D9.2v0.3 WSML Editor

WSMX Working Draft 13 June 2005

This version:
http://www.wsmo.org/TR/d9/d9.2/v0.3/20050613/
Latest version:
http://www.wsmo.org/TR/d9/d9.2/v0.3/
Previous version:
http://www.wsmo.org/TR/d9/d9.2/v0.2/20050426/
Editors:
Mick Kerrigan
Authors:
Mick Kerrigan

This document is also available in non-normative PDF version.

Table of contents

1. Introduction

1.1 Scope

The Web Service Modeling Language (WSML) [de Bruijn et al., 2004] is a formalization of the WSMO ontology [Roman et al., 2004] and provides a language for describing Semantic Web Services. There are five language variants, WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full. WSML-Core corresponds with the intersection of Description Logic and Horn Logic and 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.

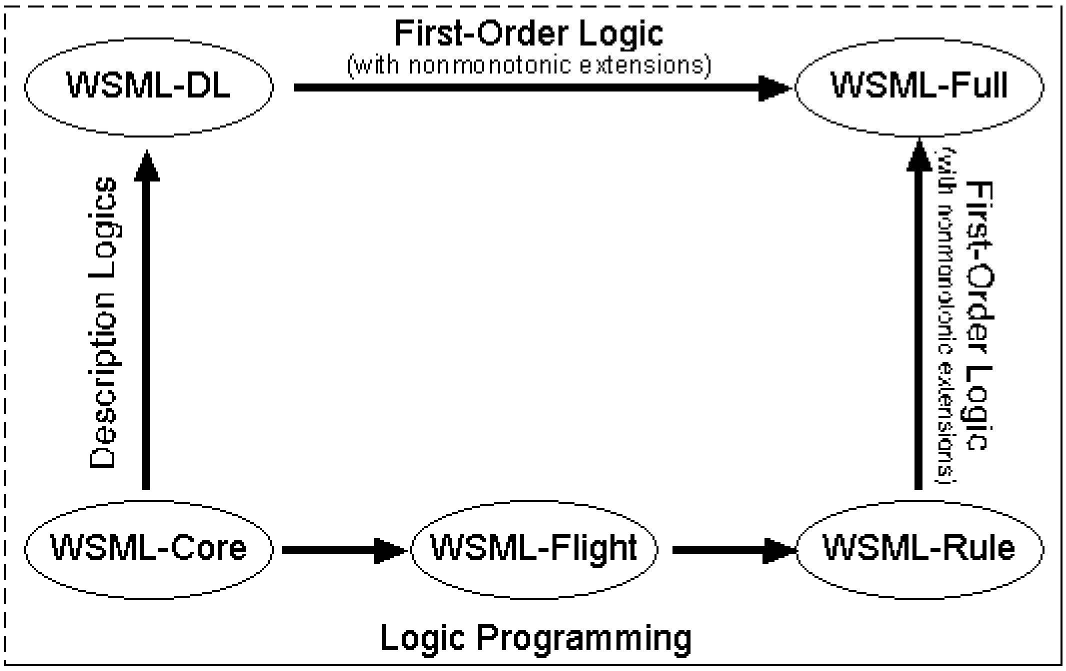


Figure 1. WSML Space [de Bruijn et al., 2004]

1.2 Purpose of this Document

The purpose of this document is to outline the WSML Editor plug-in for the Web Service Modeling Toolkit (WSMT) [Kerrigan, 2005]. The WSML Editor is a graphical tool for editing Ontologies, Goals, Mediators and Web Services described in the WSML language.

1.3 Downloading the WSML Editor

The WSML Editor is distributed as a plug-in for the Web Services Modeling Toolkit (WSMT). The latest version of the WSML Editor can be obtained by downloading the latest version of the WSMT at http://www.sourceforge.net/projects/wsmx/.

2. WSML Editor

2.1 Aim

The aim of developing the WSML Editor is to provide a useful tool for describing Semantic Web Services and publishing these descriptions to a WSMO Registry and to facilitate user experimentation with the WSMO ontology and WSML language. To achieve these goals, it is necessary to hide the syntax of WSML from the users, allowing them to focus on developing descriptions of Ontologies, Mediators, Web Services, and Goals without the overhead of syntactic correctness.

2.2 Approach

The Web Service Modeling Framework (WSMF) [Fensel & Bussler, 2002] comprises four main elements that are used to describe Semantic Web Services. As described earlier, WSML formalizes the WSMO ontology, which in turn is based on the WSMF. Therefore, the same four elements are the main elements of a WSML document. These four elements are Ontologies, Mediators, Web Services and Goals. The WSML Editor displays WSML documents in a tree structure with Ontologies, Mediators, Web Services and Goals at the top level of each document. A document may comprise one or more elements of the same or different types. Sub elements of each type (for example Concepts, Relations, Functions, Capabilities etc.) are displayed as children of the element to which they belong. Figure 2 below shows the elements in a WSML document and how they relate to one another (note: items that appear as children of the same element in the figure do not suggest the same relationship between parent and child. i.e. 'Instance' is an instance of 'Concept', where as 'Sub-concept' is a sub concept of 'Concept').

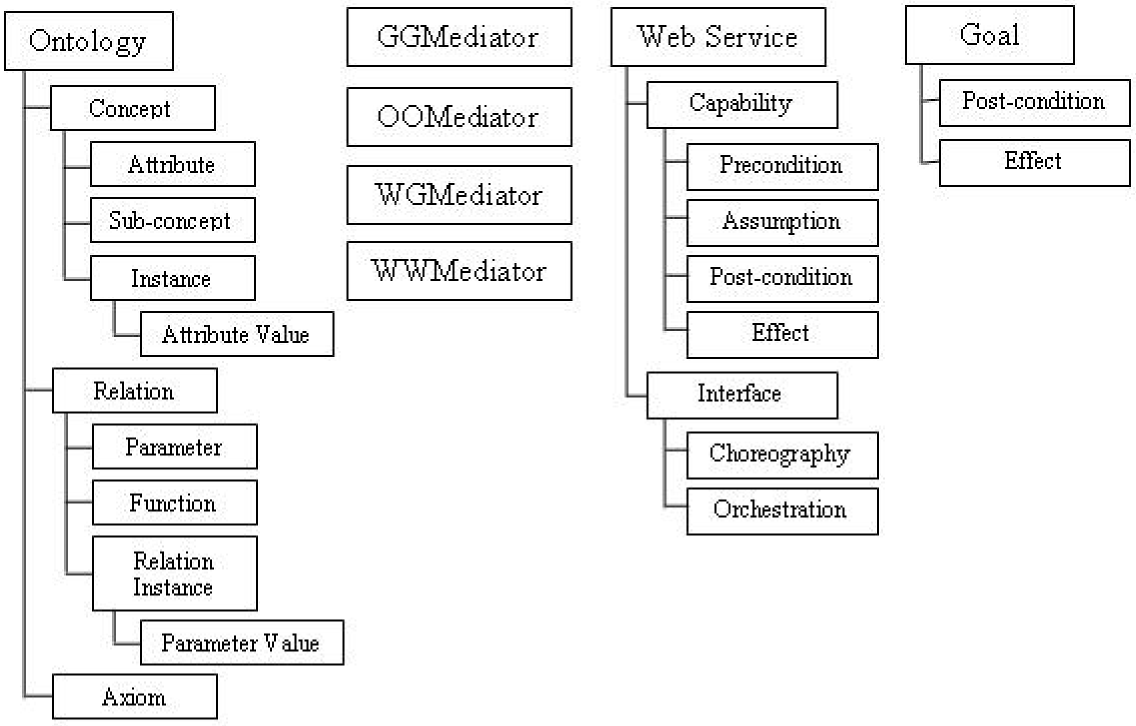


Figure 2. WSML Elements

The WSML Editor uses the libraries provided by the WSMO4J project [WSMO4J] as a basis for creating WSML documents. The WSMO4J project is an API and reference implementation compatible with WSMO v1.0 as described in WSMO Deliverable 2, Appendix A [Roman et al., 2004], and thus the WSML Editor is compatible with WSMO v1.0 also. When WSMO4J is upgraded to be inline with WSML Deliverable D16.1[de Bruijn et al., 2004], the WSML Editor will be upgraded to be compatible.

2.3 Functionality

The main aim of the first versions of the WSML Editor is to support the creation of WSML documents. Later work will focus on publishing descriptions to a WSMO Registry. This section of the deliverable focuses on how a WSML document is displayed within the WSML Editor and describes the graphical methods by which the user can modify elements of the WSML document.

Each WSML document opened in the WSML Editor is displayed in its own tab, as a tree of elements. The elements in the tree (see section 2.3.1) are laid out in hierarchy shown in Figure 2. An additional section for controlling the namespaces used in the WSML document is available in this tree also (see section 2.3.2). When an element is selected in the tree, its properties are displayed in the property panel on the right of the application. As there are many common properties across different element types the property panel is a combination of different graphical components based on the element selected. Sections 2.3.3 to 2.3.7 focus on the different graphical components that exist for changing the properties of WSML Elements.

2.3.1 Document Tree

To add a new element to the tree the user should right click on the element, which should be the super element of the new element and a list of potential new elements is displayed.

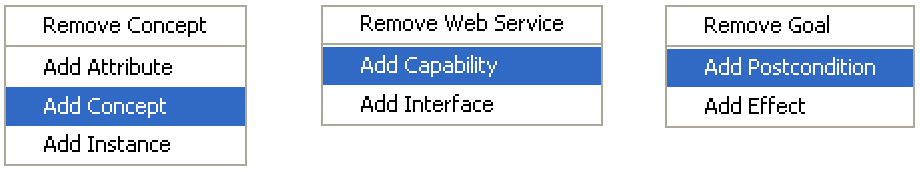
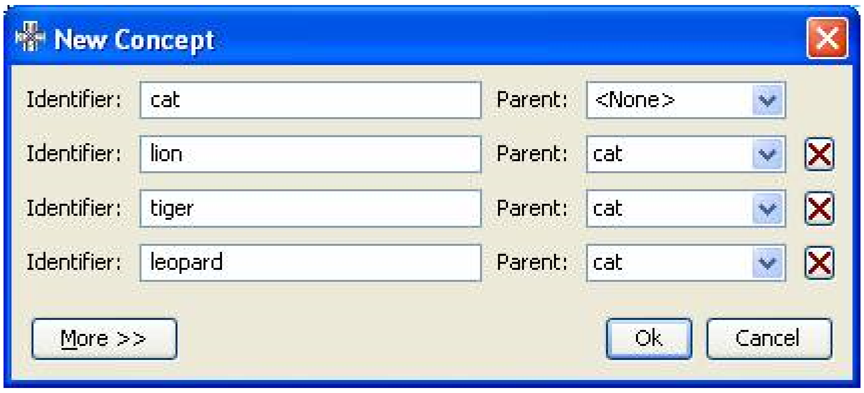


Figure 3. Document Tree Right Click Menus

Once an element type has been selected a dialog is presented where the identifier of the new element is entered (except in the case of Attribute Values and Parameter Values, see section 2.3.7). The identifier, which is entered must be unique for the document, except for attributes and parameters, which must be unique for the concept, relation or function they are being added to. The user can add multiple new items of the same type at the same time and where inheritance is present the user can choose the parent of each new item.


New Concept

Identifier: cat Parent: <None>
Identifier: lion Parent: cat
Identifier: tiger Parent: cat
Identifier: leopard Parent: cat

Figure 4. Add new concepts

2.3.2 Filtering the Document Tree

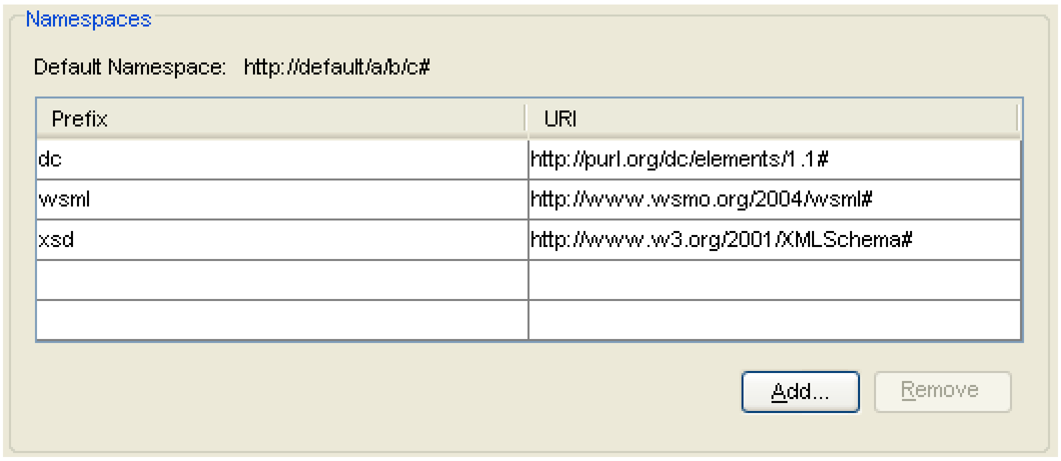
Different users of the WSML Editor may have different roles and goals, in fact the same user may have different roles at different times in the semantic description process. It is therefore very useful to provide the user with a mechanism for customizing their view at any given time. Currently the user can filter the tree using the view menu, on this menu is items which allows the user to:

- show/hide functions
- show/hide instances

- show/hide relation instances
- show/hide attributes
- show/hide parameters
- show/hide inherited attributes
- show/hide inherited parameters

2.3.3 Namespaces

A WSML document contains a default namespace and a set of other namespaces, which are used to qualify names from different XML documents. This set of namespaces is made up of tuples of prefixes and URI's. All elements in a WSML document are created in the default namespace. The user can specify the default namespace at document creation time and add/remove other namespaces, while designing the elements in the WSML document. The default namespace is not editable after document creation time, due to ongoing discussions around identifier immutability. If at any point a tuple is removed from the namespaces which is used by an element in the WSML document, the URI will continue to be used by the element in question without a prefix.

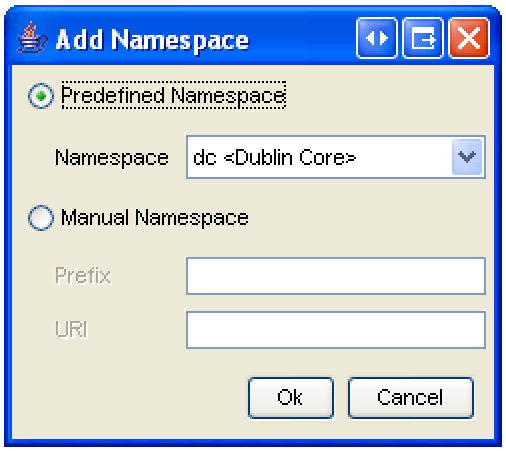


Default Namespace: http://default/a/b/c#

Prefix	URI
dc	http://purl.org/dc/elements/1.1#
wsml	http://www.wsmo.org/2004/wsml#
xsd	http://www.w3.org/2001/XMLSchema#

Figure 5. Namespaces

The WSML Editor is deployed with a number of predefined namespaces, which the user can choose from, these namespaces include the Dublin-Core [Weibel et al. 1998], WSML and XML-Schema namespaces. By providing these namespaces to the user we are encouraging their use and preventing input errors by the user when inputting properties and values. The predefined namespaces are provided in an XML document shipped with the WSML Editor, this XML document can be edited by hand or using the GUI provided in the preference dialog. Users can add namespaces to this XML document, which they use regular, to improve their own productivity. An XML document was used to enable easy maintenance and the use of a DTD for validation. Further information on the predefined namespaces and how to locate and update the XML file shipped with the WSML Editor is available in Appendix A.



[✓ Predefined Namespace]

Namespace	dc <Dublin Core>

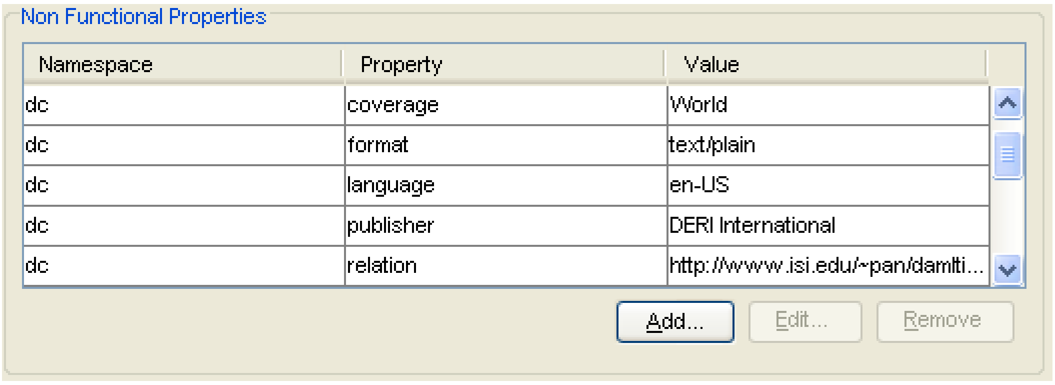
Manual Namespace

Prefix	
URI	

Figure 6. Add Namespace

2.3.4 Non-functional Properties

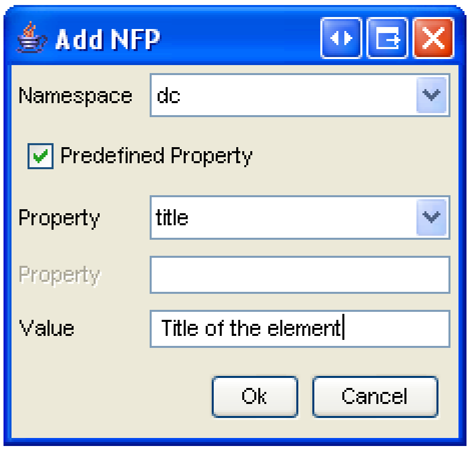
Nearly all elements in a WSML document can specify a set of non-functional properties. The recommended minimal set of non-functional properties set by the WSML Specification [de Bruijn et al., 2004] is the Dublin-Core properties [Weibel et al. 1998], however this set is extensible by the user, for example the WSML namespace defines a number of properties not in the Dublin-Core. Both of these namespaces are included in the list of predefined namespaces shipped with the WSML Editor and more information on the non-functional properties defined by these namespaces is available in Appendix A.



Namespace	Property	Value
dc	coverage	World
dc	format	text/plain
dc	language	en-US
dc	publisher	DERI International
dc	relation	http://www.isi.edu/~pan/damlti...

Figure 7. Non-functional Properties

As the set of non-functional properties which can be used is extensible by the user, the user can choose a predefined property or manually enter the property name and then assign a value to that property.



Namespace	dc

[✓ Predefined Property]

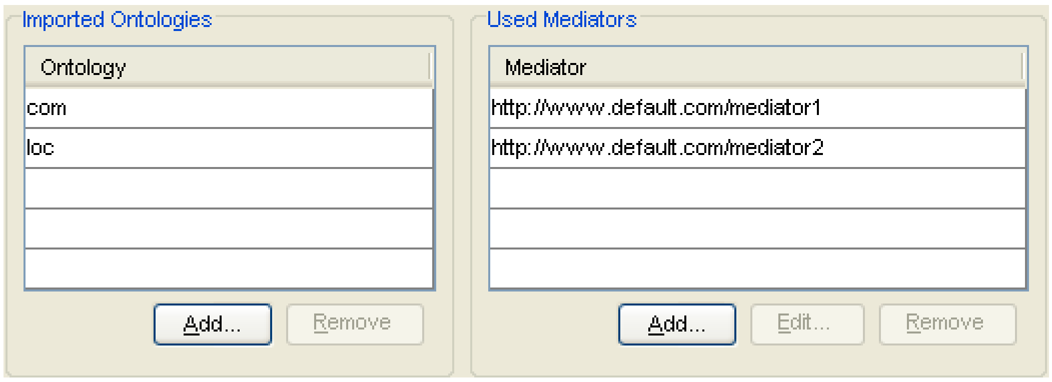
Property	title
Property	
Value	Title of the element

Figure 8. Add a Non-functional Property

2.3.5 Imported Ontologies & Used Mediators

Ontologies, Mediators, Web Services, Capabilities, Interfaces and Goals can all import ontologies specified in other WSML documents. Importing ontologies allows "a modular approach for ontology design" [Roman et al., 2004]. A given ontology may be imported into multiple objects and the URI of the ontology should be included in the namespace so that it can be referenced. The mechanism for importing an ontology into an element is to first add the ontology URI to the namespace and then add the prefix to the imported ontologies of the given element by selecting the element and clicking add in the imported ontologies panel and choosing the prefix. By ensuring the user adds the URI of the ontology to the namespace, we ensure that elements in the ontology can be referenced using qualified names and that the ontology is available for importation into other elements within the WSML documents.

Ontologies, Mediators, Web Services, Capabilities, Interfaces and Goals can all use mediators to mediate between imported ontologies where necessary. Mediators are used to link elements between different ontologies to ensure homogeneity. The user is able to add, edit and remove URI's, which identify mediators that should be used by each element to mediate between the imported ontologies and the current ontology.



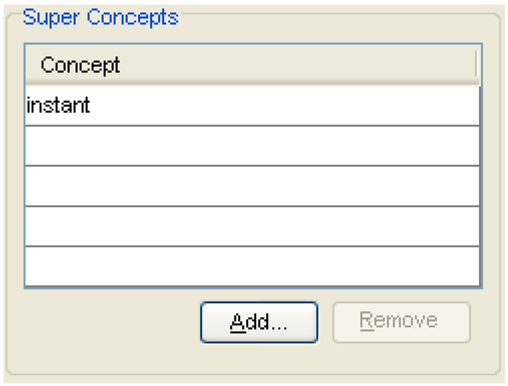
Ontology
com
loc

Mediator
http://www.default.com/mediator1
http://www.default.com/mediator2

Figure 9. Imported Ontologies and Used Mediators

2.3.6 Concepts and Relations

Concepts and Relations are displayed as trees in an ontology. Concepts and relations are in fact not trees but graphs, as they can have multiple super-concepts and super-relations. However performing graphing in a java applications is not straight-forward, so for the initial version of the WSML Editor a tree is used. For each concept and relation that is created within the ontology, additional super-concepts and super-relations can be added, if these super-concepts and super-relations are also specified in this ontology, i.e. not in imported ontologies, the concept is displayed at both points in the concept or relation tree. Future work may include adding graphing support to the WSML Editor.



Concept
instant

Figure 10. Super Concepts

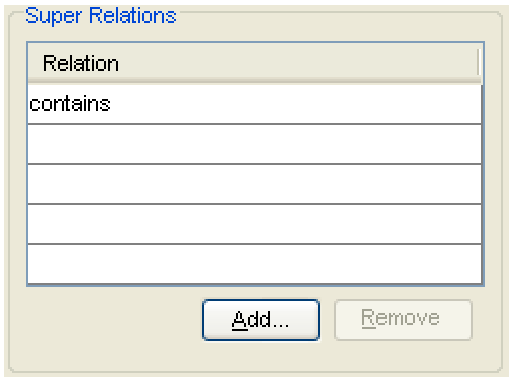


Figure 11. Super Relations

2.3.7 Attributes, Parameters and their values

When designing a concept a number of attributes can be created, when instantiating a concept these attributes are given values. The same is true when creating Relations and Functions except that these attributes are called parameters. An attribute and a parameter have a unique identifier (unique within the concept, relation or function where they are defined) and a type. This type can be a concept from the default namespace or any imported ontologies.

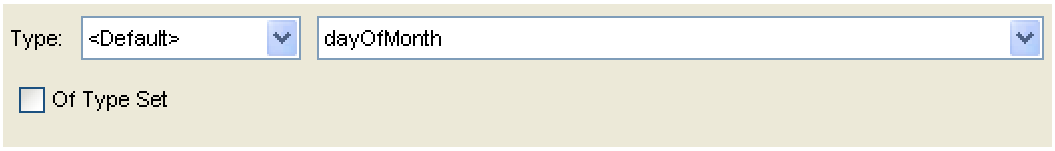


Figure 12. Set an Attribute's Type

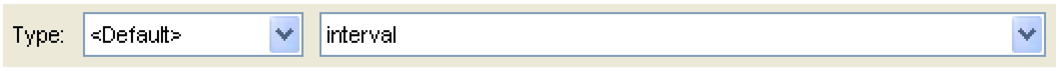


Figure 13. Set an Parameter's Type

When creating Instances or Relation Instances it is possible to assign values to the attributes or parameters created on the super concepts, relations or functions. The user chooses one of the attributes or parameters and then assigns a value to this attribute or parameter. To aid the user in attribute and parameter identification the names are displayed as superobject.identifier, so for example if the user wished to assign a value to the 'length' attribute of the 'table' concept, then user would select 'table.length' from the list of attributes.

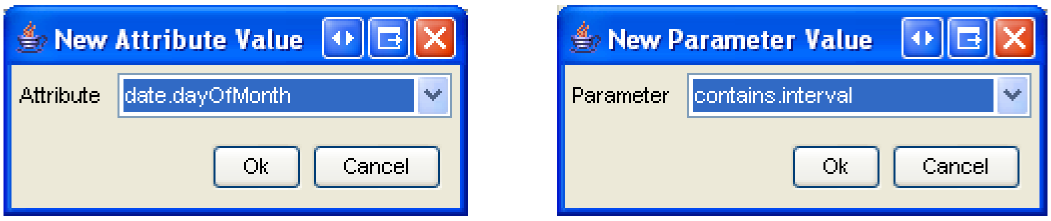


Figure 14. Add a new Attribute Value or Parameter Value

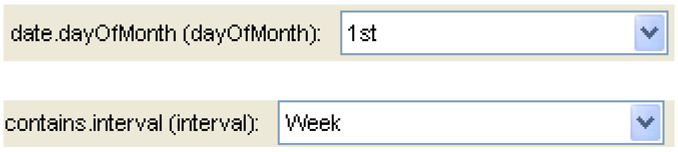
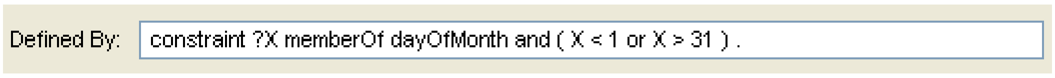


Figure 15. Set the value of an Attribute Value or Parameter Value

2.3.8 Logical Expressions

Concepts, Relations, Functions and Axioms (including capability pre-conditions, post-conditions, assumptions and effects and goal post-conditions and effects) can have a logical expression, which is used to formally specify the semantics of the concept, the set of instances of the relation or function and the statement captured by the axiom. The logical expressions are specified using the WSML Logical Expression Syntax, as defined in section 8 of the WSMO Deliverable 2 [Roman et al., 2004].

In the first version of the WSML Editor, a standard text field is provided to specify these logical expressions. The validity of the logical expressions is not checked, however with the future addition of an object model for logical expressions to the WSMOJ library, validation will be possible. Future work may also include designing a graphical method for building up these logical expressions, using the new logical expression object model.



Defined By: constraint ?X memberOf dayOfMonth and (X < 1 or X > 31) .

Figure 16. Logical Expressions

3. Future Work

As described in section 2.3, the purpose of the first version of the WSML Editor was to allow the user to create WSML documents comprising Ontologies, Mediators, Web Services and Goals. The next version of the WSML Editor will comprise updates to the current functionality and extension of the WSML Editor to allow WSML documents be exported to various other formats, published to external ontology repositories and sent to execution environments (i.e. WSMX).

3.1 Logical Expressions

As described in section 2.3.5, logical expressions are defined in a standard text field in the current version of the WSML Editor. Part of the aim of the WSML Editor is to hide the syntax of the underlying WSML language from the user and the logical expressions are the only remaining part of the syntax in the WSML Editor. Future work will look at creating a graphical mechanism for generation of logical expressions.

Currently logical expressions created in the WSML Editor are not validated, as the WSMO4J library treats logical expressions as strings. A future addition to the WSMO4J library is an object model for logical expressions. With the addition of this object model, it will be possible to ensure that the logical expressions are valid prior to serializing the object model to WSML.

3.2 Exporting WSML

It may be useful for users to convert from WSML to other file formats for modeling ontologies, for example OWL-S [Patel-Schneider et al., 2004] and FLORA-2 [Yang et al., 2003]. Future work will look at creating functionality for exporting WSML to other formats. The approach that we intend to use, is to discover Semantic Web Services, which can perform format conversion. In certain cases it may be necessary to perform a conversion via another format i.e. WSML to OWL-S and OWL-S to FLORA-2. In doing this the WSML Editor will become a Semantic Web aware application and a proof of concept for Semantic Web Service discovery, composition and invocation.

3.3 Publishing WSML

Creating a WSML document is only useful if the resulting semantic descriptions are made available for service discovery, composition and invocation. A WSML document can contain ontologies, web services, mediators and goals all of which can be published to repositories. The WSML Editor is the logical place to put functionality for interacting with these repositories. Future work will look at publishing WSML documents to and download WSML documents from these repositories.

3.4 Communication with Execution Environments

As described in section 3.3 creating a WSML document is only useful if the information within it can be used. Execution environments , for example WSMX [Cimpian et al., 2005], allow the semantic descriptions to be used for service discovery, composition and invocation. The WSML Editor should allow users to communicate with these execution environments and send WSML documents to invoke features of the execution environment, for example the user should be able to realize a goal. This communication will be implemented in a modular fashion, support for different execution environments can be added over time.

3.5 Ontology Visualization

As Ontologies get more and more complex it can become harder for an Ontology designer (or someone looking at an ontology created by another individual) to see the different relationships within the Ontology. Providing graphical mechanisms for seeing Ontologies in different ways will aid the users understanding of the Ontology they are viewing. Future work will look at providing graphical visualizations of both the Concept and Relation trees in a given WSML document.

4. Related Work

There are a number of other efforts to produce an ontology creation tool using the WSML language. This section of the deliverable takes a quick look at two, where they are and where they are going.

4.1 SWWS Studio and WSMO Studio

SWWS Studio [Dimitrov et al., 2004] is an ontology editor originally based on OWL-S, which was later ported to WSML, developed by Ontotext Lab as part of the Semantic Web enabled Web Services (SWWS) European IST Project. SWWS Studio can be used to create and edit Ontologies, Mediators, Web Services and Goals. Take up of SWWS Studio has been low due to the fact that development of the tool has completed and the tool is not up to date with the latest versions of the WSML language. SWWS Studio is also released under a closed license and therefore it is not possible for third parties to contribute to the effort.

v0.0.1 of the successor to SWWS Studio called WSMO Studio has been released by Ontotext Lab. WSMO Studio is a tool for ontology editing, publishing and browsing tool, developed as part of the DIP European IST project. This tool will allow users to create Ontologies, Mediators, Web Services and Goals in the WSML language using a collection of Eclipse plug-ins. It will then be possible to publish and manage these semantic descriptions in their relevant repositories.

4.2 OMWG - Ontology Editing and Browsing tool

The OMWG Ontology Editing and Browsing tool [Henke, 2004] is a tool for creating and managing Ontologies in the WSML language currently under development in the Ontology Management Working Group (OMWG). The tool is developed as an eclipse plug-in and provides two views for creating and modifying the elements of Ontologies. The first view, called the 'Class Explorer', is used to view and modify 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 there 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.

Appendix A: Predefined Namespaces

All files related to the WSML Editor are located within the 'wsmleditor' directory of the web services modeling toolkit installation folder. The 'namespaces.xml' file contains the list of predefined namespaces and their properties. The Dublin-Core [Weibel et al. 1998], WSML and XML-Schema namespaces are shipped by default with the WSML Editor. The following is a list of properties and data-types (values) shipped with each of the namespaces:

Dublin-Core:
Properties:

- title
- subject
- description
- type
- source
- relation
- coverage
- creator
- publisher
- contributor
- rights
- date
- format
- identifier
- language

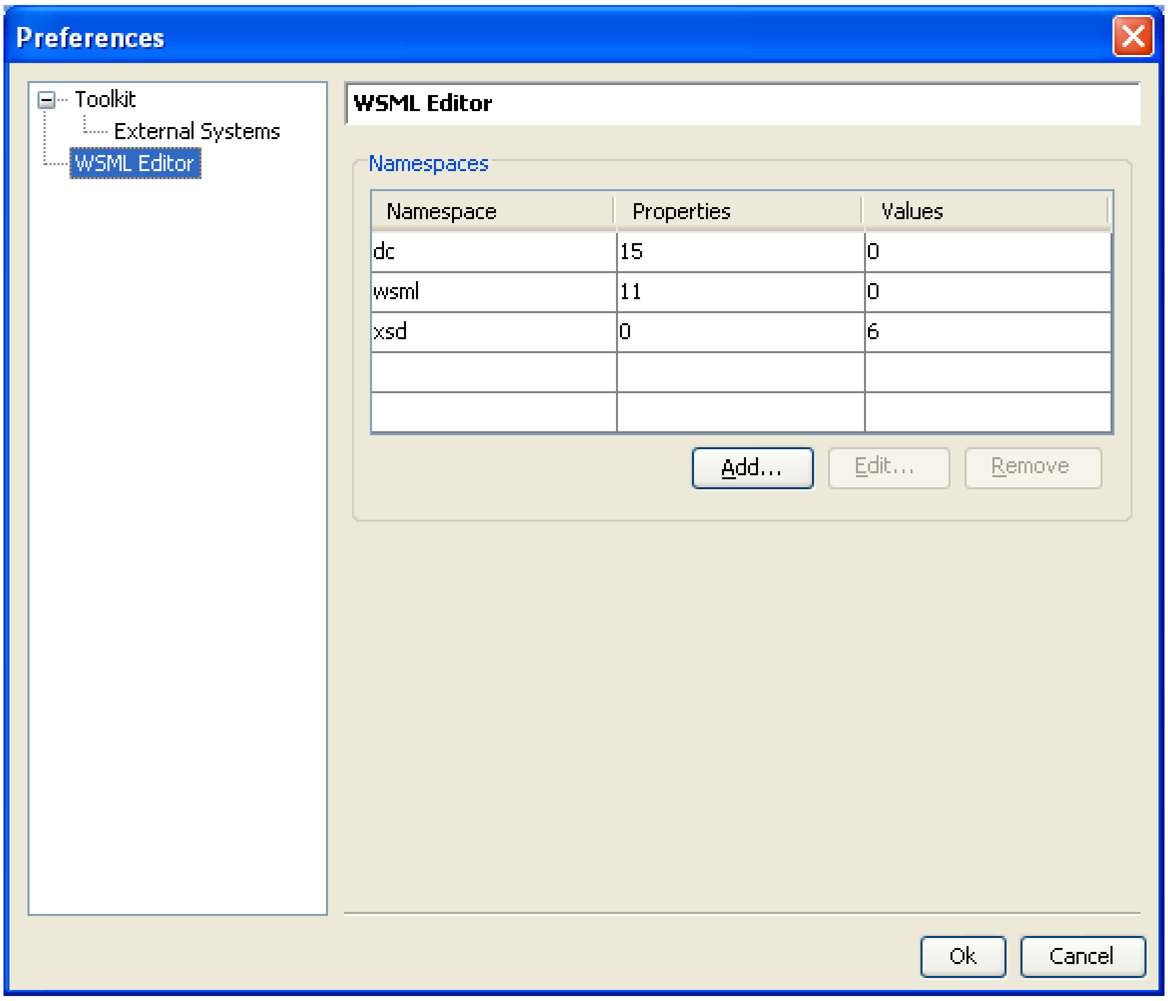
WSML:
Properties:

- version
- accuracy
- financial
- networkRelatedQoS
- performance
- reliability
- robustness
- scalability
- security
- transactional
- trust

XSD:
Values:

- string
- decimal
- integer
- boolean
- date
- time

The XML document shipped with the WSML Editor can be obtained here, the file can be changed in the application folder and changes to the document will be reflected in the application once the application is restarted, the xml document should conform to the following DTD. Alternatively the XML document can be edited via the preference dialog.



Namespace	Properties	Values
dc	15	0
wsml	11	0
xsd	0	6

Figure 17. Predefined Namespaces

References

[Cimpian et al., 2005] E. Cimpian, T. Vitvar, and M. Zaremba, Overview and Scope of WSMX, http://www.wsmo.org/TR/d13/d13.0/

[de Bruijn et al., 2004] J. de Bruijn, H. Lausen, and D. Fensel, The WSML Family of Representation Languages, http://www.wsmo.org/TR/d16/d16.1/

[Dimitrov et al., 2004] M. Dimitrov, Z. Marinova, P. Radkov, SWWS - Prototype Tools II http://swws.semanticweb.org/public_doc/D5.2.pdf

[Fensel & Bussler, 2002] D. Fensel and C. Bussler: The Web Service Modeling Framework WSMF, Electronic Commerce Research and Applications, 1(2), 2002.

[Henke, 2004] J. Henke, OMWG Editing and Browsing Implementation, http://www.omwg.org/TR/d8/d8.3/

[Kerrigan, 2005] M. Kerrigan, Web Services Modeling Toolkit (WSMT), http://www.wsmo.org/TR/d9/d9.1/

[Patel-Schneider et al., 2004] P. F. Patel-Schneider, P. Hayes, and I. Horrocks: OWL web ontology language semantics and abstract syntax . Recommendation 10 February 2004, W3C, 2004. Available from http://www.w3.org/TR/owl-semantics/ .

[Roman et al., 2004] D. Roman, H. Lausen, U. Keller (eds.): Web Service Modeling Ontology (WSMO), http://www.wsmo.org/TR/d2/

[Weibel et al. 1998] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf: RFC 2413 - Dublin Core Metadata for Resource Discovery

[WSMO4J] Available at http://wsmo4j.sourceforge.net/

[Yang et al., 2003] G. Yang, M. Kifer, and C. Zhao. FLORA-2: A rule- based knowledge representation and inference infrastructure for the Semantic Web. In Proceedings of the Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE) , Catania, Sicily, Italy, 2003.

Acknowledgement

This work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the members of the WSMX working group for their advises and inputs to this document.

---

webmaster