The Knowledge Engineering Review, Vol. 28:2, 137–156. © Cambridge University Press, 2013
doi:10.1017/S0269888912000392
First published online 18 February 2013

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] and IOANNIS VLAHAVAS[2]

[1]*Department of Informatics and Telematics, Harokopio University of Athens, Harokopou 89, 17671 Athens, Greece; e-mail: raniah@hua.gr, dimosthe@hua.gr;*
[2]*Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece; e-mail: dvrakas@csd.auth.gr, nbassili@csd.auth.gr, vlahavas@csd.auth.gr*

Abstract

This paper presents PORSCE II, an integrated system that performs automatic Semantic Web service composition exploiting artificial intelligence (AI) techniques, specifically planning. Essential steps in achieving Web service composition include the translation of the Web service composition problem into a solver-ready planning domain and 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 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 service 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 artificial intelligence (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 Web Ontology Language for Services (OWL, 2004; OWL-S, 2004) and Semantic Automations for WSDL (SAWSDL, 2007). Nevertheless, there are no tools utilizing semantic information incorporated

in OWL-S to efficiently compose Web services either accurately or approximately, taking into account the actual meaning of Web service inputs/outputs as well as the corresponding ontologies.

The PORSCE II framework aims at automated Semantic Web service composition by employing planning under semantic awareness and relaxation. Its contribution focuses on the effective utilization of semantic information present in OWL-S description of Web services to enhance the Web service composition process by facilitating approximate compositions, via planning.

The first and decisive step in the process concerns the translation of the Web service composition problem to AI planning terms (Hatzi *et al.*, 2009). This translation is based on the observation that certain correspondences exist between the two domains, which, given the appropriate mapping, enable the application of planning techniques to solve the Web service composition problem effectively. Such correspondences include the available Web services that can be combined to formulate meaningful compositions, which can be mapped to the planning domain, and user requirements about the desired composite service, which can be perceived as a planning problem over this domain.

The translation takes place between the most prominent standards in each area: OWL-S for semantic description of Web services (either atomic or composite) and PDDL (Planning Domain Definition Language) (Ghallab *et al.*, 1998) 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 related work in the area of Web service composition through planning, while Section 3 provides background related to the OWL-S standard, AI planning and the PDDL standard. Section 4 outlines 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 and poses future directions.

2 Related work

A number of approaches for automatic Web service composition can be found at Rao and Su (2004) and Dustdar and Schreiner (2005); the most closely related with the approach proposed in this paper are discussed and summarized in Table 1.

SHOP-2 (Sirin *et al.*, 2004) uses services descriptions in DARPA Agent Markup Language for Services (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 (Klusch & Gerber, 2005) uses the semantic descriptions of atomic Web services in OWL-S to derive planning domains and problems, and invokes a planning module called XPlan

Table 1 Overview and comparison of planning approaches to web service composition

System/approach	Atomic services	Composite service (inputs/outputs)	Advantages	Disadvantages
SHOP-2	Primitive tasks	Task networks, compound tasks	Heuristics for increased performance	Prior expert domain knowledge required
OWLS-XPlan	Primitive tasks	Initial state/goal state	Combination of planning techniques	Ontology information not utilized. Domain-specific knowledge required
SWORD	Entity-relation model, Horn r.	Initial state/goal state	Utilizes research in the area of rule-based expert systems	Requires user intervention. Not straightforward representation. Requires domain-specific knowledge
GOLOG	Situation calculus	High-level generic procedures and constraints	Utilizes research in the area of situation calculus	Complex encoding and translation. Decreased scalability and interoperability
Estimated-regression planning	State Trans. Ops	Initial state/goal state	Heuristics for acceleration of the process	Requires extension to the standards and planners. Decreased scalability
Knowledge-based	BPEL processes	Compound process—goal	Well-founded formalization. Non-determinism	Semantic information and ontologies not utilized. Decreased scalability

OWL-S = Web Ontology Language for Services; BPEL = Business Process Execution Language.

to generate the composite services. The system is PDDL compliant, as the authors have developed an Extensible Markup Language (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.

The SWORD framework (Ponnekanti & Fox, 2002) is a set of tools that addresses a particular subset of the automatic Web service composition problem. SWORD follows an approach close to the one of PORSCE II for modeling Web service composition. However, atomic services are represented as rules, stating that given specific inputs, an atomic service produces specific outputs. SWORD then uses a rule-based expert system to generate the composition plan, rather than classical planning.

The work described in Mcilraith and Son (2002) conceives Web service composition as a planning and execution task, where actions (services) may be complex. The GOLOG language is then employed, adapted and extended to address the issue of composition. Approaches that use knowledge-based planning include the work in Pistore *et al*. (2005). Web service descriptions are expressed in a standard process modeling and execution language, such as Business Process Execution Language for Web Services (BPEL4WS), therefore some prior, domain-specific knowledge of the composition issues is required, while another approach employing estimated-regression planning is presented in Mcdermott (2002); however, in order to be used, it requires extension to current standards.

The main advantages of the proposed framework with respect to the aforementioned systems include 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 requires neither prior domain-specific knowledge, nor any kind of extension to the standards, in order to form valid, desired composite services; the OWL-S descriptions of the atomic Web services and the corresponding ontologies suffice. Finally, PORSCE II is able to scale up for a great number of services, having the flexibility to exploit modern, advanced planners, and it also handles cases of service failure or unavailability dynamically, an important feature not covered by the aforementioned frameworks.

3 Background

This section presents the fundamental standards for the proposed approach, namely OWL-S and PDDL. Additionally, it introduces some basic planning notation that will be used throughout the paper.

3.1 Web Ontology Language for Services

OWL-S is an upper ontology based on OWL, created in the context of the Semantic Web in order to describe knowledge concerning Semantic Web services. It is used in combination with ontologies organizing 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 comprises:

- *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 concrete details of how an agent can access a service, such as communication protocols and message formats.

The proposed approach utilizes semantic information contained in the *service profile*, along with the corresponding ontologies, in order to translate the description in planning terms.

An ontology in this context refers to a formal representation of the concepts appearing as inputs and outputs in the Web service profiles. The concepts in the ontology are connected with hierarchical relationships, such as superclass, subclass and sibling.

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 and Planning Domain Definition Language

A planning domain and problem is usually modeled according to STRIPS (Stanford Research Institute Planning System) notation (Fikes & Nilsson, 1971) 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 $P = A_1, A_2, \ldots, A_n$, which, if applied to I, lead to a state S' such that $S' \supseteq G$.

PDDL (Ghallab *et al.*, 1998) 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 International Planning Competition (IPC, 2004). However, it has since been enhanced, extended and become a standard 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 (Gerevini & Long, 2005) 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 account the temporal properties of domains, as well as quality metrics, features that might prove very useful in the Web service composition case, while PDDL+ (Fox & Long, 2002), also provides a standard way to represent plans, either sequential or partially parallel.

4 Overview and architecture

PORSCE II was based on the results obtained from the prototype system PORSCE (Hatzi *et al.*, 2008). PORSCE II aims at a high degree of integration as, along with the core transformation component, it additionally contains a visual interface, more elaborate relevance metrics, the ability for composite service accuracy assessment and composite Web service manipulation features. Furthermore, PORSCE II adopts a way of modeling the Web service composition as a planning problem that reduces the complexity of the generated planning problem, thus accelerating the planning process. In order to highlight the planner independency of PORSCE II, which enables

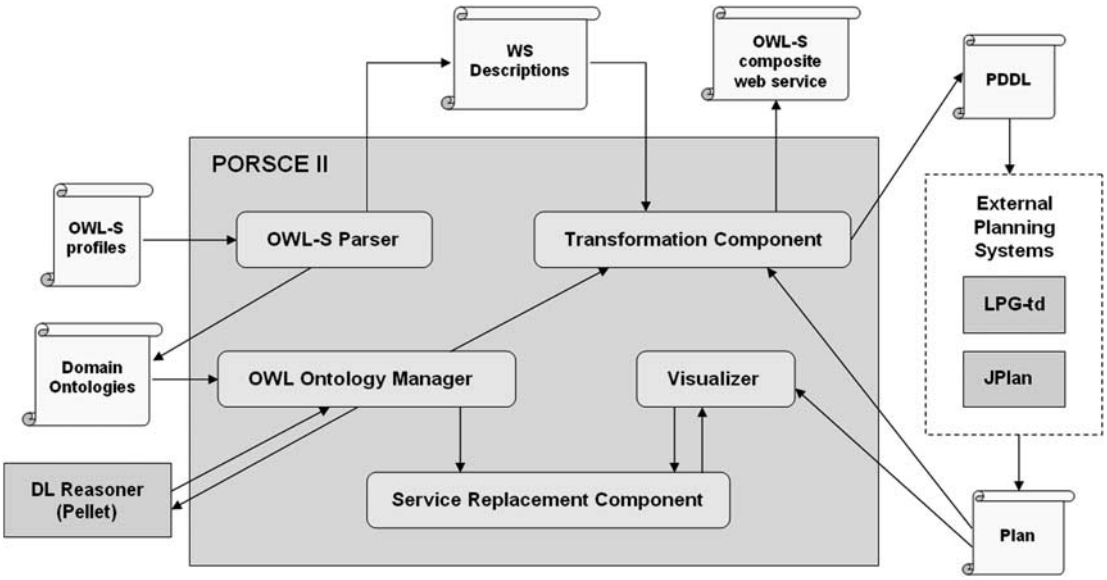

Figure 1 PORSCE II architecture. WS = Web service; DL = description logic; JPlan = Java Graphplan; OWL-S = Web Ontology Language for Services; PDDL = Planning Domain Definition Language

the use of any domain-independent planning system based on PDDL, two external planners have been included. Finally, PORSCE II 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 composite service, thus facilitating deployment.

The key features of the framework are:

- Translation of OWL-S Web service descriptions (atomic or composite) into planning operators.
- Interaction with the user in order to acquire their preferences regarding the desired 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.
- Flexibility in the choice of planner, as any PDDL-compliant external planning system can be used.
- Assessing the accuracy of the composite services.
- Visualizing and modifying the solution by atomic service replacement or replanning.
- Transformation of the solution (composite Web service) back to OWL-S.

PORSCE II comprises 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 Figure 1.

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 OOM, utilizing the inferencing capabilities of the Pellet DL reasoner (Sirin *et al.*, 2007), 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. 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 translation of the Web service composition problem into a planning problem and possible enhancement with semantic information, as well as transformation of the plan representing the produced composite service back in Web service context. 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 Web Ontology Language for Services to Planning Domain Definition Language translation

The first step in the translation process generates the planning domain by translating each available OWL-S Web service profile WSD_i into a planning action A_i (Figure 2). More specifically:

- The name of the action is the *rdf:ID* field of the profile: $name(A_i) = WSD_i.ID$
- The preconditions of the action are formed by the service input and precondition definitions: $prec(A_i) \equiv \bigcup_{k=1}^{n} WSD_i.hasInput_k \cup \bigcup_{k=1}^{m} WSD_i.hasPrecondition_k$The add effects of the action comprise of the service output and positive effect definitions: $add(A_i) \equiv \bigcup_{k=1}^{n} WSD_i.hasOutput_k \cup \bigcup_{k=1}^{m} WSD_i.hasEffect_k^+$
- The delete list is formed by the negative effect definitions. The Semantic Web Rule Language (SWRL, 2004) was used in order to model the preconditions and effects of the Web services. SWRL combines OWL DL and Rule Markup Language (RuleML, 2001) in order to model preconditions and consequences in the Semantic Web, through the use of Horn-like rules. In the PORSCE II case, preconditions are modeled by SWRL rule conditions, while positive effects are modeled as SWRL atomic expressions that are true in the world after the execution of the Web service. Since SWRL does directly support for negation and negated atomic expressions, which would model negative (delete) effects, the negation <neg> element of RuleML was employed, which is used by the transformation process in order to discriminate between add and delete effects. The delete list of the action is formed as follows: $del(A_i) \equiv \bigcup_{k=1}^{m} WSD_i.hasEffect_k^-$

This transformation can be applied either to atomic or to composite Web services described in OWL-S, provided that the outputs of the composite service are fully deterministic; PORSCE II is not concerned with the internal implementation of the services. An example of an OWL-S to PDDL transformation is presented in Figure 3, where the mapping presented above is marked.

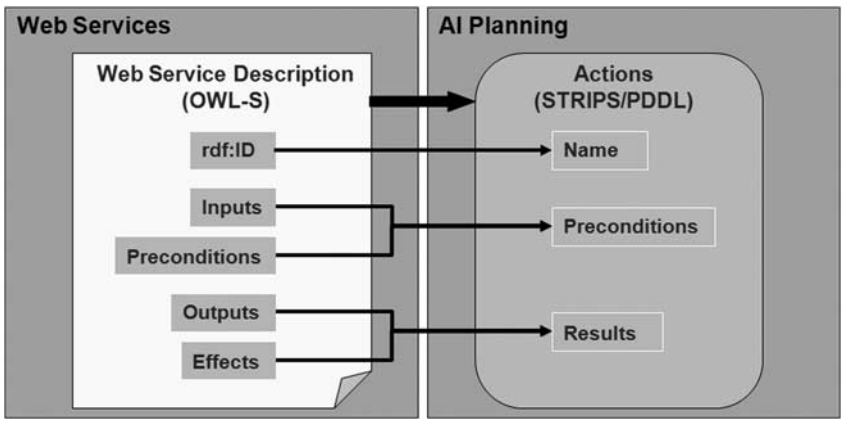

Figure 2 Web services to planning domain mapping. STRIPS = Stanford Research Institute Planning System; PDDL = Planning Domain Definition Language; OWL-S = Web Ontology Language for Services

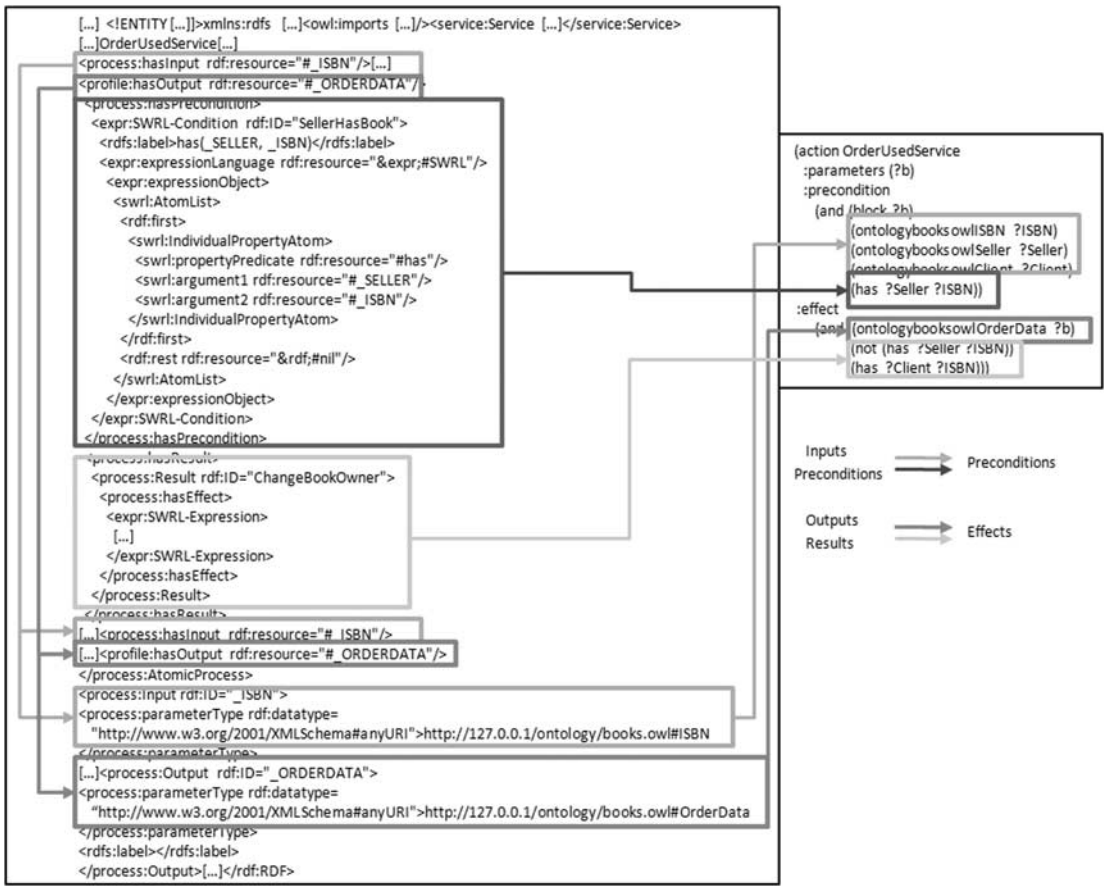

Figure 3 OWL-S to PDDL translation example. OWL-S = Web Ontology Language for Services; PDDL = Planning Domain Definition Language

The Web service description at hand concerns a Web service that accepts as inputs the ISBN, a Seller and a Client of a book, and has as a precondition that a Seller possesses the book indicated by the ISBN. The service has as output the Order Data, and as a result the change of book ownership from Seller to Client. The addition of the auxiliary argument in every PDDL predicate representing an input or output concept was a necessary technicality, in order to overcome the disadvantage of many planners that did not accept predicates without arguments, which does not in any way affect the outcome of the composition.

After the creation of the planning domain, the next step is the generation of a corresponding planning problem, based on the user requirements about the composite service. A straightforward solution adopted by PORSCE II for this step is the following: Let IC be the set of concepts that the user wishes to provide to the composite service and GC its desired outputs. If O denotes the set of all available concepts in the ontology, then $IC \subseteq O$, $GC \subseteq O$ and $IC \cap GC = \emptyset$. The inputs that the user wishes to provide formulate the initial state of the planning problem, while the desired outputs of the composite service formulate the goals: $I = IC$ and $G = GC$. Both input and output sets are provided externally by the user.

5.2 Semantic analysis

The step of semantic analysis, following that of the transformation process described in the previous subsection, enables the system to exploit semantic information. This step is implemented by the OOM. During translation, the OOM is used extensively for performing semantic relaxation, which is useful in cases when an exact input-to-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* and their *semantic distance* does not exceed a user-defined 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 identifier (uniform resource identifier (URI)) or they should be equivalent, in terms of OWL class equivalence, that is, $A = B \vee A \equiv B$.
- *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 is calculated in PORSCE II using two methods:

1. The *edge-counting distance (ec)* is based on the observation that in the hierarchical, tree-like structure of an ontology, the further two concepts are placed, the less semantically related they are. Therefore, it computes the semantic distance of two concepts in terms of the number of edges *(p)* found on the shortest path between them in the ontology tree. 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. This value is the result of normalization of the number of edges to $[0\,..\,1]$ as $p/p_{\max}$, using the maximum *ec* distance ($p_{\max}$) found in the ontology. For performance purposes, $p_{\max}$ can be approximated as $p_{\max} = 2h - 1$, where h is the maximum number of edges between the root owl:Thing concept (T) and the furthest leaf.
2. 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 (Maedche & Zacharias, 2002). In PORSCE II, the upwards cotopic distance definition has been modified 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, thus ignoring concept set multiplicity. 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 are disjoint, then their distance equals 1; otherwise, if the two concepts have a hierarchical relationship, then $d_{uc}(A, B) \in [0\,..\,1)$. The upwards cotopic measure reflects the significance of the common ancestors of two concepts in the ontology hierarchy, based on the intuition that concepts with a great fraction of common ancestors among all their ancestors tend to be semantically related.

5.3 Semantic awareness and relaxation

After the steps of the translation process and the semantic analysis are both complete, the system is able to perform semantic awareness and semantic relaxation. This step is potentially essential, because the representation of the Web service composition as a planning problem is significantly empowered if the planning system is aware of semantic similarities among syntactically different concepts.

The implementation in PORSCE II involves enhancing the domain and problem description with all the required semantic information and consequently 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 preprocessing phase, prior to actual planning, the system uses the OOM in order to acquire all 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 subsection. 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 preconditions of each operator remain the same, while the list of add effects of each operator is enhanced by including all the equivalent and semantically similar concepts for the concepts in the initial list.

Suppose, for example, that the initial state I and the two operators of the problem are the following:

```
I = {debitcard(X), dates, motel}
ActivateCard: prec = {creditcard(X),disabled(X)},
              effects(+) = {enabled(X)}, effects(-) = {disabled(X)}
BookHotel: prec = {dates,hotel}, effects(+) = {bookinginfo}, effects(-) = {},
```

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

```
debitcard ~ creditcard  motel ~ hotel  active ~ enabled
```

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

```
EIS: {debitCard(X), dates, motel, creditCard(X), hotel}
EOS: ActivateCard: prec = {creditCard(X),disabled(X)},
                   effects(+) = {enabled(X),active(X)},
                   effects(-)={disabled(X)}
     BookHotel: prec = {dates,hotel}, effects(+) = {bookinginfo},
                effects(-) = {}
```

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 Planning Domain Definition Language to Web Ontology Language for Service translation

After the acquisition of solutions, a reverse translation process has to take place, in order to provide the resulting composite Web service to the original OWL-S standard and the initial Web services domain. This reverse translation accommodates composite service deployment and execution monitoring.

For the purposes of the PORSCE II framework, the use of intricate OWL-S control constructs 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 to define alternative paths. Moreover, all the plans produced by the planning systems contain a finite number of steps and the use of loops 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 nonlinear plan (i.e. one that contains steps with parallel execution of actions), since any nonlinear STRIPS 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 nonlinear) 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 ← {r ∈ V : ∀x ∈ V, (x → r) ∉ E} // R is the set of root nodes in G
2   if |R| = 0 then return NULL
3   if |R| = 1 then
      set G' ← the tree in G with r ∈ R as the root
      return sequence(r, Basic(G'−{r}))
4   set c ← {}
5   for each r in R
      set G' ← the tree in G with r ∈ R as the root
      set c ← c ∪ 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 Figure 4. The output of the Basic function presented in Algorithm 1 will be *split*(*sequence*(WS_a, *sequence*(WS_c, *NULL*)), *sequence*(WS_b, *sequence*(WS_c, *NULL*))).

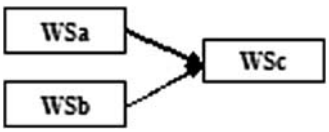

Figure 4 Web service graph example

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

```
1   set f(a1, a2, …, an) = C, where f is the name of the construct and a1 to an its arguments
2   if f = NULL then return NULL
3   if f = sequence then
      a'1 = Join(a1)
      a'2 = Join(a2)
      return f(a'1, a'2)
4   if f = split then
      for each pair (ai, aj), i, j in [1,n]
        if ai and aj have a common ending, i.e. ai = a'i ∪ k and aj = a'j ∪ k then
          C' = C−{ai, aj} ∪ seq(split + join(a'i, a'j), k)
          return C'
```

The output of the *Basic* function is then fed to the *Join* function, presented in Algorithm 2, in order to replace the *Split* 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 arguments of the *Split* construct end in *sequence*(WS_c, *NULL*). Therefore, if we apply Algorithm 2 to the output of Algorithm 1, the resulting composite service will be *split*(*sequence*(*split* + *join*(WS_a, WS_b), *sequence*(WS_c, *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*(WS_a, WS_b), WS_c).

The above algorithms, along with some additional filters, such as the one mentioned above for removing *NULLs* and unary constructs were implemented using the CMU OWL-S API (CMU, 2005).

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. This is a key issue for the ability of the system to keep up to date with advancements in planning research. Currently, two different planning modules have been incorporated in the system: Java Graphplan (JPlan, 2009), which is an open-source Java implementation of Graphplan and LPG-*td* (Gerevini *et al.*, 2004). 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 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, the produced 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, with respect to the kind of hierarchical relationship to the original concept. This provides the option to discriminate among different hierarchical relationships, to accommodate for cases when certain relationships might be more desirable than others. As a rough example, consider a user looking for 'zip_code' (subclass); in this case, superclasses, for example 'address' (which entails the zip code) are more desirable than siblings, for example 'street_number'; therefore, providing different weights for superclasses and siblings will have the desired effect on the results. These values are combined to form the plan semantic distance.

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 appropriately the semantic distance in this case:

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

The plan accuracy metric 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 Web 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 a certain 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 conditions $add(A) \supseteq add(Q)$ and $prec(A) \subseteq S$ must hold, where S is the state of the world exactly before the application of A and can be calculated by Algorithm 3 below (I_{rr}). The selected alternative service substitutes the original one both in the plan and in the visualization, and no replanning is performed. The Web service substitution can be applied an arbitrary number of times on any of the services taking part in the composite service. In cases when none of the semantically equivalent or relevant services that correspond to a certain service is considered suitable, or in cases where there are no alternative services, the system offers the option to substitute the service with a partially ordered set of 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 Web 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 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: Extended Initial State (EIS), plan $P = A_1, \ldots, A_n$, index of the action being replaced (i)
Output: The I_{rr}

1	**set** $I_{rr} \leftarrow EIS$	
2	**if** $j = 1$	// start at the beginning of the plan
3	**do**	// for every action
	$I_{rr} \leftarrow I_{rr} \cup add(A_j)$	// include the add effects of the action in the I_{rr} set
	$j = j + 1$	
4	**while** $j < i$	// until the action being replaced is reached
5	**return** I_{rr}	

Algorithm 4 Computes the goal state for replacement of action A_i through replanning (G_{rr})
Inputs: Initial Goal State (G), plan $P = A_1, \ldots, A_n$, index of the action being replaced (i)
Output: The G_{rr}

1	**set** $G_{rr} \leftarrow G$	
2	**if** $j = n$	// start at the end of the plan
3	**do**	// for every action
	$G_{rr} \leftarrow G_{rr} - add(A_j) \cup prec(A_j)$	// include preconditions and exclude the add effects
	$j = j-1$	
4	**while** $j > I$	// until the action being replaced is reached
5	**return** G_{rr}	

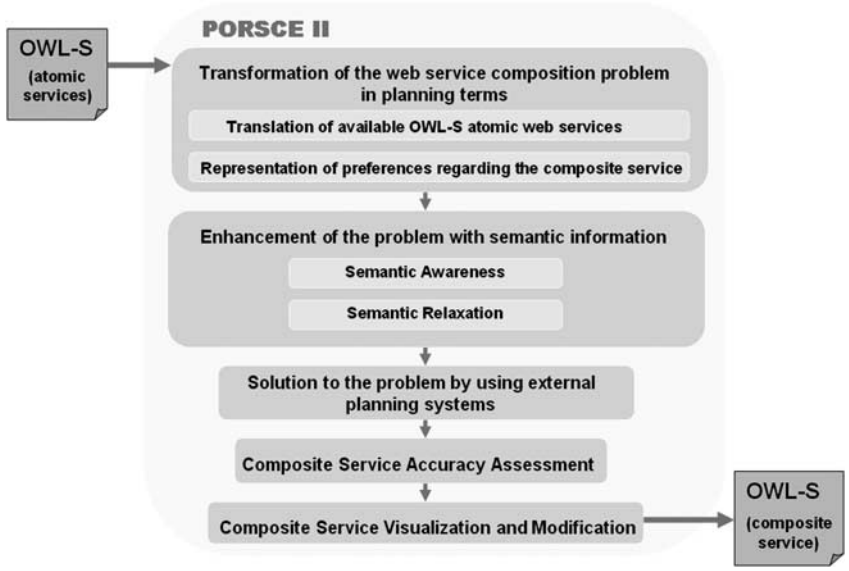

Figure 5 The demonstration steps. OWL-S = Web Ontology Language for Services

If replacement of a Web service, either by an equivalent or through replanning, is not a suitable option, or if multiple 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, Algorithm 3 is used to calculate the initial state.

7 Demonstration and system evaluation

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

The test sets used to perform experiments were obtained from the OWLS-TC (OWLS-TC, 2005). Several service descriptions were modified or added to the domains, accommodating demonstration of the capabilities of the system. Some indicative Web services that were modified or added are depicted in Table 2.

The transformation of the Web service composition problem in planning terms includes translating all available OWL-S 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 Figure 6.

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 his details (client), a book

Table 2 Added/modified web services

Service	Inputs	Preconditions	Outputs	Effects
BookToPublisher	Book, Author		Publisher	
CreditCardCharge	OrderData, CreditCard		Payment	
ElectronicOrder	Electronic		OrderData	
PublisherElectronic Order	PublisherInfo		OrderData	
ElectronicOrderInfo	Electronic		OrderInformation	
Shipping	Address, OrderData		ShippingDate	
WaysOfOrder	Publisher		Electronic	
CustomsCost	Publisher, OrderData		CustomsCost	
FindUsed	ISBN		Seller	has(Seller,ISBN)
OrderUsed	ISBN, Seller, Client	has(Seller,ISBN)	OrderData	-has(Seller,ISBN) has(Client,ISBN)
FindISBN	Book, Author		ISBN	

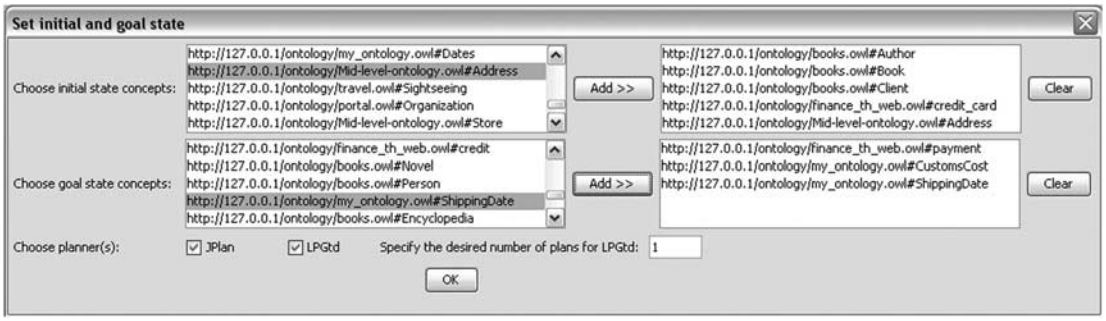

Figure 6 Initial and goal states definition and desired planners selection

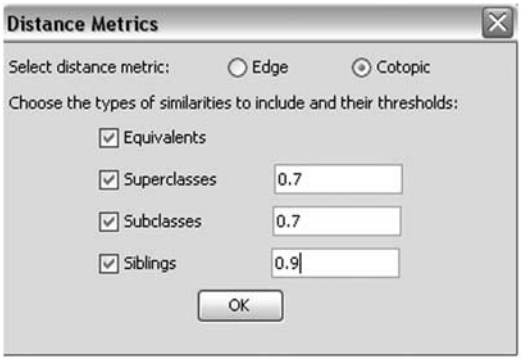

Figure 7 The semantic enhancement interface

title and author, credit card info and the address that the book will be shipped to, and wishes to use a credit card for the purchase, as well as to be informed about 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 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 Figure 7.

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)
 (ontologybooksowlISBN ?b)
 (ontologybooksowlSeller ?b)
 (ontologybooksowlClient ?b)
 (has_Seller_ISBN ?b)(has_Client_ISBN ?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))
   :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))
(:action FindUsed
   :parameters (?b)
   :precondition (and (block ?b)
     (ontologybooksowlISBN ?b))
   :effect (and (ontologybooksowlSeller ?b)
     (has_Seller_ISBN ?b)))
(:action FindISBN
   :parameters (?b)
   :precondition (and (block ?b)
     (ontologybooksowlBook ?b)
     (ontologybooksowlAuthor ?b))
   :effect (ontologybooksowlISBN ?b))
(:action OrderUsed
   :parameters (?b)
   :precondition (and (block ?b)
     (ontologybooksowlISBN ?b)
     (ontologybooksowlSeller ?b)
     (ontologybooksowlClient ?b)
     (has_Seller_ISBN ?b))
   :effect (and (ontologymyontologyowlOrderData ?b)
     (has_Client_ISBN ?b)(not (has_Client_ISBN ?b)))))
```

PROBLEM:

```
(define (problem wscompprob)
(:domain wscomp)
(:objects a)
(:init (block a)
(ontologybooksowlAuthor a)
(ontologybooksowlBook a)
(ontologyfinancethwebowlcreditcard a)
(ontologybooksowlClient a)
(ontologyMidlevelontologyowlAddress a))
(:goal (and
(ontologyfinancethwebowlpayment a)
(ontologymyontologyowlCustomsCost a)
(ontologymyontologyowlShippingDate a)
)))
```

PLANS (LPG-td):

```
; Version LPG-td-1.0
; Seed 52716286
; Command line: lpg-td-1.0 -o
domain.pddl -f problem.pddl -n 1
; Problem problem.pddl
; Actions having STRIPS duration
; Time 0.01
; Search time 0.00
; Parsing time 0.01
; Mutex time 0.00
; Quality 6

Time 0.01

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

OR

```
; Version LPG-td-1.0
; Seed 46626428
; Command line: lpg-td-1.0 -o
domain.pddl -f problem.pddl -n 1
; Problem problem.pddl
; Actions having STRIPS duration
; Time 0.03
; Search time 0.02
; Parsing time 0.02
; Mutex time 0.00
; Quality 6

Time 0.03

0:  (FINDISBN A) [1]
1:  (FINDUSED A) [1]
2:  (ORDERUSED A) [1]
3:  (CREDITCARDCHARGESERVICE A) [1]
3:  (CUSTOMSCOSTSERVICE A) [1]
3:  (SHIPPINGSERVICE A) [1]
```

Figure 8 The PDDL domain, problem and plan for the specific scenario. PDDL = Planning Domain Definition Language

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 Figure 8. Note that the domain in this case, for space purposes, contains only the necessary atomic Web services.

The produced plans are imported into the Visualizer Component, where they are represented as a Web service graph and depicted visually. The first plans produced by JPlan and LPG-*td* for this

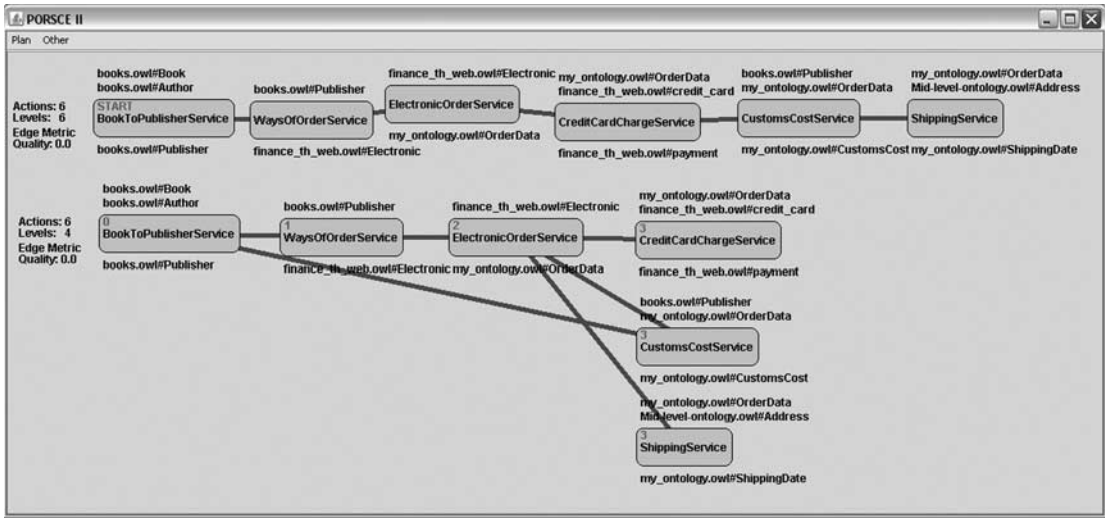

Figure 9 The plans from JPlan (top) and LPG-*td* (bottom) for the specific case study. JPlan = Java Graphplan

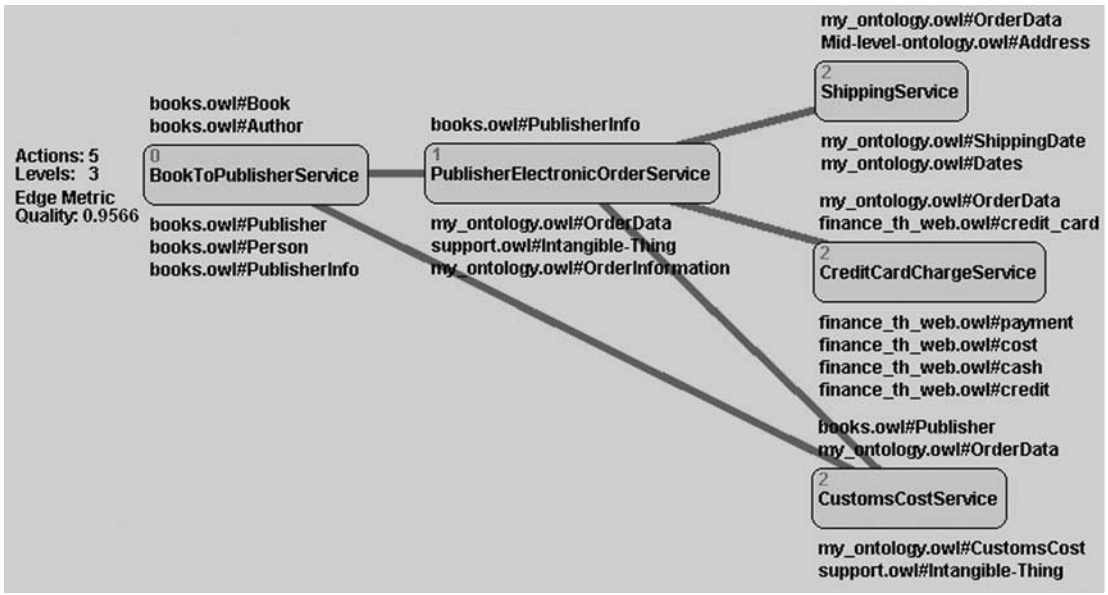

Figure 10 Approximate composite service

case study, using the operator set described above, without performing any semantic relaxation, are presented in Figure 9. The calculated statistics and metrics for the composite Web services include the number of actions and the number of levels in the plan (which coincide for sequential plans), as well as the plan accuracy metric.

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 OOM 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 Figure 10. The calculated accuracy of this service is different from the accurate ones presented in Figure 9.

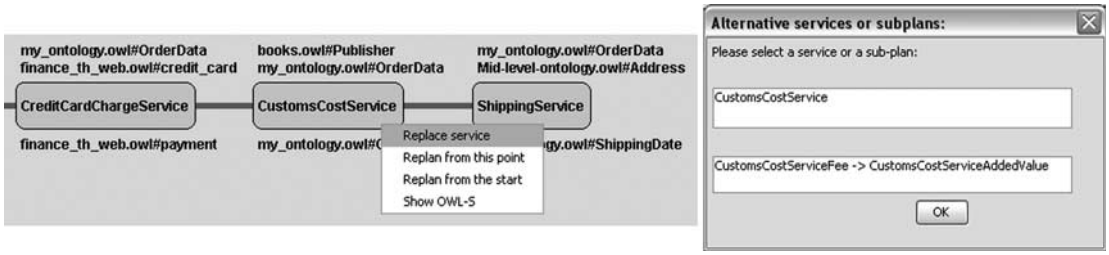

Figure 11 Service Replacement Interface

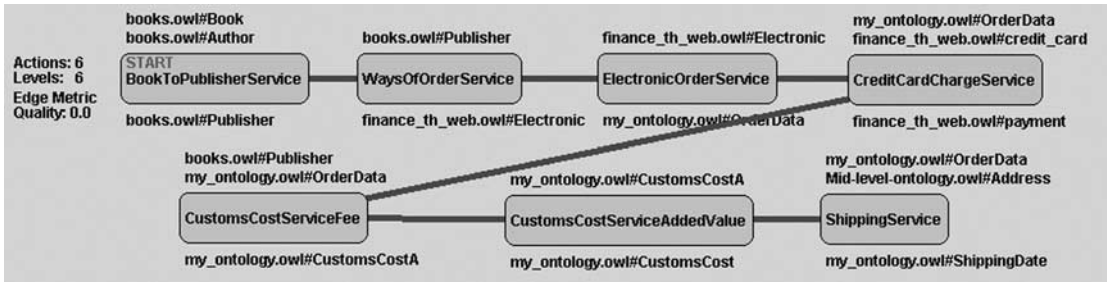

Figure 12 Composite service after replacement operation

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 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 Figure 11, while the resulting composite service after the modifications is depicted in Figure 12.

The final step is to transform the solution to Web service context. This is achieved by translating the PDDL+ plan that represents the desired composite Web service, produced by the external planning systems, into OWL-S, utilizing information retrieved from Web service descriptions and ontology 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 3 were obtained from a number of runs of the system on a machine with Dual-Core AMD Opteron Processor at 2.20 GHz with 1 GB of RAM memory and concern times for pre-processing, transformation of the OWL-S service profiles to PDDL actions and planning using LPG-*td*.

Measurements took place for domains of different sizes, namely 10, 100, 500 and 1000 OWL-S profiles. Some of the experiments were performed with exact matching, while others were performed with semantic relaxation using either the edge-counting or the upwards cotopic metric. The preprocessing time did not show significant fluctuation, as it depends 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 ~0.81 seconds for the exact matching and the edge-counting cases. In the upwards cotopic case, the increase in the average transformation time is significant as available Web services increase, due to the higher complexity of the algorithm used for calculating 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 remarkably fast. It should be noted that the scalability of the system as far as planning time is concerned is not critical, as the planners used are not embedded and could easily be replaced by more efficient ones.

Table 3 Time measurements in milliseconds

Number of web services	10	100	500	1000
Preprocessing time	5953	5794	6007	5903
Total transformation time				
Exact	4685	71 533	356 800	823 793
Edge	4635	76 558	346 212	820 700
Cotopic	4667	76 928	757 778	3 947 955
Transformation time per web service				
Exact	469	715	714	824
Edge	464	766	693	820
Cotopic	467	769	1516	3950
Planning time (LPG-*td*)				
Exact	4	17	50	119
Edge	6	16	72	123
Cotopic	4	15	56	122

8 Conclusions and future work

This paper presented PORSCE II, an integrated system that 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 accommodating approximate compositions and then transformed back in Web service terms. The system exploits the most prominent standards in both worlds, namely 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 that accommodate composite service evaluation and modification. Among the main advantages of the proposed framework are the extended utilization of semantic information, the ability to scale up for a great number of services and the capability to handle service failure or unavailability dynamically.

Future goals include the extension of the system in order to deploy the produced composite services, through OWL-S deployment systems, such as the OWL-S Virtual Machine (Paolucci *et al*., 2003), and automatically acquire feedback, which can then be utilized to partially automate the service replacement procedure. Also, another direction we plan to explore is the modification of the Web service modeling, so that a metric domain is constructed, which incorporates the concept semantic distances and requires the planner to minimize the total semantic distance. In addition, another future 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 Web services, under certain time constraints. Furthermore, integration with the VLEPPO system (Hatzi *et al*., 2007) 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

CMU. (2005). OWL-S API. Retrieved April, 17, 2010, from `http://www.daml.ri.cmu.edu/owlsapi/`.

Dustdar, S. & Schreiner, W. 2005. A survey on web services composition. *International Journal of Web and Grid Services* **1**(1), 1–30.

Fikes, R. E. & Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. In *IJCAI'71: Proceedings of the 2nd InternationalJoint Conference on Artificial Intelligence*, San Francisco, CA, USA, 608–620. Morgan Kaufmann Publishers Inc.

Fox, M. & Long, D. 2002. PDDL+: Modelling continuous time-dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, USA.

Gerevini, A. & Long, D. 2005. *Plan Constraints and Preferences in PDDL3*. Technical report, Department of Electronics for Automation, University of Brescia, Italy.

Gerevini, A., Saetti, A., Serina, I. & Toninelli, P. 2004. LPG-td: a fully automated planner for PDDL2.2 domains. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04) International Planning Competition*, Whistler, British Columbia, Canada.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. & Wilkins, D. 1998. *PDDL – the planning domain definition language*. Technical report, Yale University, New Haven, CT.

Hatzi, O., Meditskos, G., Vrakas, D., Bassiliades, N., Anagnostopoulos, D. & Vlahavas, I. 2008. A synergy of planning and ontology concept ranking for semantic web service composition. In *IBERAMIA '08: Proceedings of the 11th Ibero-American Conference on AI*, Geffner, H., Prada, R., Machado Alexandre, I. & David, N. (eds). Berlin, Heidelberg, 42–51. Springer-Verlag.

Hatzi, O., Meditskos, G., Vrakas, D., Bassiliades, N., Anagnostopoulos, D. & Vlahavas, I. 2009. PORSCE II: using planning for semantic web service composition. In *Proceedings of 3rd International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS'09), in conjunction with the International Conference on Automated Planning and Scheduling (ICAPS-09)*, Bartak, R., Frattini, S. & McCluskey, L. (eds), 38–45.

Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D. & Vlahavas, I. 2007. VLEPPO: A visual language for problem representation. In *PlanSIG 2007: The 26th workshop of the UK Planning and Scheduling Special Interest Group*, Bartak, R. (ed.), 60–66.

International Planning Competition (IPC). 2004. Retrieved October, 27, 2010, from `http://www.tzi.de/edelkamp/ipc-4/`.

Java Graphplan (JPlan). (2009). Java Graphplan Implementation. Retrieved October, 27, 2010, from `http://sourceforge.net/projects/jplan`.

Klusch, M. & Gerber, A. 2005. Semantic web service composition planning with OWLS-XPlan. In *Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 55–62.

Maedche, A. & Zacharias, V. 2002. Clustering ontology-based metadata in the semantic web. In *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, Elomaa, T., Mannila, H. & Toivonen, H. (eds). London, UK, 348–360. Springer-Verlag.

McDermott, D. 2002. Estimated-regression planning for interactions with web services. In *Proceedings of the 6th International Conference on Artificial Intelligence PlanningSystems*, Ghallab, M., Hertzberg, J. & Traverso, P. (eds). Toulouse, France, 204–211. AAAI Press.

Mcilraith, S. & Son, T. C. 2002. Adapting GOLOG for composition of semantic web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning*, Toulouse, France, 482–493.

OWL (Web Ontology Language). (2004). Retrieved October, 27, 2010, from `http://www.w3.org/TR/owl-ref/`.

OWL-S. (2004). 1.1. Retrieved October, 27, 2010, from `http://www.daml.org/services/owl-s/1.1/`.

OWLS-TC. (2005). SemWebCentral. Retrieved October, 27, 2010, from `http://projects.semwebcentral.org/projects/owls-tc/`.

Paolucci, M., Ankolekar, A., Srinivasan, N. & Sycara, K. 2003. The DAML-S virtual machine. In *The SemanticWeb—ISWC 2003*, Lecture Notes in Computer Science **2870**, 290–305. Springer.

Pistore, M., Marconi, A., Bertoli, P. & Traverso, P. 2005. Automated composition of web services by planning at the knowledge level. In *Proceedings of 19th International Joint Conferences on Artificial Intelligence*, San Francisco, CA, USA, 1252–1259.

Ponnekanti, S. R. & Fox, A. 2002. SWORD: a developer toolkit for web service composition. In *Proceedings of the 11th International WWW Conference (WWW2002)*, Lassner, D., De Roure, D. & Iyengar, A. (eds). Honolulu, HI, USA, 83–107. Elsevier.

Rao, J. & Su, X. 2004. A survey of automated web service composition methods. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, SWSWPC*, San Diego, CA, USA, 43–54.

Rule Markup Language (RuleML). (2001). The Rule Markup Initiative. Retrieved October, 27, 2010, from `http://ruleml.org/`.

Semantic Annotations for WSDL (SAWSDL). (2007). Retrieved October, 27, 2010, from `http://www.w3.org/2002/ws/sawsdl/`.

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

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. & Katz, Y. 2007. Pellet: a practical OWL-DL reasoner. *Journal of Web Semantics* **5**(2), 51–53.

Semantic Web Rule Language (SWRL). (2004). A Semantic Web Rule Language. Retrieved October, 27, 2010, from `http://www.w3.org/Submission/SWRL/`.