally contain three components:

- Dimensions
- Rating Scale
- Descriptors

Dimensions are generally referred to as performance criteria, the *rating scale* as levels of performance, and *descriptors* as definitions. Figure 4 depicts the general form of a rubric table.

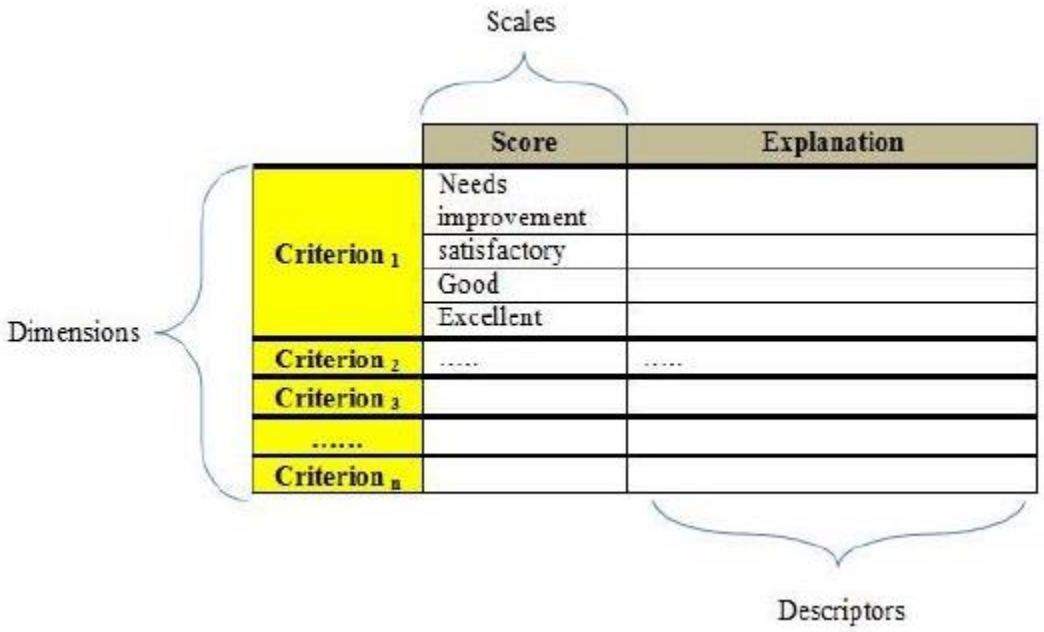


Figure 4. Structure of Rubric table

Table 1 presents a sample of analytic rubric for evaluating research reports [67].

Table 1. Sample of research report rubric table [67]

Criterion	Score	Description
Amount of information	Needs Improvement	All topics not addressed or most questions answered with words or phrases instead of sentences.
	Satisfactory	All topics are addressed, and most questions answered with 1-2 sentences about each.
	Good	All topics are addressed and most questions answered, with at least 3 sentences about each.
	Excellent	All topics are addressed, and all questions answered, with at least 3 sentences about each.
Organization	Needs Improvement	There appears to be little organization of the material.
	Satisfactory	Information is generally organized, but no headings are used.
	Good	Information is organized with headings, but some material under the headings may be out of place.
	Excellent	Information is very well organized with headings that relate clearly to the material.
Quality of Information	Needs Improvement	Information gathered has little or nothing to do with the questions posed.
	Satisfactory	Information gathered provides answers to main questions, but no details and/or examples are given.
	Good	Information gathered provides answers to main questions along with 1-2 supporting details and/or examples.
	Excellent	Information gathered provides answers to the main questions along with several supporting details and/or examples for each.
Sources	Needs Improvement	Some sources for information and graphics are not documented.
	Satisfactory	Sources for information and graphics are documented, but most are not in the correct format.
	Good	Most sources for information and graphics are documented in the designated format.
	Excellent	Sources for information and graphics are documented in the designated format.

2.8 Scalability

Scalability is defined as the capability of increasing the computing capacity of service provider‟s computer system and system‟s ability to process more operations or transactions in a given period. It is also related to performance [49].

Chapter 3

RFSWS: A NEW EVALUATION FRAMEWORK

In chapter 3 our new Rubric and Feature-based assessment and comparison framework for Semantic Web Services (RFSWS) is presented. This chapter contains five sub sections. The first section presents our strategies in creation and implementation of RFSWS framework, and the second one introduces approaches employing different methodologies for discovery and composition of WSs. The third part demonstrates tabular evaluation and appraisal of mentioned approaches in the second part, the fourth part is related work, and finally, the closing part presents our notion of what features and characteristics an ideal SWS discovery and composition approach should have.

3.1 RFSWS Framework

3.1.1 Rubric Tables

In this work for the first time, we apply the technique of rubric tables outside the area of education, and developed rubrics for the evaluation and appraisal of SWS discovery and composition approaches. Also we actually evaluat several recent SWS discovery and composition approaches through using our rubrics. Since not all comparable attributes of the approaches are amenable to comparison using rubrics, we also develop a feature-based evaluation scheme, and used that scheme in concert with the rubric to obtain a better picture of the approaches under consideration. Tables 2 and 3 contain our analytic rubric table. Our rubric table contains six

important criteria needed in composition processes of SWSs, along with their descriptions.

These criteria are *automation*, *scalability*, *adaptivity*, *dynamicity*, *heterogeneity* and *workflow pattern*. *Automation* is the automation level of WS discovery and composition approach, *scalability* shows how many WSs the system can deal with, *adaptivity* identifies the degree to which the system is flexible for modifying its behaviours in volatile environments and responding to significant changes at execution time, *dynamicity* means the degree of dynamism at which the approach can combine WSs for user's request at runtime, *heterogeneity* means the degree to which the WS discovery and composition approach can deal with heterogeneity of WSs, and *workflow pattern* illustrates which types of workflow patterns WS discovery and composition approach uses. We analysed the SWS composition approaches and filled the performace levels of the important criteria in composition process. Scores 1, 2, 3 and 4 illustrate rating scales of our rubric table namely, *needs improvement*, *satisfactory*, *good* and *excellent* respectively.

Table 2. Analytic rubric table for the appraisal of SWS discovery and composition approaches (Part1)

Criterion	Score	Explanation
Automation	**1**	Semi-automatic: User designs the overall architecture of WSs interactions and describes at a high level the requirements that participating WSs must satisfy. Actual WS discovery and composition take place automatically at runtime.
	2	Somewhat automatic: The system presents the user with its results and the user accepts the one that is most satisfactory for the job at hand.
	3	Mostly automatic: All the WS discovery and composition procedures are done automatically by the machine and without user intervention; however the approach does not consider the non-functional properties.
	4	Fully automatic: User does not intervene in discovery and composition processes and all the WS discovery and composition procedures are done automatically by machine which considers the non-functional properties and user preference as well.
Scalability	**1**	Approach works only with toy examples, using less than 10 WSs. This approach merely presents its idea in small size with possible development in future works.
	2	Approach can handle 10 to 99 WSs.
	3	Approach presents method for reasonable real-life cases, and can deal with WSs within the range of 100 to 1000.
	4	Approach is scalable to more than thousands of WSs.
Adaptivity	**1**	No adaptiveness: Presented method is not able to support any types of adaptation during WSC processes.
	2	Rule-based adaptation: Approach employs methods that rely on predefined event-condition-action (ECA) rules. Rules are activated whenever the events which they are bound to happen in the environment, but they are limited in covering all possible events and scenarios.
	3	Partial adaptation: Approach considers only some aspects of adaptation at runtime, such as, change in non-functional properties and/or addition or removal of WSs during WS discovery and composition processes.
	4	Full adaptation: Approach is able to cope with all aspects of unpredicted change in functional and non-functional properties of WSs at runtime without interrupting the whole system operation.

Table 3. Analytic rubric table for the appraisal of SWS discovery and composition approaches (Part2)

Criterion	Score	Explanation
Dynamicity	**1**	Static: All processes of discovery, selection and composition of specified WSs take place manually at design time. Thus it cannot be an appropriate solution for the unpredicted user's request.
	2	Somewhat-dynamic: The processes of discovery and selection are performed statically, while the composition of WSs is done at runtime.
	3	Mostly dynamic: Approach creating an Abstract Composition Model (ACM) at design time, while discovery, selection and composition (actual linking of WSs according to the ACM) of WSs are done at execution-time.
	4	Fully dynamic: Approach implementing this type of dynamism is able to handle all the processes of discovery, selection and composition of WSs at runtime.
Heterogeneity	**1**	Approach assumes that all the candidate WSs which participate in WSC processes use the same description language.
	2	Approach requires an adaptor for each pair of cooperating WSs. (In this case with N cooperative WSs are needs to develop N^2 adaptors).
	3	Approach has no formal mediator component, but can solve the heterogeneity problem, in non-systematic ways.
	4	Approach uses in-built mediators to overcome the problems of heterogeneity and enable interoperability between WSs.
Workflow Pattern	**1**	Sequential: The sequential pattern defines sequential execution of WSs. A WS is invoked after the completion of previous one.
	2	Sequential and and-split-join: WSC approach uses both sequential and and-split-join patterns. And-split-join represents WSs that are executed simultaneously. The term join represents the synchronization constructor, which shows that the next WS is invoked when all parallel branches of WSs have been executed.
	3	Sequential, and-split-join and conditional: WSC approach uses sequential, and-split-join and conditional patterns. A conditional pattern represents the exclusive choice of branches to invoke the proper WS. In exclusive choice, exactly one of the conditions is permitted to be true, and the corresponding WS is executed.
	4	Sequential, and-split-join, conditional and iteration: WSC approach uses all the mentioned patterns. In the iteration pattern, WSs can be called repetitively.

3.1.2 Feature-based Evaluation Scheme

There exist some important criteria which have important roles in SWS discovery and composition processes but cannot be easily evaluated using rubrics, because it is not possible to directly compare the offered solutions on a graded scale. Here we investigate those as feature-based criteria, which we use in conjunction with the analytic rubric represented in the previous subsection for the appraisal and analysis of SWS discovery and composition approaches.

Table 4. Features and sub-features that will be used to evaluate approaches

Feature	Value space
1. Accepted SWS Description Methodology	Top-down Bottom-up
2. Quality of Service	Response time Performance Cost Availability Security Reliability
3.Composition Methods	AI-planning Workflow-based Model-based Mathematics-based Other methods
4. Execution of Composite Services	Self-execution Execution by other standards
5. Personalization	User preference Context awareness

In the rest of this section, features and sub-features items are explained briefly.

1. **Accepted SWS description methodology:** This item is about methods for describing both functional and non-functional properties of WSs. Two popular ones are top-down and bottom-up methods.

(a) **Top-down:** WSs are semantically described by providing a high level declarative specification of WS functionality and non-functional properties. Two prominent models which follow this method are WSMO and OWL-S. There also exist other special purpose languages for the semantic description WSs, such as DIANE Service Description (DSD) language [50].

(b) **Bottom-up:** First the service developer generates WSs based on Web Service Description Language (WSDL) [16]. Then existing WSs are semantically annotated by different bottom up annotation models such as WS Description Language with Semantics (WSDL-S) [1] or Semantic Annotation for WSDL and XML schema (SAWSDL) [47].

2. **Quality of Service (QoS):** QoS deals with the quality aspects of a user's interaction with the WSs. The prominent QoS factors associated with the WS composition and execution are mentioned in [52]; below we explain them briefly.

(a) **Response time:** Refers to the time needed to complete a user's request.

(b) **Performance:** Refers to the overall speed of the system for completing the user's request and is measured in terms of response time, latency, execution time and throughput. *Latency* is the round-trip delay time in sending a request by the user and receiving the response from the system, *execution time* is to the time taken by a WS to fulfil its series of activities and *throughput* refers to the number of requests which are served in a given period of time. In general, low latency, high throughput, low execution time and fast response time are the desired performance characteristics of WS.

(c) **Cost:** Is the amount of money needed in order to execute the related WSs for answering the user's request.

(d) **Availability:** Is the readiness of the WS to accept and process requests. High availability shows that the WS is ready to use most of the time.

(e) **Security:** Since WS discovery and composition processes work on the public Internet, loss, theft and modification of information is a real risk; therefore security is a very important aspect of WS discovery and composition processes that must be given full attention. The service provider should have different levels of security depending upon the needs of the service requestor. Sub-aspects of security include confidentiality, traceability, authorization and non-repudiation.

3**. Composition methods:** Different approaches use various methods to combine the WSs in order to satisfy the user's demand. Due to the large number of composition methods, we use the classification scheme proposed in [7]. The authors divide composition methods into four groups, namely *AI planning*, *workflow-based*, *model-based* and *mathematic-based* methods.

(a) **AI-planning:** Employs Artificial Intelligence planning (AI-planning) algorithms in order to combine WSs. The approach solves the problems of WS discovery and composition by designing the set of actions for achieving the goals and generating a plan.

(b) **Workflow-based:** In this approach first an Abstract Composition Model (ACM) is designed, either manually or automatically by the workflow generator tools in accordance with the user's request. ACM specifies control and data flow among the tasks. Secondly, an algorithm is employed to find

specific WSs that are matched to the tasks and bind relevant WSs together respectively.

(c) **Model-based:** This approach uses modelling languages like UML, Petri net, etc. to model service composition and overcome to the problems of complex requests.

(d) **Mathematic-based:** This approach presents its solution for solving the WS discovery and composition problems based on mathematic structure and techniques such as: *graph-based techniques, logic-based techniques* and *techniques based on process algebra*.

(e) **Other Methods:** The last group of composition methods comprises all the approaches that do not fit in the aforementioned list and represents other methods for solving the WS discovery and composition problem.

4. **Execution of composite services:** These approaches present different ways to execute the qualified composite service, either directly via in-built components or by the help of other standards.

(a) **Self-execution:** All the processes of composition and execution of WSs are done within the presented approach's components.

(b) **Execution by other standards:** The approach addresses discovery and composition of WSs to create composite services but devolves the execution of the composite service to other outstanding standards, such as BPEL [38].

5. Personalization: Web personalization is the process of customizing WSs so that they match the particular user's needs and preferences [37]. Preference and context-awareness are two main factors of personalization.

(a) **User Preference:** By paying attention to the end user's desire or intention in service selection, it is possible to improve the quality of presented WSs and better achieve end user's satisfaction.

(b) **Context Awareness:** Context refers to the information about the end user and its environments, such as, name, address, current location of the user and type of device that the customer is using. Authors of [66] categorize methods for awareness of end user's context as: personal profile oriented, usage history oriented, process oriented and other methods.

3.2 Approaches Employing Different Methodologies for Discovery and Composition of Web Services

This section briefly reviews five state-of-the-art SWS composition approaches, namely: *DynamiCoS* [18], *PORSCE II & VLEPPO* [35, 36], *Bartalos* [5], *Top-k ASC* [20] and *Tang et al.* [72]. These approaches present different methods for solving the problems of SWS discovery and composition. They have been selected after a comprehensive survey of the literature on current SWS discovery and composition approaches.

3.2.1 DynamiCoS

DynamiCoS (Dynamic Composition of Services) [18] is a user-centric framework which was created for combining WSs at runtime to answer the user requirements. In this framework, services are created and published by the service provider at design-time but the processes of discovery, selection and composition are performed at runtime. As shown in figure 5, *DynamiCoS* architecture has five modules, namely service creation, service publication, service request, service discovery and service composition. In order to achieve automation in WS discovery and composition processes, the framework semantically annotates the WS as a seven-tuple < ID, I, O,

P, E, G, NF >, where ID, I, O, P, E, G and NF stand for service *identifier*, *inputs*, *outputs*, *preconditions*, *effects*, *goals*, and *non-functional* properties respectively.

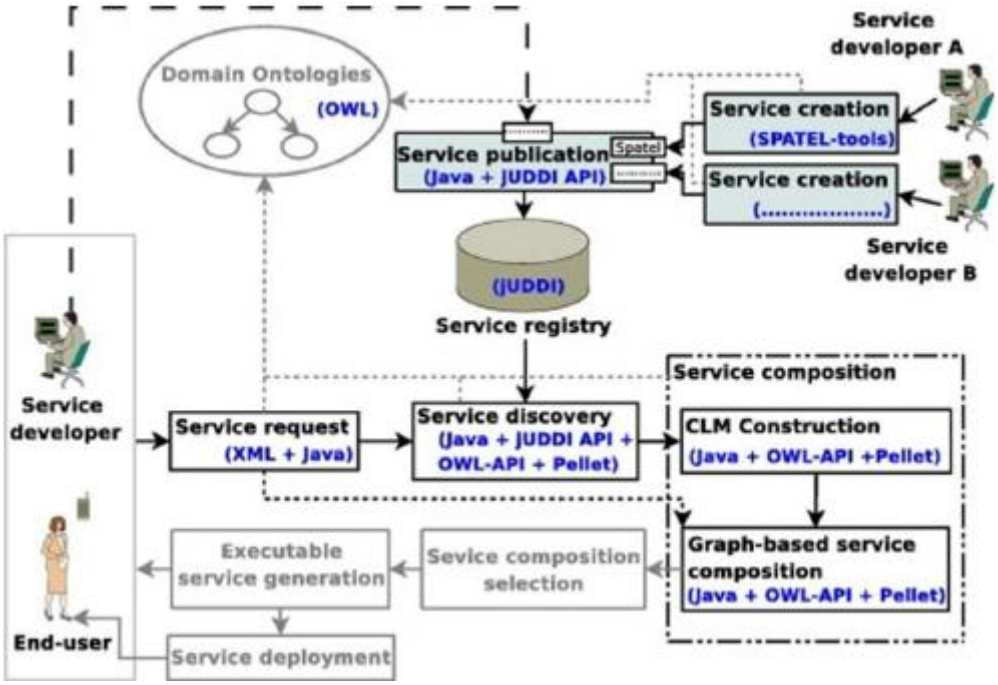


Figure 5. DynamiCoS Architecture [18]

The composition processes of DynamiCoS are:

(i) Requested WSs are discovered based on exact and partial matching of Inputs, Outputs, Preconditions and Effects (IOPE) concepts of WSs.

(ii) Description of all discovered WSs are organized in a pre-processed structure called *Casual Link Matrix* (CLM) [51] which stores all possible semantic connections (or causal links) between the discovered services' input and output concepts.

(iii) A graph-based composition algorithm [30] is used to find set of composed services according to the user demand via the prepared CLM matrix.

3.2.2 PORSCE II & VLEPPO

PORSCE II & VLEPPO [35, 36] is a framework for modelling the SWSC problem as a planning problem, expressed in the *Planning Domain Definition Language* (PDDL) [27], and then applying a variety of external planners to get a solution plan for obtaining the end user's goal. Figure 6 depicts the architecture of this approach which contains two software systems, namely *PORSCE II* [35] and *VLEPPO* [33, 34]. Implementation of the approach is performed through the integration of these two systems. The former performs all the tasks related to WSs, such as transforming OWL-S description of WSs to PDDL, accuracy measurement of the composed service, etc., while the latter deals with the planning steps. Key features of this approach are: (i) it is able to be used by the non-expert user through the dialog interface in PORSCE II, (ii) it allows the preparation of an approximate composition service, and (iii) independence between the representation and solving parts makes the approach flexible in the choice of external planners.

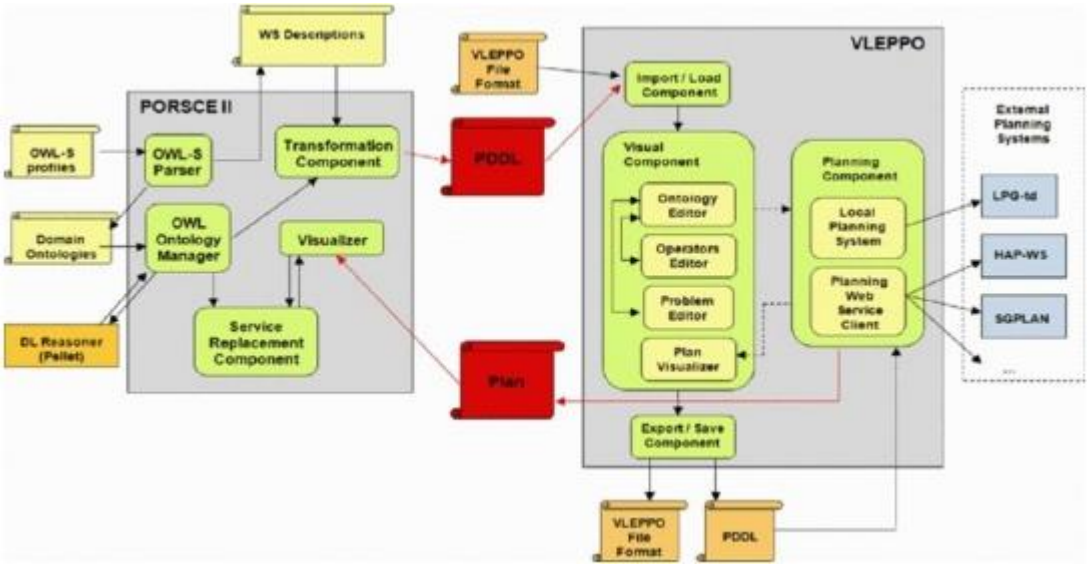


Figure 6. PORSCE II & VLEPPO Architecture [36]

SWSC steps in PORSCE II & VLEPPO are:

(i) Translation of the WS composition problem into the planning problem, which is done in *PORSCE II*, comprising the transformation of the discovered OWL-S description of WSs into PDDL elements.

(ii) Solving the transformed problem by invoking external planners in *VLEPPO*.

(iii) Visualization of the generated plans in *PORSCE II*.

(iv) Selection of one of the generated plan based on statistical methods and accuracy metrics.

In the remainder of this paper, inside tables we shall use the abbreviation PII-V to denote the PORSCE II & VLEPPO approach.

3.2.3 Bartalos

Bartalos [5] created an approach for composing SWS in a large scale environment by considering functional properties of WSs (*Input*, *Output*, *Pre/Post conditions*) along with QoS attributes. The basic steps of the framework are (i) finding WSs that can consume the provided inputs such that their preconditions are satisfied, (ii) selecting WSs that provide requested outputs and post-conditions by using a backward chaining strategy, and finally, (iii) multiple composite services are created based on produced interconnection between initial and final services by considering optimal the QoS value.

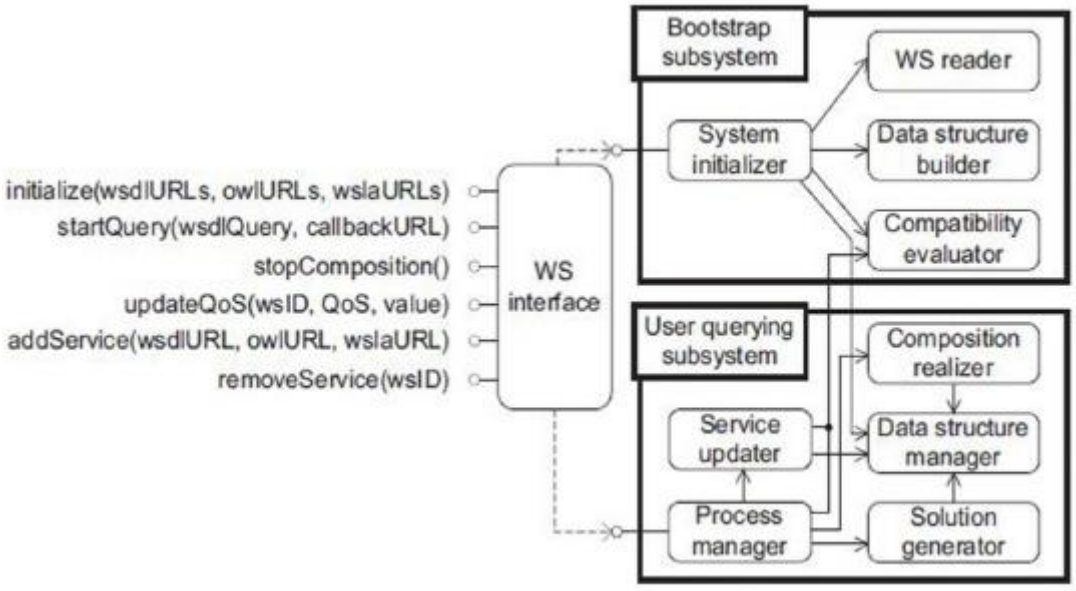


Figure 7. Bartalos Composition Processes Architecture [5]

Figure 7 illustrates the overall composition processes in the *Bartalos* approach. Its composition process architecture is divided into two main phases: *bootstrap* and *user querying*. *Bootstrap* is the pre-processing stage which is performed before receiving any user request. In this stage all the actual WSs are analysed and linked if there is any relation between them to create a Directed Acyclic Graph (DAG). *User querying* is done after receiving the goal. The initial and finals WSs are found and then the DAG of WSs found in the pre-processing stage is employed to find a set of suitable composition of services according to QoS attributes.

3.2.4 Top-k ASC

Top-k ASC (Top-K Automatic Service Composition) [20] is a method which was created for determining best k composition services based on QoS attributes with a large number of WSs. A WS is defined as a three tuple < *I, O, QoS* > where *I, O* denote the semantic concepts of *Input* and *Output,* and *QoS* denotes Quality of Service. QoS itself is defined as an n-tuple < q_1, q_2, q_3,… , q_n >, where each q_i defines one QoS attribute, such as cost, response time, availability, etc. This

framework transforms the WS composition problem into a graph searching problem, i.e. each composed service is shown in the form of a DAG. Afterward, the approach, by using a composition algorithm (based on backtracking and depth-first search) can find best k composition services in a parallel way according to the user's request.

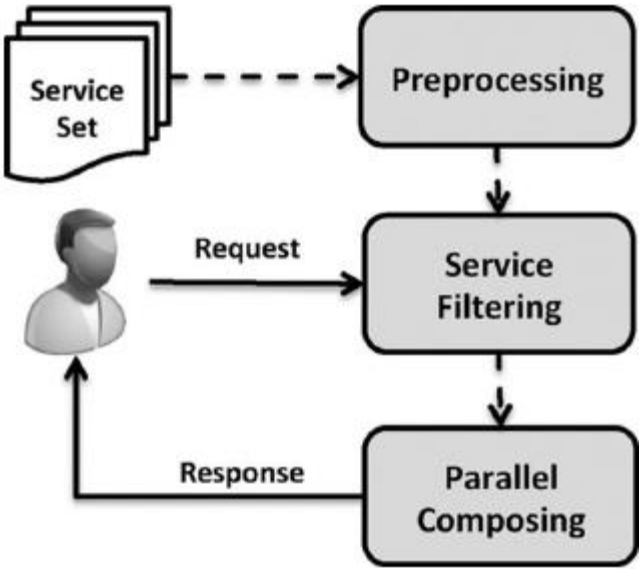


Figure 8. Architecture of Top-k ASC [20]

The composition procedure of Top-k ASC is depicted in figure 8. The approach has two phases: *run-up* and *composition*. The *run-up* phase, usually done off-line, consists in pre-processing of WSs from a large-scale registry, and then transforming the pre-processed services sets into the rule repository for being efficiently accessible to answer the user's request. The *composition* phase contains service filtering and parallel composing stages. The service filtering stage fetches rules representation of WSs which are compatible with user request from rule repository and filters out unrelated ones. The parallel composing stage uses the idea of *MapReduce* [19] (a programming model for processing parallelizable problems in a large data sets with a huge number of nodes) to find best k composition services in a parallel way, while guaranteeing optimal QoS.

3.2.5 Tang et al.

Tang et al. [72] presents a framework for composing SWSs based on a logical interface of Horn clauses and Petri nets. In this approach a WS is defined by four-tuple < *I, O, BC, QoS* > where *I, O, BC* and *QoS* stand for the semantic concepts of *Input* and *Output*, set of *Behavioural Constrains* and *Quality of Service* respectively. *Behavioural Constrains* is conditions which ensure correct execution of the WSs. Quality of Service involves attributes such as cost, response time, availability and reliability.

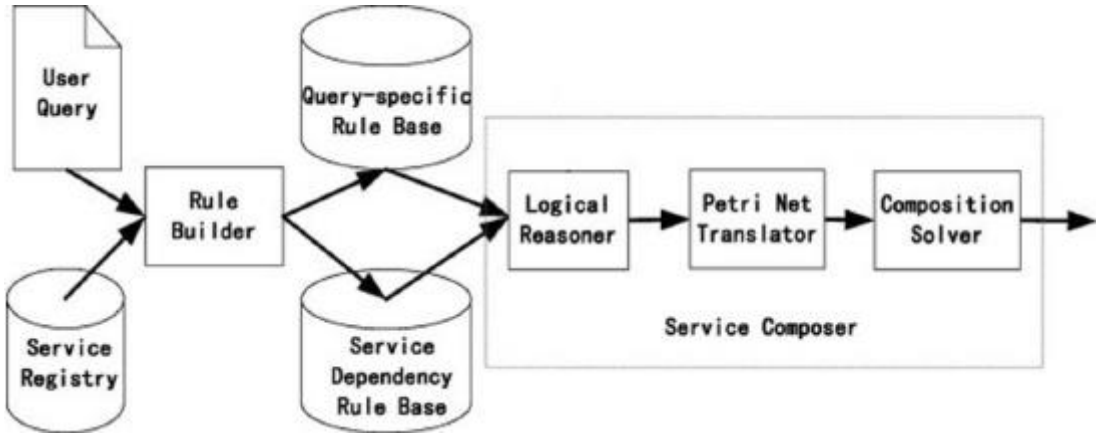


Figure 9. Framework of Tang et al., [72]

Figure 9 illustrates SWSC processes of Tang et al. which involve the following steps:

(i) Before accepting any request from the user, the *Rule Builder* component that works based on the *hypergraph* theory [9] generates dependency rules between the existing SWSs in the registry. These rules have the structure $WS_1 \wedge WS_2 \wedge \ldots \wedge WS_k \rightarrow WS_z$ which means whenever all of WS_i $(1 \leq i \leq k)$ have finished their execution, then WS_z can be invoked. Rules are stored in the *Service Dependency Rule Base*.

(ii) When an end user request comes, *rule builder* creates Horn logic rules for inputs and outputs of the request. Rules are indicated $\rightarrow WS_p$ and $WS_q \rightarrow$ for inputs and outputs respectively, and are stored in the *Query Specific Rule Base*.

(iii) The *Logical Reasoner* applies an algorithm based on forward chained deduction for propositional logic (*PL-FC-ENTSILS?*) on both *service dependency* and *query specific rules*. This algorithm determines whether there exist any compositions of services that can satisfy user request. If such a composition exists, it returns a set of Horn clauses rules which are necessary for the composition.

(iv) The *Petri Net Translator* takes the selected rules and converts them into the Petri net representation, and finally,

(v) The *Composition Solver* part generates composite services by using structural analysis techniques, such as T-invariant, of Petri nets.

3.3 Actual Comparison of Approaches in Section (3.2)

This section presents the actual comparison and appraisal of the mentioned SWS composition approaches based on the RFSWS framework which we presented in Section 3.1. Tables 5, 6 and 8 depict the evaluation of the aforementioned approaches according to the rubric tables and feature-based scheme respectively.

3.3.1 Rubric-based Appraisal and Comparison

Tables 5 and 6 illustrate tabular comparison and appraisal of SWS composition approaches based on rubric tables 2 and 3.

Table 5. Evaluation of SWS composition approaches based on rubric tables (Part1)

Criterion	Score	Explanation
Automation	**4**	***DynamiCoS*** is an automatic SWS composition framework where all the processes of discovery and composition are done automatically.
	3	***PII-V*** is a framework which automatically composes SWSs based on AI planning techniques. Furthermore, the most preferable composite service can be selected automatically among the candidate ones based on statistical methods and accuracy metrics without considering any non-functional properties.
	4	***Bartalos*** automatically combines SWSs in a large scale environment by considering both functional and non-functional attributes.
	4	***Top-k ASC*** automatically determines best k composite services based on QoS attributes.
	4	***Tang et al***. is an automatic WS composition framework based on logical interface of Horn clauses and Petri nets.
Scalability	**3**	***DynamiCoS*** prototype shows that approach is able to deal with 500 WSs in the registry.
	3	Experimental result shows that ***PII-V*** is able to handle 1000 WSs.
	4	Experimental result of ***Bartalos*** demonstrates that it has high scalability, being able to handle around 100,000 WSs.
	4	***Top-k ASC*** can handle 20,000 WSs.
	N/S[5]	***Tang et al***. does not give any information about the size of the web service registry.
Adaptivity	**1**	***DynamiCoS*** does not support any kind of adaptivity and flexibility in its WS discovery and composition processes.
	3	In ***PII-V***, service replacement component handles problems of service failure or service unavailability by replacing an alternative atomic WS into the composite plan. If it cannot find a suitable alternative WS, it performs the re-planning technique.
	3	***Bartalos*** is able to handle three types of changes in WS environment, namely addition/removal of a WS, and change in the QoS of a WS, by designing an algorithm which updates a data structure to handle the dynamic changes in the WS environment.
	1	***Top-k ASC*** does not take adaptivity issues into account during the composition processes.
	1	***Tang et al.*** is not flexible enough to adapt to any changes that may happen during the composition processes.

[5] N/S means not specified, i.e., the approach does not clearly specify values

Table 6. Evaluation of SWS composition approaches based on rubric tables (Part2)

Criterion	Score	Explanation
Dynamicity	**4**	***DynamiCoS*** supports runtime discovery, selection and composition of SWSs.
	4	In ***PII-V***, all the processes of discovery, composition and selection of composed services are done at runtime.
	4	***Bartalos*** composes SWSs in a large scale environment by considering the QoS attributes at runtime.
	4	***Top-k ASC*** performs the processes of creating best k composite services at runtime.
	4	In ***Tang et al.*** all the composition processes are done dynamically.
Heterogeneity	**3**	***DynamiCoS*** framework has two steps for solving the problem of using different WS description languages. First, it specifies an interpreter for each supported description language to extract necessary information of WS such as *I*, *O*, *P*, *E*, *G* and *NF*. Second, it publishes all the extracted WS information into the registry by using the *DynamiCos service publication* mechanism.
	1	***PII-V*** does not consider the problems of incompatibility of WSs in the registry, since it uses the same description language, OWL-S, for all the existing WSs.
	3	In ***Bartalos*** different types of WS descriptions (WSDL, OWL-S) are parsed by the WS *Reader*. Then the parsed data are processed by the WS *processor* to build the basic data structure.
	1	***Top-k ASC*** assumes that all the WSs use the same description language.
	1	***Tang et al***. does not take the incompatibility of WSs into account since it uses the same semantic description language, SAWSDL, for all existing WSs in the registry.
Workflow Pattern	**2**	***DynamiCoS*** uses a graph-based composition algorithm to find a composite service based on user request. A composite service which is represented as a DAG can have sequential and and-split-join constructors.
	2	In ***PII-V,*** plans which are generated in VLEPPO are based on the sequential and and-split-join control structure.
	2	In ***Bartalos,*** composition of WSs is presented in DAG with sequential and and-split-join control structure.
	2	Best k composite services in ***Top-k ASC*** are presented in DAG which can contain sequential and and-split-join structure.
	2	In ***Tang et al.,*** the represented composite service in Petri net can contain sequential and and-split-join constructors.

A summary of the findings is presented in table 7 (assume that all criteria have the same importance). Results show that among the evaluated SWS composition approaches, *Bartalos* [5] is the winner of rubric-based evaluation with the highest total grade of 20. Most of the studied approaches except *PORSCE II & VLEPPO* obtained highest score in the automation and dynamism criteria. These approaches achieved this score since they carried out all the steps of SWS discovery and composition processes automatically and at runtime along with considering non-functional properties of WSs beside the functional ones.

Table 7. Summary of the evaluation of SWS composition approaches based on rubric tables

Criterion	**DynamiCoS**	**PII-V**	**Bartalos**	**Top-k ASC**	**Tang et al.**
Automation	4	3	4	4	4
Scalability	3	3	4	4	N/S
Adaptivity	1	3	3	1	1
Dynamicity	4	4	4	4	4
Heterogeneity	3	1	3	1	1
Workflow Pattern	2	2	2	2	2
Total score	**17**	**16**	**20**	**16**	**12**

In the second criterion (scalability), *Bartalos* and *Top-k ASC* received the best score among the approaches. They obtained this score since they were able to deal with more than thousands of WSs in the registry. Both of the mentioned approaches pre-process existing WSs in the registry to determine dependencies between WSs and thereby enhance response time and the performance of discovery and composition processes. *Bartalos* seems better than *Top-k ASC* and *Tang et al.* in pre-processing of WSs as it takes different types of WS description languages into account.

All the mentioned approaches obtained the same score in the workflow pattern criterion. Approaches merely support sequential and and-split-join control structures for creating composite services and cannot handle complex patterns such as conditional and iteration.

Lastly, adaptivity guarantees flexibility of approaches against any changes that may happen during the composition processes. *Bartalos* and *PORSCE II & VLEPPO* provide methods to handle such changes. *Bartalos*, compared to *PORSCE II & VLEPPO*, is able to adapt itself to more changes, namely addition/removal of a WS, and change in the QoS of a WS.

3.3.2 Feature-based Appraisal and Comparison

Table 8 depicts the evaluation of SWS composition approaches based on supported features described in table 4. Unlike the summary evaluation rubric table 7, it is hard to quantitatively compare features of the approaches, since they employ a variety of methods, tools and languages.

Table 8. Evaluation of SWS composition approaches using the feature-based table

Criterion	DynamiCoS	PII-V	Bartalos	Top-k ASC	Tang et al.
Accepted SWS Description Methodology	Top-down	Top-down	Top-down	N/S	Bottom-up
Quality of Service	Cost	N/A[6]	N/S	Response time	Response time, Cost, Availability
Composition Methods	Mathematics-based	AI-planning	AI-planning	Mathematics-based	Model-based
Execution of Composite Services	N/A	N/A	N/A	N/A	N/A
Personalization	N/A	N/A	N/A	N/A	N/A

Also chart in figure 10 depicts graphically the scores of each studied approaches determined by the rubric table evaluation as well.

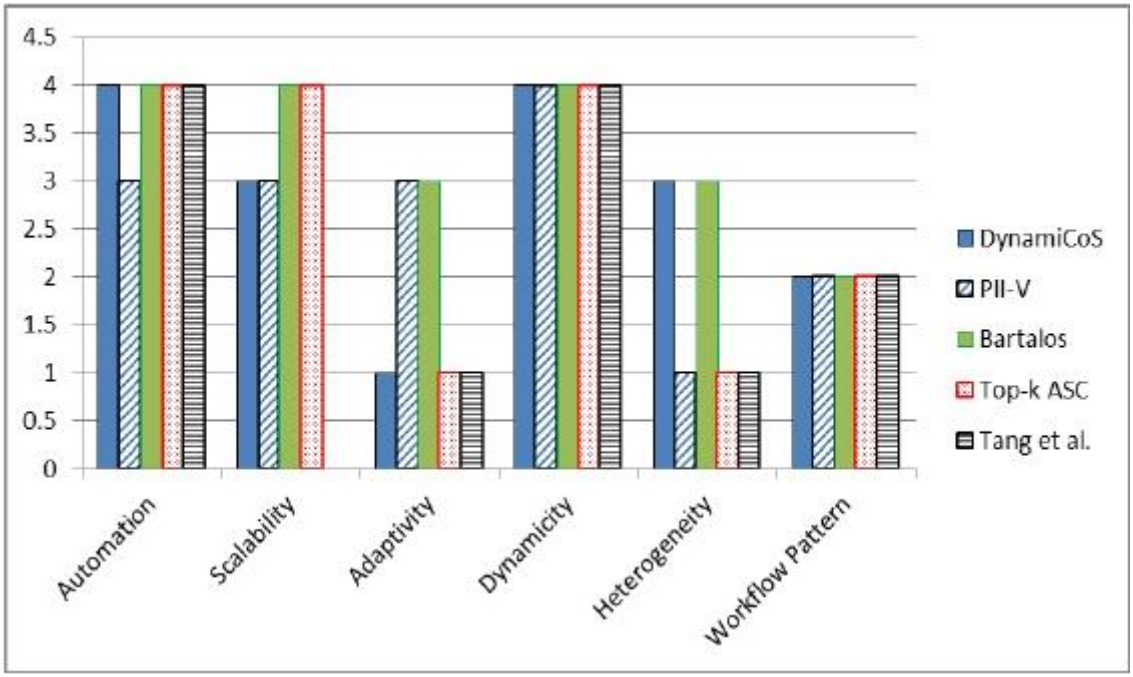


Figure 10. Evaluation chart

[6] N/A means not available

The first feature (accepted SWS description methodology) determines what kinds of semantic description languages are used by the approaches. *Bartalos* and *PORSCE II & VLEPPO* use WSs which are semantically described in OWL-S. On the other hand, *DynamiCoS* which is a service description language neutral framework, allows service providers to use different semantic description languages for describing the WSs. *Spatel* language is employed in the *DynamiCoS* prototype for the semantic annotation of WS operations. Furthermore OWL is also used in this prototype for describing ontologies. In contrast, *Tang et al.* accepts the bottom-up WS description language, SAWSDL.

The second row of table 8 depicts the QoS aspects of the approaches. *Bartalos* considers QoS attribute values for each atomic WS, and the best composite service is found based on the optimal aggregated QoS values of each atomic WS which participates in the composition. However it does not clearly define what kinds of QoS attributes are used in this approach. On the other hand, the *DynamiCoS* prototype uses an OWL ontology which defines non-functional properties of WSs, but only cost is used in the prototype. *Tang et al.* uses the QoS attributes cost, availability and response time for WSs.

Approaches, in order to achieve user's desired goals, combine WSs in various ways. *DynamiCoS* and *Top-k ASC* employ different mathematical techniques to compose the WSs. In *DynamiCoS* composite services are generated based on a CLM matrix via a graphic composition algorithm, whilst in *Top-k ASC* the WSC problem is transformed into a graph searching problem. *Top-k ASC* uses a composition algorithm based on the combination of backtracking and depth-first search. In *PORSCE II & VLEPPO* and Tang et al., similar to *Top-k ASC*, the WSC problem is

transformed into a different formalism. *PORSCE II & VLEPPO* uses AI-planning method and WSC problem is transform into a planning problem. Then, WSs are composed using classical planning techniques. *Tang et al.* uses a model-based composition method. In the *Tang et al.* WSC problem is transformed into logical interface problem of Horn clauses and Petri nets, and then a forward chaining algorithm is used to find composite services.

Finally, despite the importance of last two features (execution of composite services and personalization), most of the approaches do not deal with them. Only *Bartalos*, via Solution Generator supplies composite service in executable BPEL format.

3.4 Evaluation and Appraisal of SWS Composition Approache

In this section, we briefly review other recent prominent surveys on WS discovery and composition approaches, and then point out the ways in which our work is different from or superior to these works.

The survey paper [64] reviews WS discovery and composition approaches based on two composition methods: workflow and AI-planning, and claims that most of the WS discovery and composition approaches employ AI-planning techniques to compose the WSs.

The authors of [7] present requirements in automated WS discovery and composition such as dynamicity, automation level, semantic capability, QoS awareness, scalability, correctness, domain independence, partial observation and adaptivity. They compare state-of-the-art approaches proposed until 2010 based on these criteria.

Vardhan et al. [73] introduce a review paper which has a simple classification of WS composition approaches, namely static composition, dynamic composition and semantics based. It proceeds to evaluate and compare WS composition approaches using this classification.

Authors of [21] review only dynamic WS discovery and composition approaches. They derive a reference model along with some requirements for dynamic WS discovery and composition techniques, namely query analyser, dynamic selection, composition template, verification model, distributed execution, monitoring module, recovery module, QoS certifier, control agent, semantic based and context source. They then analyse the WS discovery and composition approaches based on these requirements.

In [69] the authors present a WS discovery and composition life cycle consisting of three phases, namely definition, service selection, and execution. For each phase, a different set of requirements is given. For definition phase, the requirements are expressibility and correctness. The requirements automation and selectability are for the service selection, and finally adaptability, scalability, monitoring and reliability are for execution phase. Then, they compare WS discovery and composition approaches using the specified requirements.

Our work is unlike the prior works because:

- We determined aspects of SWS discovery and composition processes that can be evaluated as performance criteria in rubric tables and defined rubric tables for such evaluation.

- We designed a feature-based evaluation scheme for aspects of SWS discovery and composition processes that could not be naturally evaluated using performance criteria.
- We created a novel framework called RFSWS consisting of rubric tables and the feature-based evaluation scheme for the evaluation and comparison of SWS discovery and composition approaches.
- We used the RFSWS framework to actually evaluate five state-of-the-art SWS discovery and composition approaches that contain the recent advancements in SWS discovery and composition technology.
- We proposed an idealized SWS discovery and composition approach which can be the yardstick against which any new SWS discovery and composition approaches can be judged (given in the following section).

3.5 An Idealized Approach

An idealized approach has to perform all the steps of the WSC processes in an automated manner. Specifically, it should have the following capabilities:

- Accept requests both from expert and inexpert users,
- Deal with large number of WSs in the registry (scalability),
- Handle diverse WSs in the registry (heterogeneity),
- Automatically employ partial matching as well as exact matching of requested WSs with the WSs in the registry at runtime,
- Consider both functional and non-functional properties of WSs in the composition processes.
- Involve sequential, and-split-join, conditional and iteration patterns in the control structure of the composite service, and

- Be adaptable and respond to significant changes at execution time in volatile environments.

Chapter 4

SEMANTIC WEB SERVICE SPECIFICATION AND DISCOVERY FORMALIZATION

Chapter 4 contains the following subjects: i) expression of a new logical SWS discovery framework using F-logic, ii) explanation of new employed enhancements over the original WSMO framework such as optimization and categories, and finally, iii) description of some scenarios which are part of WSMO-FL V2 test collection.

4.1 New Logical Semantic Web Service Discovery Framework

In general, WS discovery is the process of finding appropriate WSs with respect to the user request and ranking of discovered services based on user preference. Many researchers proposed various SWS discover methods [71, 68, 14] in different ways with ours. Our discovery framework receives WSMO goal descriptions and WSMO WS descriptions, all coded in F-Logic, along with related mediators and ontologies as input entities and for each goal returns an ordered list of WSs that can satisfy the needs of the goal.

Figure 11 depicts the architecture of our discovery framework. The framework consists of four stages: 1) the creation and maintenance of goals and WSs along with related domain ontologies and mediators, 2) pre-filtering stages, 3) matchmaker and 4) ranking stage. In the *creation and maintenance* stage, WS and goal descriptions which are specified based on our modified WSMO model, along with domain ontologies and mediators, are stored in different repositories. In the *pre-filtering*

stages, for a given goal, advertised WSs are filtered in two steps in order to narrow down the list of WSs that can be possible matches for the goal, the rest of the WSs being eliminated from consideration. In the *matchmaker* stage, the logical matchmaker checks whether each filtered WS can really execute in a way such that the user goal is achieved. Finally, the *ranking* stage returns lists of matched WSs based on user preference regarding the minimization of some numeric result (for example, the cost of a flight between two cities).

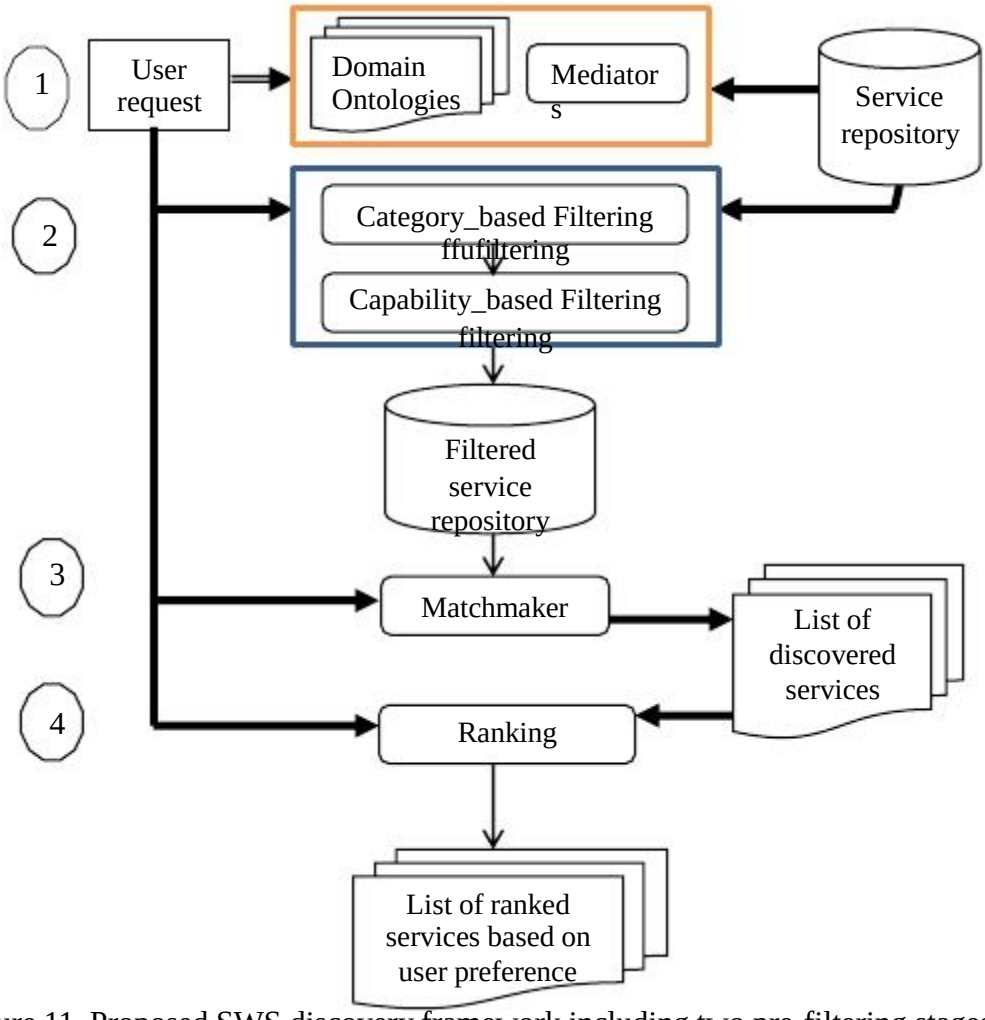


Figure 11. Proposed SWS discovery framework including two pre-filtering stages

Following are briefly description of our logical matchmaker mechanism.

Our logical matchmaker algorithm makes use of pre-conditions and post-conditions of goals and WSs, as well as related domain ontologies and mediators which are imported in service descriptions. The proof commitments (i.e. what must be proven before a match can succeed) required for our logical inference based matching are given below:

1. *Onts* ∧ *Mediator* ∧ *Goal.Pre* ⊨ *Ws.pre*: The pre-condition of the WS (*Ws.Pre*) should be logically entailed by imported ontologies, mediators, and what is provided /guaranteed by the goal pre-condition (*Goal.Pre*).

2. *Onts* ∧ *Mediator* ∧ *Goal.Pre* ∧ *(Ws.pre => Ws.post)* ⊨ *Goal.post*: the post-conditions of the goal should be logically entailed by imported ontologies, mediations as well as the implication Ws.pre=>*Ws.post*, which we assume is guaranteed by the execution of the WS.

4.2 Enhancements over the Original WSMO Framework

We enhanced the original WSMO model [24] framework in several ways. Listings 1, 2 and 3 depict the meta-level concept definitions of WSMO. Listing 1 contains the *Goal* concept, instances of which are used to specify a user‟s request. It has attributes for non-functional properties (such as quality of service, response time, security etc.), category information (such as transportation, education, food etc.), ontologies that need to be consulted that contain specific information about a domain (for example, flight information ontology, geographical information ontology, etc.), mediator information (ontologies that deal with discrepancies in terms by defining equivalence classes of terms and synonymous relationship between them), capability needed

from the WS, and the interface demanded from the WS (i.e. orchestration and choreography). The *hasCategory* attribute has been newly introduced in our framework in order to allow filtering based on categories.

```
Goal [ |
        hasNonFunctionalProperty =>NonFunctionalProperty,
        hasCategory=> Category,
        importsOntology => Ontology,
        usesMediator =>Mediator,
        requestsCapability =>Capability ,
        requestsInterface => Interface
    |].
```

Listing 1. Goal concept in our extended version of WSMO

The *Service* concept given in listing 2 is almost identical to the *Goal* concept. Its two differences are: (i) it specifies the *provided* capability instead of the *requested* capability, and (ii) it has an extra attribute called *otherSource* (not in the original WSMO specification) which lists the concepts that should be excluded from consideration in the filtering phase, since objects that are instances of the listed concepts should come from other sources, such as imported ontologies, and are not in the goal.

```
Service [ |
        hasNonFunctionalProperty =>NonFunctionalProperty,
        hasCategory=> Category,
        importsOntology => Ontology,
        usesMediator=>Mediator,
        hasCapability =>Capability,
        hasInterface => Interface,
        otherSource =>OntologyConcept
    | ].
```

Listing 2. WS concept in our extended version of WSMO

Listing 3 is the definition of the *Capability* concept. It has attributes for non-functional properties, imported ontologies, mediators used, pre-condition, assumption, post-condition, effect and optimization. The *optimization* attribute allows the user to specify that the WS returned by the discovery engine should be optimized with respect to some measure (for example, price of a flight etc.), and is an enhancement of the original WSMO specification.

```
Capability [ |
        hasNonFunctionalProperty =>NonFunctionalProperty,
        importsOntology => Ontology,
        usesMediator=>Mediator,
        hasPrecondition=>Axiom,
        hasAssumption=>Axiom,
        hasPostcondition=>Axiom,
        hasEffect=>Axiom,
       optimization=>OptSpecification
      | ].
```

Listing 3. Capability concept in our extended version of WSMO

When a capability object is part of a goal, the pre-condition is a *conjunction* of embedded objects in the form of F-logic molecules which specify the information provided by the request to the WS, and post-condition is a logical expression possibly containing embedded objects, *predicates*, *conjunction*, *disjunction* and *negation* operators. All logic variables in a goal post-condition are implicitly existentially quantified.

However, inside a WS specification, pre-condition is a logical expression possibly containing embedded objects in the form of F-logic molecules, *predicates*, *conjunction*, *disjunction* and *negation* operators and where all logic variables are existentially quantified, and post-condition is a *conjunction* of embedded objects which specify the information provided by the WS to the requester that is the result

of the WS execution. Note the similarities between the goal post-condition and WS pre-condition, as well as the goal pre-condition and WS post-condition.

4.3 Scenarios obtained from WSMO-FL test collection

Listings 4 to 9 show the F-logic descriptions of two goals and four WSs specifications respectively among various available types of goals and WSs in our repository.

Listing 4 depicts capability descriptions of a goal instance, which belongs to *AirTransportation* category and describes a request for a flight ticket from Berlin to Istanbul and specifies that the user does not want *Sabiha_Gokcen* as a destination airport. The requester also demands flights that have less than 500$ cost for one person, and that whatever flight is returned, it must have the minimum cost. Note that logic variables start with the “?” symbol.

```
Goal Instance
Goal #1: "Book a flight from Berlin to Istanbul"
hasCategory  −>  AirTransportation,

requestsCapability −> ${goal_1[
        hasPrecondition−>
            ${reqFlight[
                    originateCity−> berlin,
                    terminalCity  −> istanbul
                        ]: RequestFlightTicket
             },
        hasPostcondition −>
            (${?BookTicket[
                    fromAirport  −> ?FromAirport,
                    toAirport  −> ?ToAirport,
                    cost −> ?Cost
                        ]:Response
             },
            \+ is_equal(?ToAirport , Sabiha_Gokcen),
             less (?Cost, 500)
              ),
        optimization −>
              ${optObj[optCost  −>? Cost]}
                            ]
                        }
```

Listing 4. Part of goal instance specification request a flight reservation

Listing 5 depicts semantically descriptions of a goal instance, which belongs to *Restaurant* category. This goal describes a request for a restaurant‟s name in Berlin and specifies that the desired restaurant must be a vegetarian restaurant. The requester also demands restaurant‟s name that maximum food price of this restaurant should not be greater than 150$ for two people.

Goal Instance
Goal #2: "Reserve a restaurant in Berlin"

```
hasCategory  −>  Restaurant,

requestsCapability −> ${goal_2[
        hasPrecondition−>
            ${findRest[
                    restName−>?Name,
                    inCity−> berlin,
                    foodSource−> plants,
                    numberPeople −> 2
                        ]:RequestRest
              },
        hasPostcondition −>
            (${?BookRest[
                    restName−> ?Name,
                    address−> ?Adr,
                    maxPrice−> ?TotalCost
                            ]:ResponseRest
                },
             \+ greater (?TotalCost, 150)
              )                    ]
                                 }
```

Listing 5. Part of goal instance specification request a restaurant name

Listing 6 depicts the main part of the capability and category descriptions of a WS instance in our WS repository. SWS#1 belongs to the *AeroplaneTransportation* category and provides flight reservation through credit card or PayPal payment systems for users who request a flight from one place to another place. This WS asks for source and destination cities, desired payment types, consults two ontologies containing flight information (*FlightInfo_Simple_ont*) and geographical information (*Geographical_ont*), and returns the list of matching airports. The preconditions needs four objects, two of first three objects coming from the goal (instance of *RequestFlightTicket* and either *CreditCard* or *PayPal*) and last object coming from an imported ontology (instance of *Flight*).

SWS#1:
"Reserve a Flight by Credit Card Or PayPal"

```
hasCategory −> AeroplaneTransportation,

importsOntology −>
            { '.../ FlightInfo_Simple_ont.flr',
              '…/Geographical_ont.flr'},

hasCapability−> ${ sws_1[
    hasPrecondition−>
        (${?ReqFlight[
                startCity−>?FromCity,
                endCity−>?ToCity
                        ]: RequestFlightTicket
            }
            ,
        (${?PaymentType1[
                creditNumber−>?CreditNo,
                expireDate−>?ExpreDate
                        ]: CreditCard
            };
        ${?PaymentType2[
                accountName−>?AccName,
                accountNumber−>?AccNo
                        ]:PayPal
            }
          )
            ,
          (${?SomeFlight[
                fromAirport−>?FromAirport,
                toAirport−>?ToAirport,
                cost−>?Cost
                        ]:Flight
            }
          )
          ),
    hasPostcondition−>
          ${response[
                        fromAirport−>?FromAirport,
                        toAirport−>?ToAirport,
                        cost−>?Cost
                                ]:Response
          }
                            ]
                    }
```

Listing 6. Parts of WS instance specification dealing with flight reservation via credit card or PayPal

In order to better illustrate our work, consider listings 7, 8 and 9 which depict main parts of SWS descriptions of three different instances in our WS repository. All these services are obtained from the defined domains in our test collection (WSMO-FL V2).

According to Listing 7, SWS#2, it belongs to the *Food* category and provides a restaurant reservation for the user who demands the name of restaurant in specific place. This WS asks for name of restaurant if available, city of the desired restaurant, the number of people and type of specific food, then consults two ontologies containing restaurant information (*RestInfo_ont*) and geographical information (*Geographical_ont*) and returns the lists of matching restaurants along with their maximum meals‟ price for specific number of people. An instance of the *Food* concepts should come from the restaurant ontology (*RestInfo_ont*). Note that imported ontologies of the WS can act like the local knowledgebase consulted by the WS.

```
SWS#2:
 "Reserve a Restaurant"
hasCategory −> Food,

importsOntology −>
            { '.../ RestInfo_ont.flr',
              '…/Geographical_ont.flr'},

hasCapability−> ${ sws_2[
    hasPrecondition−>
            (${?FindRest[
                    restName −>?Name,
                    inCity −>?City,
                    numberPeople −>?HNumber,
                    foodSource −>?Source
                            ]:RequestRest
              },
             ${?SomeRestaurant[
                    restName −>?Name,
                    inCity −>?City,
                    foodSource −>?Source,
                    address −>?Adr,
                    maxPrice −>?Cost
                            ]:Food
              },
             mult(?Cost,?HNumber,?TotalCost)
              ),
    hasPostcondition−>
             ${response[
                    restName −>?Name,
                    address −>?Adr,
                    maxPrice −>? TotalCost
                        ]:ResponseRest
              }
                    ]
               }
```

Listing 7. Parts of WS instance specification dealing with restaurant reservation

SWS#3 in listing 8 fits in the *Transportation* category and provides a flight or ship reservation for a user who desires to book either a flight or ship voyage from one place to another place in world. SWS#3 asks for origin and destination cities according to user‟s desire, consults either of the two ontologies containing flight information (*FlightInfo_Simple_ont*) and geographical information

(*Geographical_ont*) or two ontologies containing ship information (*ShipInfo_Simple_ont*) and geographical information (*Geographical_ont*) respectively. As the result, it returns a list of matching airports or harbours, ordered according to minimum cost, if the goal specifies an ordered result.

SWS#3:
"Reserve a Flight or Ship"

```
hasCategory −> Transportation,

importsOntology −>
            { '.../ FlightInfo_Simple_ont.flr',
              '.../ ShipInfo_Simple_ont.flr ',
              '…/Geographical_ont.flr'},

hasCapability−> ${ sws_3[
    hasPrecondition−>
            ((${?ReqShip[
                    startCity->?FromCity,
                    toCity->?ToCity
                        ]:RequestShipTicket
                } ,
              ${?SomeShip[
                    fromHarbor ->?FromHarbor,
                    toHarbor ->?ToHarbor,
                    cost->?Cost
                        ]:Ship
                }
              ) ;
              (${?ReqFlight[
                    fromCity->?FromCity,
                    destinationCity ->?ToCity
                        ]: RequestAirplainTicket
                },
               ${?SomeFlight[
                    fromAirport->?FromAirport,
                    toAirport->?ToAirport,
                    cost->?Cost
                        ]:Flight
                }
              )) ,
    hasPostcondition−>
              ${response[
                    originateAirport ->?FromAirport,
                     toAirport->?ToAirport,
                    fromHarbor ->?FromHarbor,
                     toHarbor ->?ToHarbor,
                    cost->?Cost
                        ]:Response
                }
                        ]
                    }
```

Listing 8. Parts of WS instance specification dealing with either flight or ship reservation

Finally in listing 9, SWS#4, fits in the *Learning* category and provides school‟s name for users who request a name of school in specific city. This WS asks for the name of school, desired city, school type (preliminary_school or high_school or etc.) and gender of students who are studying in the school, then consider two ontologies containing school information (*SchGenderInfo_ont*) and geographical information (*Geographical_ont*) and returns the list of schools‟ name along with their costs. Again, the precondition needs two objects, one coming from the goal (instance of *RequestSch*) and one coming from an imported ontology (instance of *Education*).

SWS#4:
"Finding Name of School"

```
hasCategory −> Learning,

importsOntology −>
            { '.../ SchGenderInfo_ont.flr',
              '…/Geographical_ont.flr'},

hasCapability−> ${ sws_4[
    hasPrecondition−>
            (${?FindSch[
                    schName->?Name,
                    inCity->?City,
                    schoolType->?SchoolType,
                    gender->?Gender
                        ]:RequestSch
              },
             ${?SomeSch[
                    schName->?Name,
                    inCity->?City,
                    gender->?Gender,
                    address->?Adr,
                    schoolType->?SchoolType,
                    cost->?Cost
                        ]:Education
              } ),
    hasPostcondition−>
             ${response[
                    schName->?Name,
                    address->?Adr,
                     cost->?Cost
                        ]:ResponseSch
             }
                    ]
              }
```

Listing 9. Parts of WS instance specification dealing with finding a school‟s name

Chapter 5

PROPOSED TWO-PHASE PRE-FILTERING MECHANISM

In chapter 5 we explain our novel strategy improves discovery performance by adding two pre-filtering stages before the logical matchmaker stage of discovery framework. We call these two pre-processing algorithms, which offer different filtering levels, *Category-based Filtering* (Cat_Filt) and *Capability-based Filtering* (Cap_Filt).

Our algorithms that perform pre-processing reduce the input data of service matchmaking, so that the matching process is more streamlined; only logical reasoning about WSs that really matter with respect to the goal is carried out.

In the following sections, we describe the two filtering stages in more detail.

5.1 Category-based Filtering (*Cat_Filt)*

The *Cat_Filt* stage filters the original WSs repository according to both specified categories and synonyms defined in the *Global_Cat_Ont* ontology. Figure 12 illustrates part of hierarchical structure of our specified domains in *Global_Cat_Ont*, which currently contains the three major categories for *transportation*, *food* and *education*.

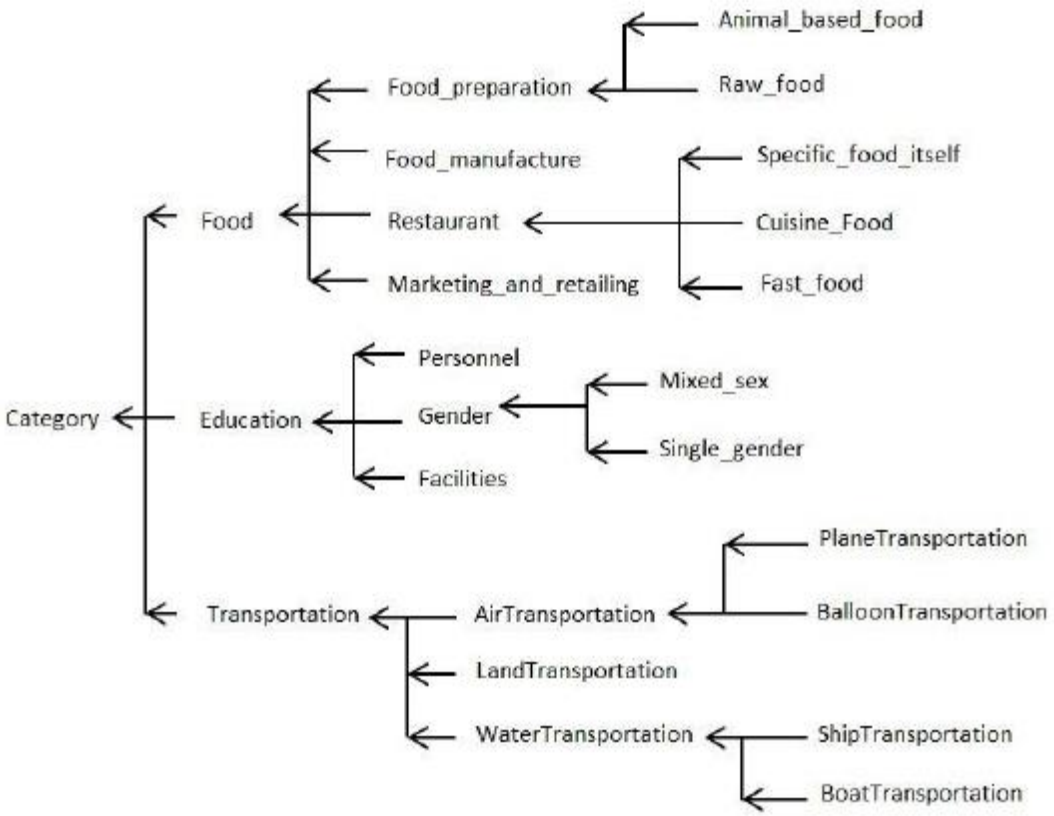


Figure 12. Part of the hierarchical structure of our specified domains in the category ontology Global_Cat_Ont

Global_Cat_Ont contains both structural knowledge (i.e. it defines subclass and superclass relationships between concepts of three specified domains) and a dictionary of synonymous concepts.

5.1.1 Abstract Definition of Cat_Filt Algorithm

Formula 1 shows the abstract definition of *Cat_Filt* in the form of a function that takes a goal as a parameter.

To understanding of the following function, let us give a brief introduction to object oriented notation used in FLORA-2. Suppose that O and C are two objects. O **:** C means that O is an instance of C (in FLORA-2, an object can simultaneously be a class). C **::** D means that C is a subclass of D. Also for user-defined equality,

suppose that O1 and O2 are different names (called id-terms in FLORA-2 terminology) that are supposed to denote the same object. This fact is stated in FLORA-2 with the notation O1**:=:** O2. This facility enables the user to state that two syntactically different (and typically non-unifiable) terms represent the same object, and can be used to define synonymy between such terms.

Here, *g* and *w* stand for goal instance and WS instance respectively, and W is the Web service repository. The result of the function is the union of three sets: (i) if the goal specifies a category (Cat_g), advertised WSs in the registry which have categories matching the goal‟s category, (ii) WSs that have no category specified, and (iii) all WSs in case no category is specified for the goal. This definition guarantees that if there is any possibility of a WS matching the goal, it is never eliminated from consideration in the next phase.

1:	*Cat_Filt(g)* =	{ *w* | *g* has a category specified,
2:		*w* ∈ W,
3:		Cat_w ∈ *Global_Cat_Ont*,
4:		Cat_g ∈ *Global_Cat_Ont*,
5:		(Cat_w **::** Cat_g or Cat_g **::** Cat_w or Cat_w**:=:** Cat_g)} ∪
6:		{ *w* | *w* ∈ W, *w* does not have a category specified } ∪
7:		{ *w* | *g* does not have a category specified, *w* ∈ W }.

Formula 1. Abstract definition of Cat_Filt as a function

5.2 Filtering According to Capability Decomposition (Cap_Filt)

Cap_Filt algorithm eliminates irrelevant WSs based on checking of the attributes and concepts of objects employed in the goal and the WS pre and post-conditions. Our *Cap_Filt* algorithm uses a novel technique of extracting attributes and concepts of objects utilized in the goal and the WS specifications. This algorithm can deal with predicates and objects that occur in a logical formula with full usage of the logical

connectives conjunction, disjunction and negation, a first in literature. We also make use of ontology-based mediation between concepts and attributes, so that two syntactically different symbols may be declared to denote the same thing semantically. Therefore, *Cap_Filt* analyses semantic equivalency between extracted attributes and concepts in order to filter out unrelated WSs.

The level of similarity between such parameters is obtained based on their hierarchical relationships inside this ontology. In this work, levels of semantic similarity between parameters are defined as *exact*, *plug-in*, *subsume* and *fail*. *Exact* means two concepts or two attributes are exactly identical in the same domain ontology. Similarity degree of two concepts or two attributes is *plug-in* only if concept or attribute of goal request is superclass of concept or attribute of the WS. Degree of two concepts or two attributes is *subsume* only if concept or attribute of goal request is subclass of concept or attribute of the WS. Finally *fail* degree expresses that there is no semantic-based relationship between two concepts or two attributes.

Also, our work, in order to gain more precise results and tackle the problem that two concepts or two attributes which are going to be investigated may not be equal syntactically, uses WorldNet [55], a dictionary of synonymous words. Thus, synonym similarity between the goal and WS parameters in the *Cap-Filt* algorithm is calculated by making use of the WordNet[7] online synonym dictionary.

[7] https://wordnet.princeton.edu/

5.2.1 Disjunctive Normal Form (DNF)

Since Cap_Filt should be able to deal with predicates and objects that occur in a logical formula with full usage of the logical connectives conjunction, disjunction and negation, we need to reach a comparable list of concepts and attributes of objects between pre-condition and post-condition parts of goal and WS. In order to achieve a comparable list, some kind of normalized form is needed. Note that, a goal pre-condition (web service post-condition) is a conjunction of embedded objects (in the form of F-logic molecules), and goal post-condition (web service pre-condition) is a logical expression possibly containing embedded objects, predicates, conjunction, disjunction and negation operators, where all logic variables are existentially quantified. The logic of matching dictates that we use Disjunctive Normal Form (DNF) for web service pre-conditions and goal post-conditions.This is due to the fact that a WS pre-condition (goal post-condition) in DNF depicts explicitly the alternatives that the web service (goal) is ready to accept for a match.

A formula in DNF has the structure of $(C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n}) \vee \ldots \vee (C_{k1} \wedge C_{k2} \wedge \ldots \wedge C_{kn})$, where C_{ij} (a literal) is either an object, predicate or the negation of an object or predicate.

5.2.2 Cap_Filt Algorithm

Algorithm 1 depicts the employed strategies in the second step of filtering.

Algorithm 1. Capability filtering Algorithm

Input: Goal.Pre, Goal.Post, Ws.Pre and Ws.Post
Output: Matching status of Web service

```
1:  Get concepts and attributes of Goal.Pre
2:  Convert Ws.Pre into DNF
3:  Eliminate the negative literals in the converted Ws.Pre
4:  WsPrePossibleMatch ← False
5:  for each OR‟ed component Comp (of Ws.Pre)
6:     get Comp's concept, attributes
7:     if concept and attributes of Comp match with concepts and attributes of Goal.Pre
8:       then
9:          WsPrePossibleMatch ← True
10:    end if
11: end for

12: Get concepts and attributes of Ws.Post
13: Convert Goal.Post into DNF
14: Eliminate the negative literals in the converted Goal.Post
15: GoalPostPossibleMatch ← False
16: for each OR‟ed component Comp (of Goal.Post)
17:    get Comp's concept, attributes
18:    if concept and attributes of Comp match with concepts and attributes of Ws.Post
19:      then
20:         GoalPostPossibleMatch ← True
21:    end if
22: end for

23: if WsPrePossibleMatch && GoalPostPossibleMatch then
24:      return (WS Matches)
25: else
26:      return (WS doesn‟t Match)
27: end if
```

Following are the steps of capability filtering procedure, given in algorithm 1. Lines 1 to 11 are related to examining the objects in goal pre-conditions and WS pre-conditions. Lines 12 to 22 compare goal post-conditions and WS post-conditions‟ objects. Lines 23 to 27 return matched WS to the goal request if and only if the results of two previous parts are true.

Starting point of algorithm begins by getting the concepts and attributes of the goal pre-condition side which contain a conjunction of components. The WS pre-condition which may contain conjunction, disjunction and negation components is

converted into DNF. Then negative components are eliminated for the next step (the filtering algorithm looks for the presence of attributes and concepts, not their absence). The attributes and concepts of WS pre-condition in DNF are extracted to compare with attributes and concepts of goal pre-condition. If they match, *WsPrePossibleMatch* flag‟s status is changed to "True".

Next step is analysing of post-conditions parts of goal and WS. Goal post-condition is converted into DNF and its negative components are removed. Then, attributes and concepts of the goal post-condition in DNF are extracted to check with WS post-conditions‟ components. If they match, the *GoalPostPossibleMatch* flag‟s status is changed to "True".

Finally, lines 23 to 27 check whether both flags, *WsPrePossibleMatch* and *GoalPostPossibleMatch*, are "True" or not. If both are "True", a positive result is returned. Otherwise, the result is negative.

Complexity of Cap_Filt Algorithm:

The conversion of logic statements to DNF naturally dominates the time complexity of the algorithm, since in the worst case it is exponential (for example, logical formulas of the form $(X_{11} \vee X_{12} \vee ... X_{1k}) \wedge (X_{21} \vee X_{22} \vee ... X_{2k}) \wedge \ldots \wedge (X_{m1} \vee X_{m2} \vee ... X_{mk})$, where X_{ij}, $1 \leq i \leq m$, $1 \leq j \leq k$ are propositional literals, have k^m terms, each term consisting of conjunctions of *m* literals when converted to DNF). In the context of pre and post-conditions, literals are objects or negations of objects. However, in practical situations, we expect that the pre-conditions of WSs and post-conditions of goals would already be in DNF, eliminating the costly conversion process. Disregarding the DNF conversion, and assuming that the matching of a single attribute of a WS concept to a goal concept attribute takes constant time

through the usage of appropriate hashing mechanisms, and the number of or‟ed component of a WS has a practical maximum limit, then the complexity of matching a list of such attributes against goal concept attributes is linear in the size of the list.

A more detailed analysis of the time complexity of Cap_Filt Algorithm is as follows. Let *n* be the maximum of the number of attributes in the decomposed objects in each or‟ed component of the pre-condition of a WS in DNF, and the pre-condition of a goal. In algorithm 1, line one takes O(n) time, since goal.pre is a conjuction of positive literals, and the expression tree grows only to the right. Line 2, in the worst case, can be exponential in the number of or‟ed components, as already mentioned. Time complexity for processing line 3 is O(n), because we need to search all the extracted attributes of WS pre-condition and then eliminate the negative ones. The processing time of line 4 is O (1), because this line refers changing the status of *WsPrePossibleMatch* flag.

Assuming that there is a practical maximum limit *r* in the number of or‟ed componets in WS pre-condition, Line 5 iterates at most *r* times. Line 6 has time complexity O (n), since each attribute needs to be individually “read”. Line 7 involves matching of attribute values, and has time complexity O(n). Line 9 involves the change of flag status, and has constant time complexity O(1). As a result, the complexity of lines 5-10 is O(rn) = O(n).

An exactly symmetrical analysis can be made for lines 12-22 of the algorithm, resulting in an overall complexity of O(n), provided that the WS pre-condition and goal post-condition are already in DNF, and there is a practical maximum limit on

the number of or‟ed components. Otherwise, the time complexity is exponential in the worst case.

Chapter 6

IMPLEMENTATION

Chapter 6 outlises the implementation part of this dissertation which introduces the main predicates used in our pre-filtering technique. All of these predicates are written in FLORA-2. Furthermore, in this chapter, results of each predicate during two filtering stages according to our predefined scenario in chapter 4 are displayed step by step.

6.1 Main Predicates of Pre-filtering Strategies Written in FLORA-2

This section explains the main predicates which are employed in our two pre-filtering techniques. The predicates are *%FilterMain, %Filter_Cap, %FindGoalOrWsAtt, %Conv_DNF, %DC* and *%Check_Att_Cnp*.

6.1.1 %FilterMain Predicate

In listing 10, both category and capability filtering strategies are performed through the main predicate named *%FilterMain*. Output of this predicate is a list of goals and their related WSs which are inserted into the knowledge based named *RelatedGoalWsModule* for the subsequent logical matchmaker phase.

```
1:  %FilterMain :- ?_Inserted = setof{ ?Ins |
                          //---------First stage of filtering- Cat_Filt--------------
2:      ?GoalName[hasCategory->?GoalCat]@?_GoalModule,
3:      ?WsName[hasCategory->?WsCat]@?_WsModule,
4:      ((?WsCat :=: ?GoalCat) ; (?WsCat :: ?GoalCat) ; (?GoalCat ::?WsCat)),

                          //----------Second stage of filtering- Cap_Filt----------
5:      %Filter_Cap (?GoalName, ?WsName),
6:       alreadySelected(?WsName, WEBSERVICE)@FilteredWsModule,
7:       alreadySelected(?WsName, GOAL)@FilteredWsModule,
8:
9:       insert{related(?GoalName,?WsName)}@RelatedGoalWsModule,
10:     ?Ins=related(?GoalName,?WsName)
11:                          }.
```

Listing 10. Pre-filtering processes containing two filtering stages (lines 2 to 4: Cat_Filt, lines 5 to 11: Cap_Filt)

For each goal and WS pair, the first stage, *Cat_Filt* uses the *Global_Cat_Ont* ontology to check semantic similarity of the *goal category* (Cat_g) against the *Web service category* (Cat_w). According to listing 10 line 4, if Cat_g and Cat_w are equal, synonym or in an inheritance relationship with one another, the WS is kept for the next stage, otherwise it is discarded.

In the second stage, *Cap_Filt*, first attributes and concepts of objects utilized in the goal and the WS pre and post-conditions are extracted by our new algorithm (described in chapter 5). Then, extracted concepts and attributes as well as our ontology-based mediation are used to select WSs which satisfy the following conditions:

(i) Their pre-condition concepts and attributes are a subset of, equal to or synonymous with the goal pre-condition concepts and attributes,

(ii) Their post-conditions‟ concepts and attributes are superset, equal to or synonymous with the goal post-condition concepts and attributes.

Each goal is then logically tested for an exact match with only the WSs that survived the two-phase filtering process.

Note that our *Cap_Filt* deals with any logical expression involving the negation and disjunction operators as well as the conjunction operator.

6.1.2 %Filter_Cap Predicate

As it is shown in line 5 of listing 10, in order to perform Cap_Filt algorithm we call another predicate *%Filter_Cap. %Filter_Cap* sequentially calls four main predicates named *%FindGoalOrWsAtt*, *%Conv_DNF*, *%DC* and *%Check_Att_Cnp*. The following sections explain all of the mentioned predicates in detail.

Listing 11 depicts critical parts of the *%Filter_Cap* predicate. Filtering based on concepts and attributes of objects in the capability specification of the WS and goal is carried out in the following manner:

1) Lines 2 to 7 read goal and WS pre and post conditions from their individual modules.
2) As the process of checking semantic and synonymous similarity of goal and WS specifications is done in the knowledge base module (*GoalWsAttModul),* in listing 11 line 8, attributes and concepts of goal pre-condition are inserted into *GoalWsAttModule* through the *%FindGoalOrWsAtt* predicate. Its source code is available in listing 12.
3) In order to find a comparable list of concepts and attributes in WS pre-condition with concepts and attributes in goal pre-condition, line 9 of listing 11 calls *%Conv_DNF* predicate to convert the WS pre-condition into

Disjunctive Normal Form (DNF). Output is a WS pre-condition in DNF. *%Conv_DNF* is completely explained in listing 13.

```
1:  %Filter_Cap (?GoalName, ?WsName):-

                    //----------Pre-Condition --------

2:      ?GoalName[requestsCapability->?GCap]@?GoalModule,
3:      ?GCap ~ ${?_GCapability[
4:       hasPrecondition->?GoalPre ,hasPostcondition->  ?GoalPost]}@?GoalModule,

5:      ?WsName[hasCapability->?Wcap]@?WsModule,
6:      ?Wcap ~ ${?_WSCapability[
7:       hasPrecondition-> ?WsPre,hasPostcondition->?WsPost]}@?WsModule,

8:       %FindGoalOrWsAtt (?GoalPre, GoalWsAttModule),
9:       %Conv_DNF(?WsPre,?DNFedWsPre),
10:      %DC (?DNFedWsPre, ?Ws_Pre_Att_Cnp),
11:      %Check_Att_Cnp (?WsName, ?Ws_Pre_Att_Cnp, WEBSERVICE),

                    //----------Post-Condition --------

12:      deleteall{?_A[?_B->?_V]:?_C @GoalWsAttModule},

13:     %FindGoalOrWsAtt (?WsPost,GoalWsAttModule),
14:     %Conv_DNF(?GoalPost,?DNFedGoalPost),
15:     %DC (?DNFedGoalPost, ?Goal_Post_Att_Cnp),
16:     %Check_Att_Cnp (?WsName, ?Goal_Post_Att_Cnp, GOAL).
```

Listing 11. Critical parts of *%Filter_Cap* predicate

4) Attributes and concepts of the WS pre-condition in DNF are extracted via *%DC* predicate. Source code of this predicates is presented in listing 14. Line 10 of listing 11 shows that this predicate is called with the WS pre-condition in DNF *(?DNFedWsPre)* as input parameter and an unbound variable *?Ws_Pre_Att_Cnp* as output parameter. *?Ws_Pre_Att_Cnp* is the list of concepts and their corresponding attributes in WS pre-conditions.
5) Finally, line 11 depicts *%Check_Att_Cnp* predicate that implements algorithm 2 explained in section 6.1.6. This predicate compares concepts and attributes related to goal pre-conditions with concepts and attributes

associated to WS pre-conditions based on their semantic and synonymous similarity.

Comparison of goal and WS post-conditions is similar to the pre-conditions, except for some changes in predicates‟ parameters.

6) As it is shown in line 12 of listing 11, contents of knowledge base *GoalWsAttModule* which already consists of goal pre-condition‟s attributes and concepts, is deleted in order to replace it with the new data.
7) Attributes and concepts of WS post-conditions are inserted into *GoalWsAttModule* module by *%FindGoalOrWsAtt* predicate in line 13 of listing 11**.**
8) In line 14 of listing 11, goal post-conditions are converted into DNF through the *%Conv_DNF* predicate. This predicate calls *?GoalPost* as input parameter and the goal post-condition in DNF is the output of this predicate.
9) Attributes and concepts of the goal post-condition in DNF *(?DNFedGoalPost)* are extracted via *%DC* predicate, then results are stored in *?Goal_Post_Att_Cnp* variable as shown on line 15.
10) Line 16, similar to line 11, *%Check_Att_Cnp* implements algorithm 2. However, this time it checks concepts and attributes related to WS post-conditions with concepts and attributes associated to goal post-conditions based on semantic equivalency between them.

If all these checks succeed, then the pair of goal and its related WSs are inserted into the knowledge base so that full checking of the proof commitments can be carried out in the next stage.

6.1.3 % FindGoalOrWsAtt Predicate

As it is shown in lines 8 and 13 of listing 11, attributes and concepts of goal pre-conditions and WS post-condition which only contain conjunction logical connectives are sequentially inserted into *GoalWsAttModule* through *%FindGoalOrWsAtt* predicate. Source code of this predicate is depicted in listing 12.

Lines 1 to 6 of listing 12 demonstrate that if goal pre-condition or WS post-condition contains only an attribute or a concept, they are inserted directly into the *GoalWsAttModule* module. However, if goal pre-condition or WS post-condition contains more than one concept and/or an attribute, they are decomposed through the meta-decomposition operator "**=..**" in FLORA-2 (FLORA-2 supports an extended version of the Prolog meta-decomposition operator "**=..**"; the main use of the "**=..**" operator in FLORA-2 is for decomposing HiLog terms or reifications of HiLog predicates and F-logic frame literals [48]). Then, the retrieved attributes and concepts are inserted into *GoalWsAttModule* module as it is shown in lines 7 to 13 of listing 12.

```
1:   % FindGoalOrWsAtt (?Cap,?GoalWsAttModule):-
2:                          ?Cap ~ ${?A[?B->?C]@?_Module1},
3:                           insert{?A[?B->?C]@?GoalWsAttModule }.

4:   % FindGoalOrWsAtt (?Cap,?GoalWsAttModule):-
5:                          ?Cap ~ ${?A:?Concept@?_Module1},
6:                            insert{?A:?Concept}@?GoalWsAttModule.

7:   % FindGoalOrWsAtt (?Cap,?GoalWsAttModule):-
8:                          ?Cap =..?L,
9:                          ?L = [?H,?F,?S],
10:                         ?H = logic(and),
11:                         !,
12:                         % FindGoalOrWsAtt (?F,?GoalWsAttModule),
13:                         % FindGoalOrWsAtt (?S,?GoalWsAttModule).
```

Listing 12. Source code of *%FindGoalOrWsAtt* predicate in FLORA-2

6.1.4 %Conv_DNF Predicate

This predicate is the heart of our Cap_Filt strategy since it can deal with objects that occur in a logical formula with full usage of the logical connectives conjunction, disjunction and negation. As already explained, the main aim of the %Conv_DNF predicate is converting WS pre-conditions and goal post-conditions into DNF. Listing 13 depicts the algorithm implemented by %Conv_DNF algorithm in Haskell syntax. This algorithm converts any propositional formula into DNF. The definition of %Conv_DNF is much more involved due to the necessity of dealing with embedded objects and the particularities of the FLORA-2 system. The full code of the %Conv_DNF predicate, however, is available in Appendix A.

```
1:  data Propositional = Var String |
2:                       And Propositional Propositional |
3:                       Or Propositional Propositional |
4:                       Not Propositional

5:  conv_dnf f = (dnf (nnf f))

6:  nnf (Var a) = (Var a)
7:  nnf (Not (Var a)) = (Not (Var a))
8:  nnf (Not (Not a)) = nnf a
9:  nnf (Not (And a b)) = (Or (nnf (Not a))
10:                           (nnf (Not b)) )

11: nnf (Not (Or a b)) = (And (nnf (Not a))
12:                           (nnf (Not b)) )

13: nnf (And a b) = (And (nnf a) (nnf b))

14: nnf (Or a b) = (Or (nnf a) (nnf b))

15: in_dnf (Or a b) = (in_dnf a) && (in_dnf b)
16: in_dnf a = all_conj a

17: all_conj (And a b) = (all_conj a) && (all_conj b)
18: all_conj (Var a) = True
19: all_conj (Not (Var a)) = True
20: all_conj a = False

21: dnf f
22:   | in_dnf f = f
23:   | otherwise = case f of
24:                   (Or a b) →  (Or (dnf a) (dnf b))
25:                   (And a b) → process_and (dnf a) (dnf b)

26: process_and (Or p q) (Or r s) =  Or
27:                                   (Or (dnf (And p r) ) (dnf (And q r)))
28:                                   (Or (dnf (And p s))  (dnf (And q s)))

29: process_and (Or p q) b2 = Or (dnf (And p b2))
30:                               (dnf (And q b2))

31: process_and a2 (Or r s) = Or (dnf (And a2 r))
32:                               (dnf (And a2 s))
```

Listing 13. Algorithm of *Conv_DNF* function in Haskell syntax

The following is a brief explanation regarding the internal steps of the *Conv_DNF* function:

i) As it is displayed in listing 13, main function of algorithm, ***conv_dnf,*** calls two other functions are named ***nnf*** and ***dnf*** step by step.

ii) Presentation of ***nnf*** function illustrated in lines 6 to 14 of listing 13. ***nnf*** converts its parameter to Negation Normal Form (NNF).

iii) Then, ***dnf*** function first checks its argument to see if it is already in DNF. If so, returns it argument as is. If not,

iv) then it must belong to one of the cases below:

A) The outermost operator is disjunction, in which case the function is called recursively on each sub-part and the result is the disjunction of the two sub-results,

B) The outermost operator is a conjunction, in which case the two sub-parts are passed as parameters to recursive calls to the function ***dnf***, and the sub-results are passed to the function ***process_and*** for proper handling of the various cases.

6.1.5 %DC Predicate

Listing 14 presents the source code of *%DC* predicate in FLORA-2. This transactional predicate sequentially extracts attributes and concepts of the WS pre-condition in DNF and the goal post-condition in DNF. Then, extracted concepts and attributes are stored in different lists.

```
1:  %DC (?X,?R):- ?X =  (?A ;?B),
2:                      !,
3:                      % DC (?A,?A1),
4:                      % DC (?B,?B1),
5:                      %append(?A1,?B1,?R).

6:  % DC (?Conj,?R):-   \true,
7:                      !,
8:                      % DC_and(?Conj,?ListAtt),
9:                      ?R=[?ListAtt].

10: % DC_and(?X,?R):-   ?X=..?L,
11:                     ?L=[?A,?_B,?C,?_D],
12:                     ?A=flogic('->',?_Module),
13:                     !,
14:                     ?R=[?C].

16: % DC_and(?X,?R):-  ?X=..?L,
17:                     ?L=[?A,?B,?C],
18:                     ?A=logic(and) ,
19:                     ?B =${?_Obj:?Concept@?_M},
20:                     !,
21:                     % DC_and(?C,?C1),
22:                     ?R=[(?Concept,?C1)].

23: % DC_and(?X,?R):-   ?X=..?L,
24:                     ?L = [?A,?B,?C],
25:                     ?A = logic(and) ,
26:                     ?B = ${?_Obj[?At->?_Val]@?_M},
27:                     !,
28:                     % DC_and(?C,?C1),
29:                     ?R = [?At|?C1].

30: % DC_and(?X,?R):-   ?X=..?L,
31:                     ?L=[?A,?B,?C],
32:                     ?A=logic(and) ,
33:                     !,
34:                     % DC_and(?B,?B1),
35:                     % DC_and(?C,?C1),
36:                     %append(?B1,?C1,?R).

37: % DC_and(?X,?R):- ?X ~ ${nothing:Nothing@main},
38:                    !,
39:                    ?R=[].
40: %append([],?_X,?_X).
41: %append([?_H|?T],?Y,[?_H|?R]):- %append(?T,?Y,?R).
```

Listing 14. Source code of *%DC* predicate in FLORA-2

6.1.6 %Check_Att_Cnp Predicate

The following are the steps of *%Check_Att_Cnp* predicate given in algorithm 2. This predicate contains two stages, (i) it first compares concepts and attributes related to goal pre-condition with concepts and attributes associated to WS pre-condition, (ii) then, concepts and attributes related to WS post-condition are investigated with concepts and attributes associated to goal post-condition based on semantic equivalency between them.

Output of *%Check_Att_Cnp* predicate is the name of related WSs whose concepts and attributes exist in the requested goal, as it is shown in algorithm 2. A WS name which is passed through this level of filtering is stored in the knowledge base named *FilteredWsModule.*

Source code of *%Check_Att_Cnp* predicate is available in appendix B.

Algorithm 2. Filtering by comparing concepts and attributes

Input: List1 of the form [(Concept, [ListOfAttributes]), …]
(extracted from **either** Goal.Pre **or** Ws.Post)

List2 of the form [(Concept, [ListOfAttributes]),…]
(extracted from **either** Ws.Pre **or** Goal.Post)

Tag (either GOAL or WEBSERVICE)

Output: (WsName,Tag) (as insertion into module FilteredWsModule)

```
1:  Let equiv(A,B) = (A::B) or (B::A) or (A:=:B)
2:  PossibleMatch ← True
3:  for all (Concept2,[ListOfAttributes2]) ∈ List2 do
4:     if not ∃ (Concept1,[ListOfAttributes1]) ∈ List1 (
5:                  equiv(Concept1,Concept2) and
6:                  ∀ (attribute2 ∈ [ListOfAttributes2]) (
7:                     ∃ (attribute1∈ [ListOfAttributes1]) (
8:                        equiv(attribute1,attribute2)
9:                     )
10:                 )
11:           )
12:     then
13:        PossibleMatch ← False
14:    end if
15: end for
16: if
17:     PossibleMatch = True
18: then
19:     Insert (WsName,Tag) into FilteredWsModule
20: end if
```

6.2 Demonstration of the Workings of Defined Pre-filtering Predicates through the Examples

In order to better understand the working of aforementioned predicates, we use WS and goal instances which were already explained in chapter 4. The results of pre-filtering predicates over the instances are shown step by step.

6.2.1 Results Obtained by Employing Category Filtering Predicates

Consider the listings 4 and 5 of chapter 4 which depict Goal #1 and Goal #2 instances. In these two requests, users are looking for a flight and a restaurant consecutively.

Category of Goal #1 is *AirTransportation* and category of Goal #2 is *Restaurant*. Besides, category of SWS #1 is *AeroplaneTransportation*, category of SWS #2 is *Food*, category of SWS #3 is *Transportation* and finally category of SWS #4 is *Learning*.

As already explained, *Cat_Filt* algorithm is based on concept relationships definition in *Global_Cat_Ont* ontology. Outputs of *Cat_Filt* strategy over the above described instances are illustrated in table 9.

Table 9. Results of Cat_Filt algorithm over the described scenarios

	Cat$_{SWS\#1}$ AeroplaneTransportation	**Cat**$_{SWS\#2}$ Food	**Cat**$_{SWS\#3}$ Transportation	**Cat**$_{SWS\#4}$ Learning
Cat$_{Goal\#1}$ AirTransportation	√	×	√	×
Cat$_{Goal\#2}$ Restaurant	×	√	×	×

As it is shown in table 9, retrieved WSs based on Goal#1 category filtering are SWS#1 and SWS#3. Since, category of SWS#1 which is *AeroplaneTransportation*, is a synonym of Goal#1 category, *AirTransportation*, and category of SWS#3 which is *Transportation*, is a superclass of *AirTransportation*. Therefore, SWS#1 and SWS#3 remain as matched WSs for Goal#1 and are forwarded to the next step of checking in the *Cap_Filt* algorithm. However, SWS #2 and SWS #4 are discarded as irrelevant WSs for Goal#1.

The only category matched WS for Goal#2 is SWS#2. Since SWS#2 belongs to *Food* category and according to concepts‟ relationships definition in *Global_Cat_Ont* ontology it is a superclass of Goal #2 category, *Restaurant*.

6.2.2 Results Obtained by Employing Capability Filtering Predicates

WSs that passed through the *Cat_Filt* algorithm are entered as inputs of *Cap_Filt* algorithm in order to filter based on attributes and concepts of objects in the capability specification of the WS and goal. Following sub-sections show results obtained by applying the available predicates in *Cap_Filt* over the goals and WSs instances in a step by step manner.

Output of %FindGoalOrWsAtt Predicate: In this predicate attributes and concepts of goal pre-conditions and WS post-conditions which only contain conjunction logical connectives are sequentially inserted into *GoalWsAttModule* module.

Table 10 depicts attributes and concepts of both Goal#1 and Goal#2 pre-condition which are inserted into *GoalWsAttModule* module.

Table 10. Attributes and concepts of both Goal #1 and Goal #2 pre-condition

Goals	Attributes and concepts
Goal #1	**Concept:** RequestFlightTicket **Attributes:** originateCity, terminalCity
Goal #2	**Concept:** RequestRest **Attributes:** restName, inCity, foodSource, numberPeople

Our predicate also inserts attributes and concepts of WS post-condition into the *GoalWsAttModule* module. Table 11 depicts attributes and concepts of all WS instances in our scenario.

Table 11. Attributes and concepts of SWS #1 to SWS #4 post-conditions

WSs	Attributes and concepts
SWS #1	**Concept:** Response **Attributes:** fromAirport, toAirport, cost
SWS #2	**Concept:** ResponseRest **Attributes:** restName, address, maxPrice
SWS #3	**Concept:** Response **Attributes:** originateAirport, toAirport, fromHarbor, toHarbor, cost
SWS #4	**Concept:** ResponseSch **Attributes:** schName, address, cost

Output of %Conv_DNF Predicate: In order to get a comparable list of concepts and attributes in WS pre-condition with concepts and attributes in goal pre-condition we need to convert WS pre-condition into DNF.

Also, in order to compare concepts and attributes of goal post-condition with WS post-condition, we need to convert goal post-condition into DNF.

This section presents the structure of WS pre-condition and goal post-condition before and after converting into DNF. Conversion into DNF is according to the algorithm described in section 6.1.4.

Tables 12 to 17 depict pre-conditions of our WSs and post-conditions of our goals instances before and after converting into DNF. Tables 12 to 15 belong to SWS #1, SWS #2, SWS #3 and SWS #4 pre-conditions. Tables 16 and 17 display Goal #1 and Goal #2 post-conditions.

Table 12 illustrates SWS #1 pre-condition before and after converting into DNF. To better understand the contents of table 12, consider the following assumptions. Suppose that we assign terms A, B, C and D instead of the objects in SWS #1 pre-condition. Term „A‟ is utilized instead of *?ReqFlight* object, term „B‟ is used instead of *?PaymentType1*, term „C‟ is used for *?PaymentType2* and finally term „D‟ is used instead of *?SomeFlight*.

According to the following assumptions, the structure of SWS #1 pre-condition before converting to DNF was ((A ∧ (B ∨ C)) ∧ D) which is not in DNF. Therefore, after employing *%Conv_DNF* predicate, the new structure of SWS #1 pre-condition is ((A ∧ (B ∧ D)) ∨ (A ∧ (C ∧ D))) which is in DNF. The new structure can be compared with the goal pre-condition.

Table 12. SWS#1 pre-condition before and after converting into DNF

SWS #1	**Before**	**(**${?ReqFlight[startCity−>?FromCity, endCity−>?ToCity]: RequestFlightTicket }**,** **(**${?PaymentType1[creditNumber−>?CreditNo, expireDate−>?ExpreDate]: CreditCard **};** ${?PaymentType2[accountName−>?AccName, accountNumber−>?AccNo]:PayPal **}),** (${?SomeFlight[fromAirport−>?FromAirport, toAirport−>?ToAirport, cost−>?Cost]:Flight } **))**
	After	**((**${?ReqFlight[startCity−>?FromCity, endCity−>?ToCity]: RequestFlightTicket **},** **(**${?PaymentType1[creditNumber−>?CreditNo, expireDate−>?ExpreDate]: CreditCard **},** ${?SomeFlight[fromAirport−>?FromAirport, toAirport−>?ToAirport, cost−>?Cost]:Flight **));** **(**${?ReqFlight[startCity−>?FromCity, endCity−>?ToCity]: RequestFlightTicket **},** **(**${?PaymentType2[accountName−>?AccName, accountNumber−>?AccNo]:PayPal **},** ${?SomeFlight[fromAirport−>?FromAirport, toAirport−>?ToAirport, cost−>?Cost]:Flight **)))**

Table 13 illustrates contents of SWS #2 pre-condition. SWS #2 pre-condition is already in DNF, so after converting it into DNF, it will be the same as before. Only predicates are eliminated, because our *Cap_Filt* algorithm only examines attributes and concepts and does not deal with predicate checking.

Table 13. SWS #2 pre-condition before and after converting into DNF

SWS #2	**Before**	($\{?FindRest[restName −>?Name, inCity −>?City, numberPeople −>?HNumber, foodSource −>?Source]:RequestRest }, $\{?SomeRestaurant[restName −>?Name, inCity −>?City, foodSource −>?Source, address −>?Adr, maxPrice −>?Cost]:Food }, mult(?Cost,?HNumber,?TotalCost))
	After	($\{?FindRest[restName −>?Name, inCity −>?City, numberPeople −>?HNumber, foodSource −>?Source]:RequestRest }, $\{?SomeRestaurant[restName −>?Name, inCity −>?City, foodSource −>?Source, address −>?Adr, maxPrice −>?Cost]:Food })

Table 14 illustrates SWS #3 pre-condition before and after conversion into DNF. To better understand the table contents, consider term „A" as the first object, *?ReqShip*,

term „B‟ as second object, *?SomeShip*, term „C‟ as third object, *?ReqFlight*, and finally term „D‟ as last object, *?SomeFlight*.

As it is displayed in table 14, the structure of SWS #3 pre-condition before converting into DNF with the new assumption terms was ((A ∧ B) ∨ (C ∧ D)) which is in DNF. So, the structure of SWS #3 pre-condition will not change after calling %*Conv_DNF* predicate.

Table 14. SWS #3 pre-condition before and after converting into DNF

SWS #3	**Before**	((${?ReqShip[startCity->?FromCity, toCity->?ToCity]:RequestShipTicket } , ${?SomeShip[fromHarbor ->?FromHarbor, toHarbor ->?ToHarbor, cost->?Cost]:Ship }) ; (${?ReqFlight[fromCity->?FromCity, destinationCity ->?ToCity]: RequestAirplainTicket }, ${?SomeFlight[fromAirport->?FromAirport, toAirport->?ToAirport, cost->?Cost]:Flight }))
	After	((${?ReqShip[startCity->?FromCity, toCity->?ToCity]:RequestShipTicket } , ${?SomeShip[fromHarbor ->?FromHarbor, toHarbor ->?ToHarbor, cost->?Cost]:Ship }) ; (${?ReqFlight[fromCity->?FromCity, destinationCity ->?ToCity]: RequestAirplainTicket }, ${?SomeFlight[fromAirport->?FromAirport, toAirport->?ToAirport, cost->?Cost]:Flight }))

Table 15 depicts contents of SWS #4 pre-condition before and after converting into DNF. As it is presented, the structure of SWS #4 is already in DNF, so after calling *%Conv_DNF* predicate, the structure will not change and it will be the same as before.

Table 15. SWS #4 pre-condition before and after converting into DNF

SWS #4	**Before**	($\{?FindSch[schName->?Name, inCity->?City, schoolType->?SchoolType, gender->?Gender]:RequestSch }, $\{?SomeSch[schName->?Name, inCity->?City, gender->?Gender, address->?Adr, schoolType->?SchoolType, cost->?Cost]:Education })
	After	($\{?FindSch[schName->?Name, inCity->?City, schoolType->?SchoolType, gender->?Gender]:RequestSch }, $\{?SomeSch[schName->?Name, inCity->?City, gender->?Gender, address->?Adr, schoolType->?SchoolType, cost->?Cost]:Education })

Tables 16 and 17 display the post-conditions of Goal #1 and Goal #2 respectively. As it presented after converting both Goal #1 and Goal #2 post-conditions into DNF, only the predicates like, *less*, *\+ is _equal* and *greater* are eliminated, because our Cap_Filt algorithm only examines attributes and concepts and don‟t deal with predicate checking.

Table 16. Goal #1 post-condition before and after converting into DNF

Goal #1	**Before**	(${?BookTicket[fromAirport −>?FromAirport, toAirport −> ?ToAirport, cost −> ?Cost]:Response }, \+ is_equal(?ToAirport, Sabiha_Gokcen), less (?Cost, 500)),
	After	${?BookTicket[fromAirport −>?FromAirport, toAirport −>?ToAirport, cost −> ?Cost]:Response }

Table 17. Goal #2 post-condition before and after converting into DNF

Goal #2	**Before**	(${?BookRest[restName−>?Name, address−>?Adr, maxPrice−>? TotalCost]:ResponseRest }, \+ greater (?TotalCost, 150))
	After	${?BookRest[restName−>?Name, address−>?Adr, maxPrice−>? TotalCost]:ResponseRest }

Output of %DC Predicate: This transactional predicate sequentially extracts attributes and concepts of the WS pre-condition in DNF and the goal post-condition in DNF. Then, it stores extracted concepts and attributes in different lists.

Tables 18 and 19 illustrate list of extracted attributes and concepts of WSs pre-conditions and goals post-conditions through the *%DC* predicate.

Table 18. List of extracted attributes and concepts of SWS #1 to SWS #4 pre-conditions

WSs	[(ConceptName, [List of attributes])]
SWS #1	[[(RequestFlightTicket,[startCity, endCity]) , (CreditCard, [creditNumber, expireDate]) , (Flight, [fromAirport, toAirport, cost])] , [(RequestFlightTicket,[startCity, endCity]) , (PayPal, [accountNam, accountNumber]) , (Flight, [fromAirport, toAirport, cost])]]
SWS #2	[[(RequestRest, [restName, inCity, numberPeople, foodSource]), (Food , [restName, inCity, foodSource, address, MaxPrice])]]
SWS #3	[[(RequestShipTicket, [startCity, toCity]), (Ship, [fromHarbor, toHarbor, cost])] , [(RequestAirplainTicket, [fromCity, destinationCity]), (Flight, [fromAirport, toAirport, cost])]]
SWS #4	[[(RequestSch, [schName, inCity, schoolType, gender]), (Education, [schName, inCity, gender, address, schoolType, cost])]]

Table 19. List of extracted attributes and concepts of Goal #1 and Goal #2 post-conditions

Goals	[(ConceptName, [List of attributes])]
Goal #1	[[(Response, [fromAirport, toAirport, cost])]]
Goal #2	[[(ResponseRest, [restName, address, maxPrice])]]

Output of %Check_Att_Cnp Predicate: This predicate compares extracted concepts and attributes related to goal with concepts and attributes associated to WS based on semantic equivalency between them. Extracted concepts and attributes are presented in tables 10, 11, 18 and 19.

All extracted concepts and attributes of WS pre-condition are checked with extracted concepts and attributes of goal pre-condition. If all were equal or synonyms or belong to the same class, name of WS along with “WEBSERVICE” tag are inserted into *FilteredWsModule*.

For instance, consider SWS #3 and Goal #1. Attributes and concepts of Goal #1 and SWS #3 pre-conditions are depicted in table 20.

Table 20. Attributes and concepts of Goal #1 and SWS #3 pre-conditions

Goal#1	**Concept:** RequestFlightTicket **Attributes:** originateCity, terminalCity
SWS#3	[[(RequestShipTicket, [startCity, toCity]), (Ship, [fromHarbor, toHarbor, cost])] , [(RequestAirplainTicket, [fromCity, destinationCity]), (Flight, [fromAirport, toAirport, cost])]]

SWS #3 provides either ship reservation service or flight reservation service. Therefore, it could be considered as a proper service for Goal #1 which is looking for a flight ticket reservation.

At this point, *%Check_Att_Cnp* predicate checks whether the concepts and attributes in goal pre-condition exist in WS pre-condition or not.

This predicate first checks whether concepts of Goal #1 pre-condition are among the concepts of WS pre-condition or not. Concept of Goal #1 pre-condition, *RequestFlightTicket*, is synonym with concept of SWS #3 pre-condition, *RequestAirplainTicket*.

Then attributes belonging to the matched concept are examined as well. As it is displayed in table 20, *originateCity* and *fromCity* are the first attribute of Goal #1 pre-condition and SWS #3 pre-condition and both belong to the same concept. Although the spelling of these two attributes is different and they may not have any relation in domain ontology, they have the identical meaning. Our approach tackles this problem and considers the attributes similar to each other through the dictionary of synonymous words in the mediator.

Next attribute of goal pre-condition, *terminalCity*, is also synonym with the WS pre-condition attribute, *destinationCity*. Therefore, pre-conditions of SWS #3 and Goal #1 are semantically equivalent, and name of SWS #3 along with "WEBSERVICE" tag are inserted into *FilteredWsModule* module.

Note that two concepts *Ship* and *Flight* are not checked by *%Check_Att_Cnp* predicate because they came from external ontology.

Similar to pre-conditions we need to examine semantic equivalency of post-conditions as well. So, next step is checking the concepts and attributes of SWS #3 and goal #1 post-conditions according to tables 11 and 19.

Table 21. Attributes and concepts of Goal #1 and SWS #3 post-conditions

Goal#1	[[(Response, [fromAirport, toAirport, cost])]]
SWS#3	**Concept:** Response **Attributes:** originateAirport, toAirport, fromHarbor, toHarbor, cost

Table 21 depicts concepts and attributes of Goal #1 and SWS #3 post-conditions. As it is shown concepts of both goal and WS post-conditions are the same. Therefore, checking the attributes of matched concepts is started.

First attributes of Goal #1 and SWS #3 are *fromAirport* and *originateAirport* respectively, and through the dictionary of synonym words [55] in our mediator, are synonym. Also the rest attributes of goal post-condition, *toAirport* and *cost* are exactly available among the existing attributes in WS post-condition. Therefore, post-conditions of SWS #3 and Goal #1 are semantically equivalent, and name of SWS #3 along with "GOAL" tag are inserted into *FilteredWsModule* module.

Output of %FilterMain Predicate: This predicate checks both pre-filtering algorithms, and output is list of goals and their related WSs which are inserted into the knowledge based named *RelatedGoalWsModule* for the subsequent logical matchmaker phase.

By considering the mentioned instances, outputs of *Cat_Filt* algorithm are SWS #1 and SWS#3 as proper WSs for Goal #1 based on their category filtering.

However, as output of Cap_*Filt* algorithm which examines goals and WSs based on their concepts and attributes only SWS#3 remains as proper WS to the requested goal, Goal #1.

Therefore, SWS #3 are inserted into *RelatedGoalWsModule* as related WS to the Goal #1 for logical checking through the logical matchmaker. In the logical

matchmaker engine retrieved related WSs are checked with their specified goals based on ontology matching of their attributes‟ value.

Chapter 7

EXPERIMENTAL EVALUATIONS

Chapter 7 contains experimental date, experimental environments, experimental results and related works. The graphical and tabular results were obtained on our test collection, WSMO-FL V2, after employing the pre-filtering steps, are shown in this chapter.

7.1 Available Test Collections

A proper test collection is needed in order to evaluate the suitability and performance of service discovery frameworks. Currently, two de-facto test collections are OWLS-TC[8] and SAWSDL-TC[9]. OWLS-TC, which mainly considers input and output parameters, is applicable for approaches that deal with OWL-S WSs descriptions, Approaches which employ SAWSDL WS descriptions use the SAWSDL-TC test collection. Another test collection uses WSMO-lite for the description WSs [13], a bottom-up approach for annotating web services.

7.1.1 OWLS-TC

The latest version of OWLS-TC at the time this dissertation is written is version 4 [45]; it consists of 1083 WSs and 42 queries which are written in the OWL-S language. Unfortunately, the majority of WSs in OWLS-TC are only partially described, being based on input and output types. Only in the last version (version 4),

[8] http://projects.semwebcentral.org/projects/owls-tc/
[9] http://projects.semwebcentral.org/projects/sawsdl-tc

160 WSs contain pre-conditions and post-conditions (effects) which are described in different languages such as, SWRL[10] and PDDL [27].

7.1.2 SAWSDL-TC

The SAWSDL-TC test collection is established to support the performance appraisal of SAWSDL matchmakers. The latest version of SAWSDL-TC, at the time this dissertation is written is version 3; it consists of 1080 semantic WSs and 42 queries which are described in the SAWSDL language. However, descriptions of WSs and queries are only based on input and output parameters [39].

7.1.3 WSMO-Lite TC

WSMO-Lite, lightweight service ontology intended for semantic annotations of the Web Service Description Language WSDL. In contrast to SWS frameworks such as OWL-S and WSMO, WSMO-Lite simplifies the semantic descriptions and enables bottom-up semantic annotation of WSs, but very importantly, it also relaxes the requirements on completeness of semantic descriptions, which enables building incremental layers of semantics on top of existing service descriptions(SAWSDL, MicroWSMO).

7.2 Why Existing Test Collections are not suitable for Evaluating our Semantic Web Service Filtering Strategy

Our semantic web service filtering strategy requires the specification of goals and web services that have complex logical expressions (including disjunction, conjunction and negation) in their pre- and post-conditions, as well as a categorization scheme. Since none of the existing test collections (including WSMO-lite) have web service and goal descriptions with these features, we could not use

[10] SWRL: A Semantic Web Rule Language, http://www.w3.org/Submission/SWRL/

them and were forced to produce a novel test collection, *WSMO-FL*[11] that does support them.

7.3 WSMO-FL: A New Test Collection for Web Services based on an Extended Version of WSMO using FLORA-2

The majority of approaches (such as [26, 46, 40, 56, 68, 2]) that work in our field and are mentioned in related works, evaluate efficiency and accuracy of their works based on OWLS-TC version 3 test collection. Among all related works, authors of [17] evaluated their proposal based on last version of OWLS-TC test collection, but only input and output parameters are considered for the evaluation of their work. As explained in the previous section, we generated our own test collection of WS and goal specifications, and used this test collection to measure the gains in efficiency obtained by employing our proposed pre-filtering strategy. We called our test collection

WSMO-FL contains three different domains, namely *transportation*, *food* and *education*, with 250 different F-Logic WSs descriptions, 6 different F-Logic goals descriptions, 22 concepts, 3100 attributes and 1225 instances.

In [26] we used WSMO-FL V1. The first version of our test collection only contains objects and predicates that occur in a logical formula with usage of the conjunction (and) logical operator.

[11] http://cmpe.emu.edu.tr/samira/WSMO-FL.htm

However, the new version, WSMO-FL V2, which is the test collection of our current thesis, improved the previous version. This second version involves WSs and goals with the objects and predicates that occur in logical formulas with full usage of the conjunction (and), disjunction (or) and negation (not) logical operators.

7.4 Experimental Environment

In this chapter, in order to validate our proposal, we performed experimental evaluations described and the results of that experimental study. For analysis, each test has been run 20 times, and it performed on a PC running Windows 7 OS, with a 2.93 GHz Intel processor and 4.00 GB of RAM.

7.5 Experimental Results

In order to determine the actual improvements of our proposed pre-filtering stages, we measured several indicators: (i) The average response time of our SWS matchmaker with filtering (*Filt_Disc*) and without filtering (*Naive_Disc*). (ii) The number of WSs that have been effectively eliminated from the initial pool of available WSs at each pre-filtering stage, (iii) Precision, recall and fallout.

Due to the fact that our filtering stages never eliminate any WS from consideration unless they are guaranteed to fail at the logical matching stage, it is no surprise that recall rate is always 100%.

7.5.1 Average Response Time

The results of the performed tests for the goal are given in table 22, showing the mean and median of the time it took to match the goal against varying number of WSs. The statistical measures (mean, median) were computed over 20 runs which yielded the raw data. Timing data was recorded for the two cases of matchmaker

using the pre-filtering phases *Filt_Disc* and matchmaker using no filtering at all *Naive_Disc*.

Table 22. Statistical comparison of Filt_Disc and Naive_Disc

No. WS	Engine	Mean time (ms)	Median time (ms)
10	Filt_Disc	57	72
	Naive_Disc	3065	3125
50	Filt_Disc	903	980
	Naive_Disc	10056	9985
100	Filt_Disc	1841	1806
	Naive_Disc	27775	27685
150	Filt_Disc	2005	1909
	Naive_Disc	39666	39295
200	Filt_Disc	2982	2983
	Naive_Disc	65892	65886
250	Filt_Disc	3569	3665
	Naive_Disc	82158	81924

Figure 13 graphically depicts the same information as a line chart. It can be seen that when using *Filt_Disc*, the average response time is in range of 57 to 3569 milliseconds, while for the same goal and WSs in *Naive_Disc* it dramatically increases and is in range of 3065 to 82158 milliseconds.

Curvefitting of the data in Table 22 (also in figure 13) using the online curvefitting tool [61] gives the linear formula:

ag *t*

.

Using this formula, we can predict that for 1000 WSs, the average response time will be 14,161.761 ms. This is a reasonable response time for most cases. For example, consider a human user being helped by an intelligent agent employing semantic web technology to find appropriate WSs for an airline reservation; and 14 seconds for automatic discovery is most likely acceptable. We can thus claim that our discovery framework, through its pre-filtering stages, handles realistic numbers of WSs well.

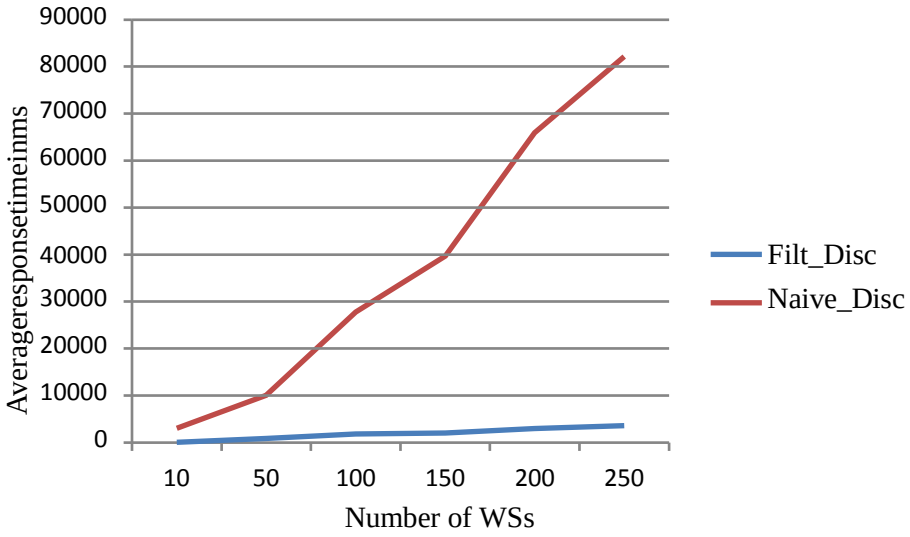


Figure 13. Comparison of Filt_Disc and Naive_Disc

7.5.2 Effectiveness of the Pre-filtering Algorithms in Eliminating Irrelevant Web Services

Figure 14 depicts the dramatic number of reductions in the number of WSs that remain after each pre-filtering phase. The data has been collected by matching six different goals and varying number of WSs for each goal. The chart indicates that *Cap_Filt* through the semantic equivalency of goal and WS concepts and attributes does a very good job of eliminating irrelevant WSs, given that most of the remaining WSs after its application passes the *Cap_Filt* stage.

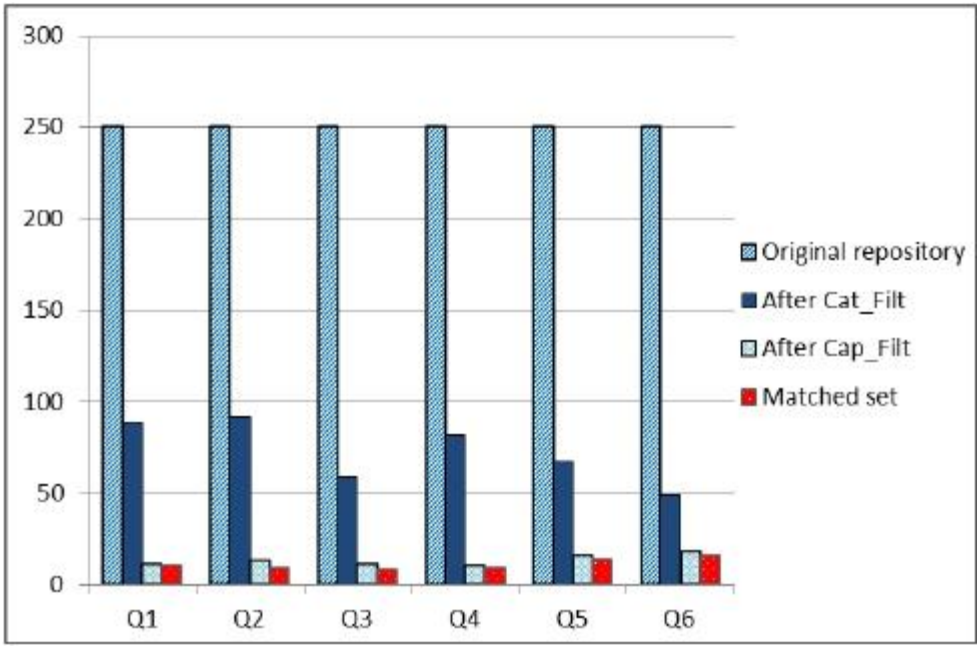


Figure 14. Effectiveness of the two pre-filtering stages in eliminating irrelevant Web services

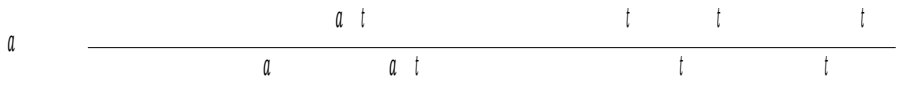
7.5.3 Definitions of Precision, Recall and Fallout

To analyse the accuracy of our pre-filtering stages, table 23 gives the precision, recall and fallout values of the combined pre-filtering stages for the same set of data obtained by running 6 requested goals against 250 WSs in the repository.

Precision is the percentage of the retrieved WSs that are actually relevant. In our context, "retrieved Web services" means the WSs that survived the two-stage elimination process, and a WS is "relevant" to a goal if the logical matchmaker says so. With these definitions, precision can be expressed as [4]:

$$\frac{a\ t \qquad\qquad t \qquad t \qquad\qquad t}{t}$$

Recall is the portion of the relevant WSs that are successfully retrieved. It can be expresed as:



Fallout is the percentage of the retrieved WSs that are non-relevant. In our context, a WS is “non-relevant” to a goal if the logical matchmaker says non-matched. With these definitions, fallout can be exprsed as:

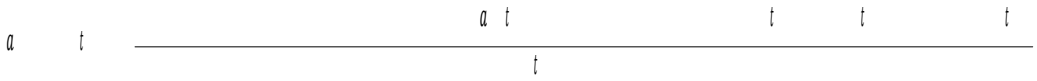


Table 23. Precision, recall and fallout of combined pre-filtering stages in each requested goal

Goal	Q1	Q2	Q3	Q4	Q5	Q6	Average
Precision	90.91%	69.23%	72.73%	90.00%	87. 5%	88.89%	83.21%
Recall	100%	100%	100%	100%	100%	100%	100%
Fallout	9.09%	30.77%	27.27%	10.00%	12.50%	11.11%	16.79%

As shown in table 23, average precision for all request queries is 83.21% which can be considered a good precision rate. It means that 83.21% of retrieved WSs are exactly matched with the requested goal and the other around 15% is irrelevant. However, the average recall of queries has the highest possible rate, 100%. With this 100% recall rate, all the relevant WSs in WS repository are retrieved through the proposed pre-filtering stages, an important feature that sets out filtering strategy apart from all the other proposals.

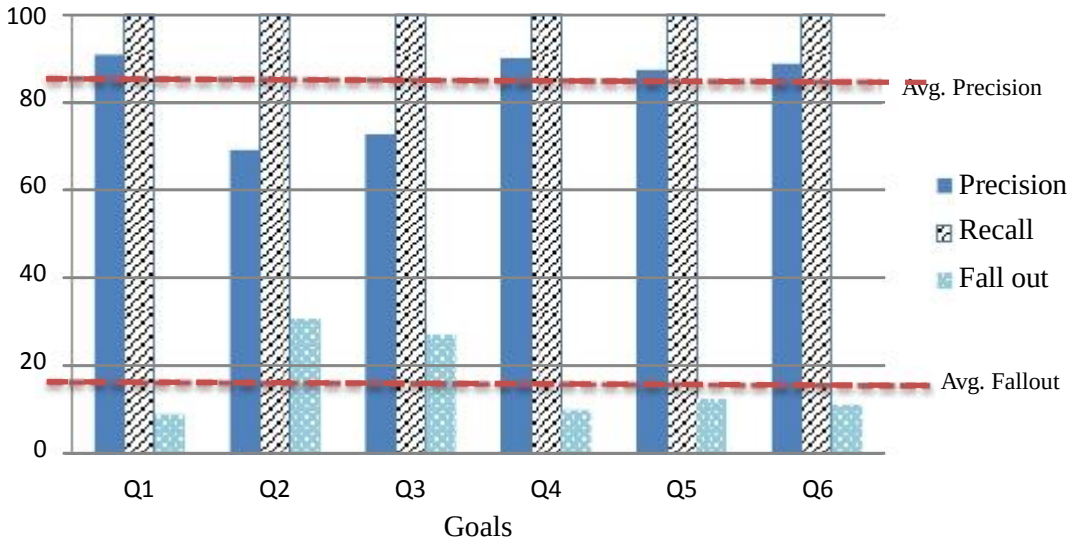


Figure 15. Precision, recall and fallout of each requested goal along with the average precision and fallout lines

Figure 15 graphically shows the precision, recall and fallout rate of each requested goal together with the average precision and fallout lines. The chart illustrates that precision rate of all requests except the second and third one (Q2, Q3) are higher than the average. Low precision rate of Q2 and Q3 indicate that there exist many WSs in the repository whose attributes and concepts are semantically similar to the concepts‟ name and attributes‟ name of requested goals, however, the value of WS attributes which were defined in ontologies do not match with the requested value of goals attributes. Such WSs fail in the actual logical matching procedure.

7.5.4 Discussion of the Results

Our prefiltering stages result in an impressive 100% recall rate. The reason for this top recall rate is that *all* relevant WSs are retrieved by Cat_Filt and Cap_Filt algorithms, which is another way of saying that in the pre-filtering stages, we only eliminate WSs that the matcher would definitely reject. The 100% recall rate of our framework implies that our method does not result in contain false negatives (FN) (i.e. WSs which are relevant, but are classified as irrelevant).

Our work, due to checking of the semantic and synonym equivalency of concepts and attributes of WSs and goals, as well as using Global_Cat_Ont ontology which contains both structural knowledge (i.e. it defines subclass and superclass relationships between concepts of three specified domains) and a dictionary of synonymous concepts, eliminates all of the irrelevant WSs through the filters stages. Therefore, the average precision and recall rate of our approach is higher than the others work in this filed.

In general, since our framework evaluation is based on our newly generated test collection, WSMO-FL V2, a comparison between the recall rate and precision rate of our work and the other available works in the literature would not be very informative. However, an average of 100% for recall, 83.21% for precision and 16.79% for fallout indicates a satisfactory accuracy of this work. It should be pointed out that this accuracy was observed in a more complex condition of goals and WSs due to pre and post-condition parameters that involve all logical connectives (i.e. and, or, not), whereas other studies mentioned in related works did not consider this much of complexities in their goals and WSs descriptions.

7.6 Related Work Regarding Approaches using Various Techniques to Improve Speed of Discovery Processes of WSs

Recently, although a wealth of insightful efforts have proposed different solutions to improve the semantic Web discovery process, we could not find any work that addresses the performance challenge of discovery process in a similar way to our work. In this section, we discuss proposals related to this field and analyse their relationship with our solution and their advantages as compared to our approach.

Table 24 compares our work with the related works based on several dimensions with respect to SWS discovery improvement. These dimensions are: *pre-processing*, *discovery methods*, *parameters*, *false negatives* and *frameworks*.

First three dimensions are further subdivided into sub-dimensions: *Pre-processing* is sub-divided into Non-Functional Properties (NFP) and Functional Properties (FP). *NFP* here stands for methods of adding some NFP elements to the WS and goal descriptions (e.g. categorization of each advertised / requested WS at design time). *FP* stands for methods to compare functional parameters of goal and WSs (i.e. IOPE).

One more level of subdivision used in pre-processing factor is: Taxonomy (TX) (i.e. relationship between two concepts/attributes is described by using a hierarchical diagram), Synonymity (SY) (i.e. syntactically two concepts/attributes are different but they have the same or identical meaning) and syntax (ST) (i.e. no synonymous or hierarchical relationships exist between two concepts/attributes and they are compared based on similarity of their string) similarity method measurements for each mentioned NFP and FP.

Discovery methods represent which kinds of service matchmakers are used in the approaches: Logic (LOG), Non-logic (NLOG) or Hybrid method (HY) which is combination of both logic and non-logic methods.

Parameters demonstrate degree of completeness of a research (whether it uses the major functional parameters of goal and WSs or not). Major functional parameters of

goal and WSs in OWL-S and WSMO models are Input (I), Output (O), Pre-condition (PRE) and Post-condition or Effect (POS/EFF).

We summarise the result of the comparative study of WS discovery approaches in table 24, where each row represents an approach, and the columns stand for main dimensions in WS discovery improvement. The symbol „„√“ used to denote that the specified approach supports the corresponding dimension, and „„-“ means that it does not.

Table 24. Comparison of this work with related works

App	Pre-processing					Discovery method			Parameters				FN	Frame work
	NFP		FP											
	TX	SY	TX	SY	ST	LOG	N-LOG	HY	I	O	PRE	POS		
[46]	-	-	-	-	-	-	-	√	√	√	-	-	-	OWL-S
[44]	-	-	-	-	-	-	-	√	√	√	-	-	-	WSMO
[70, 71]	-	-	-	-	-	√	-	-	√	√	√	√	-	WSMO
[68]	-	-	-	-	-	√	-	-	√	√	√	√	-	OWL-S
[2]	-	-	-	-	-	-	-	√	√	√	-	-	-	OWL-S
[40]	√	-	-	-	-	√	-	-	√	√	-	-	-	OWL-S
[56]	-	√	-	-	-	-	-	-	√	√	-	-	-	OWL-S
[26]	-	-	√	-	-	-	-	-	√	√	-	-	-	OWL-S
[25]	-	-	√	-	-	-	-	-	√	√	-	-	-	WSMO
[48]	-	-	-	-	√	√	-	-	√	√	√	√	-	WSMO
Our work	√	√	√	√	-	√	-	-	√	√	√	√	√	WSMO

Approach name (APP)	Logic (LOG)	Non-logic (NLOG)
Hybrid method (HY)	Functional properties (FP)	Non-functional properties (NFP)
Taxonomy (TX)	Output (O)	Pre-condition (PRE)
Synonymity (SY)	Input (I)	Post-condition /Effect (POS/EFF)
syntax (ST)	False Negatives (FN)	

In order to highlight the advantages of our work with respect to the prior researches, we classified the related works into two groups: approaches that optimize SWS discovery through a) improvement of matchmakers and b) application of pre-filtering mechanism before actual matchmakers. The former discusses the related works where the only focus is to improve the performance of their matchmaker engines by employing various methods. The latter tries to reduce the size of original repository and the filtered repository is used as input of actual matchmaker.

7.6.1 Approaches that Improve the Matchmaker Engine

Regarding the need to improve the discovery process and make it more scalable, some approaches attempt to improve the performance of the matchmaker engine without introducing any extra pre-processing stages.

Klusch et al. [46] implemented a hybrid matchmaker consisting of both approximated Information Retrieval (IR) matching, such as syntactic similarity technique, and OWL-DL logical reasoner to discover SWSs. Authors used four variants to calculate the text similarity of parameters, called *cosine*, *loss-of-information*, *extended Jacquard*, and *Jensen-Shannon*. In *OWLS-MX*, the logical reasoner only considers degree of semantic similarity between input and output parameters of OWL-S advertised/requested services and available concepts in the specified domain ontology. Later they developed their system to support WSMO services, called WSMO-*MX* [44]. Their comprehensive evaluations demonstrate that both approaches presented high precision in the S3 contest [43]. However, shortcomings of their solution are (i) they are time consuming because of high calculation costs related with both logic-based matching and text-based similarity matching, (ii) they retrieve WSs which are not related to the request.

The Klusch et al. approach can be improved if they utilize our pre-processing strategies on top of their actual matchmakers. For instance, by applying our pre-filtering stages before the hybrid matchmakers, especially on the logic-based matchmaker they can potentially decrease the size of the initial WS repository and consequently improve the overall performance of matchmaker.

Stollberg et al. in [70, 71] improved the matching process by implementing a caching mechanism that decreases the size of search space and reduces the matchmaker operations. The presented cache uses a *Semantic Discovery Caching* (SDC) graph that stores connections between client requests described as WSMO goal templates, and their relevant WSs. Thus, when a goal instance is received, first, the system compares the goal instance with cached templates with respect to semantic similarity, and if there is a match, merely the relevant WSs are stored in the SDC graph are used for subsequent discovery.

Authors of [70, 71] claim that they presented a standard approach where both advertised and requested functionalities are formally expressed in terms of pre-conditions and effects (post-condition). Also they used first-order logic as the specification language for formal description of these terms. Since our proposal also has been established in the spirit of WSMO framework and developed to work on goals and WSs capability which consist of inputs, outputs, pre and post-condition, proposed caching approach can be completed when our pre-filtering mechanisms are implemented before creating the caching graph. Thus, the number of relevant WSs which are stored in graph can be possibly decreased.

Authors of [68] introduced SPARQL as a language to describe the pre-conditions and post-conditions of OWL-S WSs as well as user requests. They implemented a matchmaker that works through agents called *SPARQLent* (SPARQL agent). In this approach, a complete discovery solution of their algorithm is discussed and shows how SPARQL queries are used to modify and query the agent‟s knowledge base. Finally, they evaluated their proposal against OWLS-MX via SME2 test tool[12].

Although the method offered in [68] is based on pre and post-conditions of WSs and goals, their evaluation is performed based on OWLS-TC V3, where presented WSs descriptions are without pre and post-conditions. Our pre-filtering stages could be also useful in helping SPARQL agent avoid the loading of all the available WSs on the repository and as a result cause to further improvements in their agent performance.

Amorim et al. [2] discuss a hybrid matchmaker called *OWL-S Discovery*. It is a combination of semantic filters based on input and output parameters of requested/advertised services and analysing each neighbour relationship in domain ontology. Authors employ five levels of semantic similarity between input and output parameters, namely exact, plug-in, subsume, fail and sibling. Also, in order to analyse each neighbour relationship in the concepts, they use a dictionary to classify the concepts. Based on this dictionary, concepts are either identical, synonymous or neither synonymous nor identical, as in our work. At the end they compare their work with Paolucci‟s approach [62] and the hybrid algorithm OWLS-MX through

[12] http://projects.semwebcentral.org/projects/sme2

OWLS-TC V3 test collection. Our proposal also can be applied to the top of *OWL-S Discovery* to further improve discovery processes. However our work uses a more expressive model to describe user requests and WSs descriptions as they contain pre and post conditions.

7.6.2 Approaches Using Pre-processing Mechanisms

These approaches make use of pre-processing mechanisms that help the optimization of automated WS discovery by narrowing down the set of existing WSs in the repository that will be considered by the service matchmaker. Pre-processing mechanisms are further are subdivided into two categories, 1) Pre-processing mechanisms based on categorization schemes of NFPs and 2) Pre-processing mechanisms based on semantic similarity of FPs.

Pre-filtering based on categorization schemes of NFPs: Most of the efforts related to pre-filtering techniques follow one of the two classification methods: they either exploit hierarchical categorization schemes of WSs on the basis of domain ontologies [40] or use dictionary of synonymous words [56]. The filtering process is separate from the matchmaker, so the results of this pre-filtering stage are then inspected through any actual process of service matchmaking. The majority of the mentioned proposals adapted OWLS-TC V3 test collection by adding one element to the request and WS NFPs that refer to service application domain.

Authors of [40, 56] implemented their categorization proposals on OWL-S WSs and verified it with respect to the OWLS-TC V3 data set. However, OWL-S service description in this test collection doesn‟t contain any information about service‟s application domain. Thus, in order to overcome the limitation of current OWL-S service profile elements both approaches added one NFP to the OWL-S service

profile. Although both used the same idea, their solution is different. In [40] the defined category concept of the service request is compared with the defined category concept of advertised WSs via hierarchical categorization scheme in global category ontology. A WS is eliminated if it has no category relationship with the request category. However, in [56] equivalency of requested and advertised WSs category concepts are computed via their relationship in the WordNet [55] dictionary of synonyms words. This approach is lacking in its own matchmaker (i.e. evaluation is done via OWLS-MX matchmaker).

Although the idea of our first filtering stage is similar to the mentioned proposals, it has the following novelties: (i) our proposed *Cat_filt* stage enrich the WSMO framework by adding an attribute called *hasCategory* to both goal and WS descriptions. (ii) In order to increase the accuracy and performance of our categorization schemes, this work takes into account semantic similarity relationship between goal category and WS category (i.e. if two categories mean the same thing or inherit the same class).

Pre-filtering based on semantic similarity of FPs: Authors of [25, 26] also used pre-processing strategies before the actual matching process. Their pre-filtering is based on only FPs of WSs. They present two different SPARQL queries to facilitate the search process on a SWS registry. They automatically create SPARQL queries (called Q_{all}, Q_{some}) by analysing the user request, and by using these two filtering queries they are able to perform two levels of filtering on the initial WS repository. Based on these two queries, only WSs containing all (in the case of Q_{all}) or some (in the case of Q_{some}) concepts referred by a user request are returned.

Our second filtering stage (*Cap_Filt*) is similar to the method proposed in [29]. Four major differences between our work and their works are:

(i) since in our pre-filtering stage service descriptions consist of all information about inputs, outputs, pre and post- conditions, we can obtain more accurate results than their strategies,

(ii) our algorithm not only considers the hierarchical relationship of concepts and attributes but also takes into account the similarity of requested/advertised WS concepts and attributes based on their synonyms,

(iii) we employ an initial filtering phase based upon a categorization scheme, which could actually improve their performance as well if they used it before Q_{all} or Q_{some} algorithm ,

(iv) Their approach consists of only a pre-processing stage to filter the preliminary WS repository and they did not implement any service matchmaking, so they cannot be evaluated on their own.

Among all the mentioned approaches, [48] is the closest to our work. The INFRAWEBS project implements a discovery framework which consists of two-components, pre-filtering and discovery. In the pre-filtering stage it uses traditional Information Retrieval techniques, and a logic-based matching implemented in Prolog is utilized as a service matchmaker.

Although the INFRAWEBS project has similarities with our work, some differences do stand out. Our pre-filtering stage considers semantic equivalency of both NFP and FP of the requested/advertised services, analysing objects, attributes and concepts. Our discovery engine works with much richer descriptions of WSs and requests, encoded in frame logic. Our implementation uses the FLORA-2 language and

execution environment, a much more powerful alternative to plain Prolog. It is conceivable that a combination of our approach and theirs can yield a discovery framework that is more effective at eliminating useless WSs than either approach alone.

Chapter 8

CONCLUSION AND FUTURE WORK

This dissertation presents two new methods and one newly generated test collection. These two methods are: i) a new evaluation framework for comparison and appraisal of SWS discovery and composition approaches, and ii) a new logical framework to improve peformance of automated SWS through pre-filtering strategies. The second method is the major contribution of this thesis.

In the first method, we presented a novel framework for the appraisal and comparison of automatic SWS discovery and composition approaches. Our method comprises of two parts: a rubric table, and a table of features appropriate for evaluating SWS discovery and composition processes. We called our method RFSWS, which stands for Rubric and Feature-based evaluation framework for SWS composition approaches.

Our usage of analytic rubric tables in the RFSWS framework is the first of its kind in the evaluation of SWS discovery and composition approaches. Aspects of SWS discovery and composition approaches that could not be assessed meaningfully using rubrics have been delegated to the feature-based evaluation scheme. When together with the feature-based evaluation scheme, the rubric we generated gives a reasonably complete picture of the capabilities, deficiencies, strong and weak points of a SWS discovery and composition approaches under review.

In the second method, we illustrated that the overall performance and accuracy of SWS discovery frameworks can be improved significantly through the introduction of pre-filtering stages that eliminate most of the irrelevant WSs from consideration at the computationally expensive matching stage. Specifically, in this thesis, we proposed *Category_based* and *Capability_based* pre-filtering mechanisms for narrowing down the number of WS descriptions that need to be considered in the matching phase to determine their relevance to the current goal.

We evaluated the effectiveness of our proposal in a novel test collection, WSMO-FL, which consists of 250 WS specifications of varying complexities and 6 goals. Our filtering stages stand out due to their 100% recall rate and 83.21% precision rate that are a consequence of their design, their ability to deal with complex specifications of goals and WSs written in an enhanced version of WSMO., as well as a reasonably high precision rate, as demonstrated experimentally, which is bound to increase considerably in the presence of a large number of categories and goals/WSs that make use of those categories. Our results also indicate that when the pre-filtering stages are employed in the system, as expected, the search space is considerably reduced, and consequently response time of the system is improved dramatically.

Our contributions in this thesis can be summed up in the following:

i) Unlike all the previous works in evaluation of SWS discovery and composition approaches, we proposed a novel idea in the context of comparison and evaluation of WSC approaches based on rubrics.

ii) Our RFSWS framework not only can be used as is or enhanced by other future researchers for the evaluation of SWSC approaches, but also it can

be adapted by researchers working on other subjects to evaluate methodologies and approaches relevant to their area of investigation.

iii) Unlike the majority of SWS discovery approaches which are only performed on input and output concepts, our SWS discovery framework deals with concepts and attributes of WS and goal pre and post conditions.

iv) Our pre-filtering stages are generic, so that they can be applied (after necessary adaptations) to improve the performance of other available service matchmakers.

v) 100% recall rate of our framework implies that our method does not result in contain false negatives (FN) (i.e. WSs which are relevant, but are classified as irrelevant): ALL relevant WSs are retrieved through the pre-filtering algorithms.

vi) Due to incomplete service descriptions in existing test collections such as OWL-S (i.e. WSs are partially described only based on input/output concepts), for the first time we created a new test collection named WSMO-FL, which contains fully defined WSs and goals capabilities (i.e. WSs and goals are described based on pre and post-conditions).

vii) To the best of our knowledge WSMO-FL is the first larger test collection which is established based on the WSMO conceptual model. It uses Frame-logic (F-logic) as a fully adequate expression language for specifying pre and post-conditions which is missing in currently available test collections.

For future work, we are planning to improve our scheme in the following ways:

i) Extending our new WSMO-FL test collection to (a) have a much larger number of Web services and goals, as well as categories, (b) increasing complexity of Web service and goal pre- and postconditions, and (c) expanding the dictionary of synonymous words in the existing domain ontologies.

ii) Implementing parallel version of our filtering strategy to get better scalability

REFERENCES

[1] Akkiraju, R., Farrell, J., Miller, J. A., Nagarajan, M., Sheth, A. P., & Verma, K. (2005). WSDL-S: Web service semantics. [Accessed on 2015 Aug 12]. Available from: www.w3.org/Submission/WSDL-S/.

[2] Amorim, R., Claro, D. B., Lopes, D., Albers, P., & Andrade, A. (2011). Improving Web service discovery by a functional and structural approach. In*Web Services (ICWS), 2011 IEEE International Conference*, P. 411-418.

[3] Angele, J. (2014). OntoBroker: Mature and approved semantic middleware, *Semantic Web 5*, No. 3, P. 221-235.

[4] Baeza-Yates, R., & Ribeiro-Neto, B. (1999). Modern information retrieval, Vol. 463. *New York: ACM press*.

[5] Bartalos, P. (2011). Effective Automatic Dynamic Semantic Web Service Composition. *Inf. Sci. and Technol. Bulletin ACM Slovakia, 3*(1), P. 61-72.

[6] Bartalos, P., & Bieliková, M. (2012). Automatic Dynamic Web Service Composition: A survey and problem formalization. *Computing and Informatics*, *30*(4), P. 793-827.

[7] Baryannis, G., & Plexousakis, D. (2010). Automated Web Service Composition: State-of-the-Art and Research Challenges. *ICS-FORTH, Tech. Rep, 409*.

[8] Belqasmi, F., Singh, J., Melhem, S. Y. B., & Glitho, R. H. (2012). SOAP-based vs. RESTful web services: A case study for multimedia conferencing. *IEEE Internet Computing*, (4), P. 54-63.

[9] Berge, C. (1989). Hypergraphs: Combinations of Finite Sets. *NorthHolland, Amsterdam, the Netherlands*.

[10] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). Semantic Web. *Scientific American*, *284*(5), P. 28-37.

[11] Bonner, A. J., & Kifer, M. (1994). Overview of transaction logic. *Theoretical Computer Science*, *133*(2), P. 205-265.

[12] Bonner, A. J., & Kifer, M. (1998). A logic for programming database transactions. In *Logics for databases and information systems* , P. 117-166. Springer US.

[13] Cabral, L., Li, N., & Kopecký, J. (2012). Building the WSMO-Lite Test Collection on the SEALS Platform.

[14] Çelik, D., & Elçi, A. (2013). A broker-based semantic agent for discovering Semantic Web services through process similarity matching and equivalence considering quality of service. Science China Information Sciences, 56(1), P. 1-24.

[15] Chen, W., Kifer, M., & Warren, D. S. (1993). HiLog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, *15*(3), P.187-230.

[16] Chinnici, R., Moreau, J. J., Ryman, A., & Weerawarana, S. (2007). Web Services Description Language (WSDL) version 2.0. *W3C recommendation*, Vol. 26.

[17] Cong, Z., Fernandez, A., Billhardt, H., & Lujak, M. (2014). Service discovery acceleration with hierarchical clustering. *Information Systems Frontiers*, 17(4), P. 1-10.

[18] Da Silva, E. G., Pires, L. F., & Van Sinderen, M. (2011). Towards runtime discovery, selection and composition of semantic services. *Computer Communications*, *34*(2), P.159-168.

[19] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), P. 107-113.

[20] Deng, S., Huang, L., Tan, W., & Wu, Z. (2014). Top-K Automatic Service Composition: A Parallel Method for Large-Scale Service Sets. *Automation Science and Engineering, IEEE Transactions on*, *11*(3), P. 891-905.

[21] Eid, M., Alamri, A., & El Saddik, A. (2008). A reference model for dynamic web service composition systems. *International Journal of Web and Grid Services*, *4*(2), P. 149-168.

[22] Elçi, A., & Rahnama, B. (2007). Human-robot Interactive Communication Using Semantic Web Tech. in Design and Implementation of Collaboratively Working Robots. *The 16th IEEE International Symposium in Robot and Human interactive Communication*, P. 273-278.

[23] Elçi, A., & Rahnama, B. (2009). Semantic robotics: cooperative labyrinth discovery robots for intelligent environments. *Complex Systems in Knowledge-based Environments: Theory, Models and Applications*, P. 163-198.

[24] Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., & Domingue, J. (2006). Enabling semantic web services: the web service modeling ontology, P. 63- 81. Springer Science & Business Media.

[25] Garcia, J. M., Ruiz, D., & Ruiz-Cortés, A. (2011). A lightweight prototype implementation of SPARQL filters for WSMO-based discovery. Tech. Rep. ISA-11-TR-01, *Applied Software Engineering Research Group-University of Seville*.

[26] Garcia, J. M., Ruiz, D., & Ruiz-Cortés, A. (2012). Improving semantic web services discovery using SPARQL-based repository filtering. *Web Semantics: Science, Services and Agents on the World Wide Web, 17*, P. 12-24.

[27] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Riedman, M., & Weld, D. (1998). PDDL-the planning domain definition language. *Yale center for computational vision and control*, Tech Report No.: CVC TR-98-003/DCS TR-1165.

[28] Ghayekhloo, S., & Bayram, Z. (2015). A Novel Rubric and Feature-Based Appraisal and Comparison Framework for the Evaluation of Semantic Web Services Composition Approaches. *Indian Journal of Science and Technology*, In press.

[29] Ghayekhloo, S., & Bayram, Z. (2015). Prefiltering Strategy to Improve Performance of Semantic Web Service Discovery. *Scientific Programming*, Vol 2015.

[30] Goncalves da Silva, E. M., Ferreira Pires, L., & van Sinderen, M. J. (2009). Supporting dynamic service composition at runtime based on end-user requirements. *CEUR Workshop Proceedings*.

[31] Gruber, T. R. (1993). A translation approach to portable ontology specification. *Knowledge acquisition*, *5*(2), P. 199-220.

[32] Hakimpour, F., & Cong, U. S. (2006). Semantic Descriptions of Web Services. *Advances in Electronic Business*, 2(2), P. 31.

[33] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2007). VLEPPO: A visual language for problem representation. *PlanSIG*, 7, P. 60-66.

[34] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2010). A visual programming system for automated problem solving. *Expert Systems with Applications*, *37*(6), P. 4611-4625.

[35] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2013). The PORSCE II framework: Using AI planning for automated semantic web service composition. *The Knowledge Engineering Review*, 28(02),P.137-156.

[36] Hatzi, O., Vrakas, D., Nikolaidou, M., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2012). An integrated approach to automated semantic web service composition through planning. *IEEE Transactions on Services Computing*, *5*(3), P. 319-332.

[37] Hong, J., Suh, E. H., Kim, J., & Kim, S. (2009). Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, *36*(4), P. 7448-7457.

[38] Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., & Yiu, A. (2007). Web services business process execution language version 2.0. *OASIS standard*, *11*(120).

[39] Khalid, M., Fries, B., Vasileski, M., Kapahnke, P., & Klusch, M. (2010). SAWSDL-TC Service Retrieval Test Collection, User Manual Version 3.0, *Saarbrücken, Germany*.

[40] Khdour, T. (2011). Towards Semantically Filtering Web Services Repository. In *Digital Information and Communication Technology and Its Applications*, P. 322-336. Springer Berlin Heidelberg.

[41] Kifer, M., Lausen, G., & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, *42*(4), P. 741-843.

[42] Kifer, M., Yang, G., Wan, H., Zhao, C., Kuznetsova, P., & Liang, S. (2014). FLORA-2: User‟s Manual Version 1.0. *Department of Computer Science, Stony Brook University, Stony Brook*.

[43] Klusch, M. (2012). The S3 Contest: Performance Evaluation of semantic web services. *Semantic Web Services: Advancement through Evaluation*.

[44] Klusch, M., & Kaufer, F. (2009). WSMO-MX: A hybrid Semantic Web service matchmaker. *Web Intelligence and Agent Systems*, *7*(1), P. 23-42.

[45] Klusch, M., Alam Khalid, M., Kapahnke, P., Fries, B., & Vasiles Saarbrücken, M. (2010). OWLS-TC: OWL-S Service Retrieval Test Collection, User Manual Version 4.0. *Saarbrücken, Germany*.

[46] Klusch, M., Fries, B., & Sycara, K. (2009). OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Web Semantics: Science, Services and Agents on the World Wide Web*, *7*(2), P. 121-133.

[47] Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. (2007). SAWSDL: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE*, *11*(6), P. 60-67.

[48] Kovács, L., Micsik, A., & Pallinger, P. (2006). Two-phase Semantic Web Service Discovery Method for Finding Intersection Matches using Logic Programming. *In Workshop on Semantics for Web Services*.

[49] Kritikos, K., & Plexousakis, D. (2008). Enhancing the Web service description and discovery processes with QoS. *Managing Web Service Quality: Measuring Outcomes and Effectiveness: Measuring Outcomes and Effectiveness*. IGI Global.

[50] Küster, U., König-Ries, B., Klein, M., & Stern, M. (2007). DIANE: A matchmaking-centered framework for automated service discovery, composition, binding, and invocation on the web. *International Journal of Electronic Commerce*, *12*(2), P. 41-68.

[51] Lécué, F., & Léger, A. (2006). A formal model for semantic web service composition. *The Semantic Web-ISWC 2006*, P. 385-398. Springer Berlin Heidelberg.

[52] Lee, K., Jeon, J., Lee , W., Jeong, S. H., & Park, S. W. (2003). Qos for web services: Requirements and possible approaches. *W3C working group note*, *25*, P. 1-9.

[53] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., & Sycara, K. (2004). OWL-S: Semantic markup for web services. *W3C member submission*, *22*, P. 2007-2004.

[54] McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE intelligent systems*, (2), P. 46-53.

[55] Miller, G., & Fellbaum, C. (1998). Wordnet: An electronic lexical database. *Blackwell Publishing Ltd*, MA.

[56] Mohebbi, K., Ibrahim, S., & Zamani, M. (2013). A Pre-matching Filter to Improve the Query Response Time of Semantic Web Service Discovery.*Journal of Next Generation Information Technology*, *4*(6).

[57] Moskal, B. M. (2009). Scoring rubrics: What, when and how? Practical assessment, research and evaluation. [Accessed on 2015 Aug 20]. Available from: pareonline.net/getvn.asp?v=7&n=3.

[58] Ngan, L. D., & Kanagasabai, R. (2013). Semantic Web service discovery: state-of-the-art and research challenges. *Personal and ubiquitous computing*, *17*(8), P. 1741-1752.

[59] Ni, Y., & Fan, Y. (2010). Model transformation and formal verification for Semantic Web Services composition. *Advances in Engineering Software*, *41*(6), P. 879-885.

[60] Nitko, A. J. (2001). *Educational assessment of students*. Prentice-Hall, Inc., Des Moines, IA 50336-1071.

[61] Online Curve Fitting [Accessed on 2015 Dec 24]. Available from: http://www.MyCurveFit.com

[62] Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. (2002). Semantic matching of web services capabilities. In *The Semantic Web—ISWC 2002*, P. 333-347. Springer Berlin Heidelberg.

[63] Papazoglou, M. P., & Van Den Heuvel, W. J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3), P. 389-415.

[64] Rao, J., & Su, X. (2005). A Survey of Automated Web Service Composition methods. In *Semantic Web Services and Web Process Composition* , P. 43-54. Springer Berlin Heidelberg.

[65] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., & Fensel, D. (2005). Web Service Modelling Ontology. *Applied ontology*, *1*(1), P. 77-106.

[66] Rong, W., & Liu, K. (2010). A survey of context aware web service discovery: from user's perspective. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium*, P. 15-22.

[67] RubiStar (2008) [Accessed on 2015 Aug 20]. Available from: http://rubistar.4teachers.org/.

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

[69] Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade‟s overview. *Information Sciences*, *280*, P. 218-238.

[70] Stollberg, M., Hepp, M., & Hoffmann, J. (2007). A caching mechanism for semantic web service discovery, P. 480-493, Springer Berlin Heidelberg.

[71] Stollberg, M., Hoffmann, J., & Fensel, D. (2011). A caching technique for optimizing automated service discovery. *International Journal of Semantic Computing*, *5*(01), P. 1-31.

[72] Tang, X., Jiang, C., & Zhou, M. (2011). Automatic Web service composition based on Horn clauses and Petri nets. *Expert Systems with Applications*, *38*(10), P. 13024-13031.

[73] Vardhan, A. V., Basha, M. S., & Dhavachelvan, P. (2011). An Overview of Web Services Composition Approaches. *International Journal of Computer Applications*, *29*(8), P. 10-15.

[74] Wan, H., Grosof, B., Kifer, M., Fodor, P., & Liang, S. (2009). Logic programming with defaults and argumentation theories. In *Logic Programming*, P. 432-448.

[75] Wolf, K., & Stevens, E. (2007). The role of rubrics in advancing and assessing student learning. *The Journal of Effective Teaching*, *7*(1), P. 3-14.

[76] Fensel, D., Fischer, F., Kopecký, J., Krummenacher, R., Lambert, D., & Vitvar, T. (2010). *WSMO-Lite: Lightweight semantic descriptions for services on the web*. W3C Member Submission, 23.

APPENDIX

Appendix A: Source code of *%Conv_DNF* predicate in FLORA-2

```
 //------------------------%NNF----------------------
// negation normal form

%nnf(?X, ?R):- %is_pred(?X,?_PredName,?_Parameters),
                !,
                ?R=?X.

%nnf(?X, ?R):- %is_neg_pred(?X),
                !,
                ?R=?X.

// base case 1
%nnf(?X, ?R):-  %is_simple(?X),
                !,
                ?R=?X.

// base case 2
%nnf((\+ ?X), ?R):- %is_simple((\+ ?X)),
                     !,
                    ?R=(\+ ?X).

// double negation
%nnf((\+ ?X), ?R):- ?X=(\+ ?Y),
                      !,
                      %nnf(?Y,?Y2),
                      ?R=?Y2.

// special case for double negation of conjunction
%nnf(?In, ?R):-  ?In = (\+ (\+ (?X,?Y))),
                   !,
                   %nnf((?X,?Y),?R).

 //negation of conjunction
%nnf(?In, ?R):-    ?In =.. ?L,
                  (( ?L = [hilog(\+), ?X,?Y]) ;
                   ( ?L = [hilog(\+), (?X,?Y)] ) ;
                   ( ?L = [flapply(hilog,\+), ?X,?Y]) ;
                   ( ?L = [flapply(hilog,\+), (?X,?Y)]) ;
                   ( ?L = [negation(\+), ?X,?Y]) ;
                   ( ?L = [negation(\+), (?X,?Y)]) ;
                   ( ?L = [flapply(negation,\+), ?X,?Y] ) ;
                   ( ?L = [flapply(negation,\+), (?X,?Y)] )
                   ),
                    !,
                  %nnf( (\+ ?X),?X2),
                  %nnf( \+ ?Y,?Y2),
                  ?R=(?X2;?Y2).
```

```
%nnf((\+ (?X;?Y)), ?R):-
                \true,
                !,
                %nnf( (\+ ?X),?X2),
                %nnf( (\+ ?Y),?Y2),
                ?R=(?X2,?Y2).

%nnf((?X,?Y), ?R):- \true,
                 !,
                %nnf(?X,?X2),
                %nnf(?Y,?Y2),
                ?R=(?X2,?Y2).

%nnf((?X;?Y), ?R):-  \true,
                 !,
                %nnf(?X,?X2),
                %nnf(?Y,?Y2),
                ?R=(?X2;?Y2).

//------------------------%in_DNF-------------------------------

%in_dnf((?X;?Y)):-  \true,
                !,
                %in_dnf(?X),
                %in_dnf(?Y).


%in_dnf(?X):- %all_conj(?X).

%all_conj((?X,?Y)):- \true,
                        !,
                        %all_conj(?X),
                        %all_conj(?Y).


%all_conj(?X):- %is_simple(?X).
%all_conj(?X):-  %is_pred(?X,?_PredName,?_Parameters),!.
%all_conj(?X):-  %is_neg_pred(?X),!.


%is_simple(?X):- isatomic{?X},!.
%is_simple(?X):- ?X ~ ${?_A[?_B->?_C]@?_Module}, !.
%is_simple(?X):- ?X ~ ${?_A:?_C@?_Module}, !.

%is_simple((\+ ?X)):- isatomic{?X},!.
%is_simple((\+ ?X)):- ?X ~ ${?_A[?_B->?_C]@?_M}, !.
%is_simple((\+ ?X)):- ?X ~ ${?_A:?_Concept}, !.
```

```
//---------------------------------%DNF--------------------

%dnf(?X,?R):- %in_dnf(?X),
                !,
                ?R ~ ?X.


%dnf(?X,?R):-?X ~ (?A ; ?B),
                !,
                %dnf(?A,?A2),
                %dnf(?B,?B2),
                ?R ~ (?A2;?B2).


%dnf(?X,?R):-  ?X ~ (?A , ?B),
                %dnf(?A,?A2),
                 %dnf(?B,?B2),
                 (
                (?A2 ~ (?P1@?M ; ?Q1@?M),
                ?B2 ~ (?H1@?M  ; ?S1@?M),
                !,
                %dnf((?P1 , ?H1),?Res1@?M),
                %dnf((?P1 , ?S1),?Res2@?M),
                %dnf((?Q1 , ?H1),?Res3@?M),
                %dnf((?Q1, ?S1),?Res4@?M),
                ?R ~ (?Res1;?Res2;?Res3;?Res4)
                )
               ;
                (
                ?A2 ~ (?P1@?M ; ?Q1@?M),
                // ?B2 ~ conjunction of literals
                !,
                %dnf((?P1, ?B2@?M),?Res1@?M),
                %dnf((?Q1, ?B2@?M),?Res2@?M),
                ?R ~ (?Res1;?Res2)
                )
                ;
                 (
                // ?A2 conjunction of literals
                ?B2 ~ (?H1@?M; ?S1@?M) ,
                !,
                %dnf((?A2@?M,?H1),?Res1@?M),
                %dnf((?A2@?M,?S1),?Res2@?M),
                ?R ~ (?Res1;?Res2)
                )
               ;
                (
                // both ?A2 and ?B2 conjunction of literals
                ?R ~ (?A2, ?B2)
                )).
```

```
//-------------------------%Remove_NOT------------------------------

// %remove_not(Input,Result)

// object is positive
%remove_not(?X,?R):- ?X =..?L,
                      ?L = [?A|?_B],
                     ((?A = flogic('->',?_M));(?A = flogic(:,?_M))),
                     !,
                     ?R=?X.

 //object is negative
%remove_not(?X,?R):-   ?X =..?L,
                     ?L = [?A,?_B],
                     ?A = negation(\+),
                     !,
                     ?R = ${nothing:Nothing@?_Module}.


%remove_not(?X,?R):-
                    (%is_pred(?X,?_PredName,?_Parameters)
                     ;
                     %is_neg_pred(?X)),
                     !,
                     ?R = ${nothing:Nothing@?_Module}.

// contain more object (and)
%remove_not(?X,?R):- ?X =..?L,
                      ?L = [?H|?T],
                     ?H = logic(and),
                     !,
                     ?T=[?C,?D],
                     %remove_not(?C,?C1),
                     %remove_not(?D,?D1),
                     ?R= (?C1,?D1).


// contain more object (or)
%remove_not(?X,?R):- ?X =..?L,
                     ?L = [?H|?T],
                     ?H = logic(or),
                     !,
                     ?T=[?C,?D],
                     %remove_not(?C,?C1),
                     %remove_not(?D,?D1),
                     ?R= (?C1;?D1).
```

```
//-----------------------------------%Convert-DNF------------------

%Conv_DNF(?In,?Result):-  %nnf(?In,?Temp),
                          %remove_not(?Temp,?RNot),
                          %dnf(?RNot,?Result).
```

Appendix B: Source code of *%Check_Att_Cnp* predicate in FLORA-2

```
%Check_Att_Cnp (?WsName, ?_X,?WsOrGoal):-
                    alreadySelected (?WsName,?WsOrGoal)@FilteredWsModule,
                    !.

%Check_Att_Cnp(?_WsName,?DisjOfConj,?_WsOrGoal):-
                   ?DisjOfConj=[],
                   !.

%Check_Att_Cnp(?WsName,?Conjunction,?WsOrGoal):-
                   ?Conjunction=[?FirstConjunction|?_Rest],
                   %processConjunction(?WsName,?FirstConjunction,?WsOrGoal).

%Check_Att_Cnp(?WsName,?Conjunction,?WsOrGoal):-
                   ?Conjunction=[?_FirstConjunction|?Rest],
                   Check_Att_Cnp(?WsName,?Rest,?WsOrGoal).

%processConjunction(?WsName,?Conj,?WsOrGoal):-
                  ?Conj=[],
                   insert{alreadySelected(?WsName,?WsOrGoal)}@FilteredWsModule,
                   !.

%processConjunction(?WsName,?Conj,?WsOrGoal):-
                  ?Conj=[?FirstPair|?RestPairs],
                  %processPair(?WsName,?FirstPair,?WsOrGoal),
                  %processConjunction(?WsName,?RestPairs,?WsOrGoal).

%processPair(?WsName,?Apair,?_WsOrGoal):-
                 ?Apair =(?Concept,?_AttList),
                 ?WsName[otherSource->?Concept]@?_WsModule,
                 !.

%processPair(?_WsName,?Apair,?_WsOrGoal):-
                   ?Apair =(Nothing,?_AttList),
                   !.

%processPair(?_WsName,?Apair,?WsOrGoal):-
                   ?Apair =(?Concept,?AttList),
                   %allAttributesPresent(?Concept,?AttList,?WsOrGoal).

%allAttributesPresent(?_Concept,?AttList,?_WsOrGoal):-
                  ?AttList = [],
                   !.
```

```
%allAttributesPresent(?Concept1,?AttList,?WsOrGoal):-
                    ?AttList = [?AnAttribute1|?RestAttributes],
                    (
                     (?AnAttribute1 :=: ?AnAttribute2) ;
                     ( ?MediatorAtt::Mediator,
                     ?AnAttribute1::?MediatorAtt,
                     ?AnAttribute2::?MediatorAtt)
                      ),
                     ?SomeObj[?AnAttribute2->?_SomeValue]@GoalWsAttModule,
                     (
                     (?Concept1 :=: ?Concept2) ;
                     (?MediatorConcept::Mediator,
                     ?Concept1::?MediatorConcept,
                     ?Concept2::?MediatorConcept)
                     ),
                     ?SomeObj:?Concept2@GoalWsAttModule,
                     %allAttributesPresent(?Concept1,?RestAttributes,?WsOrGoal).
```

Appendix C: Sample of Web Service in WSMO-FL Test Collection

```
sws3:Service.
sws3[
        hasNonFunctionalProperty -> someNonFunctional,
        hasCategory->Transportation,
        importsOntology ->{ 'C:\Matching\Areas/FlightInfo_Simple_ont.flr',
                            'C:\Matching\Areas/geographical_ont.flr',

        'C:\Matching\Areas/ShipInfo_Simple_ont.flr'},
        usesMediator -> someMediator,
        hasCapability -> ${ ws3_c [
        hasPrecondition ->
                (($${?ReqShip[
                        startCity->?FromCity,
                        toCity->?ToCity
                                ]:RequestShipTicket
                },
                ${?SomeShip[
                        fromHarbor ->?FromHarbor,
                        toHarbor ->?ToHarbor,
                        cost->?Cost
                                ]:Ship
                }
                );
                {ReqFlight[
                        fromCity->?FromCity,
                        destinationCity ->?ToCity
                                ]:RequestAirplainTicket
                },
                ${?SomeFlight[
                        fromAirport->?FromAirport[inCity->?FromCity],
                        toAirport->?ToAirport[inCity->?ToCity],
                        cost->?Cost
                                ]:Flight
                  }
                )),
        hasPostcondition ->
                 ${response[
                        originateAirport->?FromAirport,
                        toAirport->?ToAirport,
                        fromHarbor->?FromHarbor,
                        toHarbor->?ToHarbor,
                        cost->?Cost
                                ]:Response
                        }
                                        ]
                        },
        hasInterface -> someInterface,
        otherSource -> {Flight,Ship}  ].
```

Appendix D: Sample of Goal in WSMO-FL Test Collection

```
g1: Goal.
g1[
        hasNonFunctionalProperty -> someNonFunctional,
        hasCategory->AirTransportation,
        importsOntology -> somOnt,
        usesMediator -> someMediator,
        requestsCapability -> ${goal1_c1[
        hasPrecondition ->
                                ${reqFlight[
                                        originateCity->berlin,
                                        terminalCity->Istanbul
                                                ]:RequestFlightTicket
                                  },
        hasPostcondition ->
                                 (${?BookTicket[
                                        fromAirport->?FromAirport,
                                        toAirport->?ToAirport,
                                        cost->?Cost
                                                ]:Response
                                    },
                                \+ is_equal(?ToAirport ,sabiha_Gokcen),
                                less(?Cost, 500)
                                )
                                                        ]
                                        },
        requestsInterface -> someInterface
].
```

Appendix E: Source code of MainMatcher

```
//---------------------------------Pre-filtering_Part----------------------//


%FilterMain:- ?_Inserted = setof{ ?Ins |
                ?Goal[hasCategory->?GoalCat]@?_GoalModule,
               ?WS[hasCategory->?WsCat]@?_WsModule,
                ((?WsCat :=: ?GoalCat) ; (?WsCat::?GoalCat) ; (?GoalCat::?WsCat)),

                %Filter_Cap(?Goal, ?WS),
                 alreadySelected(?WS,GOAL)@FilteredWsModule,
                alreadySelected(?WS,WEBSERVICE)@FilteredWsModule,

                insert{related(?Goal,?WS)}@RelatedGoalWsModule,
                ?Ins=related(?Goal,?WS)
                                        }.


 //---------------------------------Matching_Part--------------------------//

%run_a(?Result):-
         %FilterMain,
        %run_c(?Result).

%run_c(?Result):- ?Result = setof{ ?MatchResult |
                                related(?Goal,?WS)@RelatedGoalWsModule,
                                  %match_c(?Goal,?WS,?MatchResult)
                                          }.

//--------------------------------Match-------------------------------------------------//
%match_c(?Goal,?WS,?Res):-
                WebService(?WS, ?_WsPath,?WsModule),
                Goal(?Goal, ?_GoalPath,?GoalModule),
                deleteall{?_A[?_B->?_C]@m1},

                %module_copy4(?WsModule, m1),
                %module_copy4(?GoalModule, m1),
                ?_temp = setof{ ?WsOntology |
                 (?WS[importsOntology -> ?WsOntology])@m1,

                Ontology(?_Ont,?WsOntology,?OntModule),
                %module_copy4(?OntModule, m1),
                t_insert{ontologyLoaded(?WsOntology)@m1 }},
        \if (
                ?Goal[requestsCapability->?GCap]@m1,
                ?GCap ~ ${?_GCapability[
                hasPrecondition->?_GoalPre,hasPostcondition->
                ?_GoalPost,optimization-> ?_Opt]}@m1,!
```

```
                )
        \then (
                %optimizing_main(?Goal,?WS,?optValue),!,
                ?Res = opt_match(?Goal,?WS,?optValue )
                )

        \else (
                %non_optimizing_main(?Goal,?WS),!,
                ?Res=plain_match(?Goal,?WS)
                )
        ,!.
%match_c(?Goal,?WS,?Res):-
                \true,!,
                ?Res=non_match(?Goal, ?WS).


//-----------------------------------Non_optimizing_main-------------------------//

%non_optimizing_main(?GoalName,?WsName):-
                %match(?GoalName,?WsName, ?_Cost),
                !.
//---------------------------------Optimizing_main-------------------------------//

%optimizing_main(?GoalName,?WsName,?Min_Val):-
                ?Min_Val = min{ ?Cost | %match(?GoalName,?WsName,?Cost)},
                !.

//-----------------------------------Ranking---------------------------------------//
%ranking(?Result, ?Ranked):-
                 ?Ranked = setof{?N|
                 %Goals(?AllGoals),
                 %member(?AGoal, ?AllGoals),
                 ?WsList = setof{ ?WsValuePair([asc(2)]) | member(?E,?Result),
                           ?E=opt_match(?AGoal, ?AWs, ?Value),
                             ?WsValuePair = [?AWs, ?Value]
                                      },
                ?N=(?AGoal,?WsList)
                                      }.
```