



D9.1v0.2 Web Service Modeling Toolkit (WSMT)

WSMX Working Draft 25 April 2005

This version:

<http://www.wsmo.org/TR/d9/d9.1/v0.2/20050425>

Latest version:

<http://www.wsmo.org/TR/d9/d9.1/v0.2/>

Previous version:

<http://www.wsmo.org/TR/d9/d9.1/v0.1/>

Editors:

Mick Kerrigan

Authors:

Mick Kerrigan

This document is also available in non-normative [PDF](#) version.

Copyright © 2005 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Table of contents

1. Introduction

[1.1 Overview](#)

[1.2 Purpose of this Document](#)

[1.3 Downloading the Web Services Modeling Toolkit](#)

2. Web Services Modeling Toolkit

[2.1 Contributing to Application Menubar](#)

[2.2 Contributing to Application Toolbar](#)

[2.3 Contributing to Preferences Dialog](#)

[2.4 Switching Between Plug-ins](#)

3. Core Components

[3.1 Plug-in Loader](#)

[3.2 Logger](#)

[3.3 Communication Component](#)

[3.3.1 Invoker](#)

[3.3.2 Receiver](#)

4. Creating a Tool as a Plug-in

5. Future Work

References

Acknowledgement

1. Introduction

1.1 Overview

The Web Service Modeling Ontology (WSMO) [Roman et al., 2004] is an ontology for describing Semantic Web Services. WSMO is based on the Web Service Modeling Framework (WSMF) [Fensel & Bussler, 2002] and as such is based on the four main elements of the WSMF: Ontologies, Goals, Mediators and Web Services. The Web Service Modeling Language (WSML) [de Bruijn et al., 2004] is a formalization of the WSMO ontology and provides a number of languages within which the properties of Semantic Web Services can be described. These languages are based on Description Logic and Logic Programming, each language variant providing "different levels of logical expressiveness" [de Bruijn et al., 2004]. The Web Service Execution Environment (WSMX) is a reference implementation of WSMO using the WSML family of languages. WSMX is an execution environment for the dynamic discovery, mediation, composition and invocation of Semantic Web Services. WSMX uses WSMO as its conceptual model and defines its own execution semantics [Oren, 2004], architecture [Zaremba et al., 2004] and implementation [Moran, 2004].

1.2 Purpose of this Document

The purpose of this document is to outline the Web Services Modeling Toolkit (WSMT). WSMT is a framework for the rapid deployment of graphical administrative tools, which can be used with WSMO, WSML and WSMX.

1.3 Downloading the Web Services Modeling Toolkit

The latest version (v0.2) of the WSMT is available at <http://www.sourceforge.net/projects/wsmx/>.

2. Web Services Modeling Toolkit

As research into the Semantic Web continues within the WSMO, WSML and WSMX clusters at DERI, the need for tools to manage resources and systems is becoming more apparent. The WSMT provides the functionality to deploy these tools as plug-ins to a larger system, allowing the users of Semantic Web technology to install one client side application for multiple tasks. For tool developers the framework provides the basic functionality for housing a tool and reduces the need for each developer to redevelop this functionality for each tool. This enables increased developer productivity and allows a developer to rapidly develop tools for WSMO and WSML or for testing the components in the WSMX system.

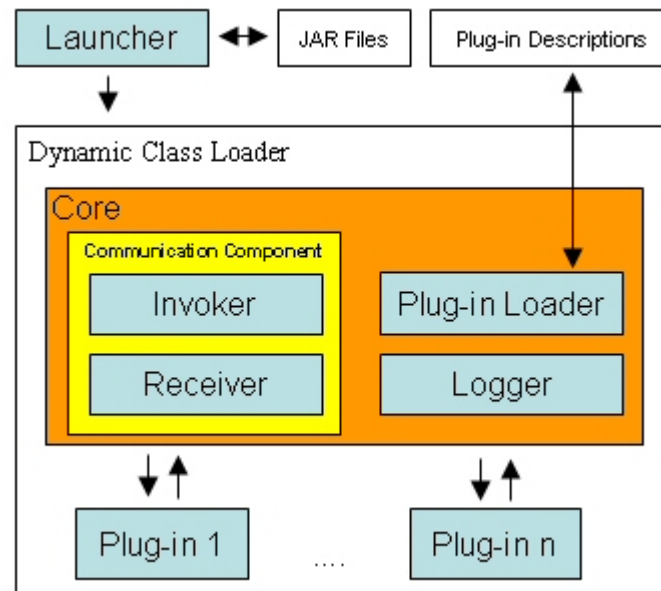


Figure 1 : Web Services Modeling Toolkit Architecture

The WSMT is a three tiered architecture. The first tier is called the launcher, its major function is to build a dynamic class loader within which the application will run. This functionality allows new plug-ins to be deployed into the application without recompiling. The second tier of the architecture is called the core. The core contains all the 'glue' functionality used by the individual plug-ins. The core provides the application frame, menu bars, toolbars and preference dialog, which can all be contributed to by the current visible plug-in. Also contained in the core are a set of components that provide reusable functionality to all the plug-ins. The third tier is made up of the individual plug-ins. The current version of the WSMT (v0.2) contains two tools. The first tool is the WSML Editor used to create and manage WSML documents, more information on which can be found in deliverable d9.2 [Kerrigan, 2005a]. The second tool is the WSMX Invoker, which was developed to replace the WSMX backend application used in WSMX demos. The WSMX Invoker provides functionality for invoking the Web Service interfaces of WSMX and for monitoring the incoming and outgoing messages in the WSMT Communication Component, more information on the WSMX Invoker can be found in deliverable d9.4 [Kerrigan, 2005c]. A stub also exists for the planned WSMX Monitor plug-in, more information on the WSMX Monitor will be available in deliverable d9.3 [Kerrigan, 2005b]. The Web Services Modeling Toolkit is currently packaged in an install for both windows and linux, with optional Java 1.5 runtime.

The WSMT architecture is designed as it is to facilitate deploying of new plug-ins to the framework as easy as possible and reduce the overhead for application development. One major decision taken during the design phase was not to use the Eclipse framework as a basis for the Web Services Modeling Toolkit. The Eclipse framework is an application framework allowing for simple plug-ins to be created and allow more complex applications to grow organically from these simple plug-ins. One major issue with Eclipse is that it uses IBM's Standard Widget Toolkit (SWT) as its graphical front-end instead of using Java's Swing graphics library. The SWT library provides a graphical look and feel which is native to the operating system on which the application is running. The SWT library is not a Java standard and is not interoperable with Swing (although efforts to make them interoperable are underway). This means that existing tools for Semantic Web Services (for example the WSMX Mediation GUI, which will be integrated with the WSMT in v0.3) would need to be rewritten to make them accessible from the WSMT and once they were rewritten the tools would be dependant on the Eclipse framework. The SWT library also goes against one of the most important principles of Java, platform independence. IBM have to release a SWT native port for each of the operating systems you wish to support, these ports are in different states of development, some being more optimized than others (for example SWT

runs quickly on windows, but slower than Swing on Linux and Mac's). Also most java developers have some experience with Swing and having the overhead of learning how to use the SWT in order to use the WSMT framework could discourage third parties from creating tools. Finally there are a large number of Swing libraries available for performing functions like graphing and 2D graphics (needed in the WSMX Monitor), while there are much less libraries available for SWT.

2.1 Contributing to Application Menubar

By default the Web Services Modeling Toolkit provides a very simple menu bar containing only items for exiting the application, showing the preferences, switching between plug-ins and viewing the about dialog.

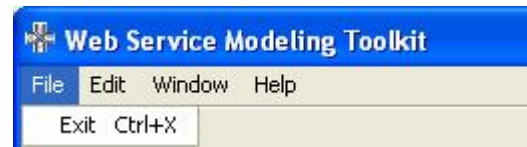


Figure 2: Basic Menubar

Individual plug-ins can add new items to the menu bar (including new top level sections e.g. File and Edit). This is done by implementing the getMenuBar method in the PluginPanel interface (see [section 4](#)). The Menubar returned by the plug-in is merged with the main application menu bar. Figure 3, shows the application menubar when the WSMML Editor plug-in (v0.3) is displayed.

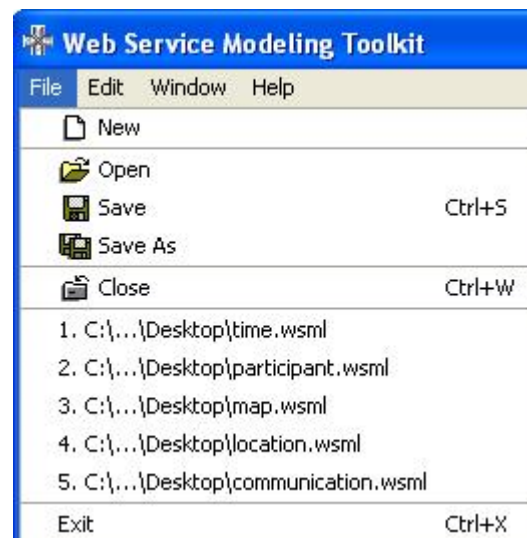


Figure 3: WSMML Editor Menubar

2.2 Contributing to Application Toolbar

The Toolbar provided by the WSMT contains only a toolbar item for launching the preferences dialog.



Figure 4: Basic Toolbar

Individual plug-ins can contribute to the application toolbar by implementing the contributeToToolbar method in the PluginPanel interface (see [section 4](#)). Figure 5 shows the application toolbar when the WSMML Editor Plug-in (v0.2) is displayed.



Figure 5: WSMML Editor Toolbar

2.3 Contributing to Preferences Dialog

The Web Services Modeling Toolkit uses the Preferences interface provided by the open source [powerswing project](#) (v0.3). The preference dialog comprises a hierarchical tree of preferences. Each node in the tree has an association preference panel which is shown when the node is selected. Each plug-in can contribute a sub tree of preference nodes to the preference dialog by implementing the `getPreferenceTreeNode` method in the `PluginPanel` interface (see [section 4](#)). The preferences that will be updated by the preference panels shown need to be registered (so that they are available to the panels in the preference dialog). This is done by registering the preferences by implementing the `registerPreferences` method in the `PluginPanel` interface (see [section 4](#)). Figure 6 shows the preference dialog, showing the basic preferences of the application along with the sub tree contributed by the WSMML Editor plug-in.

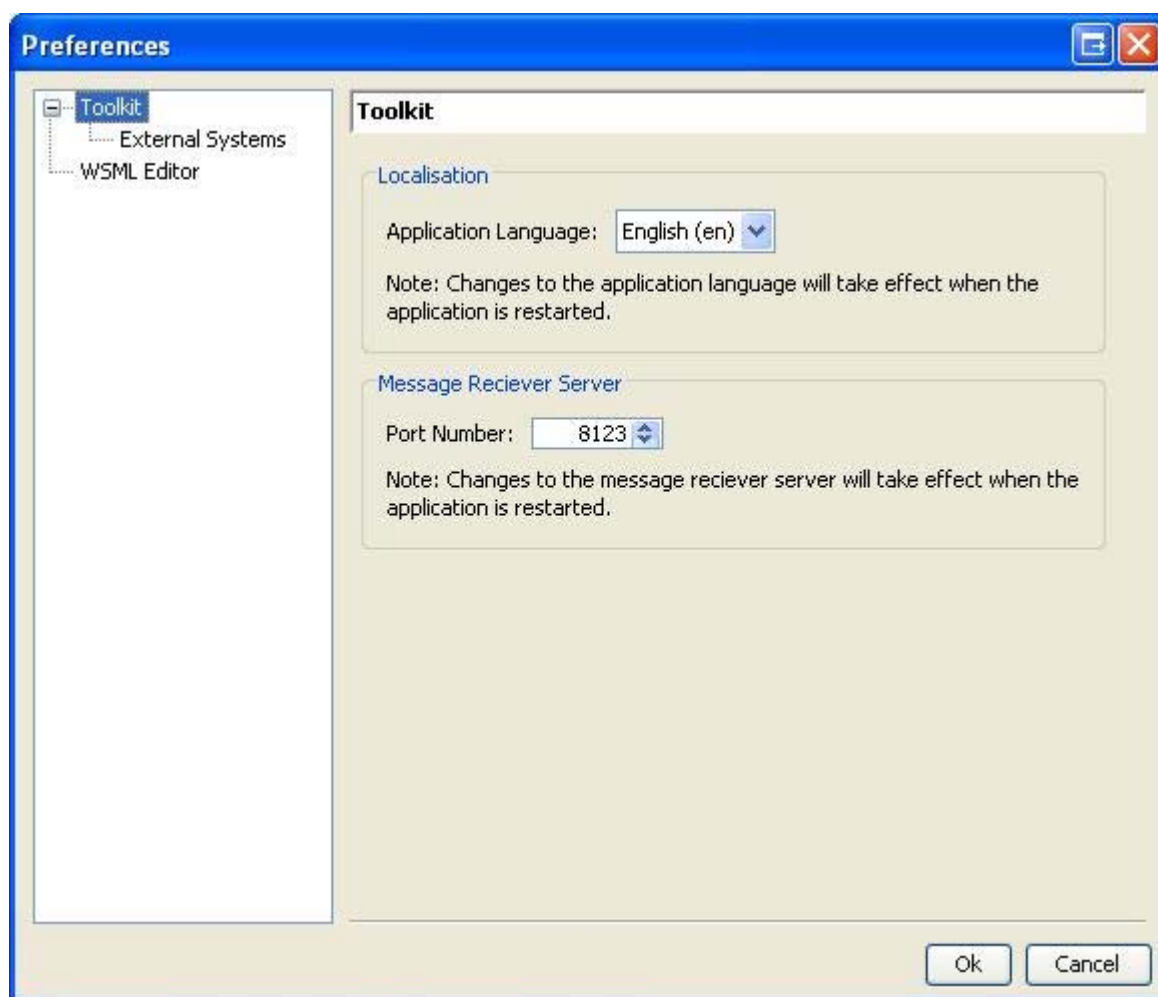


Figure 6: The Web Services Modeling Toolkit Preferences Dialog

2.4 Switching Between Plug-ins

In the previous version of the Web Services Modeling Toolkit the plug-ins were all shown in a tabbed pane and switching between them involved changing tabs. This mechanism was cumbersome and would not scale as the number of plug-ins increased. In the current version of the WSMT this mechanism has been changed to a window menu. To switch between plug-ins the user just needs to select the appropriate plug-in from the window menu. Figure 7 shows the window menu for the WSMT.



Figure 7: WSMT Window Menu

3 Core Components

The core tier of the Web Services Modeling Toolkit provides a number of components that can be used by each of the plug-ins. The components located in the core include:

3.1 Plug-in Loader

This component is responsible for loading all of the plug-ins into the framework. It is also responsible for checking that each of the individual plug-ins are compatible with the current version of the Web Services Modeling Toolkit and with the current version of the Java virtual machine. While this component is currently not directly used by the plug-ins, in the future this will be interface for inter plug-in communication.

3.2 Logger

The logging component uses Log4J and provides a generic logging interface for all the plug-ins. The logging is controlled using the log.conf file shipped in the core directory of the application. Application level logging can be switched to different levels including DEBUG, INFO, WARN, ERROR and FATAL. Logging can also be changed at the plug-in level by specifying the level for the java package which contains the plug-in.

```
log4j.category.ie.deri.wsmleditor=DEBUG  
log4j.category.ie.deri.wsmxinvoker=WARN
```

Figure 8. Setting Logging level for a particular plug-in.

3.3 Communication Component

The communication component is made up of two sub-components. These components are called the Invoker and Receiver.

3.3.1 Invoker

The Invoker sub-component of the communication component is responsible for invoking web services. It uses the Web Services invocation framework. The interface to the invoker is a simple one. The plug-in must provide a configuration containing a WSDL, a service, port-type, operation and an array data parts. Currently only synchronous communication is possible, however with the use of WS-Addressing it should be possible to inform the services how to send responses back to the WSMT in an asynchronous manner.

The Invoker also provides functionality for creating the configurations it requires as input. By supplying a WSDL document the interface provides a breakdown of the WSDL document by service, port-type, operation, expected inputs and outputs.

3.3.2 Receiver

To enable asynchronous communication in the future a web-server has been embedded in the WSMT. This web server is running a copy of apache axis and contains a single service which can receive a WSM message. Individual plug-ins can subscribe to the Receiver to receive notification when WSM messages are received. The receiver will have a more important function when the invoker is using WS-Addressing to enable Web Services respond to the WSMT asynchronously.

4. Creating a Tool as a Plug-in

The process for creating a tool as a plug-in is a very straight forward one. The source code for the tool should be packed in a jar and placed in the lib folder of the WSMT installation along with the plug-in description file (.pdesc). WSMT reads all the plug-in description files at startup and loads each of the plug-ins specified. This list of installed plug-ins can be found in the plug-ins section for the about dialog.

```
<?xml version="1.0" encoding="UTF-8"?>
<pluginDescription version="1.0">
  <id>wsmleditor</id>
  <version>0.2</version>
  <classname>ie.deri.wsmtool.wsmleditor.WSMLEditorPlugin</classname>
  <author>Mick Kerrigan</author>
  <wsmt-version>0.2</wsmt-version>
  <java-version>1.5</java-version>
</pluginDescription>
```

wsmleditor.pdesc

The plug-in description file allows WSMT to perform plug-in detection. The file gives the plug-in's id, version and the classname for creating an instance of the tool. The classname specified is that of an implementation of the `ie.deri.wsmtool.plugin.Plugin` interface. The supplied `wsmt-version` and `java-version` numbers allow the plug-in loader to perform compatibility checking between the plug-in and the current version of the WSMT and the current Java virtual machine.

```
package ie.deri.wsmtool.plugin;

import ie.deri.wsmtool.gui.PluginPanel;

import java.awt.Container;
import java.awt.Dimension;
import java.util.Locale;

public interface Plugin {

    public void init(String theInstallPath, PluginDescription
        thePluginDescription, Locale theLocale, Container theParent,
        WSMTIconRepository theRepository, ExternalSystems
        theExternalSystems) throws PluginInitializationFailedException;
    public String getId();
    public String getName();
    public String getLabel();
    public String getVersion();
    public String getDescription();
    public ArrayList <String> getAuthors();
    public String getWsmVersion();
```

```
    public String getJavaVersion();
    public PluginPanel getPanel();
    public Dimension getDimension();
}
```

The tool itself is created by creating a subclass of the abstract PluginPanel class, which is a subclass of JPanel.

```
package ie.deri.wsmtool.gui;

import javax.swing.JMenuBar;
import javax.swing.JPanel;

public abstract class PluginPanel extends JPanel{

    public abstract JMenuBar getMenuBar(ActionListener refreshMenuBar);
    public abstract void contributeToToolBar(PJToolBar theToolBar);
    public abstract void registerPreferences(Preferences thePreferences, String
thePrefix);
    public abstract PreferenceTreeNode getPreferenceTreeNode
(PreferencePanelRegistry thePreferencePanelRegistry, int theIdRange);
    public abstract void closeAll();
}
```

5. Future Work

Currently the communication component supports both message sending and message receiving using the Invoker and Receiver sub-components of the communication component. However these two are not linked in that services, which are invoked by the Invoker, are not informed of how to send messages back asynchronously to the WSMT. WS-Addressing will need to be added to the Invoker to give information to the service on how to call-back the WSMT.

As the need for further tools are required they can be developed to work within the WSMT. Already discussions regarding WSMX Management, WSMX Mediation and WSMX Choreography tools have begun.

References

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

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

[Kerrigan, 2005a] M. Kerrigan, WSML Editor, <http://www.wsmo.org/2005/d9/d9.2/>

[Kerrigan, 2005b] M. Kerrigan, WSMX Monitor, <http://www.wsmo.org/2005/d9/d9.3/>

[Kerrigan, 2005c] M. Kerrigan, WSMX Invoker, <http://www.wsmo.org/2005/d9/d9.4/>

[Moran, 2004] M. Moran, WSMX Implementation, <http://www.wsmo.org/2004/d13/d13.5/>

[Oren, 2004] E. Oren. WSMX Execution Semantics, <http://www.wsmo.org/2004/d13/d13.2/>

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

[Zaremba et al., 2004] M. Zaremba, M. Moran, WSMX Architecture, <http://www.wsmo.org/2004/d13/d13.4/>

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 program.

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



[webmaster](#)