Visual Representation of Web Service Composition Problems through VLEPpO

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

[1]*Department of Geography, Harokopio University of Athens, Athens, Greece*
{raniah, dimosthe}@hua.gr

[2]*Dept. Of Informatics, Aristotle University Of Thessaloniki, Thessaloniki, 54124, Greece*
{dvrakas, nbassili, vlahavas}@csd.auth.gr

Abstract

This paper discusses the problem of the automatic composition of Semantic Web Services. Web Services constitute a new computing paradigm, which provides a standardized framework that facilitates the interoperability among software systems and machines that are accessible through the Internet. Semantics can significantly improve software reuse and discovery and allow the automatic composition of Web Services in order to produce large scale applications.

The use of VLEPpO for the automatic composition of Web Services is proposed. VLEPpO is a visual programming tool for designing planning problems using visual elements and simple mouse operations. In the tool the user simply defines the properties of the available Web Services and the global goals of the application. Then VLEPpO automatically forms the description as a planning problem, solves it by calling an appropriate planning system and exports the solution either to a Web Service execution monitoring system or to a UDDI registry.

1. Introduction

Currently, applications and services in the web are coming to the forefront. Web services play a crucial role as they become the basic components of web-based applications [1]. The use of web services is expanding rapidly to provide a systematic and extensible framework for application-to-application interaction, built on top of existing web protocols and based on open XML standards. Web services aim to simplify the process of distributed computing by defining a standardized mechanism to describe, locate, and communicate with online software systems. Essentially, this technology enables any application to become an accessible web service component.

A web service is a software system identified by a URI (Universal Resource Identifier), which is defined and described using XML-based languages such as WSDL (Web Service Description Language) [2]. WSDL describes public interfaces and bindings of web services, while web services themselves can be viewed as remote, platform-independent implementations of these interfaces. In particular, WSDL defines the supported operations of a web service, as well as the way of exchanging data through messages. WSDL is confined only to syntactical information, excluding any semantics.

Web services can be discovered through UDDI (Universal Description, Discovery and Integration) registries. UDDI is a standard interoperable platform that enables companies and applications to quickly, easily, and dynamically locate and use web services over the Internet [4]. UDDI registries contain records of available web services and provide not only technical information and access to WSDL documents, but service categorization as well.

Software systems or agents who discover a web service through a UDDI registry may interact with it, in a manner prescribed by its definition, using SOAP (Simple Object Access Protocol) messages [5] conveyed by internet protocols such as HTTP [2]. SOAP is XML-based and provides the fundamental messaging framework for discovering and communicating with web services.

The current WSDL standard operates at the syntactic level and lacks the semantic expressivity needed to represent the requirements and capabilities of Web Services. Semantics can improve software reuse and discovery, significantly facilitate composition of Web services and enable integration of legacy applications as part of business process integration [7]. Therefore, the need for new languages who accomodate the semantic aspects of web services has emerged. Such a language is WSDL-S which constitutes a lightweight approach for adding semantics to web services [3].

In order for semantically enhanced web services to be computer comprehensible and processable, support by ontologies is required. OWL-S (previously known as DAML-S) is an OWL-based web service ontology which supplies web service providers with a core set of markup language constructs for describing the properties and capabilities of web services in unambiguous, computer-intepretable form [6].

When individual Web Services are limited in their capabilities, they can be composed to create new functionality in the form of Web Processes. Web Service composition is the ability to take existing services (or building blocks) and combine them to form new services [8] and is emerging as a new model for automated interactions among distributed and heterogeneous applications. In order to truly integrate application components on the Web across organization and platform boundaries, merely supporting simple interaction using standard messages and protocols is insufficient [9] and Web services composition languages, such as WSFL [10], XLANG [11] and BPEL4WS [12], are needed to specify the order in which WSDL services and operations are executed.

The aforementioned languages and approaches cope with the web services composition problem in a primarily syntactical way, and interaction between services is manually defined. Due to that, composing web services will become harder as the available web services increase. Therefore, automating web service composition is essential. Semantic web services try to provide a solution by employing Artificial Intelligence techniques for web service composition. A very promising direction is the use of planning techniques[14].

The contribution of this work aims at exploiting planning in order to provide solution to the web service composition problem, as well as providing an efficient planning environment oriented to web service composition. Therefore, a visual tool which enables the design and solving of planning problems has been developed. The tool can serve as a general purpose planning system, while at the same time it facilitates the requirements of planning for web service composition.

The rest of the paper is organised as follows: in Section 2 issues related to web service composition as a planning problem are discussed. Section 3 presents the visual tool developed to accomodate visual design of planning problems, while Section 4 provides a case study concerning the use of the tool to represent a web service composition problem. Finally, Section 5 concludes and presents future goals.

2. Web Service Composition as a Planning Problem

In order to employ planning, a web service composition problem must be reflected to a planning problem. The desired outcome of the complex service is described as a goal state, while simple web services play the role of planning operators, or actions. The planner then will be responsible for finding an appropriate plan, i.e. an appropriate sequence of simple web service invocations, to achieve the goal state [13]. The produced plan will eventually constitute the description of the complex service.

An important benefit of the planning approach in general is the exploitation of knowledge that has been accumulated over years of research on the field of planning. Therefore, well known planning algorithms, techniques and tools can be used to the advantage of efficient and seamless web service composition.

There are many issues to be tackled both from the planning community and the web service community in order to handle web service composition as a planning problem [15]. Some of them are the following:

- New expressive languages for representing web service actions, unifying existing standards of PDLL [16] and OWL-S [20].
- Efficient planners which produce quality plans that synthesize complex web services.
- Web Service plan verification [21].
- Efficient Web Service plan execution [22].
- Monitoring web service plan execution and repairing the plan in cases where web service execution failed [23].
- Mixing information retrieval and plan execution [24][25].

A slightly different approach within the same field is HTN (Hierarhical Task Network) planning. In this case, the desired outcome is still described as a goal state, but the planner tries to supply a solution by decomposing the goal into smaller ones. The process reaches an end when appropriate simple web services which satisfy these goals have been found.

This technique enables new web services to be created on demand, offering non-predetermined functionality to the users. Furthermore, as the entire process is automatic and dynamic, cases of service failures can be handled. Therefore, if a web service invoked at some step of a plan is unavailable, an alternative plan can be generated and executed.

In [26] and [27] a transformation method of OWL-S processes into a hierarchical task network is presented. OWL-S processes are, like Hierarhical Task Networks, pre-defined descriptions of actions to be carried out to get a certain task done, which makes the transformation rather natural. The advantage of the approach is its ability to deal with very large problem domains; however, the need to explicitly provide the planner with a task it needs to accomplish may be seen as a disadvantage, since this requires descriptions that may not always be available in dynamic environments.

Another approach in using planning techniques for Semantic Web Service composition is in [28], where the planner uses services as STRIPS operators to compose a plan, given the goal and a set of basic services. The Java Expert Shell System (JESS) [29] has been used to implement the planner and a set of JESS rules that translate DAML-S (a precursor to OWL-S) descriptions of atomic services into planning operators.

In order to enable the construction of composite web services, a number of composition languages have been proposed by the software industry. However, the handiwork of specifying a business process with these languages through simple text or XML editors is tough, complex and error prone. Visual support can ease the definition of business processes.

So far, a few visual tools that accomodate web service composition mainly through the use of BPEL4WS have appeared [30][31]. However, to the best of our knowledge, there is no visual tool that combines visual planning problem design with web service composition.

3. The VLEPpO Tool

VLEPpO (Visual Language for Enhanced Planning Problem Orchestration) is an integrated system for visually designing and solving planning problems. It offers an efficient and easy-to-use graphical interface, as well as compatibility and interoperability with PDDL (Planning Domain Definition Language), which is considered to be a standard for the definition of planning domains and corresponding problems [16][17][18][19]. The implementation language chosen is Java, therefore the tool adopts Java's portability and reliability. Further advantages that justify the choice of this language are its ability to handle graphical interfaces and its convenience when it comes to managing web services.

The main feature of the tool is the visual representation of planning domains and planning problems. Coloured shapes which correspond to various elements of PDDL are used, and ontologies which reflect the structure of a domain are created. These ontologies are combined with operators to express a planning domain in terms of the PDDL language. Furthermore, planning problems can be created based on this domain. The tool guides the designer in order to avoid mistakes and inconsistencies.

The system imports the definitions of simple, atomic web services expressed in OWL-S and translates them to planning operators. Inputs and preconditions of OWL-S web services are treated as relations to be queried in the precondition list, while outputs are treated as atoms to be added through the operator's add list. Finally, effects are also atoms to be either added through the add-list or deleted, through the delete-list. In the following case study, we assume the existence of only information requesting web services that just return results (outputs) to be consumed as inputs by other web services. However, this does not limit our tool only to information providing web services.

The system exports the plan of web service compositions into OWL-S definitions of composite web services (processes). Since plans are partially ordered, the composite processes control constructs can include sequences of actions (atomic web services) (Fig. 1a), splits of actions (or sequences of actions) (Fig. 1b), splits-joins of actions (or sequences of actions) (Fig. 1c), or combinations of the above (Fig. 1d).

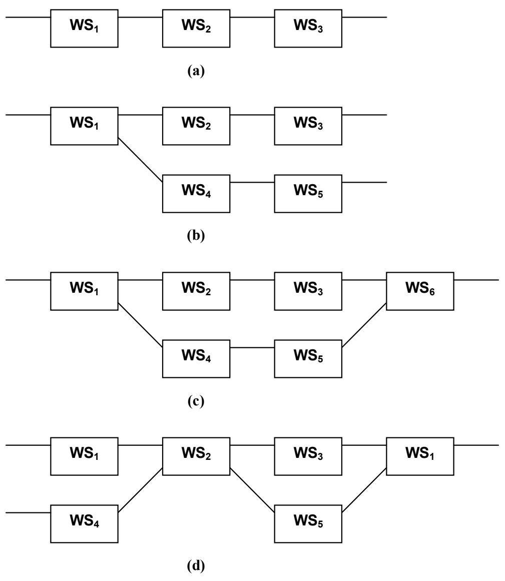

Figure 1. Various web service composite processes

The main goal during the implementation of the graphical component of the tool was to keep the interface as simple and efficient as possible, but, at the same time, represent accurately the features of PDDL. The range of PDDL elements that can be represented in the tool is quite wide, and covers the elements that are used more frequently in contemporary planning domains and problems.

Additional features of the tool include exporting the domains and problems to PDDL, as well as importing and visualising existing domains and problems from PDDL. Therefore, the designer is released by the requirement to be familiar with the language syntax, and is offered the ability to manipulate domains and problems in a more intuitive way.

Finally, in order to accomodate the interoperability of the system with other planning systems that actually perform the planning procedure, a web service client component has been provided. Thus, the user is not restricted to a single planning algorithm, but has the ability to experiment.

4. Case Study

In this chapter, a web service composition example will be presented, in order to demonstrate the expression of web service aspects in terms of planning, and the use of VLEPpO for the visualization of these elements.

The problem at hand is the specification of directions between two residences, whose occupants are specified by their full names and ID numbers. The solution to this problem will be provided by a composite web service which accepts as input the occupant names and IDs, uses the corresponding phone numbers and addresses and produces a list of directions, taking into account information about bus and metro routes. This web service will be formed by the composition of several simple web services, each offering partially the required functionality.

In this case, the web service composition problem is reflected to a planning problem. Therefore, elements such as names and addresses are represented as predicates in the planning domain, while each simple web service is expressed as an operator. Moreover, the composed web service is represented by the produced plan. Finally, the automated web service composition will take place by means of applying well known planning algorithms.

In the following, details about the example and screenshots of the visual tool will be presented to demonstrate the visualization of the aforementioned elements.

Relations, or *predicates* as they are also referred to in PDDL, are used to represent elements of the planning domain such as persons, addresses, phone numbers and areas. For each person, the elements that are considered known, and for which corresponding classes exist, are their first and last name, and their ID number. Likewise, the elements concerning addresses are street name, number and postal code. As far as phones are concerned, area codes and the actual phone numbers are represented, while for areas, larger and specific are the characteristics of interest. In addition, other predicates were used to represent directions, metro information and bus information; however these are not demonstrated here as they are trivial, due to the lack of arguments.

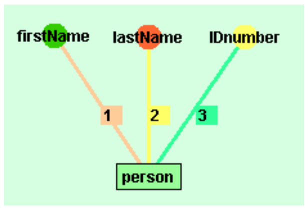

Figure 2. The person relation

In Fig. 2-4 visual representation of the aforementioned predicates and their arguments is demonstrated. Predicates are represented by rectangles, while classes are represented by circles. The corresponding PDDL code for these predicates is demonstrated in Fig. 5. In this case, typed PDDL is used, although the tool has the ability to export to non-typed PDDL as well.

In the Appendix we show how this information is represented in an OWL ontology, which automatically imported and translated to PDDL representation.

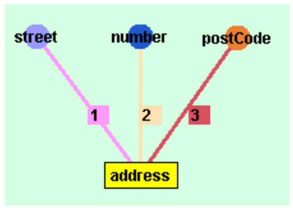

Figure 3. The address relation

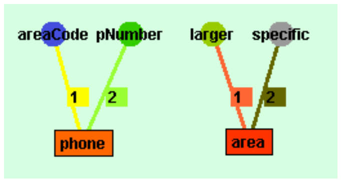

Figure 4. The phone and area relations

```
(person ?firstName1 - firstName ?lastName2 - lastName
                                        ?IDnumber3 - IDnumber)
 (address ?street1 - street ?number2 - number
                                        ?postCode3 - postCode)
 (phone ?areaCode1 - areaCode ?pNumber2 - pNumber)
 (area ?larger1 - larger ?specific2 - specific)
```
Figure 5. PDDL representation

As already mentioned, simple web services are represented as operators, and the elements mentioned above play the role of preconditions and results of these operators. Simple web services are imported from OWL-S descriptions, such as the one shown in the Appendix, which corresponds to the operator findAreafromCodes (Fig. 9 & 13). The visual notation used for representing operators is a rectangle divided into three parts. The items on the left part constitute the preconditions that must hold for the operator to be applied, the right part contains the results after the application of the operator, while the middle part holds parameters of the operator.

In the following, the available web services and the corresponding screenshots of their representations will be presented (Fig. 6-12). The available web services include:

- An address lookup service, which provides addresses of people given their names and IDs.
- Another address lookup service, which again provides addresses, given phone numbers of the occupants.
- A phone number lookup service which accepts as input the personal data of a person and provides their phone numbers.

- An area specification service, which designates the area a person lives in, based on their postal codes and phone area codes.
- Metro information and bus information services, which offer information about metro and bus routes, respectively.
- A web service which determines the most suitable directions between two specified locations, considering information about metro and bus routes.

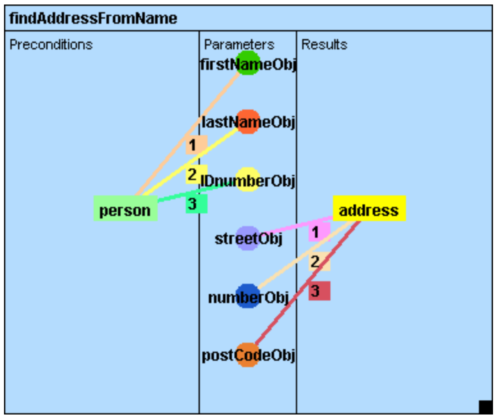

Figure 6. The findAddressFromName operator

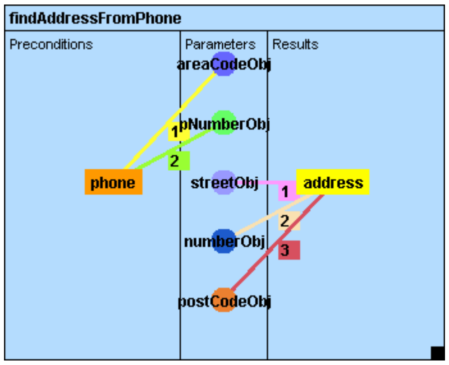

Figure 7. The findAddressFromPhone operator

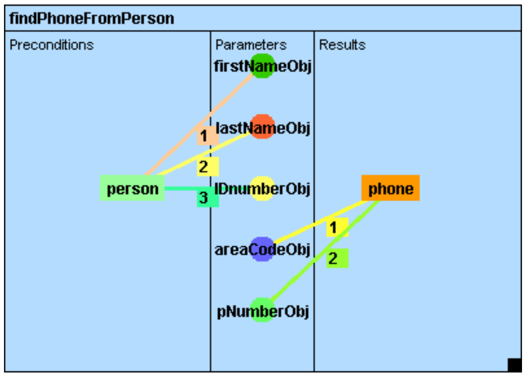

Figure 8. The findPhoneFromPerson operator

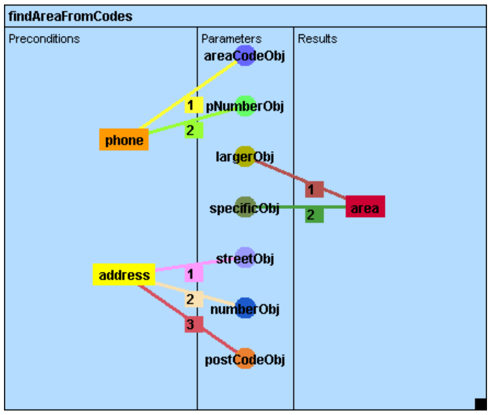

Figure 9. The findAreaFromCodes operator

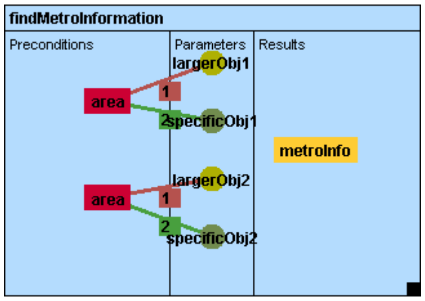

Figure 10. The findMetroInformation operator

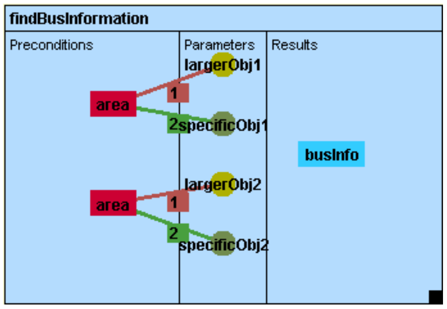

Figure 11. The findBusInformation operator

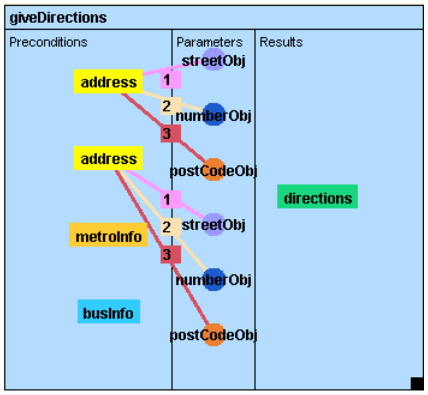

Figure 12. The giveDirections operator

At any point during the creation of the domain, the ability to export it to PDDL is offered. Indicative PDDL code for one of the aforementioned operators is demonstrated in Fig. 13.

```
(:action findAreaFromCodes
  :parameters ( ?areaCodeObj - areaCode
                ?pNumberObj - pNumber
                ?largerObj - larger
                ?specificObj - specific
                ?streetObj - street
                ?numberObj - number
                ?postCodeObj - postCode )
  :precondition (and (phone ?areaCodeObj ?pNumberObj)
                     (address ?streetObj ?numberObj
                              ?postCodeObj))
  :effect (and (area ?largerObj ?specificObj)
               (not (phone ?areaCodeObj ?pNumberObj))
               (not (address ?streetObj ?numberObj
                             ?postCodeObj))))
```

Figure 13. PDDL representation of the findAreaFromCodes operator

So far, the planning domain has been created and the inputs and outputs of the web services have been modelled. Corresponding planning problems can thus be expressed, which will provide the actual data for planning algorithms to be executed. Therefore, a plan which represents the web service composition will be produced. Such a problem is demonstrated below. The task is to find directions between the residences of two of the authors, given their names and phone numbers.

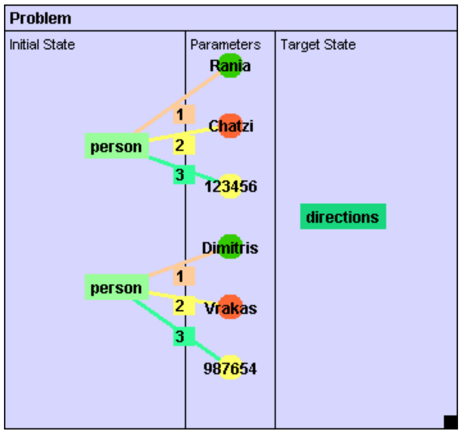

Figure 14. A typical problem

A planner will process the aforementioned information to provide a plan, which will in turn be a description of the complex web service. Alternative plans are possible, resulting in different complex web services. The final plan depends not only on the available services, but also on the planner of choice, as different algorithms may be used, and different heuristics and cost functions may be taken into account.

In the following, the shortest plan along with an alternative one will be provided for completeness. In the context of this example, the shortest plan is the plan that involves fewer simple web services than any other plan. The actions of the plan are:

1. findAddressFromName (for first person)
2. findAddressFromName (for second person)
3. findPhoneFromPerson (for first person)
4. findPhoneFromPerson (for second person)
5. findAreaFromCodes (for first person)
6. findAreaFromCodes (for second person)
7. findMetroInformation
8. findBusInformation
9. giveDirections

Figure 13. A possible plan

Altough the actions are mentioned in a specific order, the above plan is in fact partially ordered, that is, as long as certain restrictions hold, the actions may be executed in any order. In particular, these restrictions include findAreaFromCodes actions to be executed after corresponding findAddressFromName and findPhoneFromPerson actions. Furthermore, both findAreaFromCodes actions included in the plan must be executed before findMetroInformation and findBusInformation actions. Finally, the action that will be executed last is giveDirections, which will result in reaching the desired final state. The ordering of the actions of the plan and the ability to execute actions in different order depends on the planner of choice.

An alternative plan with the same length that achieves the desired goal state is the following:

1. findPhoneFromPerson (for first person)
2. findPhoneFromPerson (for second person)
3. findAddressFromPhone (for first person)
4. findAddressFromPhone (for second person)
5. findAreaFromCodes (for first person)
6. findAreaFromCodes (for second person)
7. findMetroInformation
8. findBusInformation
9. giveDirections

Figure 14. An alternative plan

Although the plan remains partially ordered, more restrictions than before must hold. Moreover, in larger domains, even more plans with varying qualities could be produced. In classic planning problems, the best plan would be chosen according to some criteria. However, in the web service domain, the ability to select between diffent plans, therefore different complex web services, is essential, as the stability and availability of simple web services cannot be assumed.

5. Conclusions and Future Work

In this paper several aspects of the web service composition issue were examined. In particular, approaching the problem using Artificial Intelligence planning techniques was described, and the visual tool VLEPpO, which was developed for this purpose, was presented. The tool is capable of describing a wide range of planning domains and problems, as most of the elements of the PDDL language can be visually represented and manipulated. Furthermore, a case study was provided in order to demonstrate the use of the tool and the process of composing web services through planning.

Our future goals include extending the tool in order to enhance its expressive ability. This will result in more efficient definitions of web service composition requirements. Specifically, including HTN planning elements and constructs will enable the description of the desired results of a composed service in terms of complex goals, which can be decomposed into simpler goals until web services which satisfy them can be found. It is estimated that HTN planning will bring the tool closer to semantic web service composition.

Moreover, another improvement will be the visual representation of produced plans. The plan readability will be significantly increased and comparisons between different plans will be possible. As far as plan comparisons are concerned, plans metrics can also be included in the tool, thus offering a wider range of options to the designer.

Acknowledgements

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03EΔ73), jointly funded by the European Union and the Greek government (General Secretariat of Research and Technology).

References

[1] Leymann F., "Web Services: Distributed Applications without Limits - An Outline", Proc. Database Systems for Business, Technology and Web (BTW 2003), Weikum G., Schöning H., Rahm E., (Eds.), GI-Edition - Lecture Notes in Informatics (LNI), P-26, Bonner Köllen Verlag, 2003.

[2] Booth D. et al., "Web Services Architecture", W3C Working Draft, Aug. 2003. http://www.w3.org/TR/ws-arch/

[3] John Miller, Kunal Verma, Preeda Rajasekaran, Amit Sheth, Rohit Aggarwal, Kaarthik Sivashanmugam, “WSDL-S: Adding Semantics to WSDL - White Paper”, July 15, 2004.

[4] Universal Description, Discovery and Integration (UDDI). Available online via http://www.uddi.org/ [accessed: May 23, 2007]

[5] Simple Object Access Protocol (SOAP). Available online via http://www.w3.org/TR/soap/ [accessed: May 23, 2007]

[6] OWL-S 1.0 Release. Available online via http://www.daml.org/services/owl-s/1.0/ [accessed: May 25, 2007]

[7] Web Service Semantics - WSDL-S. Available online via http://www.w3.org/Submission/WSDL-S/ [accessed:May 23, 2007]

[8] Piccinelli G., "Service Provision and Composition in Virtual Business Communities", Tech. Report HPL-1999-84, Hewlett-Packard, Palo Alto, CA, 1999. http://www.hplhp.com/techreports/1999/HPL-1999-84.html

[9] Aalst W. van der, "Don't Go with the Flow: Web Services Composition Standards Exposed", IEEE Intelligent Systems, Vol. 18, No. 1, pp. 72-76, 2003.

[10] Leymann F., "Web Services Flow Language (WSFL 1.0)", IBM, May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[11] Thatte S., "XLANG: Web Services for Business Process Design", Microsoft, Redmond, Wash., 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[12] Thatte S. (ed.),"Business Process Execution Language for Web Services (Version 1.1)", May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel

[13] Carman M., Serafini L., Traverso P., "Web Service Composition as Planning", ICAPS 2003 Workshop on Planning for Web Services.

[14] Milani Alfredo, "Planning and Scheduling for the Web Roadmap", Technical Coordination Unit for Planning and Scheduling for the Web, PLANET Network of Excellence, March 2003, http://www.dipmat.unipg.it/~milani/webtcu/

[15] Koehler J., Srivastava B., "Web Service Composition: Current Solutions and Open Problems", ICAPS 2003 Workshop on Planning for Web Services, pp. 28-35.

[16] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. and Wilkins, D., "PDDL -- the planning domain definition language". Technical report, Yale University, New Haven, CT (1998).

[17] Fox, M. and Long, D., "PDDL2.1: An extension to PDDL for expressing temporal planning domains". Journal of Artificial Intelligence Research, 20 (2003), 61-124.

[18] Edelkamp, S. and Hoffmann, J., "PDDL 2.2: The Language for the Classical Part of IPC-4", in Proceedings of the

International Planning Competition International Conference on Automated Planning and Scheduling (Whistler 2004), 1-7.

[19] Gerevini, A. and Long, D., "Plan Constraints and Preferences in PDDL3", Technical Report R.T. 2005-08-47, Department of Electronics for Automation, University of Brescia, Italy.

[20] OWL Services Coalition, "OWL-S: Semantic Markup for Web Services", OWL-S 1.0 Release, 2003, http://www.daml.org/services/owl-s/1.0/

[21] Berardi D., Calvanese D., De Giacomo G., Mecella M., "Reasoning about Actions for e-Service Composition", ICAPS 2003 Workshop on Planning for Web Services.

[22] Varela M. L. R., Aparicio J. N., do Carmo Silva S., "A Scheduling Web Service based on XML-RPC", ICAPS 2003 Workshop on Planning for Web Services.

[23] Blythe J., Deelman E., Gil Y., "Planning for workflow construction and maintenance on the Grid", ICAPS 2003 Workshop on Planning for Web Services.

[24] Thakkar S., Knoblock C., Ambite J.L., "A View Integration Approach to Dynamic Composition of Web Services", ICAPS 2003 Workshop on Planning for Web Services.

[25] Kuter U., Sirin E., Parsia B., Dana N. and James H., "Information gathering during planning for Web Service composition", Web Semantics: Science, Services and Agents on the World Wide Web, 3(2-3), pp. 183-205, 2005.

[26] Sirin E., Parsia B., Wu D., Hendler J. and Nau D., "HTN planning for Web Service composition using SHOP2", Web Semantics: Science, Services and Agents on the World Wide Web, 1(4), pp. 377-396, 2004.

[27] Wu D., Sirin E., Hendler J., Nau D., Parsia B., "Automatic Web Services Composition Using SHOP2", ICAPS 2003 Workshop on Planning for Web Services.

[28] Sheshagiri M., des Jardins M., Finin T., "A Planner for Composing Services Described in DAML-S", ICAPS 2003 Workshop on Planning for Web Services.

[29] Friedman-Hill, E. J., "Jess, The Expert System Shell for the Java Platform", 2002, http://herzberg.ca.sandia.gov/jess/

[30] Liu, N., Grundy, J.C., and Hosking, J.G., "A Visual Language and Environment for Composing Web Services", In Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering, Long Beach, California, Nov 7-11 2005, IEEE Press

[31] Martinez, A., Pérez-Sorrosal, F., Patiño-Martínez, M., Jiménez-Peris, R., "ZenFlow: A Visual Tool for Web Service Composition", IEEE Symp. on Visual Languages and Human-Centric Computing. Dallas, Texas. Sept. 2005

Appendix – OWL-S Case Study

Here we include the predicates and the simple web service (operator) findAreaFromCode of the case study presented in Section 4.

```
<!DOCTYPE rdf:RDF [
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY ex "http://lpis.csd.auth.gr/find-directions.owl#" >
]>

<rdf:RDF xml:base="http://lpis.csd.auth.gr/find-directions.owl"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:process="http://www.daml.org/services/owl-s/1.0DL/
                                                    Process.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://www.daml.org/services/
                                  owl-s/1.0DL/Service.owl"/>
        <owl:imports rdf:resource="http://www.daml.org/services/
                                  owl-s/1.0DL/Process.owl"/>
    </owl:Ontology>

    <owl:Class rdf:ID="Predicate"/>
    <owl:Class rdf:ID="Address">
        <rdfs:subClassOf rdf:resource="#Predicate"/>
    </owl:Class>
    <owl:Class rdf:ID="Area">
        <rdfs:subClassOf rdf:resource="#Predicate"/>
    </owl:Class>
    <owl:Class rdf:ID="Phone">
        <rdfs:subClassOf rdf:resource="#Predicate"/>
    </owl:Class>

    <owl:DatatypeProperty rdf:ID="pNumber">
        <rdfs:domain rdf:resource="#Phone"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="areaCode">
        <rdfs:domain rdf:resource="#Phone"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="postCode">
        <rdfs:domain rdf:resource="#Address"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="street">
        <rdfs:domain rdf:resource="#Address"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="number">
        <rdfs:domain rdf:resource="#Address"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="specific">
        <rdfs:domain rdf:resource="#Area"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="larger">
        <rdfs:domain rdf:resource="#Area"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>

    <process:Input rdf:ID="Phone-In">
        <process:parameterType rdf:datatype="&xsd;anyURI"
                               >&ex;Phone</process:parameterType>
    </process:Input>
    <process:UnConditionalOutput rdf:ID="Area-Out">
        <process:parameterType rdf:datatype="&xsd;anyURI"
                                >&ex;Area</process:parameterType>
    </process:UnConditionalOutput>

    <process:AtomicProcess rdf:ID="findAreaFromCodes">
        <process:name
rdf:datatype="&xsd;string">findAreaFromCodes</process:name>
        <process:hasInput rdf:resource="#Phone-In"/>
        <process:hasInput rdf:resource="#Address-In"/>
        <process:hasOutput rdf:resource="#Area-Out"/>
    </process:AtomicProcess>

</rdf:RDF>
```