PRE-FILTERING STRATEGY TO IMPROVE PERFORMANCE OF SEMANTIC WEB SERVICE DISCOVERY

Samira Ghayekhloo[1]* and Zeki Bayram[2]

[1]Department of Computer Engineering, Eastern Mediterranean University, Famagusta, Northern Cyprus
samira.ghayekhloo@emu.edu.tr
[2]Department of Computer Engineering, Eastern Mediterranean University, Famagusta, Northern Cyprus
zeki.bayram@emu.edu.tr

ABSTRACT

Discovery of semantic Web services is a heavyweight task when the number of Web services or the complexity of ontologies increases. In this paper, we present a new logical discovery framework based on semantic description of the capability of Web services and user goals using F-logic. Our framework tackles the scalability problem and improves discovery performance by adding two pre-filtering stages to the discovery engine. The first stage is based on ontology comparison of user request and Web service categories. In the second stage, yet more Web services are eliminated based upon a decomposition and analysis of concept and instance attributes used in Web service capabilities and the requested capabilities of the client, resulting in a much smaller pool of Web services that need to be matched against the client request. Our pre-filtering approach is evaluated using a new Web service repository, called WSMO-FL test collection. The recall rate of the filtering process is 100% by design, since no relevant Web services are ever eliminated in by the two pre-filtering stages, and experimental results show that the precision rate is more than 53%.

KEYWORDS:

Pre-filtering, F-Logic, Semantic Web service Discovery, Scalability

1. INTRODUCTION

Semantic Web has been a popular topic of research since its introduction by Tim Berners-Lee in 2001 [5]. Based on this idea, automation of many tasks on the Internet is facilitated through the addition of machine understandable semantic information to Web resources. For instance, automatic discovery of Web services based on their functionality or composition of Web services which cannot fulfil the user requests individually becomes possible [1].

In recent years, complexity of conceptual models (i.e. WSMO [2], OWL-S [3]) for semantic description of Web services, as well as the increasing number of advertised services in repositories made the discovery processes of semantic Web services a heavyweight task [12]. In order to deal with the problem of scalability, researchers proposed various methods, such as indexing and caching mechanism [4], pre-processing strategies before actual matching [6, 7], and hybrid matchmakers that combine logic-based and non-logic based reasoning [8, 9].

This paper presents a new logical framework and two pre-filtering strategies to improve the speed and accuracy of automated Web service discovery. Our discovery framework is based on the WSMO conceptual model for semantically describing user requests (goals), Web services and domain ontologies. During the discovery process, goal capability descriptions such as inputs, outputs, pre-conditions and post-conditions (effects) are compared with advertised Web service 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 Web service and also that the Web service 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.

Our two pre-filtering stages are used to eliminate Web services that cannot possibly be successfully matched, and reduce the number of Web services which go through the logic-based matching stage. Our first pre-filtering stage uses a categorization scheme of Web services. Our second pre-filtering stage uses a new technique of extracting attributes and concepts of objects utilized in the goal and the Web service pre and post-conditions. Our technique can deal with objects and predicates that occur in a logical formula with usage of the conjunction (and) logical operator. 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.

The remainder of this paper is organized as follows. Section 2 presents background information on the WSMO model, F-Logic, FLORA-2 and our logical discovery framework. In section 3, F-Logic formalization of goal and Web services is described. Section 4 describes our pre-processing algorithms and shows how they work to reduce the run-time processing requirements in the matching phase. Section 5 shows experimental results of utilizing our proposed algorithms, as well as the evaluation of these results. In section 6 we briefly describe related works on this field, and finally in section 7 we have the conclusions and future works.

2. BACKGROUND

Our semantic Web service discovery framework focuses on Web services, goals, ontologies and mediators that are semantically described based on the WSMO [19] model and using the F-Logic[10] language as implemented in the FLORA-2 [11] logic system. The following subsections briefly introduce WSMO and its core elements, F-Logic along with its reasoner FLORA-2 and our logical semantic web service discovery framework.

2.1. WSMO definition

Web services are semantically described by providing a high level declarative specification of Web service functionality and non-functional properties in order to facilitate automatic discovery, composition and invocation of Web Services. Two prominent models in semantic Web service descriptions are Web Service Modelling Ontology (WSMO) [2] and Web Ontology Language for Services (OWL-S) [3]. There also exist other special purpose languages for the semantic description of Web services, such as DIANE Service Description (DSD) language [15] and Semantic Annotation for WSDL and XML schema (SAWSDL) [30].

WSMO comprises four core elements, namely ontology, goal, Web service, and mediator. *Ontology* is defined as a formal, explicit specification of a shared conceptualization [28]. In the context of semantic Web services, ontology provides a common vocabulary to denote the types in the form of classes or concepts, properties and interrelationships of concepts in a domain. *Goal* describes what the requester can provide, and what it expects from a Web service. *Web service* description represents different functional and non-functional features of a deployed Web service. Finally, *Mediator* handles heterogeneity problems that possibly arise between goals, Web services and ontologies.

2.2. F-logic and FLORA-2

F-Logic (frame logic) 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 and OntoBroker [13]. Our proposed intelligent agent for semantic Web service 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 [10], HiLog [18], and Transaction Logic [14, 17] into tabled Prolog code [11]. Basically, FLORA-2 supports a programing 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 [29].

2.3. Our logical semantic Web service discovery framework

In general, Web service discovery is the process of finding appropriate Web services with respect to the user request and ranking of discovered services based on user preference. Our discovery framework receives WSMO goal descriptions and WSMO Web service descriptions, all coded in F-Logic, along with related mediators and ontologies as input entities and for each goal returns an ordered list of Web services that can satisfy the needs of the goal.

Figure 1 depicts the architecture of our discovery framework. The framework consists of four stages: (i) the creation and maintenance of goals and Web services along with related domain ontologies and mediators, (ii) pre-filtering stages, (iii) matchmaker and (iv) ranking stage. In the *creation and maintenance* stage, Web service 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 Web services are filtered in two steps in order to narrow down the list of Web services that can be possible matches for the goal, the rest of the Web services being eliminated from consideration. In the *matchmaker* stage, the logical matchmaker checks whether each filtered Web service can really execute in a way such that the user goal is achieved. Finally, the *ranking* stage returns lists of matched Web services based on user preference regarding the minimization of some numeric result (for example, the cost of a flight between two cities).

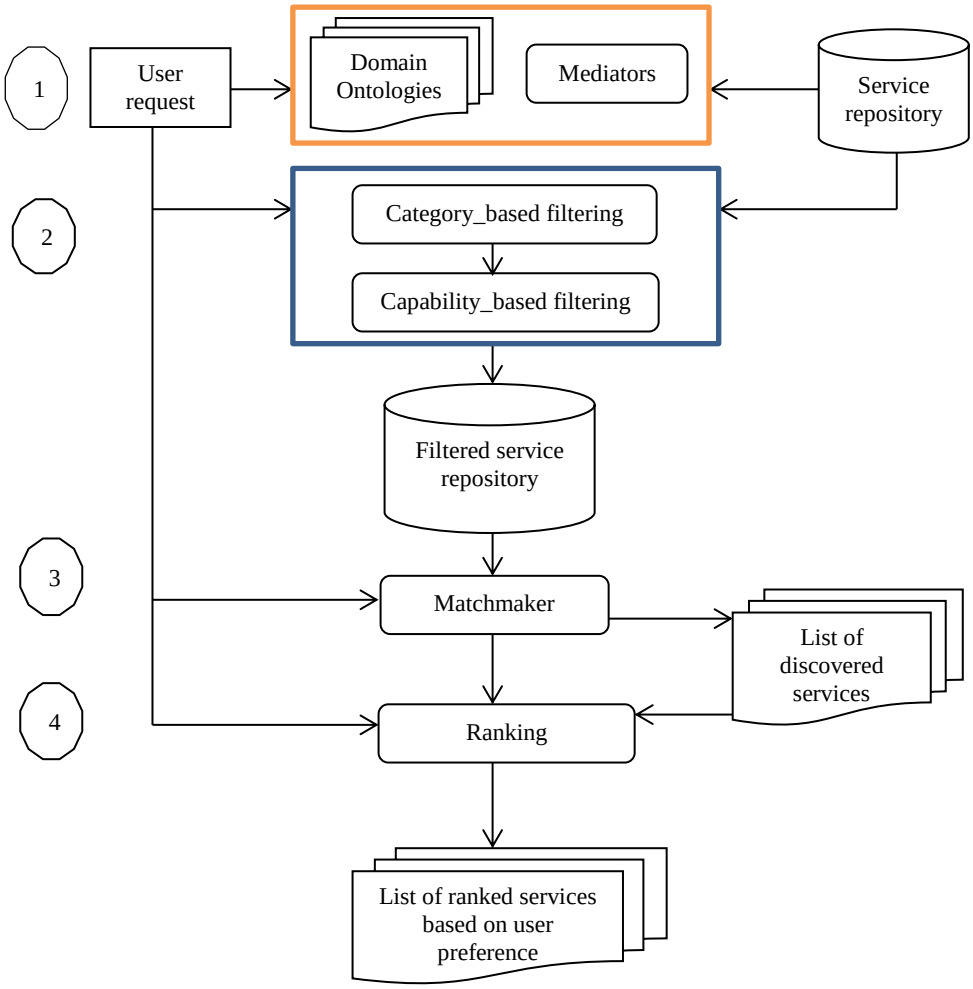

Figure 1. Proposed semantic Web service discovery framework including two pre-filtering stages

Focus of this paper is on our novel filtering strategies that we explain in detail in section 4. Here we briefly describe our logical matchmaker mechanism.

Our logical matchmaker algorithm makes use of pre-conditions and post-conditions of goals and Web services, 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 Web service (*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*: If the post-conditions for the Web service were satisfied, then the requirements of the goal should be satisfied. Note how we assume that the execution of the Web service guarantees the validity of the implication in *Ws.pre=>Ws.post*.

3. WEB SERVICE AND GOAL SPECIFICATION IN F-LOGIC

In figures 2, 3 and 4 we have the meta-level concept definitions of WSMO [19], with several enhancements. Figure 2 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 Web service, and the interface demanded from the Web service (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
    |].
```

Figure 2. Goal concept in our extended version of WSMO

The *Service* concept given in figure 3 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
      | ].
```

Figure 3. Web Service concept in our extended version of WSMO

Figure 4 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 Web service 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
      | ].
```

Figure 4. Capability concept in our extended version of WSMO

When a capability object is part of a goal, 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 Web service, 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 Web service 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 Web service to the requester that is the result of the Web service execution. Note the similarities between the goal post-condition and Web service pre-condition, as well as the goal post-condition and Web service pre-condition.

Figures 5 and 6 show the main parts of a goal and a Web service specifications respectively among various available types of goals and Web services in our repository.

Figure 5 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 wants *Sabiha_Gokcen* as a destination airport. The requester also demands flights that have a total cost less than 500$ for 2 people, and that whatever flight is returned, it must have the minimum cost. Note that logic variables start with the "?" symbol.

Goal Instance
"Book a flight from Berlin to Istanbul"

```
hasCategory  −>  AirTransportation,

requestsCapability −> ${goal_1[
        hasPrecondition−>
                ${reqFlight[
                        originateCity−> berlin,
                        terminalCity  −> istanbul,
                        departureDate −>?DDate,
                        returnDate  −>?RDate,
                        numberPeople  −> 2
                                ]:RequestTicket
                } ,
        hasPostcondition −>
                (${?BookTicket[
                        departureDate  −>?DDate,
                        returnDate  −>?RDate,
                        fromAirport  −>?FromAirport,
                        toAirport  −>?ToAirport,
                        cost  −>?TotalCost
                                ]:Response
                },
                is_equal(?ToAirport , Sabiha_Gokcen),
                less (?TotalCost, '500$')
                ),
        optimization −>
                ${optObj[optCost  −>?TotalCost]}
                                ]
                        }
```

Figure 5. Part of goal instance specification dealing with the capability desired and the category of the desired service

Figure 6 depicts part of the capability and category descriptions of a Web service instance in our Web service repository. The Web service instance belongs to the *PlaneTransportation* category and provides flight reservation for users who request a flight from one place to another place. This Web service asks for source and destination cities, desired departure and arrival date and number of people who would like to reserve this flight, consults two ontologies containing flight information (*FlightInfo_ont*) and geographical information (*Geographical_ont*), and returns the list of matching flights ordered according to minimum cost. The precondition needs two objects, one coming from the goal (instance of *RequestFlightTicket*) and one coming from an imported ontology (instance of *Flight*). The predicate *mult* multiplies its first and second parameters, and binds its third parameter to the result. It is user-defined.

Web service
"Reserve a flight"

```
hasCategory −> PlaneTransportation,

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

hasCapability−> ${ ws_x[
    hasPrecondition−>
        ($[?ReqFlight[
            startCity−>?FromCity,
            endCity−>?ToCity,
            departureDate−>?DDate,
            returnDate−>?RDate,
            numberPeople−>?HNumber
                ]: RequestAirplainTicket }
          ,
        ${?SomeFlight[
            departureDate−>?DDate,
            returnDate−>?RDate,
            fromAirport−>?FromAirport,
            toAirport−>?ToAirport,
            cost−>?Cost
                ]:Flight  }
          ,
         mult(?Cost,?HNumber,?TotalCost)),
    hasPostcondition−>
        ${response[
            departureDate−>?DDate,
            returnDate−>?RDate,
            fromAirport−>?FromAirport,
            toAirport−>?ToAirport,
            cost−>?TotalCost
                ]:Response}
            ]
        }
```

Figure 6. Part of Web service instance specification dealing with the capability and the category of the provided service

4. PROPOSED TWO-PHASE PRE-FILTERING MECHANISM

We propose a solution to tackle the scalability problem by adding two pre-filtering stages before the logical matchmaker stage of our 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 Web services that really matter with respect to the goal is carried out.

These pre-processing steps are performed through the main predicate is named *%filterMain* in listing 1. Output of this predicate is list of goals and their related Web services which are inserted into the knowledge based named *RelatedGoalWsModule* for the subsequent logical matchmaker phase. Listing 1 shows two filtering stages of this predicate.

To facilitate understanding of the code, 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.

```
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,GOAL)@FilteredWsModule,
7:                  alreadySelected(?WsName,WEBSERVICE)@FilteredWsModule,
8:
9:                  insert{related(?GoalName,?WsName)}@RelatedGoalWsModule,
10:                 ?Ins=related(?GoalName,?WsName)
11:                                 }.
```

Listing 1. Pre-filtering process containing two filtering stages (lines 2 to 4: *Cat_Filt*, lines 5 to 10: *Cap_Filt*)

For each goal-Web service pair, the first stage, *Cat_Filt* uses the *Global_Cat_Ont* to check semantic similarity of the *goal category* (Cat_g) against the *Web service category* (Cat_w). According to listing 1 line 4, if Cat_g and Cat_w are equal, synonym or in an inheritance relationship with one another, the Web service 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 Web service pre and post-conditions are extracted by our new algorithm (described in subsection 4.2). Then, extracted concepts and attributes as well as our ontology-based mediation are used to select Web services 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, and (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 Web services that survive the two-phase filtering process.

Our current scheme of *Cap_Filt* deals with logical expressions involving only the conjunction operator, positive molecules and predicates. We shall consider extension of the scheme to deal with any logical expression involving the negation and disjunction operators as well as the conjunction operator in future works.

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

4.1 Filtering according to categories (*Cat_Filt)*

The *Cat_Filt* stage filters the original Web services repository according to both specified categories and synonyms defined in the *Global_Cat_Ont* ontology. Figure 7 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*.

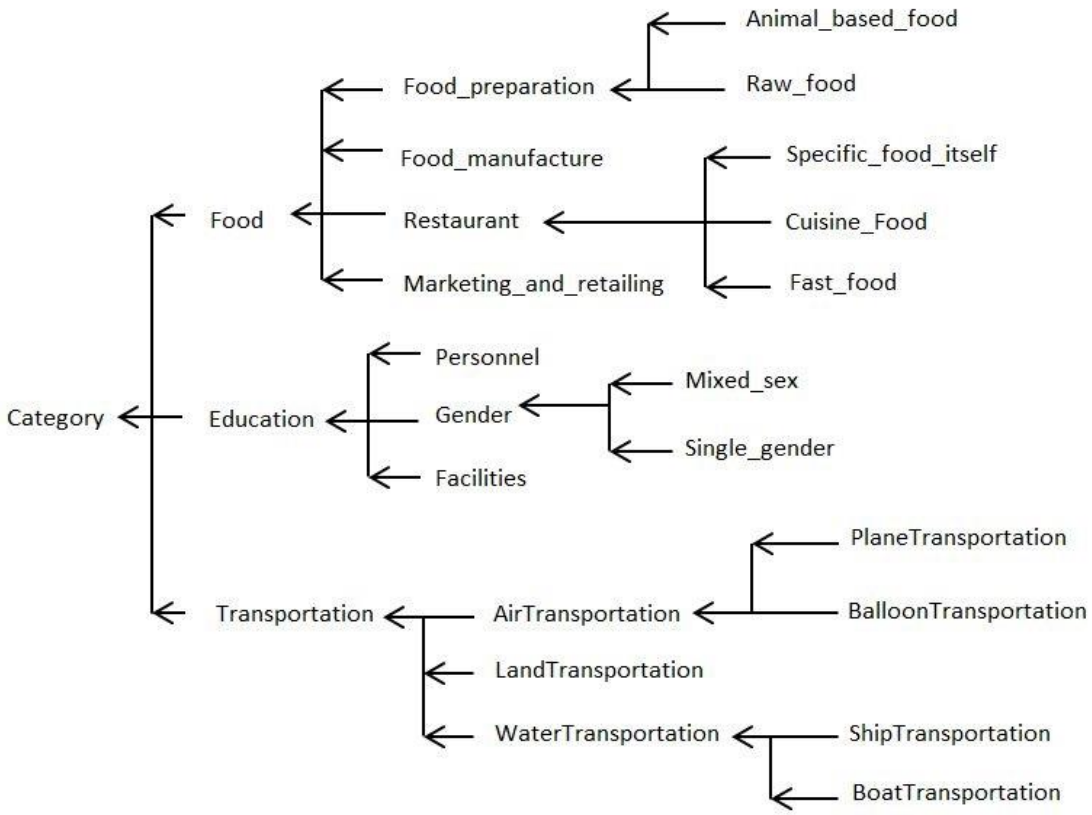

Figure 7. Part of the hierarchical structure of our specified domains in the category ontology *Global_Cat_Ont*

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.

Figure 8 shows the abstract definition of *Cat_Filt* in the form of a function that takes a goal as a parameter. Here, *g* and *w* stand for goal instance and Web service 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) does, advertised Web services in the registry which have their categories specified, and whose categories match the goal's category, (ii) Web services that have no category specified, and (iii) all Web services in case no category is specified for the goal. This definition guarantees that if there is any possibility of a Web service 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: $Cat_w \in Global\_Cat\_Ont$,
4: $Cat_g \in Global\_Cat\_Ont$,
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 }.

Figure 8. Definition of *Cat_Filt* as a function

In order to better illustrate previous definitions, consider a scenario where a user is searching for a flight lookup service among the existing Web services described in repository. Suppose that goal category is *PlaneTransportation* and categories of advertised Web services are

AeroplaneTransportation, *RailwayTransportation* and *AirTransportation* consecutively. Result of *Cat_Filt* based on *Global_Cat_Ont* ontology on the described scenario is illustrated in table 1.

Table 1. Results of *Cat_Filt* algorithm on described scenario

Cat_G	Cat_{WA}	Cat_{WB}	Cat_{WC}
	AeroplaneTransportation	RailwayTransportation	AirTransportation
PlaneTransportation	√	×	√

According to the concepts relationships definition in *Global_Cat_Ont* ontology, Cat_{WA}, *AeroplaneTransportation*, is a synonym of *PlaneTransportation* and Cat_{WC}, *AirTransportation*, is a subclass of *PlaneTransportation*. Thus, Web services A and C remain as inputs of *Cap_Filt*, and Web service (B) is discarded as irrelevant.

The result of the *Cat_Filt* stage is fed into the second stage *Cap_Filt* in order to eliminate even more Web services that cannot possibly be a match for the given goal.

4.2 Filtering according to capability decomposition (*Cap_Filt)*

Unlike other proposals mentioned in related works, our *Cap_Filt* algorithm eliminates irrelevant Web services based on checking the attributes and concepts of objects employed in the goal and the Web service pre and post-conditions. This algorithm first extracts attributes and concepts of objects in goal and Web service specifications (it can deal with predicates and objects that occur in a logical formula possibly including the conjunction logical connective), then analyzes semantic equivalency between extracted attributes and concepts in order to filter out unrelated Web services.

The level of similarity between such these parameters is obtained based on their hierarchical relationships and implication inside this ontology. In this work levels of semantic similarity between parameters are defined considered based on *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 defined as *plug-in* only if concept or attribute of goal request is superclass of concept or attribute of the Web service. Degree of two concepts or two attributes are defined as *subsume* only if concept or attribute of goal request is subclass of concept or attribute of the Web service. 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 [25] a dictionary of synonymous words. Thus, synonym similarity between the goal and Web service parameters in the *Cap-Filt* algorithm is calculated by making use of the WordNet[1] online synonym dictionary.

As an illustration of the above definition, consider the instances of a matched goal and Web service defined in section 3. List of concepts and attributes of our goal and Web service pre-conditions are shown in table 2.

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

Table 2. List of concepts and attributes obtained as a result of %DC/2 predicate

Name	**[(ConceptName, [List of attributes])]**
Goal.Pre	[[(RequestTicket, [originateCity, terminalCity, departureDate, returnDate, numberPeople])]].
Ws.Pre	[[(RequestAirplainTicket, [startCity, endCity, departureDate, returnDate, numberPeople]), (Flight, [departureDate, returnDate, fromAirport, toAirport, cost])]].

As it was shown in Table 2, *originateCity* and *startCity* are the first attribute of goal pre-condition and Web service pre-condition, respectively. 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. We employ both semantic and synonymous equivalency of pre and post conditions. This work improves the efficiency of discovery framework by avoiding false negatives (FN). (i.e. FN describes Web services which are relevant, but are classified as irrelevant).

```
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:         %DC (?WsPre, ?Ws_Pre_Att_Cnp),
10:        %Check_Att_Cnp (?WsName, ?Ws_Pre_Att_Cnp, WEBSERVICE),

                          //----------Post-Condition --------
11:        deleteall{?_A[?_B->?_V]:?_C @GoalWsAttModule},

12:        %FindGoalOrWsAtt (?WsPost,GoalWsAttModule),
13:        %DC (?GoalPost, ?Goal_Post_Att_Cnp),
14:        %Check_Att_Cnp (?WsName, ?Goal_Post_Att_Cnp, GOAL).
```

Listing 2. Critical parts of the *%filter_Cap* predicate

Listing 2 depicts the critical parts of the *%Filter_Cap* predicate. The filtering, which is based on concepts and attributes of objects in the capability specification of the Web service and goal is carried out in the following manner:

1) Lines 2 to 7 read goal and Web service pre and post conditions from their individual's modules.

2) As the process of checking semantic and synonymous similarity of goal and Web service specifications are done in knowledge base module (*GoalWsAttModule)*, in listing 2 line 8, attributes and concepts of goal pre-conditions are moved into *GoalWsAttModule* through *%FindGoalOrWsAtt/2* predicate.

3) Attributes and concepts of Web service pre-conditions are extracted via *%DC/2*[2] predicate. This transactional predicate decomposes Web service pre-conditions, and then extracted concepts and attributes are stored in different lists. Listing 2 line 9 shows that this predicate is called with *?WsPre* 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 Web service pre-conditions.

4) Finally, line 10 depicts *%Check_Att_Cnp* predicate that implements algorithm 1. It compares concepts and attributes related to goal pre-conditions with concepts and attributes associated to Web service pre-conditions based on semantic equivalency between them. Output is the name of related Web services whose concepts and attributes exist in requested goal, as explained below. Web service names that pass through this level of filtering is stored in the knowledge base named *FilteredWsModule*.

Algorithm 1 %Check_Att_Cnp predicate

Input: List of Ws_Pre_Att_Cnp and Goal.Pre **or**
List of Goal_Post_Att_Cnp and Ws.Post
Output: WsName

```
1:  for all Atts_i as attribute in List of ?Ws_Pre_Att_Cnp || ?Goal_Post_Att_Cnp do
2:      for all Atts_k as attribute of Goal.Pre || Ws.Post exist in GoalWsAttModule do
3:          if (Atts_i ∈ Mediator and Atts_k ∈ Mediator and
4:              (Atts_i :: Atts_k or
5:              Atts_k :: Atts_i or
6:              Atts_i :=: Atts_k)) then

7:              for all Cnp_i as concept in List of Ws_Pre_Att_Cnp ||Goal_Post_Att_Cnp do
8:                  for all Cnp_k as concept of Goal.Pre || Ws.Post in GoalWsAttModule do
9:                      if (Cnp_i ∈ Mediator and Cnp_k ∈ Mediator and
10:                         (Cnp_i :: Cnp_k or
11:                         Cnp_k :: Cnp_i or
12:                         Cnp_i :=: Cnp_k)) then

13:                             Insert WsName into FilteredWsModule
14:                     end if
15:                 end for
16:             end for
17:         end if
18:     end for
19: end for
```

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

5) As it is shown in line 11 of listing 2, contents of knowledge base *GoalWsAttModule* which already consisted of goal pre-condition's attributes and concepts is erased in order to replace with the new data.

6) Attributes and concepts of Web service post-conditions are moved into *GoalWsAttModule* by *%FindGoalOrWsAtt* predicate in line 12 of listing 2**.**

[2] http://cmpe.emu.edu.tr/samira/find_att_cnp.flr

7) Attributes and concepts of goal post-conditions are extracted via *%DC/2* predicate, and the results are stored in *?Goal_Post_Att_Cnp* variable as it is shown in line 13.

8) In line 14, similar to line 10, *%Check_Att_Cnp* predicate implements algorithm 1. But this time it checks concepts and attributes related to Web service 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 Web services is inserted into the knowledge base so that full checking of the proof commitments can be carried out in the next stage.

5. EXPERIMENTAL RESULTS AND DISCUSSIONS

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[3] and SAWSDL-TC[4]. OWLS-TC, which mainly considers input and output parameters is applicable for approaches that deal with OWL-S Web services descriptions, and approaches which employ SAWSDL Web service descriptions use the SAWSDL-TC test collection.

The latest version of OWLS-TC at this time is version 4 [20]; it consists of 1083 Web services and 42 queries which are written in the OWL-S language. Unfortunately, the majority of Web services in OWLS-TC are only partially described, being based on input and output types. Only in the last version (version 4), 160 Web services contain pre-conditions and post-conditions (effects) which are described in different languages such as, SWRL[5] and PDDL[6].

The SAWSDL-TC test collection is established to support the performance appraisal of SAWSDL matchmakers. The latest version of SAWSDL-TC, at this time is version 3; it consists of 1080 semantic Web services and 42 queries which are described in the SAWSDL language. However, descriptions of Web services and queries are only based on input and output parameters [31].

The majority of approaches (such as [6, 8, 21, 22, 26, 27]) that work in our field and are mentioned in related works evaluated efficiency and accuracy of their works based on OWLS-TC version 3 test collection. Among all related works, authors of [24] evaluated their proposal based on last version of OWLS-TC test collection, but only input and output parameters are considered for evaluation of their work.

Therefore, due to unavailability of an appropriate test collection that covers main functional descriptions of Web services such as pre-conditions and post-conditions, as well as a categorization scheme of Web services, we generated our own test collection of Web service/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*[7].

WSMO_FL contains three different domains, namely transportation, food and education, with 250 different F-Logic Web services descriptions, 6 different F-Logic goals descriptions, 22 concepts and 1225 instances.

[3] http://projects.semwebcentral.org/projects/owls-tc/
[4] http://projects.semwebcentral.org/projects/sawsdl-tc
[5] SWRL: A Semantic Web Rule Language, http://www.w3.org/Submission/SWRL/
[6] International Planning Competition, http://planning.cis.strath.ac.uk/competition/
[7] http://cmpe.emu.edu.tr/samira/WSMO-FL.htm

In this section, 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 performed on a machine with Windows 7, a 2.93 GHz Intel processor and 4.00 GB of RAM.

In order to determine the actual improvements of our proposed pre-filtering stages, we measured several indicators: (i) the average response time of our semantic Web service matchmaker with filtering (*Filt_Disc*) and without filtering (*Naive_Disc*). (ii) The number of Web services that have been effectively eliminated from the initial pool of available Web services at each pre-filtering stage, (iii) Precision, and (iv) Recall. Due to the fact that our filtering stages never eliminate any Web service from consideration unless they are guaranteed to fail at the logical matching stage, it is no surprise that recall rate is always 100%.

The results of the performed tests for the goal are given in table 3, showing the mean and median of the time it took to match the goal against varying number of Web services. 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 3. Statistical comparison of *Filt_Disc* and *Naive_Disc*

No. WS	Engine	Mean time (sec)	Median time (sec)
5	Filt_Disc	0.08	0.09
	Naive_Disc	0.08	0.08
50	Filt_Disc	0.27	0.27
	Naive_Disc	3.93	4.02
150	Filt_Disc	0.45	0.51
	Naive_Disc	12.39	12.39
250	Filt_Disc	0.62	0.60
	Naive_Disc	17.43	17.33

Figure 9 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 0.08 to 0.062 second, while for the same goal and Web services in *Naive_Disc* it dramatically increases and is in range of 0.08 to 17.5 seconds.

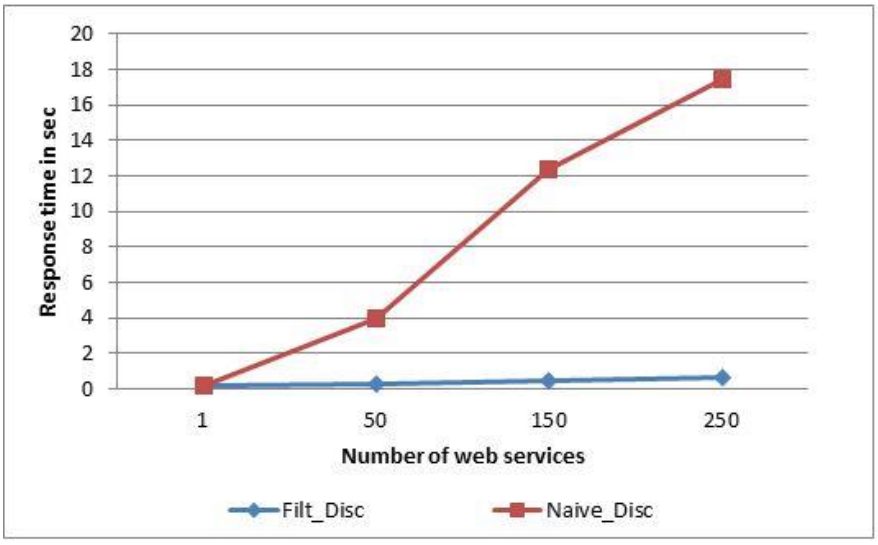

Figure 9. Comparison of *Filt_Disc* and *Naive_Disc*

Figure 10 depicts the dramatic number of reductions in the number of Web services that remain after each pre-filtering phase. The data has been collected by matching six different goals and varying number of Web services for each goal. The chart indicates that *Cap_Filt* through the semantic equivalency of goal and Web service concepts and attributes does a very good job of eliminating irrelevant Web services, given that most of the remaining Web services after its application passes the *Cat_Filt* stage.

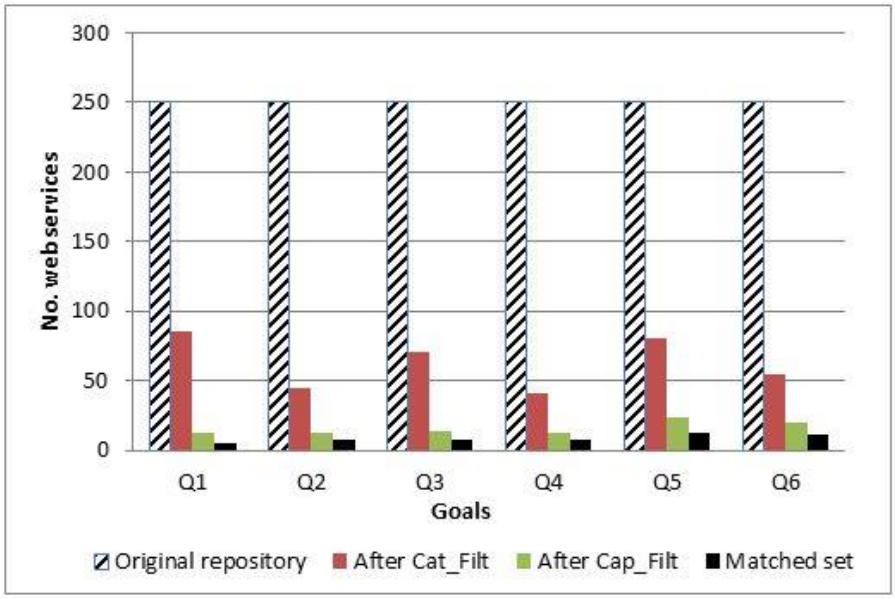

Figure 10. Effectiveness of the two pre-filtering stages in eliminating irrelevant Web services

To analyse the accuracy of our pre-filtering stages, table 4 gives the precision and recall values of the combined pre-filtering stages for the same set of data obtained by running 6 requested goals against 250 Web services in the repository.

Precision is percentage of the retrieved Web services that are actually relevant. In our context, "retrieved Web services" means the Web services that survived the two-stage elimination process, and a web service is "relevant" to a goal if the logical matchmaker says so. With these definitions, precision is formalized as [33]:

$$Precision = \frac{Number\ of\ Relevant\ Web\ services\ in\ the\ retrieved\ set}{Number\ of\ Retrieved\ Web\ services}$$

Recall is the portion of the relevant Web services that are successfully retrieved. It is formalized as [33]:

$$Recall = \frac{Number\ of\ relevant\ web\ services\ in\ the\ retrieved\ set}{Number\ of\ all\ relevant\ Web\ services\ in\ the\ repository}$$

Table 4. Precision and recall of combined pre-filtering stages in each requested goal

Goal	Q1	Q2	Q3	Q4	Q5	Q6	Average
Precision	41.6%	58.3%	57.1%	53.8%	56.5%	55%	53.72%
Recall	100%	100%	100%	100%	100%	100%	100%

As shown in table 4, average precision for all request queries is 53.72% which to some extend can be considered a good precision rate. It means that around 55% of retrieved Web services are exactly matched with the requested goal and the other around 45% are irrelevant. However, the

average recall of queries has the highest possible rate, 100%. With this 100% recall rate, all the relevant Web services in Web service repository are retrieved through the proposed pre-filtering stages, an important feature that sets out filtering strategy apart from all the other proposals

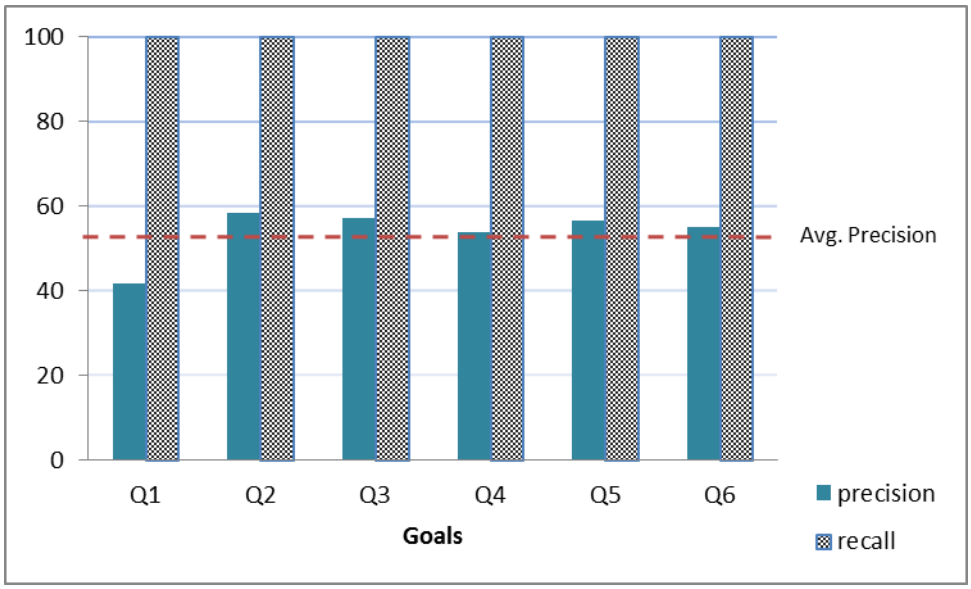

Figure 11. Precision and recall of each requested goal along with the average precision line

Figure 11 graphically shows the precision and recall rate of each requested goal together with the average precision line. The chart illustrates that precision rate of all requests except the first one (Q1) are higher than the average. Low precision rate of Q1 indicates that there exist many Web services in the repository whose attributes and concepts are semantically similar to the concepts' name and attributes' name of requested goal, however, the value of Web service attributes which defined in ontologies do not match with the requested value of goal attributes. Such Web services fail in the actual logical matching procedure.

As we explained before, the reason of top average recall rate is that all relevant Web services are retrieved by *Cat_Filt* and *Cap_Filt* algorithms, which is another way of saying that. In the pre-filtering stages, we only eliminate Web services that the matcher would definitely reject. The reason for the not-so-high average precision rate in pre-filtering stages is that although the retrieved Web services are similar with requested goal due to semantic and synonym equivalency of their concepts and attributes in domain ontology, maybe some Web services that will eventually be rejected pass through the filters. However, in a real world situation where thousands of categories exist, and Web services, as well as goals, are annotated by categories, precision would be expected to skyrocket, since the majority of Web services under consideration would be eliminated by the first stage of filtering (Cat_Filt).

Since our framework evaluation is based on our own test collection, WSMO-FL, which is new and not published yet, a comparison between the accuracy and performance of our work and the other available works in the literature would not be very informative. However, an average of 100% for recall and 53.72 % for precision 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 Web services due to pre and post-condition parameters. The other studies mentioned in related works did not consider this much complexities in their goals and Web services.

6. OVERVIEW OF RELATED WORKS

Recently, although a wealth of insightful efforts have proposed different solutions to improve the semantic Web discovery process and their related scalability issues, 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, as well as their advantages are compared to our approach.

Table 5 compares our work with the related works based on several dimensions with respect to semantic Web service discovery improvement. These dimensions are: pre-processing, discovery method, parameters 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 Web service and goal descriptions (e.g. categorization of each advertised / requested Web service at design time). *FP* stands for methods to compare functional parameters of goal and Web services (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 Syntactic (ST) (i.e. no synonym 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 method represents 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 demonstrates degree of completeness of a research (whether it uses the major functional parameters of goal and Web services or not). Major functional parameters of goal and Web services in OWL-S and WSMO models are Input (I), Output (O), Pre-condition (PRE) and Post-condition (effect) (POS).

We summarise the result of the comparative study of Web service discovery approaches in table 5, where each row represents an approach, and the columns stand for main dimensions in Web service discovery improvement. The symbol ‘‘√’’ used to denote that the specified approach supports the corresponding dimension, and ‘‘-’’ means that it does not.

Table 5. Comparison of this work with related works

App	Pre-processing					Discovery method			Parameters				Framework
	NFP		FP										
	TX	SY	TX	SY	ST	LOG	NLOG	HY	I	O	PRE	POS	
[8]	-	-	-	-	-	-	-	√	√	√	-	-	OWL-S
[9]	-	-	-	-	-	-	-	√	√	√	-	-	WSMO
[4, 16]	-	-	-	-	-	√	-	-	√	√	√	√	WSMO
[26]	-	-	-	-	-	√	-	-	√	√	√	√	OWL-S
[27]	-	-	-	-	-	-	-	√	√	√	-	-	OWL-S
[21]	√	-	-	-	-	√	-	-	√	√	-	-	OWL-S
[22]	-	√	-	-	-	-	-	-	√	√	-	-	OWL-S
[6]	-	-	√	-	-	-	-	-	√	√	-	-	OWL-S
[7]	-	-	√	-	-	-	-	-	√	√	-	-	WSMO
[34]	-	-	-	-	√	√	-	-	√	√	√	√	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)
Syntactic (ST)

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 semantic Web service 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 later tries to reduce the size of original repository and the filtered repository is used as input of actual matchmaker.

a) 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.[8] implemented a hybrid matchmaker consisting of both approximated Information Retrieval (IR) matching, such as syntactic similarity technique, and OWL-DL logical reasoner to discover semantic Web services. Authors used four variants to calculate the text similarity of parameters, named *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, named *WSMO-MX* [9]. Their comprehensive evaluations demonstrate that both approaches presented high precision in the S3 contest [32]. However, shortcomings of their solution are (i) it is time consuming because of high calculation costs related with both logic-based matching

and text-based similarity matching, (ii) they retrieve Web services 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 Web service repository and consequently improving the overall performance of matchmaker.

Stollberg et al. in [4, 16] 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 Web services. 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 Web services are stored in the SDC graph are used for subsequent discovery.

Authors of [4, 16] 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 Web services 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 Web services which are stored in graph can be possibly decreased.

Authors of [26] introduced SPARQL as a language to describe the pre-conditions and post-conditions of OWL-S Web services 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 SME_2 test tool[8].

Although the method offered in [26] is based on pre and post-conditions of Web services and goals, their evaluation is performed based on OWLS-TC V3. While presented Web services descriptions in this test collection are without pre and post-conditions. In addition, our pre-filtering stages could be also useful to avoiding SPARQL agent to load the all available Web services on the repository and as a result cause to further improvement to their agent performance.

Amorim et al. discuss a hybrid matchmaker called *OWL-S Discovery*. It is combination of semantic filters based on input and output parameters of requested/advertised services and analysing each neighbour relationship in domain ontology [27]. Authors employed 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 [23] and the hybrid algorithm OWLS-MX through OWLS-TC v3 test collection. Our proposal also can be applied to the top of *OWL-S Discovery* to further improve discovery processes. But our work uses a more expressive model to describe user requests and Web services descriptions as they contain pre and post conditions.

b) Approaches using pre-processing mechanisms

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

These approaches make use of pre-processing mechanisms that help to optimization of automated Web service discovery by narrowing down the set of existing Web services 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.

1) Pre-filtering based on categorization schemes of NFPs:

Most of the efforts related to pre-filtering techniques follow the classification methods: either exploit hierarchical categorization schemes of Web services on the basis of domain ontologies [21] or use dictionary of synonymous words [22].The filtering process is separate from 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 Web service NFPs that refer to service application domain.

Authors of [21, 22] implemented their categorization proposals on OWL-S Web services 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 [21] the defined category concept of the service request is compared with the defined category concept of advertised Web services via hierarchical categorization scheme in global category ontology. A Web service is eliminated if it has no category relationship with the request category. However, in [22] equivalency of requested and advertised Web services category concepts are computed via their relationship in the WordNet [25] 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 one concept to both goal and Web service descriptions, it named *hasCategory*. (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 Web service category (i.e. if two categories mean the same thing or inherit the same class).

2) Pre-filtering based on semantic similarity of FPs:

Authors of [6, 7] also used pre-processing strategies before the actual matching process. However, their pre-filtering is based on FPs of Web services, not NFPs. They present two different SPARQL queries to facilitate the search process on a semantic Web service registry. They automatically create SPARQL queries (named 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 Web service repository. Based on these two queries, only Web services 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 [7]. Four major differences between our work and theirs are that:

(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 Web service 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 only a pre-processing stage to filter the preliminary Web service repository and they did not implement any service matchmaking, so they cannot be evaluated on their own.

Among all the mentioned approaches, [34] 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, analyzing objects, attributes and concepts. Our discovery engine works with much richer descriptions of Web services 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 Web services than either approach alone.

7. CONCLUSIONS AND FUTURE WORKS

We have shown that the overall performance and accuracy of semantic Web service discovery frameworks can be improved significantly through the introduction of pre-filtering stages that eliminate most of the irrelevant Web services from consideration at the computationally expensive matching stage. Specifically, in this paper, we proposed *Category_based* and *Capability_based* pre-filtering mechanisms for narrowing down the number of Web service descriptions that need to be considered in the matching phase according to their relevance to the current goal.

We evaluated the effectiveness of our proposal in a novel test collection, *WSMO-FL*, which consists of 250 Web service specifications of varying complexities and 6 goals. Our filtering stages stand out due to their 100% recall rate that is a consequence of their design, their ability to deal with complex specifications of goals and Web services 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/Web services 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 work has several further advantages, summed up in the following:

- Unlike the majority of semantic Web service discovery approaches which are only performed on input and output concepts, our semantic Web service discovery framework deals with concepts and attributes of Web service and goal pre and post conditions.

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

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

- Due to incomplete service descriptions in OWLS-TC test collections (i.e. Web services 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 Web services and goals capabilities (i.e. Web services and goals are described based on pre and post-conditions).

- 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 extend our scheme in the following ways:

(i) extend the second stage so that it can work on *any* logical expression containing the logical connectives *conjunction (and), disjunction* (*or)* and *negation* (*not)* to any nesting depth.

(ii) extend our new *WSMO-FL* test collection to a) have a much larger number of Web services and goals, as well as categories, b) extend complexity of Web service and goal pre and post-conditions, and c) expand the dictionary of synonymous words in the existing domain ontologies.

REFERENCES

[1] Sheila A. McIlraith, Tran Cao Son & Honglei Zeng, (2001) "Semantic web services", IEEE Intelligent Systems, Special Issue on the Semantic Web, Vol. 16, No. 2, pp. 46-53.

[2] Dumitru Roman, Uwe Keller, Holger Lausen, & et al. (2005) "The Web service modelling ontology", Applied ontology 1, No. 1, pp. 77-106.

[3] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott & et al. (2004) "OWL-S: Semantic Markup for Web Services", W3C member submission 22.

[4] Michael Stollberg, Martin Hepp & Jörg Hoffman, (2007) "A Caching Mechanism for Semantic Web service Discovery", ISWC/ASWC, Vol. 4825 of LNCS, Springer, pp. 480-493.

[5] Tim Berners-Lee, James Hendler & Ora Lassila, (2001) "The Semantic Web", Scientific American 284, No. 5, pp. 28-37.

[6] Jos Mara García, David Ruiz & Antonio Ruiz-Corts, (2012) "Improving semantic web services discovery using SPARQL-based repository filtering". Web Semantics: Science, Services and Agents on the World Wide Web, Vol.17, pp.12-24.

[7] Jos Mara García, David Ruiz & Antonio Ruiz-Corts, (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.

[8] Matthias Klusch, Benedikt Fries & Katia Sycara, (2009) "OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services", Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 7, pp. 121-133.

[9] Matthias Klusch & Frank Kaufer, (2009) "WSMO-MX: A hybrid Semantic Web service matchmaker", Web Intelligence and Agent Systems, Vol. 7, pp. 23-42.

[10] Michael Kifer, Georg Lausen & James Wu, (1995) "Logical foundations of object-oriented and frame-based languages", Journal of ACM, Vol. 42, pp. 741-843.

[11] Michael Kifer, Guizhen Yang, Hui Wan & Chang Zhao, (2014) "FLORA-2: User's manual", Version 1.0, Stony Brook University, U.S.A.

[12] Le Duy Ngan & Rajaraman Kanagasabai, (2013) "Semantic Web service discovery: state-of-the-art and research challenges", Personal and ubiquitous computing, Vol. 17, No. 8, pp. 1741-1752.

[13] Jürgen Angele, (2014) "OntoBroker: Mature and approved semantic middleware", Semantic Web 5, No. 3, pp. 221-235.

[14] Anthony Bonner & Michael Kifer, (1998), "A logic for programming database transactions", In Logics for Databases and Information Systems, pp. 117-166.

[15] Ulrich Kuster, Birgitta Konig-Ries, Michael Klein & Mirco Stern, (2007) "DIANE: a matchmaking-cantered framework for automated service discovery, composition, binding, and invocation on the web". International Journal of Electronic Commerce, Vol.12, No. 2, pp. 41-68.

[16] Michael Stollberg, Joerg Hoffmann & Dieter Fensel, (2011) "A caching technique for optimizing automated service discovery". International Journal of Semantic Computing, World Scientific, Vol. 5, No.1, pp.1-31.

[17] Anthony J. Bonner & Michael Kifer, (1994) "An overview of transaction logic". Theoretical Computer Science, Vol. 133, No. 2, pp. 205-265.

[18] Weidong Chen, Michael Kifer & David S. Warren, (1993) "HiLog: A foundation for higher-order logic programming", Journal of Logic Programming, Vol. 15, No. 3, pp. 187-230.

[19] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn , Michael Stollberg, Dumitru Roman & John Domingue, (2007) "The Concepts of WSMO, Enabling Semantic Web Services: The Web service modelling Ontology", Springer Science & Business Media, pp. 63- 81.

[20] Matthias Klusch, Mahboob Alam Khalid, Patrick Kapahnke, Benedikt Fries & Martin Vasileski Saarbrücken, (2010) "OWLS-TC -OWL-S Service Retrieval Test Collection, User Manual", Version 4.0, Saarbrücken, Germany.

[21] Thair Khdour, (2011) "Towards Semantically Filtering Web Services Repository". In Digital Information and Communication Technology and Its Applications, Springer Berlin Heidelberg, pp. 322-336.

[22] Keyvan Mohebbi, Suhaimi Ibrahim & Mazdak Zamani, (2013) "A Pre-matching Filter to Improve the Query Response Time of Semantic Web Service Discovery", Journal of Next Generation Information Technology, Vol. 4, No.6.

[23] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne & Katia Sycara, (2002) "Semantic matching of web services capabilities", Proceedings of the 1st International Semantic Web Conference, LNCS, Vol. 1, pp. 333– 347.

[24] Zijie Cong & Alberto Fernandez, (2014) "Service discovery acceleration with hierarchical clustering". Information Systems Frontiers, Vol. 17, No. 4, pp. 1-10.

[25] Christiane Fellbaum, (1998) “WordNet: An electronic lexical database”. Blackwell Publishing Ltd, MA.

[26] Marco Luca Sbodio, David Martin & Claude Moulin, (2010) “Discovering Semantic Web Services using SPARQL and Intelligent Agents”, Web Semantics: Science, Services and Agents on the World Wide Web ,Vol. 8, No. 4, pp. 310-328.

[27] Rodrigo Amorim, Daniela Barreiro Claro, Denivaldo Lopes, Patrick Albers & Aline Andrade, (2011) “Improving Web service discovery by a functional and structural approach”, IEEE International Conference on Web Services (ICWS), pp. 411-418.

[28] Thomas R Gruber, (1993) “A translation approach to portable ontology specifications. Knowledge acquisition”, Knowledge acquisition, Vol. 5, No. 2, pp. 199-220.

[29] Hui Wan, Benjamin Grosof, Michael Kifer, Paul Fodor & Senlin Liang, (2009) “Logic programming with defaults and argumentation theories”. In: Logic Programming, Springer Berlin Heidelberg, pp. 432–448.

[30] Holger Lausen & Joel Farrell, (2007) “Semantic annotations for WSDL and XML schema”, W3C recommendation.

[31] Mahboob Alam Khalid, Benedikt Fries, Martin Vasileski, Patrick Kapahnke &Matthias Klusch, (2010) ”SAWSDL-TC Service Retrieval Test Collection, User Manual”, Version 3.0, Saarbrücken, Germany.

[32] Matthias Klusch, (2012) " The S3 Contest: Performance Evaluation of semantic web services ", In Semantic Web Services: Advancement through Evaluation, Springer.

[33] Ricardo Baeza-Yates & Berthier Ribeiro-Neto, (1999) “Modern information retrieval”, New York: ACM press, Vol. 463.

[34] László Kovács, András Micsik & Péter Pallinge, (2006) "Two-phase Semantic Web Service Discovery Method for Finding Intersection Matches using Logic Programming", In Workshop on Semantics for Web Services.