The WSML Editor Plug-in to the Web Services Modeling Toolkit

Mick Kerrigan

Digital Enterprise Research Institute (DERI),
National University of Ireland, Galway, Ireland
mick.kerrigan@deri.org

Abstract. Research into Semantic Web Services is continuously producing new technologies. It is important that these technologies are made accessible to end users, as technologies are adopted based on the ease with which they can be applied. The level of tool support in Semantic Web Services is relatively low and existing tools are not centralized. This is due to the large overhead in creating applications and a lack of communication between different groups. The Web Services Modeling Toolkit (WSMT) is a framework within which tools for Semantic Web Services can be centralized, it provides the 'glue' code required to produce a high-quality application, reducing the overhead for developers and encouraging third-party tool creation. The WSML editor, for describing Ontologies, Mediators, Web Services and Goals in the WSML language, is the first tool developed for the WSMT.

1 Introduction

It is important that any new technology should be accessible to end users to ensure that this technology is adopted. As work in the field of Semantic Web Services continues, the need for tools increases. The Web Service Modeling Toolkit (WSMT) [7] is an attempt to reduce the effort needed to create tools for Semantic Web Services by supplying a framework within which these tools can be created and deployed. The Web Service Modeling Toolkit will also act as a location to centralize tools for Semantic Web Services, enabling end users to obtain one application for all their needs.

One such technology, which currently lacks adequate tool support, is the Web Service Modeling Language (WSML) [3]. Within the Web Service Execution Environment (WSMX[1]) [1] working group there is a constant need for WSML documents containing Ontologies, Mediators, Web Services and Goals. Creating these documents is time-consuming and a tool for creating these documents would greatly help the productivity of end users. Current tools for editing WSML documents are either not up-to-date with the current version of WSML used by WSMX or lack the support for all the elements of a WSML document. The WSML Editor [8] is the first tool developed for the WSMT and it attempts

[1] http://www.wsmx.org

to provide an up-to-date, extensible editor for modifying all the elements of a WSML document.

An introduction to the WSMO, WSML and WSMX tecnologies is presented in Section 2. The Web Services Modeling Toolkit (WSMT) and its architecture is described in Section 3. The WSML Editor, the first plug-in to the WSMT, is introduced in Section 4. Section 5 presents some related work in the area of tools for Semantic Web Services. Finally, section 6 concludes the paper and presents some conclusions and future work.

2 WSMO, WSML and WSMX

The Web Service Modeling Ontology (WSMO) [11] is an ontology for describing Semantic Web Services. WSMO is based on the Web Service Modeling Framework (WSMF) [5] and as such, is based on the four main elements of the WSMF: Ontologies, Mediators, Web Services, and Goals. The aim of WSMO is to solve the integration problem by describing Web Services semantically and by removing ambiguity about the capabilities of a Web Service and the problems it solves.

The Web Service Modeling Language (WSML) [3] is a formalization of the WSMO ontology, providing a language within which the properties of Semantic Web Services can be described. There are five language variants, based on Description Logic and Logic Programming. Each language variant provides different levels of logical expressiveness [3]. These variants are: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full. WSML-Core, which corresponds with the intersection of Description Logic and Horn Logic, provides the basis for all the variants, while WSML-Full unites the functionality of all variants. WSML Core is extended in the direction of more expressive Description Logic by WSML-DL and towards Logic Programming by WSML-Flight and WSML-Rule.

The Web Service Execution Environment (WSMX) [1] is a reference implementation of WSMO using the WSML language. This implementation is an execution environment for the dynamic discovery, mediation, composition and invocation of Semantic Web Services. It uses WSMO as its conceptual model and defines its own execution semantics [10], architecture [12] and implementation [12]. The completed WSMX system will allow service providers to describe their Web Services in WSML and publish these descriptions to the WSMX system. When end users send goals described in WSML to WSMX, these goals are matched against the capabilities of the Web Services registered with the WSMX system. These services can then be invoked to realize the users' goals.

3 Web Services Modeling Toolkit

The Web Services Modeling Toolkit (WSMT)[2] [7] is a framework for the rapid creation and deployment of homogeneous tools for Semantic Web Services. A

[2] The latest version of the Web Services Modeling Toolkit is available from http://sourceforge.net/projects/wsmx/

homogeneous toolkit improves the users experience while using the toolkit, as the tools have a common look and feel. Usability is also improved as the user does not need to relearn how to use the application when switching between tools.

The WSMT enables developers of tools to focus on the tool's functionality and provides the framework within which they can be deployed and executed. The WSMT is implemented in the Java programming language in order to benefit from its multi-platform support and the existing Java libraries available for Semantic Web technologies, for example WSMO4J[3]. Using the WSMT framework does not require the user to learn any new technologies. This can be a problem with other frameworks like Eclipse[4], which uses a different graphical library (IBM's SWT - The Standard Widget Toolkit) than most Java users are accustomed to (Sun's Java Swing library).

The aims of the WSMT are to encourage the development of tools for Semantic Web Services and, once developed, to centralize these tools in a common application. This will allow users who wish to describe and manage Semantic Web Services, to install a single application from which all the tools are available. By continuing to use Sun's Java Swing graphics library over IBM's SWT library, we enable existing tools to be ported to the WSMT, thus centralizing existing tools.

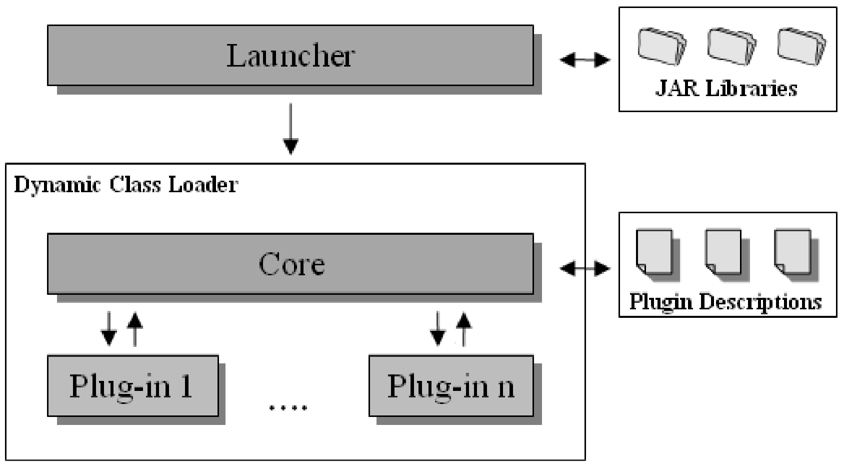


Fig. 1. Architecture of the WSMT

Figure 1 shows the architecture of the Web Services Modeling Toolkit, which consists of three tiers. The first tier contains the compact launcher, the second

[3] http://wsmo4j.sourceforge.net/
[4] http://www.eclipse.org/

contains the core and the third contains the individual plug-ins. Each tool is implemented as a plug-in to the WSMT framework. Deploying the tool into the framework is just a matter of compiling the plug-in into a jar file, which implements a number of interfaces, and placing the jar file, along with any third-party jars used, into the lib folder of the WSMT installation. This means that new tools can be deployed into the application without the requirement of recompiling the application.

Building the classpath dynamically is a major issue when developing applications where it is not known in advance what jar files will be in the classpath. When executing an application in a Unix environment, scripting can be used to build this classpath. However, this is only possible in operating systems where scripting is supported. The job of the launcher is to build the dynamic classpath. It does this by locating all jar files in the lib folder of the WSMT installation and building a dynamic classloader. This classloader is then used to launch the WSMT core.

The WSMT core is responsible for supplying the 'glue' code to the plug-ins (tools), providing the main application frame, the menu bar, and the configurations for multi-language localization. The core loads all the available tools by searching for plug-in description files in the lib folder of the WSMT installation. Each description specifies a unique identifier and the class that the core should instantiate in order to load the plug-in. This class must implement the Plugin interface, which allows access to the plug-in itself.

The WSMT is wrapped in a full installation system, which allows the end user to choose the tools that are installed during the installation process. A fully private Java 1.5 run-time environment is also installed, which means there is no dependency on the user to install any third-party software. A third-party tool provider can choose to supply their own tool for inclusion in the WSMT installation or provide their own installation for their tool.

An initial set of tools for the WSMT includes: a WSML Editor [8] for editing WSML documents and publishing them to WSMO repositories, which will be described in more detail in the next section; a WSMX Monitor [9] for monitoring the state of the components in the WSMX environment; a WSMX Mediation tool for creating mappings between Ontologies (which is in fact an existing tool that will be ported to the WSMT), and finally, a WSMX Management tool for managing the WSMX environment.

4 The WSML Editor

The aims of developing the WSML Editor are to provide a useful tool for describing Semantic Web Services using the WSMO ontology and publishing these descriptions to a WSMO repository and also to facilitate user experimentation with the WSMO ontology and WSML language. The first version of the WSML Editor focuses on the creation of semantic descriptions in WSMO and reading and writing these semantic descriptions to and from a local hard-disk using the WSML syntax. Future work, which we will look at in more detail in section 6, will focus on communicating with WSMO repositories and execution environments in order to register semantic descriptions with these systems.

The WSML Editor is capable of opening multiple WSML documents in a tabbed interface, each tab containing all the information about a given document. A WSML document is comprised of four types of top-level element, Ontologies, Mediators, Web Services and Goals. These elements are grouped together and displayed in a tree structure, as can be seen in figure 2. This means that, for example, all of the Web Services described in a given document are displayed together. A node in the document tree represents either an element or a group of elements, by expanding a group the elements within can be seen.

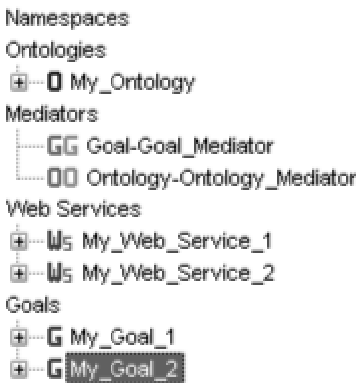


Fig. 2. WSML Document Tree

The namespaces node can be seen at the top of the WSML document tree, as shown in figure 2. WSML documents use namespaces in a similar manner to XML Schema, with the namespaces providing shortcut prefixes so that identifiers from different spaces can be differentiated. When this node is selected, the user is able to add and remove namespaces from the WSML document. The available namespaces can be used throughout the elements in the document, where values and ranges of elements must be chosen.

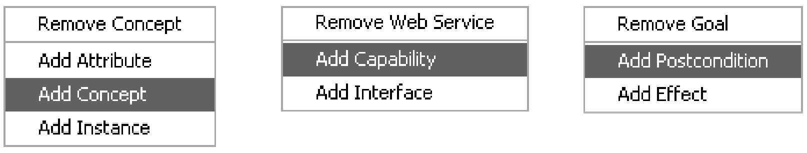


Fig. 3. Right-Click Menus

A new element is added to the document by right-clicking on the parent element in the tree. A number of potential right-click menus that can be displayed, are shown in figure 3. The content in the right-click menu depends upon the element which was selected. When adding an element to the document, an identifier must be added. The dialog for doing this can be seen in figure 4.

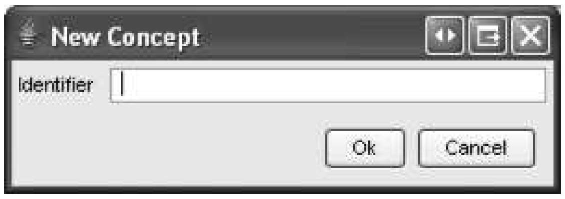
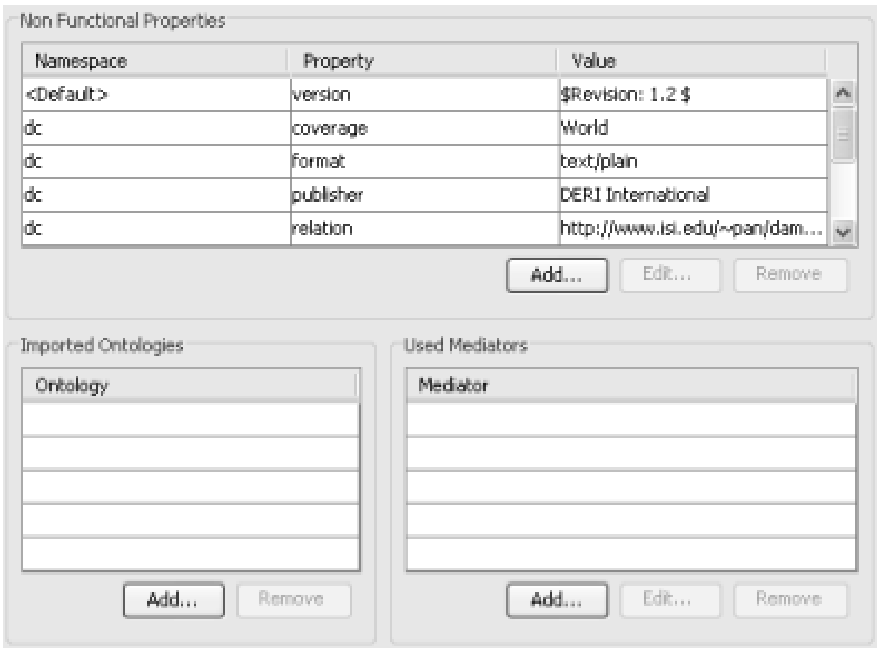


Fig. 4. Adding a new Element

When selecting nodes that represent an element in the document tree, the properties of that element are displayed in the panel on the right-hand side of the application. There are many common properties between different WSML elements, which means that this panel is dynamically built based on the functionality supported by the given element. For example, when displaying a concept, the non-functional properties, super concepts, and logical expressions panels are displayed.

Dynamically building the panel in this way ensures that if new elements are added to the WSML standard, they can easily be incorporated into the editor. This also enables an encapsulated application design. Examples of the sub-panels which can be used to make up the dynamic panel include panels for adding and removing non-functional properties, imported ontologies, used mediators, super concepts, super relations, setting logical expressions, choosing ranges and choosing values. Figure 5 shows the properties of an ontology displayed in the WSML Editor, the dynamic panel is made up of the non-functional property, imported ontology, and used mediator sub-panels.

WSMO4J is used as the object model that is built up as the user creates their semantic descriptions. WSMO4J is an open source library comprising a Java API and reference implementation compatible with WSMO v1.0, as described in WSMO Deliverable 2, Appendix A[11]. Future work on the WSMO4J library will upgrade the library to be compatible with WSML Deliverable 16.1[3]. When this work is completed the WSML Editor will be upgraded so that it is also compatible. As part of the reference implementation, a WSML parser and serializer are supplied, which allows the object model to be written to hard-disk and re-read later.



Namespace	Property	Value
<Default>	version	$Revision: 1.2 $
dc	coverage	World
dc	format	text/plain
dc	publisher	DERI International
dc	relation	http://www.isi.edu/~pan/dam...

Fig. 5. The properties of an ontology displayed in the WSML Editor

5 Related Work

This section outlines a number of tools (in development or completed), which can be used for the management of Ontologies in the WSML language.

5.1 OMWG - Ontology Editing and Browsing Tool

The Ontology Editing and Browsing Tool [6] is a plug-in for the Eclipse framework currently under development by the Ontology Management Working Group (OMWG)[5]. The tool can be used to edit Ontologies described within WSML documents. The tool uses two different views of the WSML document to aid in ontology management. The first view, called the 'Class Explorer', is used to view Concepts, Relations, Functions, Axioms, Attributes and Parameters. The second view, called the 'Instance Explorer', is used to view Instances of Concepts and Relation Instances of Relations, along with their associated Attribute and Parameter values. The main difference between this editor and the WSML Editor is the focus. Where the WSML Editor looks at Ontologies, Mediators, Web Services and Goals, the OMWG editor focuses only on Ontologies. This essentially means that while the OMWG editor provides useful functionality to users for building Ontologies, it does not provide the much needed functionality for Semantic Web Services

[5] http://www.omwg.org/

5.2 SWWS Studio

Originally based on OWL-S [2], the SWWS Studio [4] was developed by Ontotext Lab[6] as part of the Semantic Web enabled Web Services (SWWS)[7] European IST project. The tool can be used to edit Ontologies, Mediators, Web Services, and Goals located in a local or remote registry. SWWS Studio is based on early versions of the WSMO ontology, and has not recently been updated to support features added to the WSMO ontology and the WSML language. The closed license on SWWS studio makes it difficult to update and extend. This is one of the main driving forces behind making the WSMT and the WSML Editor available under an open source license.

5.3 WSMO Studio

WSMO Studio[8] is a new tool in the design phase that will be made available as a set of Eclipse Plug-ins. The tool is a joint project developed by Ontotext Lab and the University of Innsbruck[9] as part of the DIP[10] European IST project and builds on the work done in the SWWS Studio project. The aim of the WSMO Studio is to create an open source, extensible editor for the WSMO ontology, allowing users with different roles to use the editor to create and version Ontologies, Mediators, Web Services and Goals in WSML and to publish and manage these descriptions in their relevant repositories.

6 Conclusions

The Web Services Modeling Toolkit provides a useful framework for the rapid development of tools for Semantic Web Services. As more and more tools are developed using the framework the WSMT will become more popular, encouraging third-party developers to use it. Future work to the WSMT will include the addition of more graphical components to which individual tools can contribute; for example an application toolbar and preferences dialog. Additional components will also be added to the core to reduce the effort required to develop tools. These components will include an application logging component and a communication component. The logging component will enable tool developers to perform high-quality logging without the overhead of developing a logging framework. The communication component will encapsulate the communication with external execution environments and repositories.

The first version of the WSML Editor is the first step towards a fully functional tool for creating semantic descriptions in the WSML language and publishing these semantic descriptions to external systems. The WSMO4J library is

[6] http://www.ontotext.com/
[7] http://swws.semanticweb.org/
[8] http://www.wsmostudio.org/
[9] http://www.uibk.ac.at/
[10] http://dip.semanticweb.org/

currently being upgraded to bring it into line with WSML Deliverable 16.1[3]. When this upgrade is complete the WSML Editor will undergo the same process. Currently, the WSMO4J library deals with logical expressions, which are used throughout the WSML documents, as strings. This means that there is currently no validation of logical expressions within the object model. Development of an object model for logical expressions is underway and once this development is incorporated into the WSMO4J library, the WSML Editor will be able to validate all logical expressions and provide a more graphical mechanism for building up these expressions. The main functionality to be added in later versions, is communication with external execution environments and repositories so that semantic descriptions can be registered with these systems. This communication will be done through the new communication component in the WSMT core.

References

1. E. Cimpian, T. Vitvar, and M. Zaremba, Overview and Scope of WSMX, http://www.wsmo.org/TR/d13/d13.0/v0.2/
2. M. Dean, G. Schreiber, (Eds.). OWL Web Ontology Language Reference, W3C Recommendation, 10 February 2004. Available from http://www.w3.org/TR/2004/REC-owl-ref-20040210/.
3. J. de Bruijn, H. Lausen, and D. Fensel, The WSML Family of Representation Languages, http://www.wsmo.org/TR/d16/d16.1/
4. M. Dimitrov, Z. Marinova, P. Radkov, SWWS - Prototype Tools II, http://swws.semanticweb.org/public_doc/D5.2.pdf
5. D. Fensel and C. Bussler: The Web Service Modeling Framework WSMF, Electronic Commerce Research and Applications, 1(2), 2002.
6. J. Henke, OMWG Editing and Browsing Implementation, http://www.omwg.org/TR/d8/d8.3/
7. M. Kerrigan, Web Services Modeling Toolkit (WSMT), http://www.wsmo.org/TR/d9/d9.1/v0.1/
8. M. Kerrigan, WSML Editor, http://www.wsmo.org/TR/d9/d9.2/
9. M. Kerrigan, WSMX Monitor, http://www.wsmo.org/TR/d9/d9.3/
10. E. Oren. WSMX Execution Semantics, http://www.wsmo.org/TR/d13/d13.2/
11. D. Roman, H. Lausen, U. Keller (eds.): Web Service Modeling Ontology (WSMO), http://www.wsmo.org/TR/d2/
12. M. Zaremba, M. Moran, WSMX Architecture, http://www.wsmo.org/TR/d13/d13.4/v0.2/