Web Services for Adaptive Planning

G. Tsoumakas, G. Meditskos, D. Vrakas, N. Bassiliades, I. Vlahavas
Dept. of Informatics, Aristotle University of Thessaloniki, 54124 Greece

Abstract. This paper presents the design and development of an adaptive planning system using the technology of Web services. The Web-based adaptive planning system consists of two modules that can work independently. The first one is called HAP-WS and is the Web service interface to the domain independent planner HAP (Highly Adjustable Planner) that can be customized through the adjustment of several parameters, either manually or automatically. In the manual mode, the user itself adjusts planner parameters giving explicitly the values. In the automatic mode, the second subsystem, called LAMP-WS, computes the values of the planning parameters of HAP. LAMP-WS is the Web service interface to the learning system LAMP (Lazy Adaptive Multicriteria Planning) that can automatically configure a planning system using instance-based learning on past performance data of that system. The two subsystems are implemented as independent Web services, which can be used stand-alone and reside in different servers in potentially different geographical locations.

1 Introduction

Web services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where Web services are becoming the platform for application integration. Applications are constructed using multiple Web services from various sources that work together regardless of where they reside or how they were implemented.

Planning has an important role to play in the orchestration of Web services. It can be used to construct efficient execution plans of multiple Web services for the achievement of a complex task, by viewing service composition as a planning problem [4]. By doing so, various already available planning techniques and systems can be used to tackle Web service composition, probably with the addition of a few new techniques. The ability to perform automated service composition would revolutionize many application areas for Web service technology including e-commerce and systems integration.

Within this scenario, planning systems could also be deployed themselves as Web services, allowing their interoperation with other information integration Web services and incorporation into larger Web Information Management Systems. To the best of our knowledge, no planner has yet been provided as a Web service, with the exception of O-Plan [10]. Although O-Plan offers its functionality to users and programs on the World Wide Web, it uses the outdated technology of CGI scripts. Therefore, O-Plan cannot be dynamically discovered and invoked by users or other programs. They would have to know apriori its Internet address and the specifications of its input and output. Inevitably, bridging theory to practice requires the use of technology that is based on open standards, such as those adopted by the Web services architecture.

This paper presents the design and development of a Web-based system for Adaptive Planning that utilizes the technology of Web services. The system consists of two independent Web services: HAP-WS is a Web service interface to the HAP (Highly Adjustable Planner) planning system [14]. A user or program can directly use HAP-WS by providing it the planning domain and problem definition files, and optionally the values for its 7 planning parameters. LAMP-WS is a Web service interface to the LAMP (Lazy Adaptive Multicriteria Planning) learning system [12] that predicts a configuration for the planning parameters of HAP based on the characteristics of the given planning problem. LAMP-WS can also be used directly by a user or program for consultation on a suitable parameter configuration for the HAP planner. For automatically configuring HAP, a combination of the two services is required, which can be easily handled by a suitable client application. Such an application example developed in ASP.NET is also presented in this paper.

The rest of the paper is organized as follows. Section 2 presents a brief reference to the technologies that the Web service architecture entails. Section 3 introduces the concept of Adaptive Planning and related work. Section 4 describes the Web services that were developed for the purposes of Adaptive Planning and Section 5 presents an example ASP.NET application that uses these Web services. Finally the last section concludes this work and points areas for improvements.

2 Background

A Web service is a software system identified by a URI (Uniform Resource Identification), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols [5].

The use of the Web services paradigm is expanding rapidly to provide a systematic and extensible framework for application-to-application (A2A) 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, each application becomes an accessible Web service component that is described using open standards. The basic architecture of Web services includes technologies capable of:

- Exchanging messages.
- Describing Web services.
- Publishing and discovering Web service descriptions.

2.1 Exchanging messages

The standard protocol for communication among Web services is the Simple Object Access Protocol (SOAP) [2]. SOAP is a simple and lightweight XML-based mechanism for creating structured data packages that can be exchanged between network applications. SOAP consists of four fundamental components: an envelope that defines a framework for describing message structure, a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls and responses, and a set of rules for using SOAP with HTTP. SOAP can be used with a variety of network protocols, such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol.

SOAP is currently the de facto standard for XML messaging for a number of reasons. First, it is relatively simple, defining a thin layer that builds on top of existing network technologies such as HTTP that are already broadly implemented. Second, it is flexible and extensible, because rather than trying to solve all of the various issues developers may face when constructing Web services, it provides an extensible, composable framework that allows solutions to be incrementally applied as needed. Thirdly, it is based on XML. Finally, SOAP enjoys broad industry and developer community support.

SOAP defines four XML elements:

- *env:Envelope* is the root of the SOAP request. At the minimum, it defines the SOAP namespace. It may define additional namespaces.
- *env:Header* contains auxiliary information as SOAP blocks, such as authentication, routing information, or transaction identifier. The header is optional.
- *env:Body* contains one or more SOAP blocks. An example would be a SOAP block for RPC call. The body is mandatory and it must appear after the header.
- *env:Fault* is a special block that indicates protocol-level errors. If present, it must appear in the body.

2.2 Describing Web services

The standard language for formally describing Web services is Web Services Description Language (WSDL) [6]. WSDL is an XML document format for describing Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented (RFC) messages. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints may be combined into services. WSDL is sufficiently extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A complete WSDL definition of a service comprises a service interface definition and a service implementation definition, as depicted in Figure 1.

A service interface definition is an abstract or reusable service definition that may be instantiated and referenced by multiple service implementation definitions. A service interface definition can be thought of as an IDL (Interface Definition Language), Java interface, or Web service type. This allows common industry standard service types to be defined and implemented by multiple service implementers.

In WSDL, the service interface contains elements that comprise the reusable portion of the service description: binding, portType, message and type elements. In the portType element, the operations of the Web service are defined. The operations define what XML messages can appear in the input, output and fault data flows. The message element specifies which XML data types constitute various parts of a message. The message element is used to define the abstract content of messages that comprise an operation. The use of complex data types within the message is described in the types element. The binding element describes the protocol, data format, security and other attributes for a particular service interface (portType).

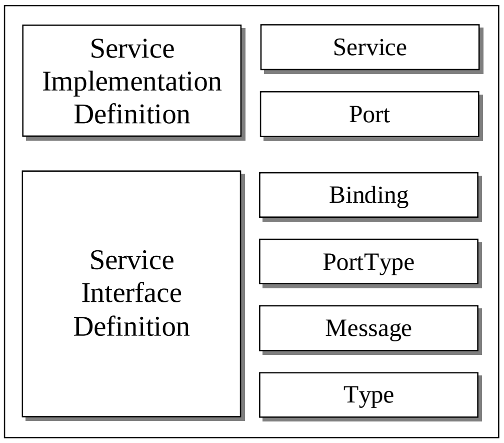

Service Implementation Definition
Service
Port
Service Interface Definition
Binding
PortType
Message
Type

Figure 1: WSDL service implementation and interface definitions.

The service implementation definition describes how a particular service interface is implemented by a given service provider. It also describes its location so that a requester can interact with it. In WSDL, a Web service is modelled as a service element. A service element contains a collection of port elements. A port associates an endpoint (e.g. a network address location) with a binding element from a service interface definition.

2.3 Publishing and discovering Web services

While there are some established standards for Web service description and communication, the publishing and discovery of Web services can be implemented with a range of solutions. Any action that makes a WSDL document available to a requestor, at any stage of the service requestor's lifecycle, qualifies as service publication. In the same way, any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as service discovery.

The simplest case of publishing a Web service is a direct publish. This means that the service provider sends the service description directly to the service requestor. This can be accomplished using an email attachment, an FTP site, or even a CDROM distribution. Slightly more dynamic publication uses Web Services Inspection Language (WSIL) [3]. WSIL defines a simple HTTP GET mechanism to retrieve Web services descriptions from a given URL. Another means of publishing service descriptions available to Web services is through a Universal Description, Discovery and Integration (UDDI) registry [1]. There are several types of UDDI registries that may be used depending on the scope of the domain of Web services published to it. When publishing a Web service description to a UDDI registry, complete business context and well though out taxonomies are essential if the service is to be found by its potential consumers.

3 Adaptive Planning

Domain independent heuristic planning relies on ingenious techniques, such as heuristics and search strategies, to improve the execution speed of planning systems and the quality of their solutions in arbitrary planning problems. However, no single technique has yet proved to be the best for all kinds of problems. Many modern planning systems incorporate more than one such optimizing techniques in order to capture the peculiarities of a wider range of problems. However, to achieve the optimum performance these planners require manual fine-tuning of their run-time parameters.

We call Adaptive Planning the enhancement of Planning systems with Machine Learning techniques, in order to perform automatic configuration of their Planning parameters. Adaptive Planning can assist in the task of creating flexible Planning systems that can automatically adapt themselves to each problem, achieving best performance.

Two adaptive planners that were build upon the HAP planning system [14] are HAP$_{RC}$ [15] and HAP$_{NN}$ [13]. Both are capable of automatically fine-tuning the planning parameters of HAP based on features of the problem in hand. The tuning of HAP$_{RC}$ is performed by a rule system, the knowledge of which has been induced through the application of rule learning over a dataset containing performance data of past executions of HAP. The tuning of HAP$_{NN}$ is performed through instance-based learning that enables the incremental enrichment of its knowledge and allows users to specify their level of importance on the criteria of plan quality and planning speed.

The LAMP (Lazy Adaptive Multicriteria Planning) system [12] uses a generalization of the learning methodology of HAP$_{NN}$ that can be utilized by any modern parameterized domain-independent planning system. In addition, it incorporates a feature weighting approach.

4 Web Services for Adaptive Planning

This section describes the two Web services and the way they can be combined in order to perform adaptive planning. The implementation of the Web services and additional information can be found in [11].

4.1 HAP-WS

HAP-WS is a Web service interface to the HAP (Highly Adjustable Planner) planning system [14]. The customization of HAP is feasible through 7 parameters that are outlined in Table 1.

Table 1: The seven planning parameters and their value sets

Name	Value Set
Direction	{0, 1}
Heuristic	{1, 2, 3}
Weights (w_1 and w_2)	{0, 1, 2, 3}
Penalty	{10, 100, 500}
Agenda	{10, 100, 1000}
Equal Estimation	{0, 1}
Remove	{0, 1}

The first one refers to the planning direction, which can be either backward (0) or forward (1). The second parameter allows the user to select one of the three available heuristic functions in order to use it as a guide during the search. The third parameter sets the values for the weights used during planning in the weighted A* search technique. The fourth parameter sets the penalty put on states violating pre-computed fact orderings, while the next one sets the size of the planning agenda (maximum number of states in the frontier set). The last two parameters enable or disable techniques for overcoming plateaus in the search space and simplifying the definition of sub problems, respectively.

Here HAP is deployed as a planning Web service. The output of HAP-WS is a string containing the solution of the given problem. Apart from the 7 planning parameters, HAP-WS requires two additional string inputs, one for the domain of the problem and another for the problem itself, both in PDDL format. In the future we will replace string inputs with XML ones, following the XML syntax of PDDL [8], should the latter is stabilized. This will allow for automatic validation checks through commodity XML software. Figure 2 depicts an excerpt from the WSDL file of HAP-WS, showing the inputs.

```
<s:element name="HAP">
   <s:complexType>
      <s:sequence>
         <s:element minOccurs="0" maxOccurs="1" name="domain" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" name="problem" type="s:string" />
         <s:element minOccurs="0" maxOccurs="1" name="inputs" type="s0:HAP_Inputs" />
      </s:sequence>
   </s:complexType>
</s:element>
```

Figure 2: HAP-WS input description.

Note that the planning parameters are organized as a separate complex type, because they are optional. Figure 3 shows another excerpt from the WSDL file of HAP-WS defining the complex type of the parameters. As we see, the WSDL file also informs potential clients about the default values of these parameters.

```
<s:complexType name="HAP_Inputs">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" default="0" name="direction" type="s0:directionType"/>
    <s:element minOccurs="0" maxOccurs="1" default="1" name="heuristic" type="s0:heuristicType"/>
    <s:element minOccurs="0" maxOccurs="1" default="2" name="search" type="s0:searchType"/>
    <s:element minOccurs="0" maxOccurs="1" default="10" name="penalty" type="s0:penaltyType"/>
    <s:element minOccurs="0" maxOccurs="1" default="100" name="agenda" type="s0:agendaType"/>
    <s:element minOccurs="0" maxOccurs="1" default="1" name="closer" type="s0:closerType"/>
    <s:element minOccurs="0" maxOccurs="1" default="0" name="remove" type="s0:removeType"/>
  </s:sequence>
</s:complexType>
```

Figure 3: HAP-WS parameters complex type.

We also notice that the type of each parameter is further specified (directionType, heuristicType, etc.). This ensures that potential clients know the allowed values of each parameter. An example of such a type about the direction parameter is given in Figure 4.

```
<s:simpleType name="directionType">
  <s:restriction base="s:int">
    <s:enumeration value="0"/>
    <s:enumeration value="1"/>
  </s:restriction>
</s:simpleType>
```

Figure 4: HAP-WS direction type.

4.2 LAMP-WS

LAMP-WS is a Web service interface to the LAMP (Lazy Adaptive Multicriteria Planning) learning system [12]. LAMP automatically configures the parameters of a planner (such as search direction and agenda size) based on 26 measurable characteristics of planning problems (such as number of actions per operator and mutual exclusions between facts).

Learning data are produced by running the planner under consideration off-line on several planning problems using all combinations of values for its parameters. When LAMP is faced with a new problem, it first retrieves the recorded performance (execution time and plan length) for all parameter configurations of the k nearest problems. It then selects the configuration with the optimal performance.

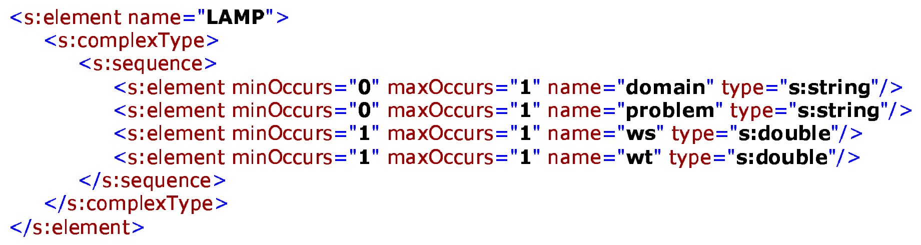
Optimality is however susceptible to user preferences. Usually the most important performance metric is plan quality, i.e. a shorter plan is preferred over a longer one, but there are cases (e.g. real time systems) where the planner must respond promptly even at the expense of the quality of the resulting plan. LAMP tries to mitigate these two mostly contradicting metrics by combining them into an overall score, using the Weighted Sum multicriteria method [9]. Users can specify the weights of plan quality and planning speed, reflecting their relative preference to either of the criteria. LAMP computes the overall score for each configuration according to the user specifications and recommends the one with the highest score.

Experimental results on the automatic configuration of the HAP [14] and LPG [7] planners have shown that LAMP manages to improve the performance of their default configurations by 10 and 6 percent respectively. In addition, setting the weights in favor of either the performance criteria has the expected effect on the planner performance.

There are various interesting discussion issues on LAMP. A first one is whether the 26 features manage to model effectively the different properties of planning problems. The experimental results show that they work, but perhaps there are additional features that should be included. This requires further work in terms of knowledge engineering for planning.

A second one is whether the *space* of different planner configurations (864 for HAP and 72 for LPG) is large enough for covering all the different properties of a planing problem. A first impression is given from the experimental results. HAP has a larger space of configurations than LPG and it was also the planner with the highest performance profit. It seems plausible that the higher the configuration space, the better the adaptation. However, a larger configuration space imposes a higher computational cost for collecting training data.

Here, LAMP is deployed as a Web service for the automatic configuration of HAP. The output of LAMP-WS is a structure containing the values for the 7 parameters of HAP. The inputs to LAMP-WS are the planning domain and problem files and the multicriteria weights for plan steps (ws) and planning time (wt). An excerpt from the WSDL file of LAMP-WS showing its input is given in Figure 5.



```
<s:element name="LAMP">
   <s:complexType>
      <s:sequence>
         <s:element minOccurs="0" maxOccurs="1" name="domain" type="s:string"/>
         <s:element minOccurs="0" maxOccurs="1" name="problem" type="s:string"/>
         <s:element minOccurs="1" maxOccurs="1" name="ws" type="s:double"/>
         <s:element minOccurs="1" maxOccurs="1" name="wt" type="s:double"/>
      </s:sequence>
   </s:complexType>
</s:element>
```

Figure 5: LAMP-WS input description.

4.3 *Using the Web Services*

Having obtained the WSDL documents of the Web services, a user or program can create proxy classes in order to establish a communication medium with them. As far as HAP-WS is concerned, a program can use it directly by providing the planning domain and problem definition files and optionally the values for its 7 planning parameters. Default values are used for the unset parameters. LAMP-WS can also be used directly by a user or program for consultation on a suitable parameter configuration for the HAP planner.

For automatically configuring HAP, a combination of the two services is required. First LAMP-WS must be called with the problem and domain description files and the multicriteria weights for plan steps and execution time. Then, HAP-WS is called passing as parameters the values that were returned by LAMP-WS. This process is shown in Figure 6.

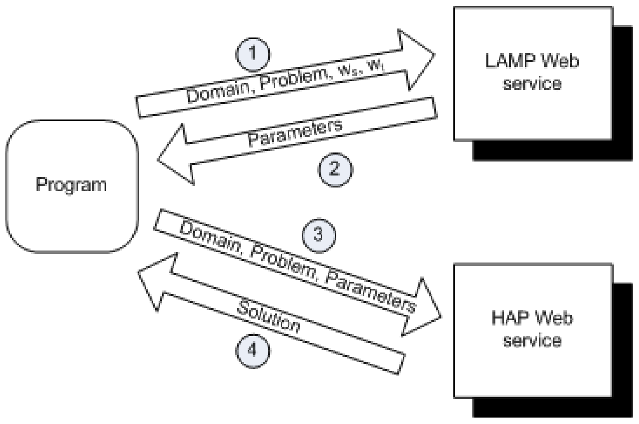


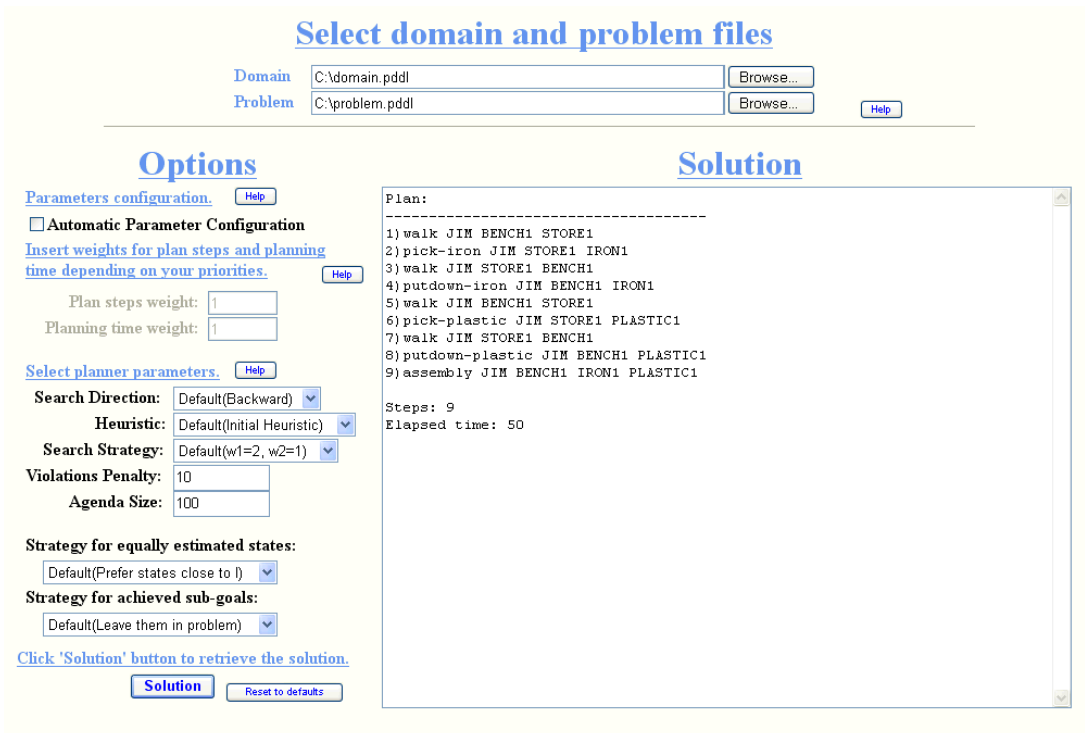
Figure 6: Combination of the two Web services for Adaptive Planning.

5 Web-based Client Application

For the purpose of testing the Web services, a Web-based client application has been developed as an ASP.NET Web page. Figure 7 depicts the layout of this page that plays the role of the application users would create for communicating with our Web services.

Users can provide the PDDL domain and the problem files by clicking the corresponding Browse buttons at the top of the page. They can then either manually configure HAP's options themselves or utilize the LAMP-WS Web service. Manual configuration is available by default, allowing users to change any of the 7 parameters of HAP through the corresponding input boxes. In each parameter, the default value is initially selected in order to inform the user about the default values of the planner and decide whether he/she wants to change the value or not. For automatic parameter configuration the corresponding check-box must be checked. This disables the input boxes for the 7 parameters.

Having finished with the configuration of the planner parameters (either manual or automatic), users can click the Solution button in order to obtain the problem solution. In the case of manual configuration, the ASP application calls HAP-WS passing the parameter values that the user has selected. In the case of automatic configuration, the application first calls LAMP-WS, then gets back the selected parameter values and finally calls HAP-WS with these values. In both cases HAP-WS returns the solution for the submitted problem and the ASP application displays it in the area of the page named Solution.


Select domain and problem files
Browse...
Browse...
Help
Options
Solution
Parameters configuration.
Help
Insert weights for plan steps and planning time depending on your priorities.
Help
Select planner parameters.
Help
Click 'Solution' button to retrieve the solution.
Solution
Reset to defaults

Domain: C:\domain.pddl
Problem: C:\problem.pddl

☐ Automatic Parameter Configuration

Plan steps weight: 1
Planning time weight: 1

Search Direction: Default(Backward)
Heuristic: Default(Initial Heuristic)
Search Strategy: Default(w1=2, w2=1)
Violations Penalty: 10
Agenda Size: 100

Strategy for equally estimated states: Default(Prefer states close to I)
Strategy for achieved sub-goals: Default(Leave them in problem)

```
Plan:
-------------------------------------
1)walk JIM BENCH1 STORE1
2)pick-iron JIM STORE1 IRON1
3)walk JIM STORE1 BENCH1
4)putdown-iron JIM BENCH1 IRON1
5)walk JIM BENCH1 STORE1
6)pick-plastic JIM STORE1 PLASTIC1
7)walk JIM STORE1 BENCH1
8)putdown-plastic JIM BENCH1 PLASTIC1
9)assembly JIM BENCH1 IRON1 PLASTIC1

Steps: 9
Elapsed time: 50
```

Figure 7: ASP.NET Web page

6 Conclusions and Future Work

This paper demonstrated the deployment of an Adaptive Planning system on the Web using the technology of Web services. Adaptive Planning successfully bridges domain-independent problem-solving theories to practical problems with specific properties. In general, we trust that Machine Learning will continue to play a prominent role in the development of effective planning systems. In addition, we believe that developing planning Web services is a step forward towards Planning for Web service composition and towards bridging the theory of Planning to its practical use as a component in larger AI systems and the Semantic Web.

From a technical point of view, delivering an existing (planning) system as a Web service is not very hard, once one becomes familiar with this technology. The task becomes much easier if the system is implemented using Object Oriented Programming, because Web services are based on this software model. Otherwise, there are two solutions. The first is to port the code of the system to an Object Oriented Programming language, which could be quite labor intensive. The second is to build an Object Oriented wrapper of the executable of the system. This is much more simple to implement but is platform-dependent and adds another layer of execution with potential computational complexity and security overheads. We followed the first solution for LAMP and the second for HAP.

As future work, we intend to develop a distributed adaptive planning system, loosely connected with different distributed planning Web services, that would be dynamically invoked and perhaps combined in order to solve a specific problem in an optimal way according to some user-specified performance criteria. We would also like to explore the application of our planning Web service to the problem of composing and orchestrating multiple general purpose Web services to achieve a complex task on the Semantic Web.

Acknowledgements

This work is partly funded from the eCONTENT FP5 European Programme under the EUROCITIZEN project, contract No. 22089.

References

[1] T. Bellwood, L. Clement, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI version 3.0. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, July 2002.

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) version 1.1. http://www.w3.org/TR/SOAP/, May 2000.

[3] P. Brittenham. An Overview of Web Services Inspection Language. http://www.ibm.com/developerworks/webservices/library/ws-wsilover, 2001.

[4] M. Carman, L. Serafini, and Traverso P. Web service composition as planning. In *ICAPS 2003 Workshop on Planning for Web Services*, pages 1636–1642, Trento, Italy, June 2003.

[5] M. Champion, C. Ferris, E. Newcomer, and D. Orchard. Web Services Architecture. http://www.w3c.org/TR/ws-arch/, November 2002.

[6] R. Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana. Web Services Description Language (WSDL) version 1.2 working draft. http://www.w3c.org/TR/wsdl12/, July 2002.

[7] A. Gerevini and I. Serina. LPG: a Planner based on Local Search for Planning Graphs with Action Costs. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems, AIPS02*, pages 13–22, 2002.

[8] J. Gough. Xpddl v.0.1. http://www.cis.strath.ac.uk/ jg/, March 2004.

[9] C.L. Hwang and K. Youn. *Multiple Attribute Decision Making - Methods and Applications: A State of the Art Survey*. Springer-Verlag, New York, USA, 1981.

[10] A. Tate and J. Dalton. O-Plan: a Common Lisp Planning Web Service. In *Proceedings of the International Lisp Conference 2003*, pages 12–25, New York, USA, October 2003.

[11] G. Tsoumakas. Web Services for Adaptive Planning. http://lpis.csd.auth.gr/systems/wsap/, May 2004.

[12] G. Tsoumakas, D. Vrakas, N. Bassiliades, and I. Vlahavas. Lazy Adaptive Multicriteria Planning. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI04*, Valencia, Spain, 2004. (to appear).

[13] G. Tsoumakas, D. Vrakas, N. Bassiliades, and I. Vlahavas. Using the k nearest problems for adaptive multicriteria planning. In *Proceedings of the 3rd Hellenic Conference on Artificial Intelligence, SETN04*, pages 132–141, Samos, Greece, 2004.

[14] D. Vrakas. The Highly Adjustable Planner (HAP). http://lpis.csd.auth.gr/systems/hap/, September 2002.

[15] D. Vrakas, G. Tsoumakas, N. Bassiliades, and I. Vlahavas. Learning rules for Adaptive Planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling, ICAPS03*, pages 82–91, Trento, Italy, 2003.