The PORSCE II Framework: Using AI Planning for Automated Semantic Web Service Composition

OURANIA HATZI[1], DIMITRIS VRAKAS[2], NICK BASSILIADES[2], DIMOSTHENIS ANAGNOSTOPOULOS[1], 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 PORSCE II, an integrated system that performs automatic semantic web service composition exploiting AI techniques, mainly planning. Essential steps in achieving the web service composition include the translation of the web service composition problem into a solver-ready planning domain & problem, followed by the acquisition of solutions, and the translation of the solutions back to web service terms. The solutions to the problem, that is, the descriptions of the desired composite service, are obtained by means of external domain-independent planning systems, they are visualized and finally evaluated. Throughout the entire process, the system exploits semantic information extracted from the semantic descriptions of the available atomic web services and the corresponding ontologies, in order to perform composition under semantic awareness and relaxation.

1. Introduction

The World Wide Web has evolved from a collection of documents into a more integrated environment, where not only information but also system functionality is exposed and the provision of services plays an important role. The web services technology is a fundamental part of the web, as it provides a standard way to interact with information systems, independent from platform and internal implementation, thus accommodating interoperability between heterogeneous systems. However, in many cases, the need for integrated functionality cannot be fulfilled by a simple atomic web service, leading to the requirement for *web service composition*; that is, the appropriate combination of atomic web services in order to achieve a complex goal. The task of web service composition becomes significantly difficult, time-consuming and inefficient as the number of available atomic services increases continuously; therefore, the possibility to automate the web service composition process is proved essential.

Automated web service composition is significantly facilitated by the development of the Semantic Web, which permits the representation of knowledge about the actual meaning of information and services. The existence of such semantic information enables composition using intelligent techniques, such as AI Planning. Without the presence of semantic information, a high degree of human expertise would be required in order to compose web services meaningfully and not based on circumstantial syntactic similarities. The incorporation of semantics in the description of web services is accommodated through the development of a number of standard languages such as OWL-S [6] and SAWSDL [11].

The PORSCE II framework aims at automated semantic web service composition by utilizing planning under semantic awareness and relaxation. The first and decisive step in this process concerns the translation of the web service composition problem to AI planning terms [20]. This translation takes place between the most prominent standards in each area: OWL-S for semantic description of web services and PDDL (Planning Domain Definition Language) [5] for definition of planning domain and problem. According to user preferences, the translation process may take into account semantics, resulting from the semantic analysis of the domain and the corresponding ontologies; if so, semantically equivalent or relevant concepts are also included, in order to cope with cases when no exact plans can be found and approximations must take place. The result of this phase of the transformation process is a fully formulated, solver-ready planning problem which incorporates all the required semantic information. PORSCE II consequently exports the planning problem to PDDL and invokes external planning systems to acquire plans, which constitute descriptions of the desired composite service. Each composite service is evaluated in terms of statistic and accuracy measures, while a visual component is also integrated, which accommodates composite service visualization and manipulation. Modification in the composite service is performed by atomic service replacement, either with an alternative equivalent atomic service, or through finding a sub-plan that can substitute it. If necessary, the composite service can also be modified through replanning.

Finally, in order to provide full-cycle support, and render the result of the composition process independent from planning, the composite service is translated back to OWL-S, presenting the user with a description in the same standard as the initial atomic services and facilitating composite service deployment.

The rest of the paper is organized as follows: Section 2 discusses some related work in the area of web service composition through planning, while Section 3 provides some background related to the OWL-S standard, AI planning and the PDDL standard. Section 4 outlines the system architecture, Section 5 elaborates on the main knowledge engineering aspect that this paper focuses on, that is, the transformation process, and Section 6 presents the rest of the system operations. Section 7 presents a case study and performance evaluation results and finally, Section 8 concludes the paper and poses future directions.

2. Related Work

One of the first systems that attempted automatic web service composition through AI Planning is SHOP-2 [13]. The system uses services descriptions in DAML-S, the predecessor of OWL-S, and performs Hierarchical Task Network (HTN) planning to solve the problem. The main disadvantage of this approach lies in the fact that the planning process, due to its hierarchical nature, requires the specification of certain decomposition rules, which have to be encoded in advance by an expert in the specific domain, with the help of a DAML-S process ontology.

OWLS-Xplan [14] uses the semantic descriptions of atomic web services in OWL-S to derive planning domains and problems, and invokes a planning module called Xplan to generate the composite services. The system is PDDL compliant, as the authors have developed an XML dialect of PDDL called PDDXML. Although the system imports semantic descriptions, the semantic information provided from domain ontologies is not utilized and semantic awareness is not achieved; therefore the planning module requires exact matching for service inputs and outputs.

Other approaches for automatic web service composition can be found at [15][16][17]; however, they are not further discussed here as they do not deal with the important issue of transforming knowledge about semantic web services into planning terms in order to perform composition, which is the main focus of this paper.

The main advantage of the proposed framework with respect to the aforementioned systems is the extended utilization of semantic information, in order to perform planning under semantic awareness and relaxation, and find better and, when necessary, approximate solutions. Furthermore, PORSCE II does not require any prior, domain-specific knowledge to form valid, desired composite services; the OWL-S descriptions of the atomic web services and the corresponding ontologies suffice. Finally, the system is able to handle cases of service failure or unavailability dynamically, an important feature not covered in the aforementioned frameworks.

3. Background

This section serves as an introduction to OWL-S and PDDL, as they are the fundamental standards for the approach presented in this paper. Additionally, it introduces some basic planning notation that will be used throughout the paper.

3.1. OWL-S

OWL-S is an upper ontology based on OWL [12], created in the context of the Semantic Web in order to describe knowledge concerning semantic web services. It is used in combination with additional ontologies, which organize the concepts appearing in the OWL-S descriptions. The use of OWL-S renders the semantics of the descriptions machine comprehensible; therefore it enables intelligent agents to discover, invoke and compose web services automatically.
A web service description in OWL-S is comprised of three parts [6]:

- *Service Profile:* describes what the service accomplishes, limitations on service applicability and quality, and requirements that the service requester must satisfy to use the service.
- *Process Model:* describes the way a client can communicate and use the service.
- *Service Grounding:* specifies the details of how an agent can access a service, such as a communication protocols and message formats.

The approach presented here utilizes the semantic information contained in the *Service Profile* of a specific web service, along with the corresponding ontologies, in order to translate the description in planning terms. An example of an OWL-S Profile and the correspondences between its elements of interest and the planning terms they are translated into is provided in Section 5.1.

Apart from atomic web services, which involve atomic processes, OWL-S establishes a framework for defining composite processes as well. A composite process consists of a set of atomic processes, combined together using a number of control constructs, such as Sequence, Split, Split+Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until. The main reasons for using these constructs while defining a composite web service are: a) to enable the definition of compact services (e.g. through the use of Iterate, Repeat-While and Repeat-Until), b) to facilitate the definition of alternative paths (i. e. through the use of Conditions and If-Then-Else constructs) and c) to speed up the invocation of the composite web service, by allowing multiple atomic processes to be invoked concurrently (i.e. through the use of Split and Split+Join constructs).

3.2. Planning & PDDL

A planning domain and problem is usually modeled according to STRIPS (Stanford Research Institute Planning System) notation [4] as 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, the facts that are added, and the facts that are deleted from the world state after the application of the action, noted as $prec(A_i)$, $add(A_i)$ and $del(A_i)$ respectively.

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 $S' = S - del(A_i) \cup add(A_i)$
- The solution to a planning problem (plan P) is a sequence of actions $P=\{A_1, A_2, ..., A_n\}$, which, if applied to I, lead to a state S’ such that $S' \supseteq G$.

Planning Domain Definition Language (PDDL) [5] was initially designed for providing a standard means of encoding planning domains and corresponding problems used as input test sets for planners that took part in planning competitions such as IPC [18]. However, it has since been enhanced, extended and become a standard in the planning community for modeling planning domains and problems.

PDDL provides structures to represent all the aforementioned STRIPS elements, such as predicates (atomic facts), actions and problems. Newer versions of the language added more features in order to enable the representation of more complex domains. These features include constants, variables, functions, and numeric expressions. PDDL also provides separate structures that can be used to represent problems, which are associated with specific planning domains. The latest extensions to the PDDL standard take into the temporal properties of domains, as well as quality metrics, features which might prove very useful in the web service composition case. A newer version of PDDL, PDDL+, also provides a standard way to represent plans, either sequential or partially parallel.

4. Overview and Architecture

PORSCE II is the evolution of the prototype system PORSCE [7]. The core transformation component exists in both systems; however, PORSCE II aims at a higher degree of integration as it additionally contains a visual interface, more elaborate relevance metrics, the ability for composite service accuracy assessment, and composite web services manipulation features. Furthermore, PORSCE II adopts a different way of modeling the web service composition as a planning problem, which reduces the complexity of the planning problem, thus accelerating the planning process. In order to highlight the planner independency of PORSCE II, which enables the use of any domain independent planning system based on PDDL, another external planner has been included in addition to the original one. Finally, PORSCE II also supports seamless composition, by initiating the process with the OWL-S descriptions of atomic web services, and concluding with the OWL-S description of the produced complex service, thus facilitating deployment.

The key features of the framework are:

- Translation of OWL-S atomic web service descriptions into planning operators.
- Interaction with the user in order to acquire their preferences regarding the composite service and desired metrics for semantic relaxation.
- Enhancing the planning domain and problem with semantically similar concepts.
- Exporting the web service composition problem as a PDDL planning domain and problem.
- Acquisition of solutions by invoking external planners.
- Assessing the accuracy of the composite services.
- Visualizing and modifying the solution by atomic service replacement of replanning.
- Transformation of the solution (composite web service) back to OWL-S.

PORSCE II comprises of the OWL-S Parser, the Transformation Component, the OWL Ontology Manager (OOM), the Visualizer and the Service Replacement Component. An overview of the architecture and the interactions among the components is depicted in Fig. 1.

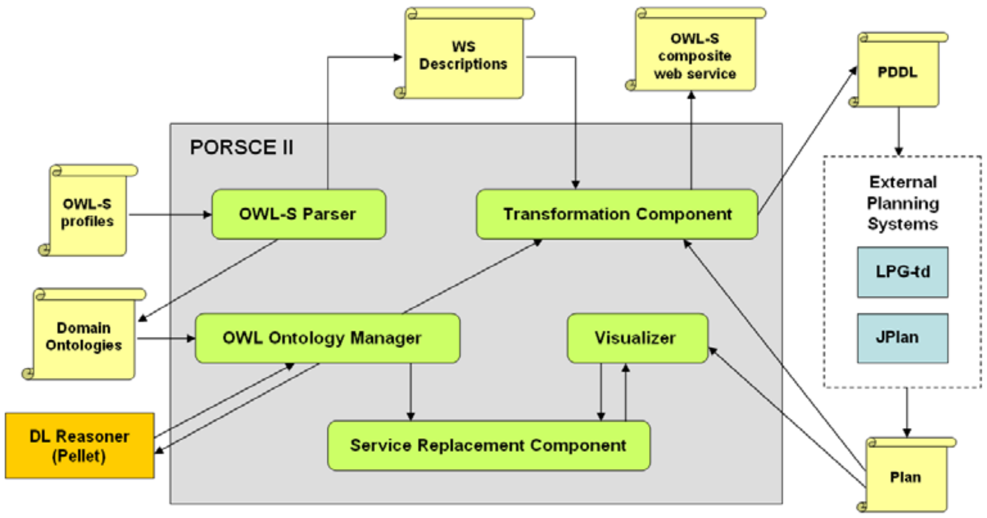


Fig. 1 The architecture of PORSCE II.

The OWL-S Parser is responsible for parsing a set of OWL-S web service profiles and determining the corresponding ontologies that the concepts appearing in the web service descriptions belong to. The OWL Ontology Manager (OOM), utilizing the inferencing capabilities of the Pellet DL Reasoner [2], applies the selected algorithm for discovering concepts that are similar to a query concept. The Transformation Component is responsible for a number of operations that result in the formulation of the planning problem from the initial web service composition problem, its consequent solving, and the transformation of the produced composite service back to OWL-S. The purpose of the Visualizer is to provide the user with a visual representation of the plan, which in fact is the description of the composite service. Finally, the Service Replacement Component enables the user to employ a number of alternative techniques in order to replace a specific atomic web service in the composite service sequence. Details on the system functionality regarding the translation process and the rest of its operations are provided in Sections 5 and 6 respectively. PORSCE II is implemented in Java and it is available online, along with example problems, at http://www.dit.hua.gr/~raniah/porsceII_en.html.

5. Transformation Process

The transformation process includes the translation of the web service composition problem into a planning problem and its possible enhancement with semantic information, as well as the transformation of the plan representing the produced composite service back in web service standards. The process starts at the OWL-S Parser, which parses the OWL-S profiles of the available atomic web services and forwards them to the Transformation Component. The Transformation Component is responsible for a number of operations, including translating the web service descriptions received from the OWL-S Parser to planning operators and enhancing them with similar concepts derived from the OOM. Moreover, it interacts with the user in order to formulate the planning problem, and exports both the planning domain and problem to PDDL. Finally, it translates the PDDL+ plan back to OWL-S, completing the composition process.

5.1. OWL-S to PDDL Translation

A straightforward solution adopted by PORSCE II for mapping the web service composition problem to a planning problem is the following (using the notation introduced in Section 3): Let *IC* be the set of concepts that the user wishes to provide to the composite service and *GC* its desired outputs (goals). If *O* denotes the set of all the available concepts in the ontology, then $IC \subseteq O$, $GC \subseteq O$ and $IC \cap GC \equiv \varnothing$. The inputs that the user wishes to provide formulate the initial state, while the desired outputs of the composite service formulate the goals of the problem: I = *IC* and G=*GC*. Both the input and output sets are provided externally by the user.

The available OWL-S web service profiles are used in order to obtain the planning operators: each web service description WSD_i is translated to an operator A_i, using the information in the each profile (Fig. 2).

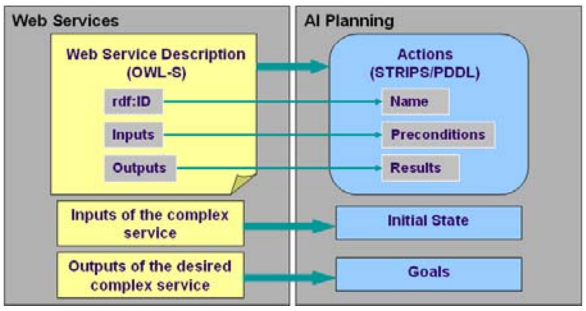


Fig. 2. Mapping web services to planning.

More specifically:

- The name of the action is the rdf:ID of the profile:

$$\mathrm{name}\left(\mathrm{A_i}\right) = WSD_i.\mathrm{ID}$$

- The preconditions of the action are formed based on the service's input definitions (concepts):

$$\mathrm{prec}\left(\mathrm{A_i}\right) \equiv \bigcup_{k=1}^{n} \{\mathrm{WSD_i.hasInput_k}\}$$

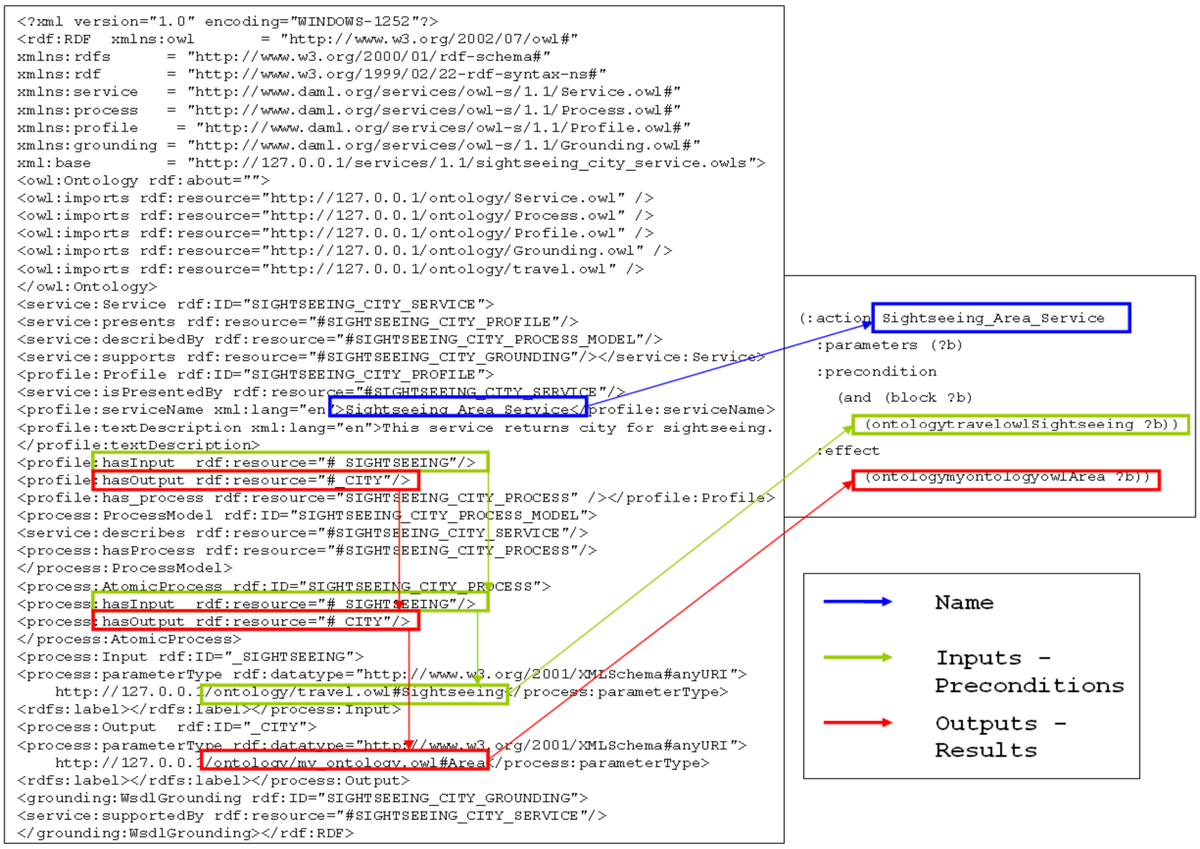
- The add effects of the action comprises of the service's output definitions (concepts):

$$\mathrm{add}\left(\mathrm{A_i}\right) \equiv \bigcup_{k=1}^{m} \{\mathrm{WSD_i.hasOutput_k}\}$$

- The delete list is left empty, since in the current study only services that do not have any negative effects in the world model are dealt with:

$$\mathrm{del}\left(\mathrm{A_i}\right) \equiv \varnothing$$

An example of an OWL-S to PDDL transformation is presented in Fig. 3, where the mapping presented above is marked. The web service description at hand concerns a web service that accepts as input the activity Sightseeing and presents the user with areas that offer this activity.


```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF  xmlns:owl       = "http://www.w3.org/2002/07/owl#"
xmlns:rdfs      = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf       = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:service   = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
xmlns:process   = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmlns:profile    = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
xml:base        = "http://127.0.0.1/services/1.1/sightseeing_city_service.owls">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/Grounding.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/travel.owl" />
</owl:Ontology>
<service:Service rdf:ID="SIGHTSEEING_CITY_SERVICE">
<service:presents rdf:resource="#SIGHTSEEING_CITY_PROFILE"/>
<service:describedBy rdf:resource="#SIGHTSEEING_CITY_PROCESS_MODEL"/>
<service:supports rdf:resource="#SIGHTSEEING_CITY_GROUNDING"/></service:Service>
<profile:Profile rdf:ID="SIGHTSEEING_CITY_PROFILE">
<service:isPresentedBy rdf:resource="#SIGHTSEEING_CITY_SERVICE"/>
<profile:serviceName xml:lang="en">Sightseeing_Area_Service</profile:serviceName>
<profile:textDescription xml:lang="en">This service returns city for sightseeing.
</profile:textDescription>
<profile:hasInput  rdf:resource="#_SIGHTSEEING"/>
<profile:hasOutput rdf:resource="#_CITY"/>
<profile:has_process rdf:resource="SIGHTSEEING_CITY_PROCESS" /></profile:Profile>
<process:ProcessModel rdf:ID="SIGHTSEEING_CITY_PROCESS_MODEL">
<service:describes rdf:resource="#SIGHTSEEING_CITY_SERVICE"/>
<process:hasProcess rdf:resource="#SIGHTSEEING_CITY_PROCESS"/>
</process:ProcessModel>
<process:AtomicProcess rdf:ID="SIGHTSEEING_CITY_PROCESS">
<process:hasInput  rdf:resource="#_SIGHTSEEING"/>
<process:hasOutput rdf:resource="#_CITY"/>
</process:AtomicProcess>
<process:Input rdf:ID="_SIGHTSEEING">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
    http://127.0.0.1/ontology/travel.owl#Sightseeing</process:parameterType>
<rdfs:label></rdfs:label></process:Input>
<process:Output  rdf:ID="_CITY">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
    http://127.0.0.1/ontology/my_ontology.owl#Area</process:parameterType>
<rdfs:label></rdfs:label></process:Output>
<grounding:WsdlGrounding rdf:ID="SIGHTSEEING_CITY_GROUNDING">
<service:supportedBy rdf:resource="#SIGHTSEEING_CITY_SERVICE"/>
</grounding:WsdlGrounding></rdf:RDF>
```

```
(:action Sightseeing_Area_Service
  :parameters (?b)
  :precondition
    (and (block ?b)
      (ontologytravelowlSightseeing ?b))
  :effect
      (ontologymyontologyowlArea ?b))
```

Fig. 3. Example of an OWL-S to PDDL translation.

5.2. Semantic Analysis

The step of semantic analysis, parallel to the transformation process, enables the system to translate the web service composition problem to a planning problem by taking into account semantic information. This step is implemented by the OWL Ontology Manager (OOM). During translation, the OOM is used extensively for performing semantic relaxation, which is useful in cases when an exact input/output matching plan is not available. The OOM locates equivalent and semantically relevant concepts; therefore, approximate plans can be created.

In our approach, two ontology concepts are considered semantically similar if and only if

- they have a *hierarchical relationship*
- their *semantic distance* does not exceed a threshold

As far as the *hierarchical relationship* is concerned, four hierarchical filters are used for its definition for two ontology concepts *A* and *B*:

- *exact*(*A*, *B*): The two concepts should have the same URI or they should be equivalent, in terms of OWL class equivalence, i.e. $A = B \vee A \equiv B$.
- *plugin*(*A*, *B*): The concept *A* should be subsumed by the concept *B*, i.e. $A \sqsubseteq B$.
- *subsume*(*A*, *B*): The concept *A* should subsume the concept *B*, i.e. $B \sqsubseteq A$. In both the *plugin* and the *subsume* filters the subsumption relationships of equivalent concepts are not considered.
- *sibling*(*A*, *B*): The two concepts should neither have a hierarchical relationship, nor be disjoint; instead, they should have a common superclass *T*, such as $A \sqsubseteq T \wedge B \sqsubseteq T$.

The *semantic distance* between two ontology concepts can be calculated in PORSCE II using two methods:

The *Edge-Counting Distance (ec)* computes the distance of two concepts in terms of the number of edges found on the shortest path between them. An edge exists between two concepts *A* and *B* if *A* is the direct subclass of *B*, denoted as $A \sqsubseteq_d B$. The implementation of the *ec* distance between two concepts, denoted as $d_{ec}(A, B)$, returns a value between 0 and 1, with 1 denoting absolute mismatch. It is normalized to [0..1] as $p/p_{\max}$, using the maximum *ec* distance (p_{max}) found in the ontology, which can be approximated as $p_{max} = 2h - 1$, where *h* is the maximum edge distance from a leaf concept to the owl:Thing concept ($\top$).

The *Upwards Cotopic Distance*, denoted as $d_{uc}(A, B)$, is defined in terms of the upwards cotopic measure, denoted as *uc*(*A*) that represents the set of the superclasses of the concept *A*, including *A* itself [10]. In PORSCE II, the upwards cotopic distance definition has been modified in order to incorporate the semantics of an ontology hierarchy. More specifically, the owl:Thing concept is not considered in the *uc* measure, while the union and intersection set operators take into account the concept equivalence semantics. In that way concept set multiplicity is ignored, that is, if $A \equiv B$, then $\{A, B, C\} = \{A, C\} \vee \{B, C\}$, and the concept set membership is semantically checked, that is, if $A \equiv B$ and $D = \{A\}$, then $A \in D \wedge B \in D$. Based on these remarks, the upwards cotopic distance is defined as

$$d_{uc}(A,B) = 1 - \frac{|uc(A) \cap uc(B)| - 1}{|uc(A) \cup uc(B)| - 1}.$$

If two concepts have only the owl:Thing class as the common superclass or they are disjoint, then their distance equals to 1; otherwise, if the two concepts have a hierarchical relationship, then $d_{uc}(A, B) \in [0..1)$.

5.3. Semantic Awareness and Relaxation

The representation of the web service composition as a planning problem is empowered if the planning system is aware of semantic similarities among syntactically different concepts (semantic awareness). The solution adopted by PORSCE II involves enhancing the domain and problem description with all the required semantic information in a pre-processing phase and letting the planner handle it as a classical planning problem. This solution is employed in order to: a) be able to use any PDDL compliant planner, as the semantic enhancement applied to the domain remains transparent to the planner, and b) minimize the interactions between the planner and the OOM, which introduce an overhead on the planning time.

In the pre-processing phase, the system uses the OOM in order to acquire all the semantically relevant concepts for both the facts of the initial state and the outputs of the operators, discovered by the semantic analysis process described in the previous section. The enhancement of the problem by PORSCE II is based on the following rules:

- The original concepts of the initial state together with the semantically equivalent and similar concepts form a new set of facts noted as the Expanded Initial State (EIS).
- The goals of the problem remain the same.
- The Enhanced Operator Set (EOS) is produced, by altering the description of each operator, while preserving the initial size of the set. More specifically, the effects list of each operator is enhanced by including all the equivalent and semantically similar concepts for the concepts in the initial effects list.

Suppose, for example, that the initial state I of the problem is the following:

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

and that there are only the following two operators:

```
CityHotelMapService: prec={City, Hotel}, effect={Map}
SightSeeingAreaService: prec={Sightseeing}, effect={Area}
```

The OOM for a given distance metric and threshold discovers the following relevant concepts:

```
Dates ≈ Duration, Area ≈ County, Map ≈ GPSRoute
```

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

```
EIS: {Sightseeing, Dates, Duration}
EOS: CityHotelMapService: prec={City, Hotel}, effect={Map, GPSRoute}
     SightSeeingAreaService: prec={Sightseeing}, effect={Area, Country}
```

The new problem, namely <EIS,EOS,G> is encoded into PDDL and forwarded to the planning system in order to acquire a solution. Note that the semantic information is encoded in such a way that it is transparent to the external planners, which can solve the problem as any other classical planning problem.

5.4. PDDL to OWL-S Translation

By enabling the representation of composite processes, OWL-S facilitates the translation of the results, that is, the produced composite services, back to the initial web services domain. The translation accommodates composite service deployment and execution monitoring.

For the purposes of the PORSCE II framework, the use of the intricate control constructs described in Section 3.1 is not mandatory, as far as the proper invocation of the atomic processes is concerned. Since the modeling of the web service composition problem to a planning problem is merely based on the STRIPS formalism, there is no need for defining alternative paths. Moreover, all the plans produced by the planning systems contain a finite number of steps (i.e. atomic processes) and the use of loops (e.g. iterate) is rare and not mandatory. Therefore, any plan produced by the framework can be expressed as a composite web service by using only the Sequence control construct, without risking the proper invocation of the composite service. This is true even for the cases where the external planning system used, such as LPG-td, returns a non linear plan (i.e. one that contains steps with parallel execution of actions), since any non linear plan has one or more equivalent topological orderings. However, in order to accelerate the invocation of the composite service by allowing the parallel execution of certain atomic processes, an algorithm that translates plans (linear or non linear) to composite web services using the Sequence, Split and Split+Join constructs has been developed.

Algorithm 1 (Basic) Computes an initial composite service with *Sequence* and *Split* constructs
Inputs: G=(V,E), the web service graph
Output: C: a composite service

```
1: set $R \leftarrow \{r \in V: \forall x \in V, (x \to r) \notin E\}$ // R is the set of root nodes in G
2: if $|R| = 0$ then return NULL
3: if $|R| = 1$ then
       set $G' \leftarrow$ the tree in G with $r \in R$ as the root
       return sequence(r, Basic(G'-{r}))
4: set $c \leftarrow \{\}$
5: for each r in R
       set $G' \leftarrow$ the tree in G with $r \in R$ as the root
       set $c \leftarrow c \cup Basic(G'-\{r\})$
6: return split(c)
```

Algorithm 1 presents the basic algorithm that creates a composite service, given a web service graph. A web service graph is a graph *G*=(*V*, *E*), where the nodes in *V* correspond to all the atomic services in the plan and the edges ($x \rightarrow y$) in *E*, where x and y are nodes in *V*, define that web service *x* produces an output that is required by *y* as an input. The process of obtaining a web service graph from the plan is straightforward and due to space limitations, we will not further elaborate on that. The output of the *Basic* function in Algorithm 1 is either a composite construct of the form *sequence*(c_1, c_2), or *split*(c_1, c_2,.., c_n), where c_1 to c_n are either *NULL* or composite constructs. For example, consider the web service graph presented in Fig. 4. The output of the *Basic* function presented in Algorithm 1 will be:

```
split(sequence(WSa,sequence(WSc,NULL)),sequence(WSb,sequence(WSc,NULL)))
```

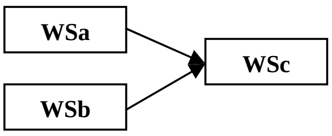


Fig. 4. Example of an OWL-S to PDDL translation.

The output of the *Basic* function is then fed to the *Join* function, presented in Algorithm 2, in order to replace the J*oin* construct with *Split+Join* wherever this is possible. The *Join* function searches in all possible pairs of split arguments, in order to find a common ending part. For instance, in the example composite service given above, both argument of split construct end in `sequence(WSc,NULL)`. Therefore, if we apply Algorithm 2 to the output of Algorithm 1, the resulting composite service will be:

```
split(sequence(split+join(WSa,WSb),sequence(WSc,NULL)))
```

The above composite service is then simplified by removing NULLs and constructs with single arguments and the final outcome is a construct of the form:

```
sequence(split+join(WSa,WSb),WSc)
```

Algorithm 2 (Join) Replaces *split* with *Split+Join* where possible in a composite service
Inputs: C=$f(a_1,a_2,..,a_n)$: a composite service with *Sequence* and *Split* constructs
Output: C: a composite service with *Sequence, Split* and *Split+Join* constructs

```
1:  set $f(a_1,a_2,..,a_n)=C$, where f is the name of the construct and $a_1$ to $a_n$ its arguments
2:  if $f = NULL$ then
        return NULL
3:  if $f = sequence$ then
        $a_1' = Join(a_1)$
        $a_2' = Join(a_2)$
        return $f(a_1',a_2')$
4:  if $f = split$ then
        for each pair ($a_i$, $a_j$), i,j in [1,n]
            if $a_i$ and $a_j$ have a common ending, i.e. $a_i = a_i' \cup k$ and $a_j = a_j' \cup k$ then
                $C' = C - \{a_i, a_j\} \cup seq(split+join(a_i', a_j'),k)$
                return $C'$
```

The above algorithms, among with some additional filters, such the one mentioned above for removing NULLs and unary constructs were implemented in PORSCE II using the CMU OWL-S API [21].

6. Solution and Integration

PORSCE II aims at a high degree of integration of the composition process; therefore, its features include solving the problem through invocation of external planning systems, visualization, solution evaluation and composite service modification.

6.1. Acquiring Solutions

Since the transformation process results in the export of both the planning domain and problem in PDDL, any PDDL-compliant domain independent external planning system can be used.

Currently, two different planning modules have been incorporated in the system: JPlan [1], which is an open-source Java implementation of Graphplan and LPG-*td* [8]. Both planners proved to be remarkably fast and can handle a respectable number of operators, which is very important as the number of available web services is expected to increase significantly over time. After the planning process is completed, JPlan provides the plan, in its own format, which comprises of a simple sequential list of actions. LPG-*td*, on the other hand, provides the plan in a format that complies with PDDL+. The plan in this case might not be sequential, but structured in levels; actions belonging to the same level can be executed in an arbitrary sequence, however all actions of a certain level must be completed before any action of the following level can be executed. Subsequently, these plans are visualized and their accuracy is evaluated.

6.2. Composite Service Accuracy Assessment

Semantic relaxation and the use of multiple planners may produce a number of composite services, for which statistics and quality metrics have to be calculated. Such metrics include the number of actions and the number of levels in the plan, as well as a plan distance quality metric, which indicates the accuracy of the plan, when semantic relaxation takes place.

For the calculation of the plan semantic distance, each concept appearing in the inputs or outputs of the actions of the plan is annotated by the OOM with a semantic distance d_i with respect to the original concept it was derived from, using the selected similarity metric. A concept distance of 0 reveals identical or equivalent concepts. Additionally, each concept is annotated with a weight w_i, which represents the kind of hierarchical relationship to the original concept, as in some cases certain hierarchical relationships might be more desirable than others.

These values are combined to form a plan semantic distance. For example, when the upwards cotopic distance metric is used, the plan semantic distance is calculated as a weighted product of these concepts, as the product represents better the semantic distance in this case:

$$PSD_{uc} = \prod_{i=0}^{n} w_i d_i , d_i \neq 0$$

The plan accuracy metric in both cases is calculated as 1-*PSD*; therefore, if there is exact input to output matching, or if only equivalent concepts are used, then the plan quality metric value is 1, while it decreases as the plan becomes less accurate.

6.3. Visualization and Composite Service Modification

The Visualizer enhances comprehensibility by providing a visual representation of the composite service and enables the user to interfere by manipulating it. The composite service is represented as a schema of simple service invocations, possibly structured in levels, showing inputs and outputs, as well as dependencies among atomic services.

The Visualizer module invokes and interacts closely with another module of the system, the Service Replacement Component, which allows the user to select among a series of alternatives in order to modify the produced composite service.

The first alternative for composite service modification is the replacement of an atomic service included in the composite service (plan) with a semantically equivalent or relevant service. In order to perform this operation, the system needs to discover all actions that could be used alternatively instead of the chosen one, using advice from the OOM as far as concept equivalence and semantic relevance are concerned. An action A is considered an alternative for an action Q of the plan as far as it does not disturb the plan sequence and the intermediate states. In order to ensure that, both the following conditions must hold:

- $\text{prec}(A) \subseteq \text{prec}(Q)$
- $\text{add}(A) \supseteq \text{add}(Q)$

The selected alternative service substitutes the original one both in the plan and in the visualization, and no replanning is performed. The atomic service substitution can be applied an arbitrary number of times on any of the atomic services taking part in the composite service.

In cases when none of the semantically equivalent or relevant services that correspond to an atomic service is considered suitable, or in cases where there are no alternative services, the system offers the option to substitute the atomic service with a partially ordered set of atomic services, which are found through planning. In this case, the world states right before and after the execution of the action being replaced serve as the initial and goal states for the planning process, respectively. In order to find the initial state I_{rr} and the goal state G_{rr} for the replanning process Algorithms 3 and 4 are used respectively.

Note that the replanning process is bound to return the atomic service being replaced itself, especially if the external planner used produces the optimal plan in each case. In order to prevent that, this specific service has to be removed from the set of available services before the replanning process proceeds. As the new plan produced substitutes the atomic service, its quality metrics have to be incorporated in the quality metrics of the entire plan.

Algorithm 3 Computes the initial state for replacement of action A_i through replanning (I_{rr})
Inputs EIS: the extended initial state,
P = {A_1, A_2, ..., A_n}: the plan
i: the plan index of the action being replaced.
Output The I_{rr}

```
1: Set $I_{rr} \leftarrow EIS$
2:   $j = 1$
3: do
4:     $I_{rr} \leftarrow I_{rr} \cup add(A_j)$
5:     $j = j + 1$
6: while $j < i$
7: return $I_{rr}$
```

Algorithm 4 Computes the goal state for replacement of action A_i through replanning (G_{rr})
Inputs G: the goal state of the initial problem,
P = {A_1, A_2, ..., A_n}: the plan
i: the plan index of the action being replaced.
Output The G_{rr}

```
1: Set $G_{rr} \leftarrow G$
2:   $j = n$
3: do
4:     $G_{rr} \leftarrow G_{rr} - add(A_j) \cup prec(A_j)$
5:     $j = j - 1$
6: while $j > i$
7: return $G_{rr}$
```

If replacement of an atomic service, either by equivalents or through replanning, is not a suitable option, or if multiple atomic services are undesirable or unavailable, the user can resort to replanning from a certain point in the plan, or even replanning from scratch. When replanning from a certain point, the aforementioned Algorithm 1 has to be used in order to calculate the initial state.

7. Demonstration and System Evaluation

This section aims at demonstrating the use and evaluating the performance of the system through a case study, following the general course depicted in Fig. 5.

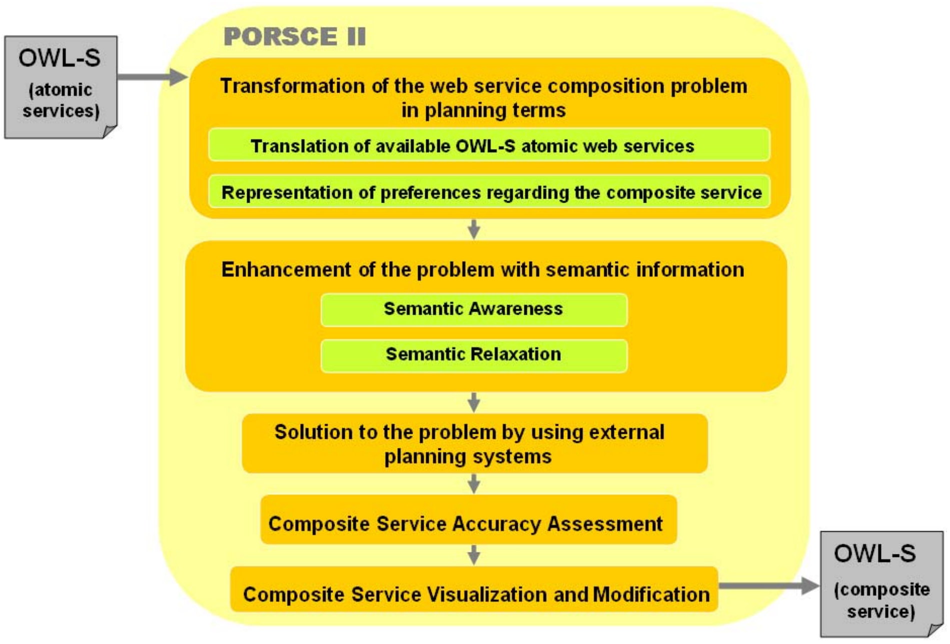

OWL-S
(atomic services)
PORSCE II
Transformation of the web service composition problem in planning terms
Translation of available OWL-S atomic web services
Representation of preferences regarding the composite service
Enhancement of the problem with semantic information
Semantic Awareness
Semantic Relaxation
Solution to the problem by using external planning systems
Composite Service Accuracy Assessment
Composite Service Visualization and Modification
OWL-S
(composite service)

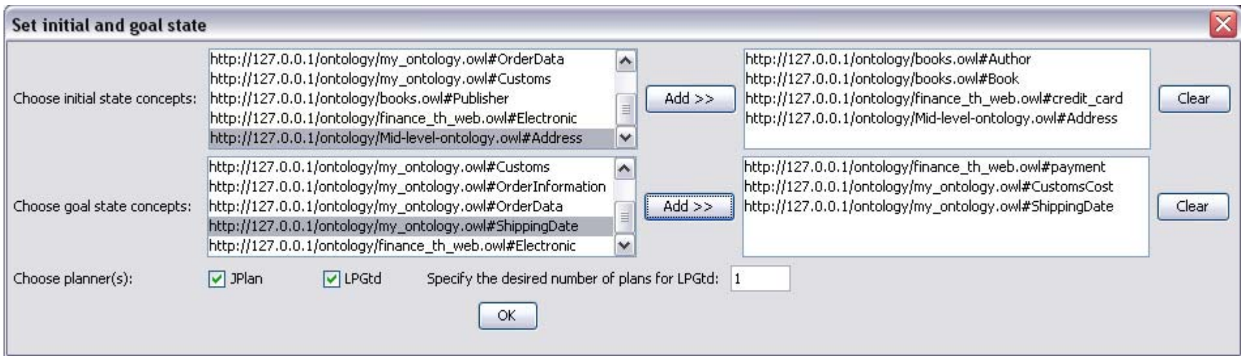
Fig. 5. The steps for the demonstration.

The test sets used to perform experiments were obtained from the OWLS-TC version 2.2 revision 1 [3], while several service descriptions were modified or added to these domains, accommodating the demonstration of the capabilities of the system. Some indicative web services that were modified or added to the domain are depicted in Table 1.

Table 1. Added / modified web services.

Service	**Inputs**	**Outputs**
BookToPublisher	Book, Author	Publisher
CreditCardCharge	OrderData, CreditCard	Payment
ElectronicOrder	Electronic	OrderData
PublisherElectronicOrder	PublisherInfo	OrderData
ElectronicOrderInfo	Electronic	OrderInformation
Shipping	Address, OrderData	ShippingDate
WaysOfOrder	Publisher	Electronic
CustomsCost	Publisher, OrderData	CustomsCost

The transformation of the web service composition problem in planning terms includes translating all available OWL-S atomic web services, including the aforementioned ones, to PDDL operators, so that the size of the resulting domain is maintained on realistic levels. The transformation process also incorporates the representation of the requirements about the composite service, which the user can express through a dialog interface such as the one depicted in Fig. 6.


Set initial and goal state

Choose initial state concepts:

http://127.0.0.1/ontology/my_ontology.owl#OrderData
http://127.0.0.1/ontology/my_ontology.owl#Customs
http://127.0.0.1/ontology/books.owl#Publisher
http://127.0.0.1/ontology/finance_th_web.owl#Electronic
http://127.0.0.1/ontology/Mid-level-ontology.owl#Address

http://127.0.0.1/ontology/books.owl#Author
http://127.0.0.1/ontology/books.owl#Book
http://127.0.0.1/ontology/finance_th_web.owl#credit_card
http://127.0.0.1/ontology/Mid-level-ontology.owl#Address

Choose goal state concepts:

http://127.0.0.1/ontology/my_ontology.owl#Customs
http://127.0.0.1/ontology/my_ontology.owl#OrderInformation
http://127.0.0.1/ontology/my_ontology.owl#OrderData
http://127.0.0.1/ontology/my_ontology.owl#ShippingDate
http://127.0.0.1/ontology/finance_th_web.owl#Electronic

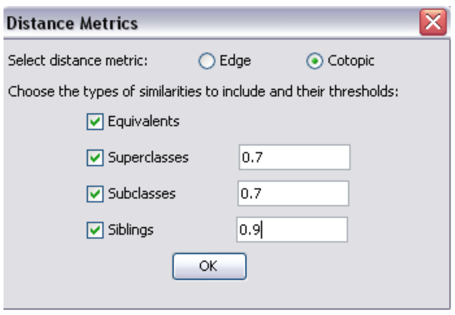
http://127.0.0.1/ontology/finance_th_web.owl#payment
http://127.0.0.1/ontology/my_ontology.owl#CustomsCost
http://127.0.0.1/ontology/my_ontology.owl#ShippingDate

Choose planner(s): ✓ JPlan ✓ LPGtd Specify the desired number of plans for LPGtd: 1

Fig. 6. Defining initial and goal states and desired planners.

The scenario implemented here belongs to the OWLS-TC books and finance domains, and concerns the electronic purchase of a book. The user provides as inputs a book title and author, credit card info and the address that the book will be shipped to, and requires a charge to credit card for the purchase, as well as the shipping dates and the customs cost for the specific item. The initial state corresponds to the inputs of the composite service, while the goal state represents the desired composite service outcome.

In order to accommodate semantic awareness, all the ontologies that organize the concepts appearing as inputs and outputs of the available atomic web services are parsed and analyzed. This enables semantic relaxation, performed through semantic enhancement of the planning domain and problem. The degree of the semantic relaxation is user-defined, and can be specified by selecting semantic distance metrics and thresholds through the interface depicted in Fig. 7.


Distance Metrics

Select distance metric: Edge [✓ Cotopic]

Choose the types of similarities to include and their thresholds:

✓ Equivalents
✓ Superclasses 0.7
✓ Subclasses 0.7
✓ Siblings 0.9

Fig. 7. The semantic enhancement interface.

At this point, the system exports the formulated and possibly semantically enhanced planning domain and problem to PDDL. Consequently, it invokes external planners to acquire solutions. The PDDL domain, problem and produced plan for this scenario are depicted in Fig. 8. Note that the domain in this case, for space purposes, contains only the necessary atomic web services.

DOMAIN:

```
(define (domain wscomp)
  (:predicates (block ?b)
 (ontologybooksowlAuthor ?b)
 (ontologybooksowlBook ?b)
 (ontologybooksowlPublisher ?b)
 (ontologyfinancethwebowlcreditcard ?b)
 (ontologymyontologyowlOrderData ?b)
 (ontologyfinancethwebowlpayment ?b)
 (ontologymyontologyowlCustomsCost ?b)
 (ontologyfinancethwebowlElectronic ?b)
 (ontologyMidlevelontologyowlAddress ?b)
 (ontologymyontologyowlShippingDate ?b))
(:action BookToPublisherService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologybooksowlAuthor ?b)
  (ontologybooksowlBook ?b))
   :effect (ontologybooksowlPublisher ?b))
(:action CreditCardChargeService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologyfinancethwebowlcreditcard ?b)
  (ontologymyontologyowlOrderData ?b))
   :effect (ontologyfinancethwebowlpayment ?b))
(:action CustomsCostService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologymyontologyowlOrderData ?b)
  (ontologybooksowlPublisher ?b))
   :effect (ontologymyontologyowlCustomsCost ?b))
(:action ElectronicOrderService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologyfinancethwebowlElectronic ?b))
   :effect (ontologymyontologyowlOrderData ?b))
(:action ShippingService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologyMidlevelontologyowlAddress ?b)
  (ontologymyontologyowlOrderData ?b))
   :effect (ontologymyontologyowlShippingDate ?b))
(:action WaysOfOrderService
   :parameters (?b)
   :precondition (and (block ?b)
  (ontologybooksowlPublisher ?b))
   :effect (ontologyfinancethwebowlElectronic ?b)))
```

PROBLEM:

```
(define (problem wscompprob)
(:domain wscomp)
(:objects a)
(:init (block a)
 (ontologybooksowlAuthor a)
(ontologybooksowlBook aaa)
(ontologyfinancethwebowlcredi
tcard a)
(ontologyMidlevelontologyowlA
ddress a))
(:goal (and
(ontologyfinancethwebowlpayme
nt a)
(ontologymyontologyowlCustoms
Cost a)
(ontologymyontologyowlShippin
gDate a)
)))
```

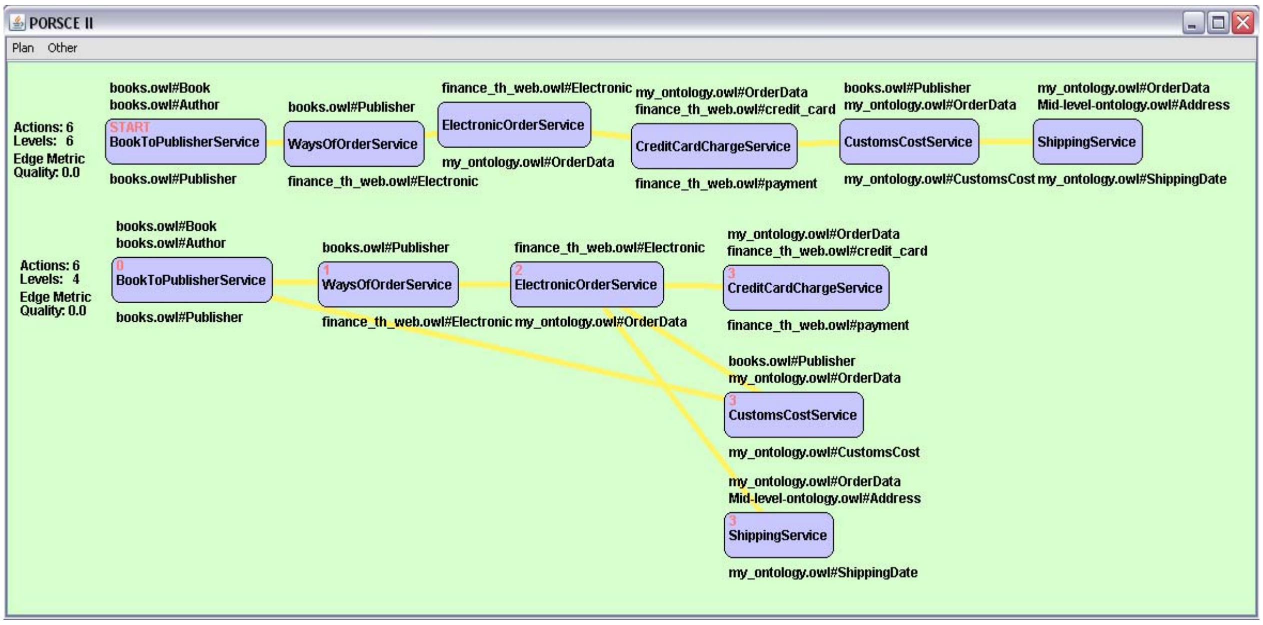
PLAN:

```
; Version LPG-td-1.0
; Seed 105421875
; Time 0.02
; Search time 0.00
; Parsing time 0.00
; Mutex time 0.00
; Quality 6

Time 0.02

0:(BOOKTOPUBLISHERSERVICE A)
[1]
1:(WAYSOFORDERSERVICE AAA)
[1]
2:(ELECTRONICORDERSERVICE A)
[1]
3:(CREDITCARDCHARGESERVICE A)
[1]
3:(CUSTOMSCOSTSERVICE AAA)
[1]
3:(SHIPPINGSERVICE AAA) [1]
```

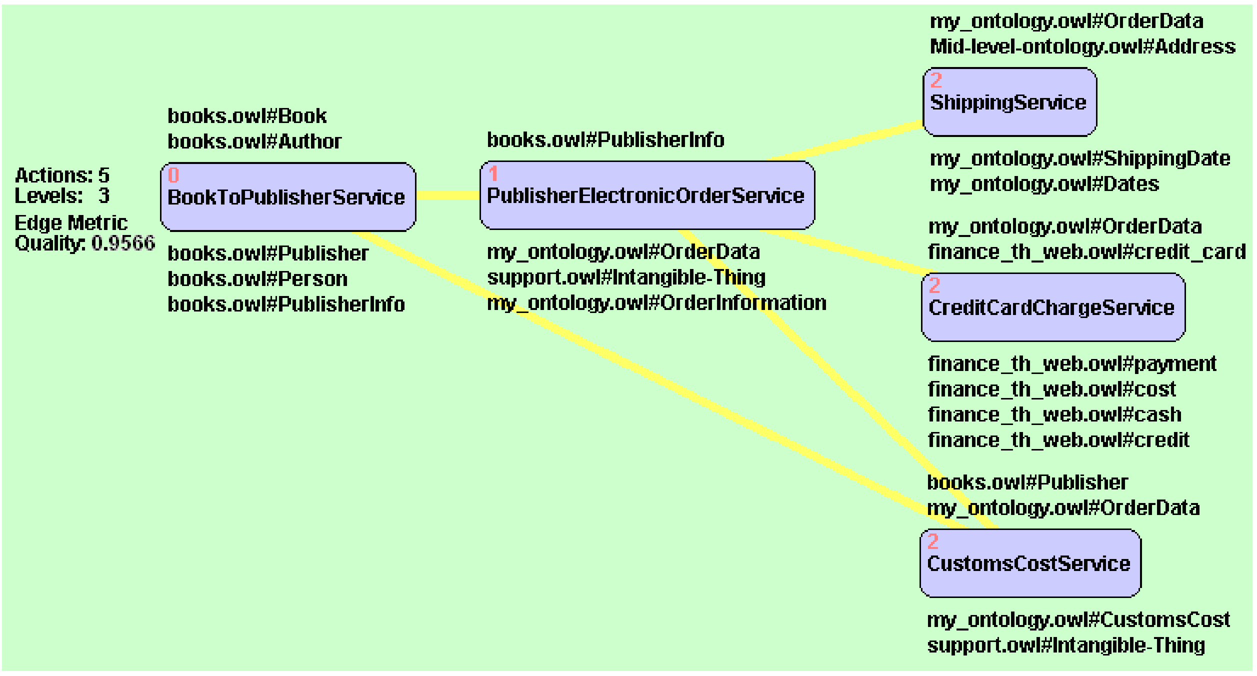
Fig. 8. The PDDL domain, problem and plan for the specific scenario.

The produced plans are imported into the Visualizer Component, where they are represented as a web service graph and depicted graphically.

The plans produced by JPlan and LPG-td for the specific case study, using the operator set described above, without including any semantically relevant concepts are presented in Fig. 9.


PORSCE II
Plan
Other
Actions: 6
Levels: 6
Edge Metric
Quality: 0.0
books.owl#Book
books.owl#Author
START
BookToPublisherService
books.owl#Publisher
books.owl#Publisher
WaysOfOrderService
finance_th_web.owl#Electronic
finance_th_web.owl#Electronic
ElectronicOrderService
my_ontology.owl#OrderData
my_ontology.owl#OrderData
finance_th_web.owl#credit_card
CreditCardChargeService
finance_th_web.owl#payment
books.owl#Publisher
my_ontology.owl#OrderData
CustomsCostService
my_ontology.owl#CustomsCost
my_ontology.owl#OrderData
Mid-level-ontology.owl#Address
ShippingService
my_ontology.owl#ShippingDate
Actions: 6
Levels: 4
Edge Metric
Quality: 0.0
books.owl#Book
books.owl#Author
0
BookToPublisherService
books.owl#Publisher
books.owl#Publisher
1
WaysOfOrderService
finance_th_web.owl#Electronic
finance_th_web.owl#Electronic
2
ElectronicOrderService
my_ontology.owl#OrderData
my_ontology.owl#OrderData
finance_th_web.owl#credit_card
3
CreditCardChargeService
finance_th_web.owl#payment
books.owl#Publisher
my_ontology.owl#OrderData
3
CustomsCostService
my_ontology.owl#CustomsCost
my_ontology.owl#OrderData
Mid-level-ontology.owl#Address
3
ShippingService
my_ontology.owl#ShippingDate
Fig. 9. The plans from JPlan (top) and LPG-td (bottom) for the specific case study.

While exact matching of input to output concepts is obligatory in the classical planning domains, in the web services world the case can be different, as it is preferable to present the user with a composite service that approximates the required functionality than to present no service at all. The semantically similar concepts obtained from the OWL Ontology Manager enable the system to compose alternative services that approximate the desired one in case there are no exact matches, by performing semantically relaxed concept matching. Such an approximate service for the specific case study is presented in Fig. 10. The calculated accuracy of this service is different from the accurate ones presented in Fig. 9.


Actions: 5
Levels: 3
Edge Metric
Quality: 0.9566
books.owl#Book
books.owl#Author
0
BookToPublisherService
books.owl#Publisher
books.owl#Person
books.owl#PublisherInfo
books.owl#PublisherInfo
1
PublisherElectronicOrderService
my_ontology.owl#OrderData
support.owl#Intangible-Thing
my_ontology.owl#OrderInformation
my_ontology.owl#OrderData
Mid-level-ontology.owl#Address
2
ShippingService
my_ontology.owl#ShippingDate
my_ontology.owl#Dates
my_ontology.owl#OrderData
finance_th_web.owl#credit_card
2
CreditCardChargeService
finance_th_web.owl#payment
finance_th_web.owl#cost
finance_th_web.owl#cash
finance_th_web.owl#credit
books.owl#Publisher
my_ontology.owl#OrderData
2
CustomsCostService
my_ontology.owl#CustomsCost
support.owl#Intangible-Thing
Fig. 10. Approximate composite service.

In case service replacement is required, for example on the CustomsCost service, and there is no alternative service available, service replacement through replanning will be employed. The algorithms described in the corresponding section will yield new initial and goal states, and the corresponding planner, in this case JPlan, will be re-invoked, finding a new sequence of actions that can substitute the selected service. The user interface for the replacement options is depicted in Fig. 11, while the resulting composite service after the modifications is depicted in Fig. 12.

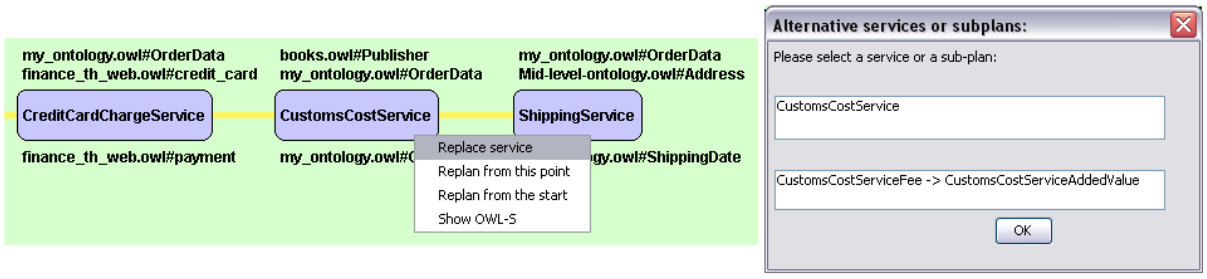

my_ontology.owl#OrderData
finance_th_web.owl#credit_card
CreditCardChargeService
finance_th_web.owl#payment
books.owl#Publisher
my_ontology.owl#OrderData
CustomsCostService
my_ontology.owl#C
my_ontology.owl#OrderData
Mid-level-ontology.owl#Address
ShippingService
gy.owl#ShippingDate
Replace service
Replan from this point
Replan from the start
Show OWL-S
Alternative services or subplans:
Please select a service or a sub-plan:
CustomsCostService
CustomsCostServiceFee -> CustomsCostServiceAddedValue
OK

Fig. 11. Service Replacement Interface.

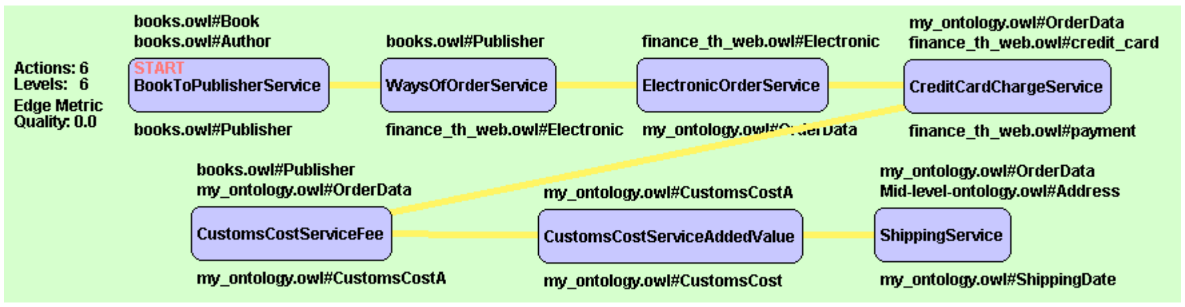

Actions: 6
Levels: 6
Edge Metric
Quality: 0.0
books.owl#Book
books.owl#Author
START
BookToPublisherService
books.owl#Publisher
books.owl#Publisher
WaysOfOrderService
finance_th_web.owl#Electronic
finance_th_web.owl#Electronic
ElectronicOrderService
my_ontology.owl#OrderData
my_ontology.owl#OrderData
finance_th_web.owl#credit_card
CreditCardChargeService
finance_th_web.owl#payment
books.owl#Publisher
my_ontology.owl#OrderData
CustomsCostServiceFee
my_ontology.owl#CustomsCostA
my_ontology.owl#CustomsCostA
CustomsCostServiceAddedValue
my_ontology.owl#CustomsCost
my_ontology.owl#OrderData
Mid-level-ontology.owl#Address
ShippingService
my_ontology.owl#ShippingDate

Fig. 12. Composite service after replacement operation.

The final step, in order to complete the cycle is to transform the solution back to the initial web services domain. This is achieved by translating the PDDL+ plan produced by the external planning systems that represents the desired composite web service, into OWL-S, utilizing information retrieved from the atomic web service descriptions, the ontologies and semantic analysis.

In order to study the behavior of the system as the number of available web services increases, web service profiles were added to the domain progressively in batches. The time performance results presented in Table 2 were obtained from a number of runs of the system on a machine with Dual-Core AMD Opteron Processor at 2.20GHz with 1GB of RAM memory and concern times for preprocessing, transformation of the OWL-S service profiles to PDDL actions and planning using LPG-td.

Table 2. Time measurements in milliseconds.

Number of web services		**10**	**100**	**500**	**1000**
Preprocessing time		5857	6104	5875	5703
Total transformation time	**Exact**	4594	70062	350836	792109
	Edge	4531	75725	335477	796797
	Cotopic	4585	74688	728633	3901141
Transformation time per web service	**Exact**	459	700	702	792
	Edge	453	671	757	797
	Cotopic	459	746	1457	3901
Planning time (LPG-td)	**Exact**	1	13	16	17
	Edge	4	6	15	16
	Cotopic	3	5	16	16

Measurements took place for domains of different sizes, namely 10, 100, 500 and 1000 OWL-S profiles. Some of the experiments were performed without semantic relaxation (X), while others were performed with semantic relaxation using either the edge-counting distance metric (E) or the upwards cotopic metric (C). The preprocessing time did not show significant fluctuation, as it depends only on the number and structure of the processed ontologies and not on the number of available web services. The total transformation time evidently increased as the number of available web services increased, however the average transformation time per web service profile converged to approximately 0.8 seconds for the exact matching and the edge-counting distance metric cases. In the upwards cotopic metric distance, the increase in the average transformation time is significant as available web services increase, due to the higher complexity of the algorithm used for the calculation of the upwards cotopic relevance between two concepts. As far as average planning time is concerned, LPG-td shows an increase in planning time as the number of actions increases; however, it is still proved remarkably fast.

8. Conclusions and Future Work

This paper presented PORSCE II, an integrated system which combines planning with semantic object relevance in order to approach automated semantic web service composition. The web service composition problem is transformed into a planning problem, solved under semantic awareness, and then transformed back in web service terms. The system exploits the most prominent standards in both words, namely the OWL-S and PDDL. PORSCE II aims at a high degree of interoperability with external planning systems which perform planning with the desired degree of semantic relaxation. Finally, the system is integrated with a visual environment and components which accommodate composite service evaluation and modification.

Future goals include the extension of the system in order to deploy the produced composite services, through OWL-S deployment systems such the OWL-S Virtual Machine [19], and automatically acquire feedback, which can be utilized to partially automate the service replacement procedure. In addition, another goal concerns the exploration of the possibility to accelerate the composition process by asserting the produced OWL-S profiles in the base of the available atomic services, under certain time constraints. Furthermore, integration with the VLEPPO system [9] is a promising future direction, in order to accommodate design and solving of the web service composition problems. Finally, it lies in our immediate plans to study ways to enhance the services representation and explore the ability to produce various composite services according to non-functional properties.

References

[1] JPlan: Java Graphplan Implementation, http://sourceforge.net/projects/jplan

[2] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz, Pellet: A Practical OWL DL Reasoner, J. Web Semantics, 2007

[3] OWLS-TC version 2.2 revision 1, http://projects.semwebcentral.org/ projects/owls-tc/

[4] R. Fikes, N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence, Vol 2 (1971), 189-208.

[5] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, "PDDL -- the Planning Domain Definition Language". Technical report, Yale University, New Haven, CT (1998).

[6] OWL-S 1.1. http://www.daml.org/services/owl-s/1.1/

[7] O. Hatzi, G. Meditskos, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, I. Vlahavas, A Synergy of Planning and Ontology Concept Ranking for Semantic Web Service Composition, IBERAMIA 2008, Lisbon, Portugal, 2008. Proceedings Lecture Notes in Computer Science 5290 Springer 2008, pp 42-51.

[8] A. Gerevini, A. Saetti, I. Serina, LPG-TD: a Fully Automated Planner for PDDL2.2 Domains" (short paper), in International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04).

[9] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, I. Vlahavas, VLEPpO: A Visual Language for Problem Representation, PlanSIG 2007, pp. 60 – 66.

[10] A. Maedche and V. Zacharias, Clustering Ontology-Based Metadata in the Semantic Web, European Conf. Principles of Data Mining and Knowledge Discovery, 2002.

[11] SAWSDL, http://www.w3.org/2002/ws/sawsdl/

[12] OWL, http://www.w3.org/TR/owl-ref/

[13] E. Sirin, B. Parsia, D. Wu, J. Hendler and D. Nau, 2004. HTN planning for web service composition using SHOP2. Journal of Web Semantics, 1(4) 377–396.

[14] M. Klusch, A. Gerber, M. Schmidt: Semantic Web Service Composition Planning with OWLS-XPlan. AAAI Fall Symposium on Semantic Web and Agents, USA, 2005.

[15] A. Bucchiarone and S. Gnesi, 2006. A Survey on Service Composition Languages and Models, 1st International Workshop on Web Services Modeling and Testing (WsMaTe 2006).

[16] J. Rao and X. Su, 2004. A Survey of Automated Web Service Com-position Methods. Lecture Notes in Computer Science, Volume 3387/2005, Springer, p. 43-54.

[17] S. Dustdar, W. Schreiner, 2005. A survey on web services composition, Int. J. Web and Grid Services, Vol. 1, No. 1, pp.1–30.

[18] IPC 2004. International Planning Competition, http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/

[19] M. Paolucci, A. Ankolekar, N. Srinivasan and K. Sycara, "The DAML-S Virtual Machine," In Proceedings of the Second International Semantic Web Conference (ISWC), 2003, Sandial Island, Fl, USA, October 2003, pp 290-305.

[20] O. Hatzi, G. Meditskos, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, I. Vlahavas, "PORSCE II: Using Planning for Semantic Web Service Composition", ICKEPS2009, in conjunction with ICAPS-09, Thessaloniki, Greece, 2009.

[21] CMU OWL-S API. http://www.daml.ri.cmu.edu/owlsapi/