DIP Deliverable Template V1.1 MS 04/01/2006

DIP
Data, Information and Process Integration with Semantic Web Services

FP6 - 507483

Deliverable

WP 4b: WSMO Platform & Tools
D4.11
WSMO Studio v2

Alex Simov (Sirma)
Marin Dimitrov (Sirma)
Mick Kerrigan (UIBK)
Vassil Momtchev (Sirma)
Martin Hepp (UIBK)
Jan Henke (UIBK)
Marc Richardson (BT)

Jan 4th, 2006
SUMMARY

This deliverable presents the second prototype of an Integrated Service Environment for WSMO, called WSMO Studio.

The Studio is a standalone, Eclipse based application that supplies the following functionality:

• Creating WSMO descriptions of ontologies, goals, web services and mediators
• Export and import of the WSMO descriptions
• Storing and retrieving WSML descriptions to/from remote service, goal, mediator and ontology repositories
• Creating WSMO centric choreography descriptions
• Management of Semantic Web Service runtime environments, such as WSMX

This is the final deliverable of a set of deliverables presenting WSMO Studio. The two previous deliverables – D4.4 [5] and D4.5 [6] presented respectively the design of the Studio and the first prototype.

It is important to note that the presentation in the deliverable assumes that the reader is already acquainted with D4.4 and D4.5, since these deliverables contain detailed explanation of concepts and components that will be referred throughout the text.

The target audience of this deliverable consists of:

• **End users** that will use WSMO Studio to work with WSMO service descriptions. Sections 2, 3 and 4 contain information for such end users
• **Solution providers** that provide new functionality relevant to WSMO/WSMX and that would like to integrate this functionality with WSMO Studio. These readers should focus on section 5 as well as the relevant appendixes.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
Abstract (for dissemination)
This is the final deliverable that describes WSMO Studio – an Integrated Service Environment for WSMO. The deliverable focuses on following main topics:

- Changes in WSMO Studio since M18
- Presentation of the new functionality in WSMO Studio – the Choreography Editor and the WSMX Manager
- A guide for developers that intend to extend WSMO Studio functionality

Keywords
Semantic Web Services, WSMO, tools, Eclipse, GUI
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01-Dec-2005</td>
<td>001</td>
<td>Marin Dimitrov</td>
<td>Initial version</td>
</tr>
<tr>
<td>08-Dec-2005</td>
<td>002</td>
<td>Alex Simov</td>
<td>List of changes from previous version</td>
</tr>
<tr>
<td>12-Dec-2005</td>
<td>003</td>
<td>Mick Kerrigan</td>
<td>WSXM Management plug-in section</td>
</tr>
<tr>
<td>14-Dec-2005</td>
<td>004</td>
<td>Marin Dimitrov</td>
<td>Description of Choreography plug-in integrated</td>
</tr>
<tr>
<td>15-Dec-2005</td>
<td>005</td>
<td>Alex Simov</td>
<td>Developer’s section integrated</td>
</tr>
<tr>
<td>16-Dec-2005</td>
<td>006</td>
<td>Marin Dimitrov</td>
<td>Version submitted for internal review</td>
</tr>
<tr>
<td>04-Jan-2006</td>
<td>007</td>
<td>Alex Simov, Marin Dimitrov, Vassil Momtchev</td>
<td>Modifications according to the internal review recommendations</td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway | NUIG    | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway Ireland  
Email: chris.bussler@deri.org  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. Bankinter  
Paseo Castellana, 29  
28046 Madrid, Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH          | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH  
Oranienburger Str. 32  
10117 Berlin, Germany  
Email: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc. | BT      | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE, United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL    | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council            | Essex   | Mary Rowlett  
Essex County Council  
PO Box 11, County Hall, Duke Street  
Chelmsford, Essex, CM1 1LX  
United Kingdom.  
Email: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik    | FZI     | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe Germany  
Email: abecker@fzi.de  
Tel: +49 721 9654 0 |
<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | UIBK | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA | ILOG | Christian de Sainte Marie  
9 Rue de Verdun, 94253  
Gentilly, France  
Email: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG | inubit | Torsten Schmale  
inubit AG  
Lützowstraße 105-106  
D-10785 Berlin  
Germany  
Email: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A. | iSOCO | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
Email: rbenjamins@isoco.com  
Tel. +34 913 349 797 |
| NIWA WEB Solutions | NIWA | Alexander Wahler  
NIWA WEB Solutions  
Niederacher & Wahler OEG  
Kirchengasse 13/1a  
A-1070 Wien  
Email: wahler@niwa.at  
Tel:+43(0)1 3195843-11 | |
| The Open University | OU | Dr. John Domingue  
Knowledge Media Institute  
The Open University, Walton Hall  
Milton Keynes, MK7 6AA  
United Kingdom  
Email: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG | SAP | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
Email: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Person</th>
<th>Address</th>
<th>Email</th>
<th>Tel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov</td>
<td>Ontotext Lab, - Sirma AI EAD</td>
<td><a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td>+359 2 9768 303</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Office Express IT Centre, 3rd Floor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>135 Tzarigradsko Chausse</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sofia 1784, Bulgaria</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +359 2 9768 303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
<td>Unicorn Solutions Ltd,</td>
<td><a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td>+972 2 6491111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Malcha Technology Park 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jerusalem 96951</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Israel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +972 2 6491111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Pieter De Leenheer</td>
<td>Starlab- VUB</td>
<td><a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
<td>+32 (0) 2 629 3749</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vrije Universiteit Brussel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pleinlaan 2, G-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1050 Brussel ,Belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:Pieter.De.Leenheer@vub.ac.be">Pieter.De.Leenheer@vub.ac.be</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +32 (0) 2 629 3749</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## List of Key Words/Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>ISE</td>
<td>Integrated Service Environment</td>
</tr>
<tr>
<td>J2SE</td>
<td>Java 2 Platform Standard Edition</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management eXtensions</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>NFP</td>
<td>Non-Functional Property</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>LGPL</td>
<td>GNU Lesser General Public Licence</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Service Modeling Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSMX</td>
<td>Web Service Execution Environment</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

## SUMMARY .................................................................................................................................................. I

## LIST OF KEY WORDS/ABBREVIATIONS ................................................................................................. VII

## TABLE OF CONTENTS ............................................................................................................................... VIII

## 1 INTRODUCTION .......................................................................................................................................... 1

## 2 SUMMARY OF CHANGES ........................................................................................................................ 3

2.1 Update site .................................................................................................................................................. 3
2.2 Choreography editor ................................................................................................................................. 3
2.3 WSMX Manager ....................................................................................................................................... 3
2.4 WSML text editor ..................................................................................................................................... 3
2.5 Import from OWL-DL ............................................................................................................................... 4
2.6 Integrated local repository ...................................................................................................................... 4
2.7 Integrated WSML validator ................................................................................................................... 5
2.8 User preferences ...................................................................................................................................... 6
2.9 Drag & Drop support ............................................................................................................................... 7
2.10 New extension points ............................................................................................................................ 7
2.11 Bugfixes .................................................................................................................................................. 8

## 3 CHOREOGRAPHY EDITOR ....................................................................................................................... 9

3.1 WSMO based choreography ................................................................................................................... 9
3.2 User Interface .......................................................................................................................................... 10
3.2.1 State Signature Editor ....................................................................................................................... 13
3.2.2 Transition Rules Component ........................................................................................................... 13

## 4 WSMX MANAGER .................................................................................................................................... 16

4.1 WSMX Server ......................................................................................................................................... 16
4.2 Adapter Framework ............................................................................................................................... 17
4.3 WSMX Manager plug-in ....................................................................................................................... 17
4.3.1 Adapter Framework View ............................................................................................................... 19
4.3.2 WSMX Server View ....................................................................................................................... 20

## 5 DEVELOPERS GUIDE ............................................................................................................................. 23

5.1 Summary of changes ............................................................................................................................... 23
5.1.1 Migration to Java 5 ............................................................................................................................ 23
5.1.2 Improved MVC infrastructure ........................................................................................................ 23
5.1.3 Load-on-demand of entities in the workspace ................................................................................ 24
5.1.4 Improved logging ................................................................. 24
5.1.5 New extension points ......................................................... 24
5.1.6 Updated extension points ................................................. 25
5.2 Extension point summary ........................................................ 25
5.2.1 org.wsmostudio.runtime.wsmoimpl .................................... 25
5.2.2 org.wsmostudio.ui.editors .................................................. 26
5.2.3 org.wsmostudio.repository.Repository ................................ 28
5.2.4 org.wsmostudio.ui.navigator_content ................................ 30
5.3 Examples .............................................................................. 32
6 CONCLUSION ........................................................................... 36
REFERENCES .............................................................................. 37
APPENDIX A – 3RD PARTY COMPONENTS AND LICENCES ................. 38
APPENDIX B – TECHNICAL INFORMATION .................................... 39

LIST OF FIGURES
Figure 1 WSML Text Editor .............................................................. 4
Figure 2 WSMO Validator ................................................................. 5
Figure 3 Preference dialog (WSMO editors) ................................. 6
Figure 4 Preference Dialog (WSML text editor) ............................ 7
Figure 5 Choreography Editor (full) ............................................. 10
Figure 6 Web Service components ................................................ 11
Figure 7 Choreography editor (partial) .......................................... 11
Figure 8 State Signature editor ...................................................... 13
Figure 9 Rule viewer ................................................................. 14
Figure 10 Rule editor (simple rules) ............................................ 14
Figure 11 Rule editor (composite rules) .......................................... 15
Figure 12 Choreography definition in the WSML text editor ........ 15
Figure 13 Adding a new WSMX Server ........................................ 18
Figure 14 Adding an Adapter Framework .................................... 18
Figure 15 Functionality of Adapter Framework ............................ 19
Figure 16 Deployed adapters ....................................................... 19
Figure 17 Testing an Adapter ..................................................... 20
Figure 18 Viewing deployed components .................................... 21
Figure 19 Component properties ................................................. 21
Figure 20 Invoking operations of a component ........................... 22
Figure 21 WSMX Management Console ................................................................. 22
Figure 22 Initial choreography visualisation ....................................................... 32
Figure 23 Extended WSMO Navigator ................................................................. 35

LIST OF TABLES
Table 1 3rd party components integrated with WSMO Studio ............................. 38
1 INTRODUCTION

The role of WP4b in DIP is to provide a set of tools for using and exploiting semantically described Web Services. Within WP4, three deliverables covered the design and implementation of an Integrated Service Environment (ISE) for WSMO: D4.4 [5], D4.5 [6] and the current deliverable, D4.11.

To summarise – the goals of the WSMO Studio effort are:

- Providing a GUI tool that assists the users working in the WSMO domain with tasks related to semantic web service annotation.
- Providing a extensible tool and architecture that will allow 3rd parties to integrate and extend WSMO Studio functionality

This deliverable presents the second prototype of such an ISE, called WSMO Studio. The Studio is a standalone, Eclipse based application that supplies the following functionality:

- Creating WSMO descriptions of ontologies, goals, web services and mediators
- Export and import of the WSMO descriptions
- Storing and retrieving WSML descriptions to/from remote service, goal, mediator and ontology repositories
- Creating WSMO centric choreography descriptions
- Managing Semantic Web Service runtime environments, such as WSMX.

Future versions of this prototype will provide improvements of the existing functionality as well as new functionality related to WSMO centric choreography and orchestration, discovery and interaction with Semantic Web Services runtime environments such as WSMX.

The target audience of this deliverable consists of:

- **End users** that will use WSMO Studio to work with WSMO service descriptions. The sections that explain the user interface and the basic functionality of WSMO Studio are intended for such readers.
- **Solution providers** that provide new functionality relevant to WSMO/WSMX and that would like to integrate this functionality with WSMO Studio. These readers should focus on the sections that contain technical details of the WSMO

---

1. [http://www.wsmo.org](http://www.wsmo.org)
2. [http://www.wsmostudio.org](http://www.wsmostudio.org)
4. [http://www.wsmx.org](http://www.wsmx.org)
Studio architecture and specifically the extension point definitions that make it possible for new functionality to be dynamically integrated within the tool.

This document is structured as follows:

- Section 2 presents a summary of the changes, since first official release of *WSMO Studio* in July 2005
- Sections 3 and 4 present two new components (plug-ins) of WSMO Studio – the Choreography Editor and the WSMX Manager.
- Finally, section 5 (and the relevant appendixes) presents technical details and a guide for developers that intend to extend existing WSMO Studio functionality or integrate new functionality

It is important to note that:

- The present document assumes that the respective reader is already acquainted with D4.4 and D4.5, since these documents introduce concepts and components that will be referred throughout the text
- A lot of additional information (both for end users and for developers) is available at the *WSMO Studio* web site: [http://www.wsmostudio.org](http://www.wsmostudio.org)
2 SUMMARY OF CHANGES

The purpose of this section is to provide a summary of the new features and modifications to WSMO Studio since its first public release (0.1.2, 13 Jul 2005).

2.1 Update site

Eclipse provides a standard functionality to provide end-user updates upon request by the means of the so called update site [9]. The user can easily perform updates of the installed functionality, without any knowledge of the underlying technical details regarding the plug-in deployment.

A dedicated component (update manager) is responsible for checking for new versions and installing them (if available). The location of the WSMO Studio update site is: http://www.wsmostudio.org/updates/ and the user can activate the update manager from the standard menu (Help → Software Updates → Find and Install).

2.2 Choreography editor

A full-featured WSMO Choreography Editor is introduced in this version of WSMO Studio. The editor provides a GUI for creating WSMO centric choreography descriptions: state signatures, modes, rules, etc. Detailed information about the Choreography Editor is available in section 3.

2.3 WSMX Manager

A plug-in for managing of WSMX servers has been integrated with this WSMO Studio. The WSMX Manager provides functionality for listing the active WSMX components, access to their attributes, ability to invoke certain operations on each components and functionality for deploying and undeploying components at runtime. More information about the WSMX Manager is available in section 4.

2.4 WSML text editor

A new WSML text editor was introduced, that allows editing plain text WSML descriptions (see Figure 1). To facilitate users work, the editor highlights the keywords in the text. It allows several levels of syntax highlighting, i.e. subsets of keywords can be coloured independently. The list of words specified for highlighting is open and the
user can additionally include other keywords. The highlighting colours are customisable by the end user as well. The editor possesses typical text-based functionalities for such components: Copy/Cut/Paste/Find/Replace, etc. The editor does not require valid input, so it can be used for repairing invalid WSML files.

```
relation ageOfHuman( ofType Human, ofType _integer)
   nfp
   dc#relation hasValue FunctionalDependencyAge
endnfp

axiom isAlive
definedBy
   ?x[isAlive hasValue _boolean("true")]
   naf (?x[hasObit hasValue ?obit] memberOf Human).
   ?x[isAlive hasValue _boolean("false")]
   impliedBy
   ?x[hasObit hasValue ?obit] memberOf Human.
```

Figure 1 WSML Text Editor

2.5 Import from OWL-DL

This feature enables files in OWL\(^5\) format to be used in the WSMO studio. The importing is supported by a wizard-like interface where the user selects an input file and a location in the workspace to store it. In order to make it compatible with the studio's environment the result is stored in the workspace in WSML format (i.e. an OWL-DL to WSML transformation is performed\(^6\)).

2.6 Integrated local repository

A local repository, based on Sesame\(^7\), is now integrated in WSMO Studio. To use the local repository the user must choose “ORDI Repository” from the context menu of the Repository plug-in.

\(^5\) A subset of OWL-DL is supported at present. The relevant functionality for importing OWL-DL is provided from the wsmo4j framework.

\(^6\) Note that an OWL-DL to WSML mapping is not always possible.

\(^7\) http://www.openrdf.org
Since its first release in July 2005, WSMO Studio provided an extensible mechanism for the end user to interact with ontology / goal / service repositories. The local ORDI plug-in makes it possible for the end user to store WSMO descriptions in a local ORDI repository (based on Sesame).

Future versions of WSMO Studio may provide adapters for other types of repositories as well.

2.7 Integrated WSML validator

The wsmo4j validator for WSML is integrated in WSMO Studio. All errors, warnings and notifications produced from the validator are reflected in the studio's graphical environment (see Figure 2) in the standard for Problems view of Eclipse. The information associated with each problem is:

- severity,
- explanation message
- problematic location.

If the error concerns parsing problems, the line/position in the text content is indicated.

![Figure 2 WSMO Validator](image)

To avoid unnecessary overhead, validation is performed only when a WSML file is opened or saved. The information about the problems identified is preserved between working sessions. The WSML validation can be disabled from the WSMO Studio's preference pages.
2.8 User preferences

*WSMO Studio* defines a set of preference pages which allow workbench customisations that will be preserved between working sessions (see Figure 3 and Figure 4). The set of preferences may vary depending on the installed components (plug-ins). It is extensible, so 3rd party components can contribute/use additional properties.

![Figure 3 Preference dialog (WSMO editors)](image-url)
2.9 Drag & Drop support

Different entities can now be “dragged” from the WSMO Navigator, and dropped in various editors. Certain restrictions apply – for example, in order to be able to “drop” an object into a visual component, the latter should be capable to handling the respective object.

Drag & Drop is supported for repositories in the Repository plug-in as well. In the previous version the user had to export / import WSML files from the local workspace into the remote repository using the context menu – the same result can now be achieved by dragging & dropping WSML files from the Navigator into the repository (and vice versa).

2.10 New extension points

Two new extension points were added to existing plug-ins:

- For plugging custom wsmo4j parsers (incl. parsers that support different formats of WSML, in addition to the supported default WSML, XML and OWL)
- For customising the WSMO Navigator – this way the user may override the default navigator visualisation (incl. the exact entity hierarchy, the labels of the entities, and the visibility of specific entity types).
More information about extension points is available in section 5.2.

2.11 Bugfixes

Many minor bugs were fixed since the first public release of WSMO Studio. The complete list of bugfixes is available at http://www.wsmostudio.org/changes-report.html.
3 Choreography Editor

The Choreography Editor provides functionality for creating WSMO based choreographies [12] (e.g. the choreography part of the service interface).

3.1 WSMO based choreography

In brief, a WSMO based choreography is comprised of:

- A *state signature* that specifies the state ontology for the choreography. The concepts and relations of this ontology will be used to express the state of the choreography, e.g. the actual states are instances of concepts / relations from the state ontology. In addition to the specification of the exact state ontology, the state signature also specifies several *roles* (also called *modes*) for the concepts and relations of the state ontology. There are five predefined roles:
  - *Static* (default) – the instances of such concepts/relations cannot be changed by the choreography execution
  - *In* – instances can only be read by the choreography, but can be changed by the environment (e.g. the agents external to the service). A *grounding* may be provided, that specifies means for the environment to modify such instances.
  - *Controlled* – instances of such concepts / relations can be created and modified only by the choreography execution and cannot be accessed by the environment
  - *Shared* – instances can be created or modified both by the choreography execution and by its environment. A grounding mechanism is specified as well.
  - *Out* – instances can be created / modified by the choreography execution, but the environment can only read such instances. A grounding mechanism must be specified, that provides read-only access for the environment.

- A set of *transition rules* that express state changes (e.g. changes in the set of instances that comprise the choreography state). The most basic form of transition rule are update functions, which add / remove / modify instances of concepts and relations. The three different types of update functions are:
  - *add(fact)*
  - *delete(fact)*
  - *update(fact\textsubscript{old} \rightarrow fact\textsubscript{new})*
where a fact is either a membership fact (e.g. “X memberOf Y”) or an attribute fact (e.g. “X hasValue Y”). With the help of basic update rules, more complex transition rules can be defined recursively:

- **if** _condition_ **then** _rules_ **endif**
- **forAll** _variables_ with _condition_ **do** _rules_ **endForAll**
- **choose** _variables_ with _condition_ **do** _rules_ **endChoose**

... where conditions are restricted WSML logical expressions and variables are WSML variables (see [2] for details).

### 3.2 User Interface

This section provides more details in the Choreography Editor UI (see Figure 5).

![Figure 5 Choreography Editor (full)](image)

When a Web Service is selected in the Project Navigator (top-left pane of the user interface), the WSMO Navigator, located in the bottom-left part of the user interface, will show a summary of the Web Service components (see Figure 6):

- The Capability of the service
- The Interface of the service
• And the Choreography, part of the interface definition

![Image of web service components]

Figure 6 Web Service components

When the choreography is open for editing (by double-clicking on the respective element or selecting Edit from the context menu), the choreography editor is activated (see Figure 7):

![Image of choreography editor]

Figure 7 Choreography editor (partial)
The editor reuses several standard graphical components of WSMO Studio, such as Identifier and NonFunctionalProperties editor, and the imported ontologies container.

The interface area of the Choreography Editor is divided in two parts, for the State Signature editor and the Rules editor.
3.2.1 State Signature Editor

The State Signature editor provides functionality for describing the state signature of the choreography, e.g. importing a state ontology, and defining the modes / roles of the concepts and relations that will define the instance states (see Figure 8):

For each type of mode, the following functionality is provided:

- **Adding** a new mode (entity) – instances of the selected concept / relation will comprise the state of the choreography
- **Removing** a mode (entity)
- **Add / remove grounding** (only for in, out and shared modes) – provides a grounding mechanism for the selected mode (for example, a reference to a WSDL message)

3.2.2 Transition Rules Component

The Rules component provides functionality for visualising and editing transition rules. The left part of the component is comprised of a tree-based viewer that shows the list of transition rules, defined for the choreography (see Figure 9). If the rules are composite (e.g. if-then, forAll and choose rules), then the nested rules will be shown in the hierarchy too.

New rules can be added\(^8\), or existing rules can be removed from the hierarchy by using the actions in the context menu.

\(^8\) Note that the transition rules are unordered, e.g. the specific position in which they appear in the list is insignificant.
When a particular rule is selected, a rule editor is activated in the bottom-right part of the user interface (see Figure 10 and Figure 11). The specific user interface components of the editor differ according to the nature of the rule – the editor for simple rules (Figure 10) is comprised of a single text area for providing the contents of the rule. If the rule is a composite one, then the editor also provides means for specifying conditions and variables (Figure 11). The conditions (in the form of restricted WSML axioms) are validated upon saving the choreography definition using the validator described in section 2.7.

Note that at present the user interface for rules is very simple and rules are edited as plain text. Future versions of WSMO Studio may provide more advanced rule editors.
The complete choreography definition can be viewed and edited in the default WSML text editor, described in section 2.4, as well (see Figure 12):
4 WSMX Manager

The Web Service Execution Environment (WSMX) [3] is the reference implementation for the WSMO. The aim of WSMX is to provide a framework for the dynamic discovery, negotiation, selection and invocation of semantic services described using WSMO. From a high level perspective the WSMX architecture is made up of three systems:

- WSMX Server
- The Adapter Framework
- GUI tools for management (WSMX Manager)

The last component – WSMX Management plug-in, integrated with WSMO Studio – provides the UI for managing and interacting with the WSMX environment.

The following sections provide a short introduction to the WSMX server and Adapter Framework, as well as detailed information on the role and functionality of the WSMX Management plug-in.

4.1 WSMX Server

The WSMX Server is made up a central core that manages a configurable set of components. An integration API written in the Java programming language exists that allows a 3rd party to create a component for WSMX as long as they adhere to the interfaces described in the API. Examples of the sorts of components that can be written for WSMX include:

- Communication Manager – the responsibilities of such a component include receiving messages from the user, acting on these messages and eventually invoking the web service that fulfils the users Goal.

- Discovery – the responsibility of the Discovery component is to find web services that could be used to fulfil the users Goal [10]. The discovery component may also create compositions of multiple services if an individual service that meets the users Goal is not available.

- Data Mediation – when performing discovery and at many other points in WSMX, it may be the case that there heterogeneity issues exist in the data in WSMX. Data mediation [7] can be employed to mediate between the different ontologies and allow the matching to occur.
• Selection – the result of the Discovery component is a list of Web Services that could be used to full the users Goal. The process of choosing which of these services to actually use is the responsibility of the Selection Component [11]. This process involves matching the non-functional properties of each web services against the user’s service preferences and choosing the most suitable.

• Choreography – the Choreography Component is responsible for managing the state of the overall conversation between the user and the Web Service.

• Process Mediation – During invocation it may be possible that heterogeneity problems exist at the process level. The job of the Process Mediation [4] component is to resolve these sorts of mismatches.

Once deployed in the WSMX Environment, the order in which these components are executed depends on the execution semantics of WSMX. The execution semantics is a formal specification of a given behaviour of WSMX, thus depending on how the user enters WSMX and the data they provide WSMX can be behave in a number of different ways.

4.2 Adapter Framework

The WSMX System itself can only be communicated with using the WSML language, while this means that all information passed into WSMX is semantically described it does produce an additional overhead on the requestor to send all messages in WSML. To reduce this overhead the Adapter Framework resides in front of the WSMX System and performs syntactic adaptation of data and low-level protocol adaptation.

An example of this would be where the user is using EDI messages to communicate with its partners using the EDI protocol, instead of changing all outgoing messages to WSML messages the user can communicate with the Adapter Framework in front of WSMX using EDI, the framework will then convert all incoming messages into WSML messages and communicate with WSMX using Web Service technologies.

4.3 WSMX Manager plug-in

The WSMX Manager is an Eclipse plug-in integrated with WSMO Studio, which is accessible through the WSMX Management perspective.

The perspective is implemented in a manner consistent with existing Eclipse plug-ins that users are already familiar with, thus enabling users who are familiar with WSMX to quickly pick up and use the tools.
The perspective is broken down into two sections, on left – the servers view is displayed and on the right there is space for opening editors and viewers on the servers. The majority of the functionality in the perspective exists within the left side of the GUI in the servers view.

![New WSMX Server](image1)

**Figure 13 Adding a new WSMX Server**

The servers view (see Figure 15) shows a list of WSMX Servers and Adapter frameworks that have been configured within the WSMX Management perspective. Initially the list is empty; however by right clicking on the servers view the user can add new configurations.

![New Adapter Framework](image2)

**Figure 14 Adding an Adapter Framework**

Once configured the functionalities of the different types of systems are available by expanding the systems entry in the servers view and by right clicking on the given entry.
4.3.1 Adapter Framework View

This UI component provides functionality for:

- *Viewing deployed adapters* – it is possible to view the list of adapters deployed in a given adapter framework by expanding the given adapter framework node (see Figure 16).

![Figure 16 Deployed adapters](image)

- *Deploying a new adapter* – to deploy a new adapter right click on the adapter node and choose ‘Deploy Adapter’, at this stage you will be asked for the name of the adapter and for the ‘.adapter’ file that contains the code and XML document that make up the adapter. Once the user clicks OK the name of the adapter and the adapter itself is sent to the adapter framework via a web service interface where it is deployed.

- *Un-deploying an adapter* – to un-deploy an adapter, expand the adapter framework node and select the adapter which is to be un-deployed\(^\text{10}\). Right click on the adapter and choose ‘Undeploy Adapter’. The name of the adapter to be un-deployed is sent to the adapter framework via a web service interface where it is un-deployed.

\(^{10}\) Note that the “Default” adapter cannot be un-deployed
• Testing an adapter – to test an adapter, expand the adapter framework node and select the adapter to be tested (see Figure 17). Right click on the adapter and choose ‘Test Adapter’. At this stage you will be asked for the input data that should be provided to the adapter framework to be adapted. Once the input data has been added the ‘Test Adapter’ will become activated, upon clicking this button the input data along with the adapter name will be sent to the adapter framework and the data will be adapted and sent back to the user where it will be displayed in the output test area.

![Figure 17 Testing an Adapter](image)

4.3.2 WSMX Server View

This component provides functionality for:

• Viewing deployed components – to view the list of components deployed in WSMX expand the given WSMX server in the servers view (see Figure 18).
• **Viewing the properties of a component** – all components deployed in a JMX instance [13] have a given set of properties attached to them (see Figure 19). The properties of each component in the WSMX architecture can be seen by right clicking on the given component of the WSMX server in the servers view and choosing properties.

![Figure 19 Component properties](image)

• **Invoking operations on deployed components** – to invoke an operation on a given component, deployed in the WSMX server, right click on the given component of the WSMX server in the servers view and choose ‘Operations’ followed by the desired operation to invoke. If the operation requires input a dialog will be displaying giving one text area for each of the values required by the operation. Once the operation has been invoked the output from the method will be displayed, if no output is returned by the operation then a dialog showing whether the invocation of the method was successful or not is displayed.
Accessing the WSMX Management Console – WSMX exposes a web-based management console which can be accessed through the WSMX management perspective, to do this right click on the given WSMX Server and choose ‘Open Management Console’. A web browser style editor is opened on the right hand side of the perspective showing the management console.
5 DEVELOPERS GUIDE

This section is intended for developers who will extend and customise the existing WSMO Studio functionality by providing 3rd party extensions that can be integrated with WSMO Studio.

The section provides a summary of the changes (from a developers’ point of view) since the first public release of WSMO Studio, followed by some details about the available extension points and their intended usage, and finally and example for building a plug-in extension and integrating it with WSMO Studio.

5.1 Summary of changes

The following improvements and new features were added to WSMO Studio since its first public release (July 2005)

5.1.1 Migration to Java 5

WSMO Studio is now fully compatible with Java 5. A JRE 1.5+ is required in order to run WSMO Studio.

5.1.2 Improved MVC infrastructure

A new Model-View-Controller infrastructure was introduced in WSMO Studio, so that the visual components and their underlying data models are fully decoupled, and the latter may be shared.

The standard Java mechanism was used – models extend the java.util.Observable\(^\text{11}\) class, while GUI components (e.g. views) implement the java.util.Observer\(^\text{12}\) interface.

The models introduce an additional abstraction layer above wsmo4j objects, for example a model may include several wsmo4j objects, and a specific wsmo4j object may be part of more than one model.

UI components (views) register with the respective model and receive notifications upon a model change. There is a many-to-one correspondence between UI components (views) and the models, e.g. one model may be associated with many views.

Dependencies between models can be introduced too, e.g. a hierarchy of models exists, so that when a model is changed, all dependent models (for example models, that include wsmo4j objects from this model) are notified too, so that in turn they can notify their respective observers (views).

\(^{11}\) http://java.sun.com/j2se/1.5.0/docs/api/java/util/Observable.html

\(^{12}\) http://java.sun.com/j2se/1.5.0/docs/api/java/util/Observer.html
The MVC related functionality is not fully finalised and will be further extended.

5.1.3 Load-on-demand of entities in the workspace

In certain cases the user needs to refer to WSMO entities (ontologies, mediators, services and goals) that are not described in the active WSML document. WSMO Studio offers the ability to refer to any WSMO entity that is available in a WSML document from the active workspace.

In order to avoid inefficient resource usage, such entities are not fully loaded in WSMO Studio, but instead, upon starting the Studio, a lightweight scanner inspects the active workspace and builds an in-memory repository, where only the identifier and the type of the entity are stored (but not the actual object representation).

This way the user is fully aware of all WSMO entities, described in various WSML files in the workspace, but at the same time, an object representation for an entity is built in memory only when such entity is actually used (e.g. opened by an editor or loaded by a dependent object).

5.1.4 Improved logging

WSMO Studio reuses the standard logging ability of Eclipse. All messages for errors/warnings which appear at runtime are recorded in a standard log output file located in a service folder of the workspace.

The logging functionality is managed by a utility class (LogManager), part of the studio runtime plug-in. Its functionality is thus available by any other plug-in loaded within WSMO Studio.

5.1.5 New extension points

The following new extension points were introduced:

- *org.wsmostudio.ui.navigator_content* — this extension point allows customisation of the WSMO Navigator (e.g. the visual component that provides summary view of ontologies, services, goals and mediators; see Figure 6). The hierarchy of elements can be modified and custom actions can be associated with each element type.

- *org.wsmostudio.runtime.wsmoimpl* — this extension point allows 3rd parties to plug custom wsmo4j implementations, e.g. to change the implementation of the underlying data model, provided that they preserve the compatibility with the WSMO API.

More details on extension points are available in section 5.2.

---

13 A workspace is the active working directory that contains the files WSMO Studio will work with

14 [http://wsmo4j.sourceforge.net](http://wsmo4j.sourceforge.net)
5.1.6 Updated extension points

The following extension points were further extended:

- `org.wsmostudio.ui.editors` – related to the introduction of a flexible MVC model above the `wsmo4j` data layer (as described in section 5.1.2), the relevant extension point was further refined to allow custom models to be plugged-in

- `org.wsmostudio.repository.Repository` – minor changes were introduced in this extension point, that allow custom implementations to provide icons for the new repositories and to restrict the number of active repositories.

More details on extension points are available in section 5.2.

5.2 Extension point summary

This section provides a summary of the functionality exposed by each extension point\(^\text{15}\) and the approaches to plug 3rd party extensions via the standard Eclipse extension point mechanism.

5.2.1 `org.wsmostudio.runtime.wsmoimpl`

*Identifier:* `org.wsmostudio.runtime.wsmoimpl`  
*Since:* version 0.1.6  
*Description:* This extension point allows 3rd parties to plug their custom implementations of the WSMO API in the Studio's runtime environment.

*Configuration Markup:*

```xml
<!ELEMENT extension (implementation)>  
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>
```

- `point` – a fully qualified identifier of the target extension point.
- `id` – an optional identifier of the extension instance.
- `name` – an optional name of the extension instance

\(^{15}\) Note that D4.4 contains a detailed introduction to Eclipse specific terminology (such as “extension point”) and we will not re-introduce the terms here
A configuration element describing a custom wsmo-api implementation which can be used in the WSMO Studio

- **id** – a user friendly identifier of the implementation
- **WsmoFactory** – the name of a class implementing `org.wsmo.factory.WsmoFactory` interface
- **LEFactory** – the name of a class implementing `org.wsmo.factory.LogicalExpressionFactory` interface
- **WSMLParser** – the name of a class implementing `org.wsmo.wsml.Parser` interface
- **WSMLSerializer** – the name of a class implementing `org.wsmo.wsml.Serializer` interface

**Supplied Implementation:**

Since not all of the implementation properties are required, the missing ones are substituted by the default implementation classes.

### 5.2.2 [org.wsmostudio.ui.editors](http://example.org)

**Identifier:** org.wsmostudio.ui.editors

**Since:** version 0.1.4

**Description:**

This extension point is used to add new editors to the WSMO Studio. An editor is a visual component within a workbench page which is typically used to edit or browse a wsmo object. When an object definition must be opened, the studio's registry is consulted to determine an appropriate editor for the object type and then a new instance of the editor type is created.

As WSMO Studio is strongly tied with the Eclipse Platform, the specific WSMO editors are also Eclipse editors. Further more, each new WSMO editor must extend Eclipse's extension point `org.eclipse.ui.editors`. Additionally, bindings between WSMO objects and corresponding editors are needed. Each such binding registers an editor which will be used when a certain WSMO object is opened.

**Configuration Markup:**

```xml
<!ELEMENT extension (editorBinding*)>
<!ATTLIST extension
   point CDATA #REQUIRED>
```
• `point` – the identifier of the target extension point.
• `id` – an optional identifier of the extension instance.
• `name` – an optional name of the extension instance.

```xml
<!ELEMENT editorBinding EMPTY>
<!ATTLIST editorBinding
   editorId CDATA #REQUIRED
   wsmoType CDATA #REQUIRED
   default (true|false)
   uiModel CDATA #IMPLIED>
```

• `editorId` – an identifier of an Eclipse's editor extension capable to handle a certain WSMO object.
• `wsmoType` – a class name of a certain WSMO API interface.
• `default` – if true, this editor will be used as the default editor for the type. This is only relevant in a case where more than one editor is registered for the same type.
• `uiModel` – the name of a class implementing the `org.wsmostudio.ui.editors.model.ModelFactory` interface. This factory class is intended to provide a specific UI model for the editor. This UI model is responsible for event notification and synchronisation between different editors when the underlying `wsmo4j` model is changed. If this value is not specified, the editor will receive a default model.

**Examples:**

```xml
<-- eclipse editor extensions -->
<extension point = "org.eclipse.ui.editors">
  <editor
class = "org.wsmostudio.ui.editors.AxiomEditor"
icon = "icons/axiom.gif"
default = "false"
name = "Axiom Editor"
id = "org.wsmostudio.ui.editors.axiomEditor"/>
  <editor
class = "org.wsmostudio.ui.editors.ServiceDescriptionEditor"
icon = "icons/webservice.gif"
default = "false"
name = "WebService Editor"
id = "org.wsmostudio.ui.editors.webserviceEditor"/>
</extension>

<-- binding editors with wsmo objects -->
<extension point = "org.wsmostudio.ui.editors">
  <editorBinding
default = "true"
</extension>
```
API Information:
The value of `editorId` attribute is a reference to an editor identifier (value of `id` attribute within a `org.eclipse.ui.editors` extension definition).

In the extension definition of `org.eclipse.ui.editors`, the `class` attribute specifies a class which should implement `org.eclipse.ui.IEditorPart`.

The implementing class may extend `org.wsmostudio.ui.WSMOEditor` (recommended) which supplies an implementation of `IEditorPart` with some additional facilities.

The mechanism for retrieving the WSMO object from the `IEditorInput` (supplied to the editor implementation) uses the `IAdaptable` interface (which `IEditorInput` extends).

The following snippet demonstrates the technique:

```java
/**
 * implements IEditorPart.init()
 */
public void init(IEditorSite site, IEditorInput input) throws PartInitException {
    Ontology inputOntology = (Ontology)input.getAdapter(Ontology.class);
}
```

If the implementation requests an object of incorrect type - `null` is returned.

5.2.3 `org.wsmostudio.repository.Repository`

**Identifier:** `org.wsmostudio.repository.Repository`

**Since:** version 0.1.0

**Description:**
An extension point that allows 3rd parties to supply WsmoRepository implementations as WSMO Studio plug-ins.

---

16 The `org.wsmo.datastore.WsmoRepository` interface ([http://wsmo4j.sourceforge.net/multiproject/wsmo-api/apidocs/org/wsmo/datastore/WsmoRepository.html](http://wsmo4j.sourceforge.net/multiproject/wsmo-api/apidocs/org/wsmo/datastore/WsmoRepository.html)) is part of the `wsmo4j` library. Its purpose is to provide an abstraction over ontology/goal/service repositories, so that they can be treated in an unified way from within WSMO Studio. The extension mechanism allows that 3rd parties provide such custom adapters to specific repositories so that WSMO Studio will be able to interact with such repositories. Technically, the adapter is an implementation of the `org.wsmo.datastore.WsmoRepository` interface. At present there are two such adapters – the local ORDI repository (based on Sesame) and an adapter that
Each repository class must implement two interfaces:

- `org.wsmo.datastore.WsmoRepository` (part of `wsmo4j`) and
- `org.wsmostudio.runtime.extension.Initialisable` (part of WSMO Studio Runtime plug-in).

The `WsmoRepository` interface provides functionality related to repositories' content: addition, retrieval, removal of WSMO elements (ontologies, goals, mediators and services).

`Initialisable` provides means for configuring a concrete repository instance. The `Initialisable#initialise(Map)` method is used to provide each repository instance with a map with specific configuring property/value pairs.

Each repository extension must have a dedicated configuration class, instances of which supply configuration settings (such as endpoint address, user name, password, etc. which are specific to each type of repository). The configuration class must implement the `org.wsmostudio.runtime.extension.Configurator` interface. The configurator is instantiated the first time a specific repository instance is activated.

A successfully configured repository extension appears as a type entry in the New Repository dialog (invoked by Repositories Explorer view). Each repository extension is identified by a short description label (supplied in its definition).

Configuration Markup:

```xml
<!ELEMENT extension (repository)+>  
<!ATTLIST extension  
  point CDATA #REQUIRED 
  id    CDATA #IMPLIED 
  name  CDATA #IMPLIED>
```

- **point** – a fully qualified identifier of the target extension point.
- **id** – an optional identifier of the extension instance.
- **name** – an optional name of the extension instance.

```xml
<!ELEMENT repository EMPTY>  
<!ATTLIST repository  
  class          CDATA #REQUIRED 
  rtype          CDATA #REQUIRED 
  configurator   CDATA #REQUIRED 
  allowsMultiple (true|false) "true" 
  icon           CDATA #IMPLIED>
```

- **class** – the name of a class implementing the `WsmoRepository` and `Initialisable` interfaces
- **rtype** – A repository type identifier

allows communicating with remote implementations of WsmoRepository (the communication is Web Service based)
• **configurator** – A class implementing the *Configurator* interface

• **allowsMultiple** – An indicator if this repository extension can have more than one runtime instance. The default is `true`.

• **icon** – a custom icon to appear in the Repositories Explorer for this type

**Examples:**

```xml
<extension point = "org.wsmostudio.repository.Repository">
  <repository
    class = "com.ontotext.wsmo4j.io.repository.webservice.RepositoryClient"
    configurator = "com.ontotext.wsmo4j.io.repository.webservice.RepositoryConfigurator"
    rtype = "Default WSMO Studio Repository"/>
</extension>
```

**API Information:**

The value of the `class` attribute must implement both *WsmoRepository* and *Initialisable* interfaces.

The `rtype` attribute must contain a one-line user-friendly repository extension type identifier.

The `configurator` attribute points to a *Configurator* implementing class instances of which set up the repository instances.

Note: Implementors are responsible for cross-session configuration properties persistence (if such are needed).

### 5.2.4 org.wsmostudio.ui.navigator_content

**Identifier:** org.wsmostudio.ui.navigator_content

**Since:** version 0.2.2

**Description:** This extension point enables 3rd parties to contribute/introduce new entities to the WSMO Navigator and to hide/filter existing ones.

**Configuration Markup:**

```xml
<!ELEMENT extension (entry | filter | action)>
<!ATTLIST extension
  point CDATA #REQUIRED
  id CDATA #IMPLIED
  name CDATA #IMPLIED>

<!ELEMENT entry EMPTY>
<!ATTLIST entry
  icon CDATA #REQUIRED
  class CDATA #REQUIRED
  manager CDATA #IMPLIED>
```
• **icon** – an image icon to appear in the navigator view for the target entity
• **class** – the class/interface name of the target entity
• **manager** – the name of a manager class (implementation of org.wsmostudio.views.navigator.TreeProvider) responsible for delivering target entity's content (if any). If this property is not specified, no content will be visualized and the label will be determined by calling `toString()`.

```xml
<!ELEMENT filter EMPTY>
<!ATTLIST filter
    name    CDATA #REQUIRED
    class   CDATA #REQUIRED
    manager CDATA #IMPLIED>
```

• **name** – A user friendly identifier of the filter (used in Window/Preferences section)
• **class** – a class name of an entity to be filtered out.
• **manager** – An optional field, defining a custom filter. The value is a class name of implementation of org.wsmostudio.views.navigator.Filter.

```xml
<!ELEMENT action EMPTY>
<!ATTLIST action
    class    CDATA #REQUIRED
    name     CDATA #REQUIRED
    handler  CDATA #REQUIRED
    category (add|remove|misc) "misc">
```

• **class** – A class name for object to which this action is applicable.
• **name** – The name (label) of this action. It appears in the context menu of the WSMO Navigator.
• **handler** – The action delegate which is executed if this action is selected. The value is a name of a class name implementing these two interfaces: org.eclipse.jface.action.IAction and org.wsmostudio.runtime.extension.Initialisable
• **category** – The actions are classified in three categories: add, remove and misc which determines their order of appearance on the context menu.

**Examples:**

This example shows how to enable parameters visualization in the navigator:

```xml
<extension point="org.wsmostudio.ui.navigator_entry">
    <entry
        icon="icons/param.gif"
        class="org.omwg.ontology.Parameter"/>
    <entry
        icon="icons/rel.gif"/>
</extension>
```
class="org.omwg.ontology.Relation"
manager="custom.relation.ContentProviderWithParams"/>

<filter
name="Axioms Filter"
class="org.omwg.ontology.Axiom"/>

<filter
name="Custom relation filter"
class="org.omwg.ontology.Relation"
manager="custom.binary.FilterManager"/>
</extension>

5.3 Examples

The following example demonstrates how the content of certain entities, represented in the WSMO Navigator can be extended by external plug-ins. The sample code in this example is based on part of the Choreography plug-in of the studio. For simplicity, the insignificant details of the real code will be omitted.

The task of this example is to customize the representation of WebServices/Goals in the WSMO Navigator, in particular to reveal the choreography content structure.

The initial configuration presents the Choreography entity as a terminal node in the tree structure. This is visible on Figure 22:

![Figure 22 Initial choreography visualisation](image)

In the next steps we will create an extension which provides extended visualization of the Choreography entity, i.e. the state signature and its content. We'll provide separate image icon for each newly introduced entity.

In order to define the content of a certain entity, we have to implement a content provider (implementation of `org.wsmostudio.ui.views.navigator.TreeProvider`) for each entity type.
In this case we need two content providers: one to provide choreography's content (its state signature and capability) and one for the StateSignature content (e.g. the modes / roles).

Here is the source code of Choreography's content provider:

```java
public class ChoreographyTreeProvider implements TreeProvider {

    public Object[] getChildren(Object parent) {
        Choreography chore = (Choreography) parent;
        return new Object[] { chore.getStateSignature() }; // 1
    }

    public boolean hasChildren(Object parent) { // 2
        return true;
    }

    public String labelFor(Object entity) { // 3
        return ((Entity) entity).getIdentifier().toString();
    }
}
```

Listing 1 Choreography content provider

Details about the code:

- 1 when requested for the content of choreography object, the provider returns its StateSignature object as single content entry. Additional entries can be included too (for example, transition rules).

- 2 as choreographies always have state signatures, there will always be visible content.

- 3 here the label representation of the choreography object in the Navigator view is determined. Here the unique identifier is used as a label.

Note: the TreeProvider interface signature is more general (arguments are Objects) which allows more flexible modelling even of non-wsmo Entity objects.

The next step is to implement a State Signature content provider:
Now, as we have all required modules implemented, it is time to register them as extensions in the plug-in descriptor file (plugin.xml). The relevant XML fragment is shown below:

```
public static class StateSigTreeProvider implements TreeProvider {

    public Object[] getChildren(Object parent) {
        StateSignature sSig = (StateSignature)parent;
        List<Node> bufferModes = new LinkedList<Node>();

        bufferModes.addAll(sSig.listInModes());
        bufferModes.addAll(sSig.listOutModes());
        bufferModes.addAll(sSig.listSharedModes());
        bufferModes.addAll(sSig.listStaticModes());
        bufferModes.addAll(sSig.listControlledModes());

        return bufferModes.toArray();
    }

    public boolean hasChildren(Object parent) {
        return true;
    }

    public String labelFor(Object entity) {
        return ((Entity)entity).getIdentifier().toString();
    }
}
```

Listing 2 State Signature content provider

Now, as we have all required modules implemented, it is time to register them as extensions in the plug-in descriptor file (plugin.xml). The relevant XML fragment is shown below:

```
<extension
    point="org.wsmostudio.ui.navigator_content">
<entry
    class="org.wsmostudio.service.choreography.Choreography"
    icon="icon/chor.gif"
    manager="org.wsmostudio.service.choreography.navigator.ChoreographyTreeProvider"/>
<entry
    class="org.wsmostudio.service.choreography.signature.StateSignature"
    icon="icon/sig.gif"
    manager="org.wsmostudio.service.choreography.navigator.ChoreographyTreeProvider&StateSigTreeProvider"/>
<entry
    class="org.wsmostudio.service.choreography.signature.Node"
    icon="icon/conceptdef.gif"/>
</extension>
```

Listing 3 Plug-in descriptor for the extension

- Declare the extension of the extension point `org.wsmostudio.ui.navigator_content`
- Declare the new entry object `Choreography`. This declaration is needed to override the default behaviour.
- Specify the dedicated content provider for this object, i.e. `ChoreographyTreeProvider`.
• declare the new entry object `StateSiganChoreography`

• declare a new entry `Mode`. Here the class name refers to a more general type (instead of concrete type specification), which enables equal treatment for all subtypes of modes (In, Out, Shared, etc.). Also no content provider is supplied which indicates that no sub-content is expected to appear.

As a result of successfully installed extension the following structure appears is the WSMO Navigator view (Figure 23):

![Extended WSMO Navigator](image)

Figure 23 Extended WSMO Navigator
6 CONCLUSION

This deliverable presented the second prototype of an Integrated Service Environment for WSMO, called *WSMO Studio*. The deliverable presented information for both end users (sections on Choreography Editor and WSMX Manager) and 3rd party developers intending to provide additional functionality (Developers Guide).
REFERENCES


APPENDIX A – 3RD PARTY COMPONENTS AND LICENCES

Table 1 shows the 3rd party modules or libraries part of the WSMO Studio distribution, together with their respective licences.

Table 1 3rd party components integrated with WSMO Studio

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
<th>Licence</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>3.1.1</td>
<td>Eclipse Public Licence</td>
<td><a href="http://www.eclipse.org/">URL</a></td>
</tr>
<tr>
<td>Sesame</td>
<td>1.2.3</td>
<td>LGPL</td>
<td><a href="http://www.openrdf.org">URL</a></td>
</tr>
<tr>
<td>Axis</td>
<td>1.3</td>
<td>Apache Software Licence</td>
<td><a href="http://ws.apache.org/axis/">URL</a></td>
</tr>
<tr>
<td>wsmo4j</td>
<td>0.5.1</td>
<td>LGPL</td>
<td><a href="http://wsmo4j.sourceforge.net">URL</a></td>
</tr>
</tbody>
</table>

17 [URL](http://www.eclipse.org/org/documents/epl-v10.html)
18 [URL](http://www.opensource.org/licenses/lgpl-license.php)
19 [URL](http://ws.apache.org/LICENSE.txt)
APPENDIX B – TECHNICAL INFORMATION

Additional information about WSMO Studio is available at the web site: http://www.wsmostudio.org

The major points of interest include:

- List of changes: http://www.wsmostudio.org/changes-report.html
- Bugs (open and fixed):
- JavaDoc:
- Additional documentation: http://www.wsmostudio.org/reference.html