A Synergy of Planning and Ontology Concept Ranking for Semantic Web Service Composition

Ourania Hatzi[1], Georgios Meditskos[2], Dimitris Vrakas[2], Nick Bassiliades[2], Dimosthenis Anagnostopoulos[1], and Ioannis Vlahavas[2]

[1] Department of Informatics and Telematics, Harokopio University of Athens, Greece
{raniah,dimosthe}@hua.gr
[2] Department of Informatics, Aristotle University of Thessaloniki, Greece
{gmeditsk,dvrakas,nbassili,vlahavas}@csd.auth.gr

Abstract. This paper presents a prototype system that exploits planning and an ontology concept ranking algorithm for composing semantic Web services (PORSCE). The system exploits the inferencing capabilities of a Description Logics Reasoner in order to compute the subsumption hierarchy of the ontologies whose concepts are used in the OWL-S Profile descriptions as input and output concepts. The concept ranking algorithm is applied over this hierarchy in order to determine similar concepts based on different degrees of semantic matching relaxation, such as subclass or sibling hierarchical relationships. The domain independent planning system's role is to semantically search the space of possible compositions of Web services, generating plans according to the desirable level of relaxation.

Keywords: Semantic Web Service Composition, Planning, Ontology Concept Ranking, Semantic Matching Relaxation.

1 Introduction

The advent of web services (WS) is a proof that nowadays the need for communication among loosely coupled distributed systems is bigger than ever. Web services offer a well-defined interface through which the major problem of interoperability on the Web can be dealt with. In order to exploit the web service technology to its full extent, Semantic Web languages, such as OWL-S [11], WSDL-S [12] SAWSDL [14], and tools, such as ontology reasoners [7], are used for the semantic annotation and processing of WS, leading to a new notion of web services, referred to as Semantic Web Services (SWS). The SWS paradigm is motivated by the fact that while the XML representation of services' characteristics (WSDL [15]) guarantees syntactic interoperability, it is unable to capture the semantics of information, which is essential for the automation of WS-related procedures.

The procedure of combining simple WS in order to create a complex service of enhanced functionality is fundamental. Composition can be either manual, where the user participates by selecting appropriate Web services from a set of available ones,

H. Geffner et al. (Eds.): IBERAMIA 2008, LNAI 5290, pp. 42–51, 2008.

or automated, where the composition plan is generated automatically, based on initial requirements about functional and non-functional properties.

In this paper the automated web service composition paradigm is approached as a planning problem over the OWL-S profile descriptions of web services. More specifically, PORSCE is proposed; a combination of a domain-independent planning system and a concept ranking module for computing similarities among OWL ontology concepts. The planning module searches for composition plans by matching OWL-S Profile input and output (I/O) parameters, while the concept ranking module semantically alters the I/O requirements of the Web services, selecting semantically related ontology concepts based on different notions of concept similarity.

The rest of the paper is organized as follows: Section 2 outlines the system architecture and points out the way the various modules cooperate. Sections 3 and 4 elaborate on the core of the system, namely the OWL Ontology Manager and the Planning System respectively. Section 5 presents some experimental results, while Section 6 concludes the paper and poses future directions.

2 System Architecture

PORSCE is a synergy of an OWL-S parser utilizing the OWL API [16], the OWL Ontology Manager, the PDDL Converter and an external planning system. The OWL-S Parser parses OWL-S profiles that correspond to a set of SWS. The output of the OWL-S Parser is a description of the WS which is provided to the PDDL converter and the domain ontologies which are forwarded to the OWL Ontology Manager (OOM). The OOM, utilizing the DL reasoner, applies the algorithm of Section 3.2 for determining similar concepts to a query concept q. The PDDL converter is responsible for expressing the problem of SW composition as a planning problem, interacting with the user in order to set the required threshold of conceptual similarities, enhancing the planning problem with semantic information retrieved from OOM and cooperating with the external planning system in order to acquire a solution to the problem. More details on both the OOM and the Planning System are provided in Sections 3 and 4 respectively.

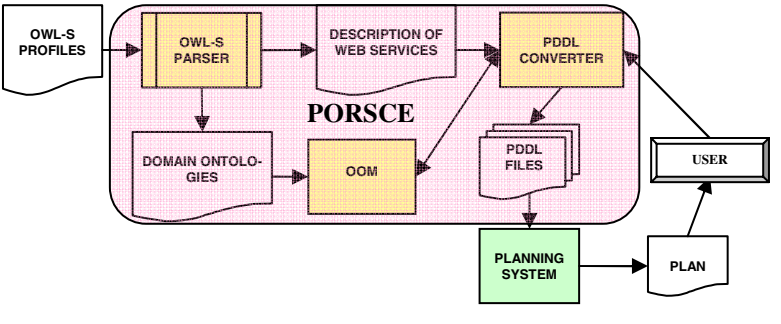

Fig. 1. The architecture of PORSCE

3 OWL Ontology Manager

The OWL Ontology Manager (OOM) is responsible for retrieving "similar" ontology concepts to a specific query concept, according to a degree of relaxation that is defined based on the hierarchical relationship *H* and the distance *D* between the concepts. OOM utilizes the inferencing capabilities of the Pellet DL Reasoner [7] in order to compute the subsumption relationships among the concepts. The general idea is to relax the concept matching criterion in order to obtain plans that might be useful, especially in cases where an exact input/output matching plan is not available.

3.1 Hierarchical Relationships

OOM utilizes three logic-based types of concept match [13] between a query q and an ontology concept C, and extends them by also introducing the *sibling* match:

- **exact.** The two concepts are inferred to be equivalent or have the same URI.
- **plugin.** This type of match holds when q is superclass of C.
- **subsume.** This type of match holds when the q is subclass of C.
- **sibling.** The two concepts have a common superclass T.
- **fail.** Nothing of the above holds, for example in the case of disjoint concepts.

These matching types represent the hierarchical relationships that two concepts could have in a hierarchy.

3.2 Concept Distance

Apart from the hierarchical relationships, the degree of relaxation is defined based on the distance *D* of two concepts, following the logical assumption that the more concepts exist between two concepts, the less "similar" they are. In simple subclass relationships, the distance between two concepts is defined as the sum of the concepts that exist between them, also including in the sum the two concepts themselves. For example, the distance between two concepts with a direct subclass relationship is 2. In the case of a sibling relationship between two concepts *q* and *C* with a common superclass *T*, the distance is defined as $D = d_{q,T} + d_{T,C} - 1$. For example, the distance of two concepts *B* and *C* with a direct common superclass *A* is 3, as Fig 2 depicts.

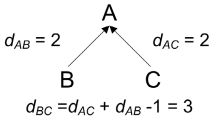

Fig. 2. An example of a sibling relationship between two concepts

The algorithm for determining all the concepts of an ontology that are similar to a query concept q is presented below. The listSubclasses(C) and listSuperclasses(C) notations are used to denote the set of subclasses and superclasses of an ontology concept C, and dist(C, C′) to denote the distance between the concepts C and C′. In case of exact matches, the algorithm returns only the equivalent to q concepts, while

in the case of plugin matches, the algorithm finds the subclasses of the concept q and returns as a result each concept in the subclass list that satisfies the distance threshold. A similar approach is followed in the case of subsume matches where the superclasses of q are retrieved. Finally, in the case of sibling matches, the superclasses of q are retrieved, and for each of them, its subclasses are retrieved. The goal is to determine concepts in the subclass lists which satisfy the distance threshold.

Algorithm: Finds the similar concepts to the query concept q according to the hierarchical relationship H and the distance threshold D.
Input: Query concept q, Hierarchical relationship H, Distance threshold D.
Output: A list with the similar concepts to q.

```
function similarConcepts(q, H, D) {
   var result = ∅
1    if H = exact then
2       for each concept C | C ≡ q do
3          result ← add(C)
4    else if H = plugin then
5       subclasses ← listSubclasses(q)
6       for each C ∈ subclasses | dist(q, C) ≤ D do
7          result ← add(C)
8     else if H = subsume then
9       superclasses ← listSuperclasses(q)
10      for each C ∈ superclasses | dist(q, C) ≤ D do
11         result ← add(C)
12   else if H = sibling then
13      superclasses ← listSuperclasses (q)
14      for each T ∈ superclasses do
15         subclasses ← listSubclasses (T) - listSubclasses(q)
16         for each C′ ∈ subclasses do
17            if dist(q, T) + dist(T, C′ ) -1 ≤ D then
18            result ← add(C′ )
19   return result  }
```

4 Problem Representation and Solving

The problem of composing simple web services in order to come up with a complex one that fulfills the user's needs can be easily transformed into a planning problem and solved using a domain independent planning system. A planning problem is usually modelled according to STRIPS (Stanford Research Institute Planning System) notation [9]. A planning problem in STRIPS is a tuple <I,A,G> where I is the Initial state, A is a set of available actions and G is a set of goals. States in STRIPS are represented as sets of atomic facts.

Set A contains all the actions that can be used to modify states. Each action A_i has three lists of facts containing the preconditions of A_i (noted as prec(A_i)), the facts that are added to the state (noted as add(A_i)) and the facts that are deleted from the state (noted as del(A_i)).

The following formulae hold for the states in the STRIPS notation:

- An action A_i is applicable to a state S if $prec(A_i) \subseteq S$.
- If A_i is applied to S, the successor state S' is calculated as $S' = S \setminus del(A_i) \cup add(A_i)$
- The solution to a planning problem is a sequence of actions, which, if applied to I, lead to a state S' such that $S' \supseteq G$.

The representation of a WS composition problem in planning terms requires simple WS to be viewed as actions, and complex WS to be viewed as plans. More details of the representation in the proposed system will be discussed in the following sections.

4.1 Problem Representation

Consider the case were a user wishes to use a complex web service which, when provided with some input data, will return some required information. There may be a number of alternatives when formalizing the problem of web service composition as a planning problem. A straightforward solution adopted by PORSCE is the following: The inputs provided by the user form the initial state of the problem, while the desired outputs form the goals of the problem. The available OWL-S profiles are used to obtain the actions available in the planning domain. For each action the following statements hold: a) its name is the same as the name of the corresponding web service, b) its preconditions list is formed by the inputs of the service, c) the add effects of the actions are the outputs of the service and d) the delete list is left empty.

The formalization presented above requires the planning system to be aware of possible semantic similarities among syntactically different concepts. This situation can be dealt with in two ways. The first solution is to alter the planning system in order to constantly advise the OWL Ontology Manager (OOM), whenever it is required, such as to determine the applicability of an action in a given state. The second solution is to enhance the problem description given above with all the required semantic information in a pre-processing phase.

In order to maintain the independency of the planner from the rest of the PORSCE system, the second solution was adopted. Therefore, the planning module can be replaced by any planning system compliant with PDDL [10] input files. In the pre-processing phase, the system uses the OOM in order to return all the semantically similar concepts for both the facts of the initial state and the preconditions of the available actions. The original concepts of the initial state together with the semantic equivalent and similar concepts form a new set of facts noted as the Expanded Initial State (EIS) (note that the term *state* is used improperly). Moreover, for each action the pre-processor creates all the possible combinations of the original preconditions and their semantically equivalents in order to form the Extended Action Set (EAS). Suppose, for example, that the initial state I of the problem is the following

```
I = {Sightseeing, Dates}
```

There are two available actions:

```
CityHotelMapService: prec={City, LuxuryHotel}
SightSeeingAreaService: prec={Sightseeing}
```

The OOM for a given threshold discovers the following similarities:

```
Dates≡Duration, Hotel≡Motel, Hotel≡LuxuryHotel
```

The pre-processor alters the problem definition to the following:

```
EIS = {Sightseeing, Dates, Duration}
EAS:   CityHotelMapService: prec={City, LuxuryHotel}
       CityHotelMapService1: prec={City, Hotel}
       CityHotelMapService2: prec={City, Motel}
       SightSeeingAreaService: prec={Sightseeing}
```

The new problem, namely <EIS,EAS,G> is then encoded into PDDL and is forwarded to the planning system in order to acquire a solution.

4.2 The Planning System

The planner module incorporated in the system is JPlan [1], an open-source implementation of Graphplan in Java. Graphplan [2] is a general-purpose planner for STRIPS-like domains, which exploits the benefits of graph algorithms in order to reduce search space and provide better solutions.

JPlan proved to be remarkably fast and can handle a respectable number of operators, unlike other implementations of Graphplan. Currently, it has been tested for more than 2000 operators in a single planning domain. This is very important as the number of operators increases significantly when equivalent or similar concepts are taken into account. After the planning process is completed, JPlan provides not only the plan, if found, but also the mutual exclusion description of the leveled graph.

JPlan supports predicates with an arbitrary number of arguments, but not predicates without arguments, a disadvantage which had to be overcome in order to be adopted for the web service case by adding a "dummy" argument, as web service inputs and outputs are usually represented as predicates without arguments. Another technical issue that had to be dealt with is the fact that JPlan does not support PDDL, but text files with a similar structure. Finally, JPlan does not seem to offer a way to supply alternative plans, if there are any, as it expands the graph to a predefined level and does only one search through it. Despite its disadvantages, JPlan proved to be efficient and serve the purposes of testing the system.

5 Experimental Results

In this section, the experiments performed and a specific example will be presented in order to illuminate the aspects of this approach. The test sets used to perform experiments were obtained from the OWLS-TC version 2.2 revision 1 [8] and included SWS descriptions in OWL-S from various domains such as travel, food and economy, and the corresponding ontologies. Initially, attempts were made to obtain all possible plans that could be produced from each domain, using only one concept of the domain as input and another concept as output, in order to familiarize with the domains and detect the flow of information among the various services. The results included only plans of length 1, namely the simple web services that had these exact concepts as input and output, and some plans of length 2. Unfortunately, the nature of these domains did not permit the creation of longer plans, and therefore more complex web services. Using more relaxed restrictions on the similarity of concepts did not alter the results. Using more than one concept in the inputs and outputs sets increased the length of the produced plans, however, this case was not examined further because

the produced plans represented more likely collections of independent web services, rather than in fact complex web services.

In order to test the abilities of the system in managing simple web services that can be composed into longer plans, therefore more complex web services, some of the service descriptions were modified. The scenario that was chosen to implement referred to the travel domain, and included a user who wants to go sightseeing at some specific dates. The user requires to know the price of the hotel, place a reservation and be presented with a map of the area. As there is no simple web service that has such functionality, but there are web services that provide this functionality partially, the problem must be solved through composition.

In order to demonstrate the dependence of the solution on the distance between concepts in the ontology we permit, we added to the domain the following service descriptions:

1. SightseeingAreaService: A service that accepts as input the activity Sightseeing and returns areas of a city that offer this activity.
2. DatesToDurationService: A service that accepts as input dates and supplies the duration of the time period specified by these dates.
3. AreaCityService: A service that accepts as input areas of a city and returns the city they belong to.
4. CityLuxuryHotelService: A service that accepts as input a city name and returns luxury hotels in this city. Luxury hotel is a subclass of Hotel.
5. CityHotelMapService: A service that accepts as inputs a city and a hotel name, and presents a map of the area of the city the hotel resides.
6. HotelPriceInfoService: A service that accepts as input a hotel name and provides information about its prices.
7. HotelReserveService: A service that accepts as inputs a city and a hotel name, as well as a duration, and places a reservation at this hotel for the specified dates.

For the sake of the example, we assume that there are no other web services in the domain that interfere with these in any way. However, the descriptions of the rest domain services are still parsed and turned into operators, in order to maintain the size of the domain in realistic levels, and therefore obtain plausible time measurements.

When no equivalent concepts are allowed in the planning process, is it obvious that these web services will not be composed to form a complex web service with the desired results, because the CityLuxuryHotelService yields an object of the class LuxuryHotel, while all other services accept objects of the class Hotel. While LuxuryHotel is a subclass of Hotel, a planner without the help of a reasoner perceives them as two totally separate concepts.

This restriction can be relaxed by taking into account concepts with distance 1 from the concept at hand. In that case, planning can match concepts with other concepts which are direct subclasses and superclasses in some specified ontology. Thus, the result set is expanded and a plan involving all 7 web services presented above is produced. (Fig. 3).

If the restrictions are relaxed even more, by increasing the distance to 2 or more, concepts can be matched with other concepts that are even further, in ontology terms. This could potentially provide more plans, and in several cases shorter plans,

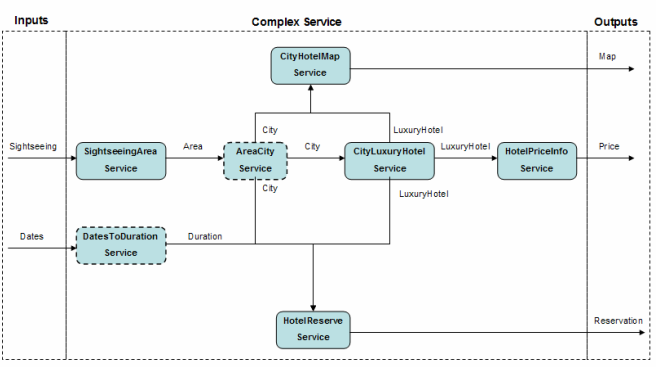

Fig. 3. A Plan of 7 steps for the Travel domain

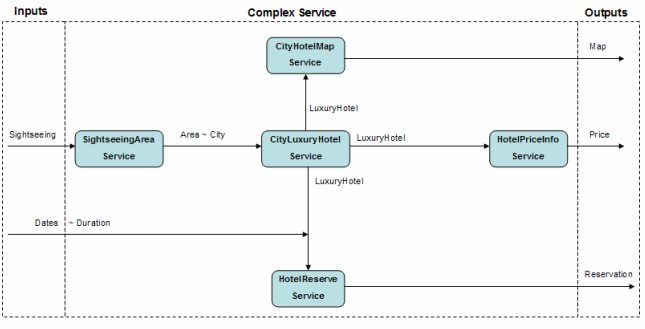

Fig. 4. A Plan of 5 steps for the Travel domain

as Fig. 4 depicts. However, the more relaxed the restrictions are, the less accurate the results become, and the complex WS provided might not fulfill the prospects of the user.

Experiments were run on a machine with an Inter Core2 CPU at 1.66GHz, with 2 GB or RAM memory. In all cases, the preprocessing step took approximately 14 seconds to parse and process 23 ontologies. However, this is a step that has to be performed only once, as long as the ontologies remain unchanged. In the first case, when the acceptable distance between concepts that can be considered equivalent is 1, the step where service descriptions are parsed and operators are generated for each service lasted approximately 47 seconds. This time includes the invocation of the object ranker module to provide equivalent and similar concepts, and the generation of additional operators for each combination of these concepts. In the travel domain, 150 web service descriptions were parsed, while the number of operators that were produced eventually was 690. In the case of distance 2, which is still considered a close relation, this time increased to approximately 97 seconds, while the number of operators was 2010. Finally, planning time was insignificant compared to parsing time, as in both cases the planner produced the desired result in less than a second.

6 Related Work

Other efforts that attempt to exploit the benefits of planning techniques to tackle the problem of automatic web service composition will be presented in this section.

One of the first systems that attempted automatic web service composition was SHOP-2 [3]. The system uses services descriptions in DAML-S, the predecessor of OWL-S, and performs HTN planning to solve the problem. The disadvantage of this approach lies in the fact that the planning process, due to its hierarchical nature, requires given decomposition rules, or *methods*, as they are referred to, which have to be encoded in advance with the help of a DAML-S process ontology.

OWLS-Xplan [4] uses semantic descriptions of web services in OWL-S to derive planning domains and problems, and then invokes a planning module called Xplan to generate the complex services. The system is PDDL compliant, as the authors have developed an XML dialect of PDDL called PDDXML. However, semantic information provided from domain ontologies is not utilized, therefore the planning module requires exact matching for service inputs and outputs.

Other approaches that use knowledge-based planning include the system described in [5] which composes web services with the PKS planning system. However, this effort does not deal with the important issue of translating semantic web service descriptions into planning terms. The work in [6] also uses knowledge-level planning to approach the automated web service composition problem. The web service descriptions in this case have to be expressed in some standard process modeling and execution language, such as BPEL4WS, therefore some prior, domain-specific knowledge of the composition issues is required.

The advantages of the proposed framework lie in the fact that the OWL-S descriptions of the web services and the corresponding ontologies are adequate information for the system to determine how to form valid complex services that satisfy given goals. Even in the case that no exact match can be found, the system is still able to find a complex service that approximates best the desired goal. No prior or additional knowledge is demanded since the ontologies capture the semantics of the concepts used, while the trade-off between the quantity and quality of the results, i.e. between the number of complex services produced and their accuracy in achieving the given goals, is up to the user to decide, by selecting desirable concept distances.

7 Conclusions and Future Work

The work presented in this paper concerns the development of a prototype system that combines planning with object ranking in order to approach the semantic web service composition problem. Each web service composition problem is mapped into a planning problem by representing simple web services as operators, inputs as the initial state of the planning problem and outputs as the goal state. Such representations are derived from the OWL-S descriptions of the web services. However, before the planning problem is fed into to planning module in order to obtain a plan, which will represent the description of the desired complex web service, the object ranker module is utilized. The object ranker exploits knowledge contributed by domain ontologies, and returns semantically equivalent or similar concepts, which are in turn used to form an

extended initial state and set of actions. As a result, the requirement for exact matching of the service inputs and outputs is eliminated, and the planning procedure can be performed with the desired degree of semantic relaxation. The system was implemented and tested with different web service domains, and experimental results were presented.

Future goals include the extension of the system in order to cooperate with different planners which are capable of providing alternative plans. Moreover, the possibility of experimenting with different metrics for semantic similarity, other than distance, should be explored, and their effect on the planning procedure and the produced plans should be examined.

References

1. JPlan: Java Graphplan Implementation, http://sourceforge.net/projects/jplan
2. Blum, A.L., Furst, M.L.: Fast Planning Through Planning Graph Analysis. Artificial Intelligence 90, 281–300 (1997)
3. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2004)
4. Klusch, M., Gerber, A., Schmidt, M.: Semantic Web Service Composition Planning with OWLS-XPlan. In: AAAI Fall Symposium on Semantic Web and Agents, USA (2005)
5. Martinez, E., Lesperance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: Proceedings of the ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services, pp. 62–69 (2004)
6. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated Composition of Web Services by Planning at the Knowledge Level. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK (August 2005)
7. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL DL Reasoner. J. Web Semantics (2007)
8. OWLS-TC version 2.2 revision 1, http://projects.semwebcentral.org/projects/owls-tc/
9. Fikes, R., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
10. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL – the Planning Domain Definition Language. Technical report, Yale University, New Haven, CT (1998)
11. OWL-S 1.1 Release, http://www.daml.org/services/owl-s/1.1/
12. WSDL-S, http://www.w3.org/Submission/WSDL-S/
13. Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. In: First International Semantic Web Conference (2002)
14. Semantic Annotations for WSDL, http://www.w3.org/2002/ws/sawsdl/
15. Booth, D., et al.: Web Services Architecture. W3C Working Draft (August 2003), http://www.w3.org/TR/ws-arch/
16. The OWL API, http://owlapi.sourceforge.net/