DIP
Data, Information and Process Integration with Semantic Web Services
FP6 – 507483

Deliverable

WP6: Interoperability and Architecture
D6.14
WSMO API v2

Marin Dimitrov
Vassil Momtchev
Damyan Ognyanov
Alex Simov
Mihail Konstantinov
Michael Altenhofen

July 3, 2006
SUMMARY

This deliverable presents the second version of a Java API (WSMO API) and reference implementation (wsmo4j) for building Semantic Web Service applications based on the Web Service Modelling Ontology (WSMO).

The results presented in this deliverable directly contribute to two of the main DIP goals, namely:

- **open source semantic Web services architecture**, by providing the WSMO API, standardising the interface definitions of low-level infrastructure components (e.g. parsers, serializers, validators, datastores, factories, etc.)
- **exploitable tools**, by providing an implementation of the WSMO API in the form of a common infrastructure layer, and thus application level components (such as SWS editors or SWS runtime environments) do not need to spend time and effort on these tasks, but instead can focus on their specific added value.

This deliverable is directly relevant to the following DIP work packages:

- **WP2**, where the ontology tools (DOME) use the ontology related functionality of WSMO API
- **WP4**, where the tools (WSMO Studio) use the API for creating and parsing the descriptions of WSMO ontologies, services and mediators as well as service choreographies.
- **WP6**, where the runtime environment (WSMX) uses the API for parsing WSMO descriptions and service choreographies.

The target audience of this deliverable is comprised of software architects and software developers providing end-user applications (such as editors) or infrastructure components (such as execution environments).

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.
# Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP6 – 507483</td>
<td>DIP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Full Title</th>
<th>Project URL</th>
<th>Document URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td><a href="http://dip.semanticweb.org/">http://dip.semanticweb.org/</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EU Project Officer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kai Tullius</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.14</td>
<td>WSMO API v2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work Package Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Interoperability and Architecture</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of Delivery</th>
<th>Status</th>
<th>Nature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contractual</td>
<td>M30</td>
<td>version 1.0</td>
</tr>
<tr>
<td>Actual</td>
<td>30-Jun-06</td>
<td>final</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marin Dimitrov (Sirma), Vassil Momtchev (Sirma), Damyan Ognyanov (Sirma), Alex Simov (Sirma), Mihail Konstantinov (Sirma), Michael Altenhofen (SAP)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resp. Author</th>
<th>Partner</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marin Dimitrov</td>
<td>Sirma</td>
<td>+359 (2) 976-8310 x 161</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract (for dissemination)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the abstract for dissemination here.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP, WSMO, API, Java, wsmo4j</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Date</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>08-06-2006</td>
</tr>
<tr>
<td>20-06-2006</td>
</tr>
<tr>
<td>29-06-2006</td>
</tr>
<tr>
<td>30-06-2006</td>
</tr>
<tr>
<td>03-07-2006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reviewers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacek Kopecký</td>
</tr>
<tr>
<td>Partner</td>
</tr>
<tr>
<td>Carlos Pedrinaci</td>
</tr>
<tr>
<td>Partner</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 Galway | NUIG | Dr. Sigurd Harand  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: sigurd.harand@deri.org  
Tel: +353 91 495112 |
| 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 |
| 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  
E-mail: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
E-mail: 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  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
| 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  
E-mail: csma@ilog.fr  
Tel: +33 1 49082981 |
|---|---|---|
| inubit AG | inubit | Torsten Schmale,  
inubit AG,  
Lützowstraße 105–106  
D-10785 Berlin,  
Germany  
E-mail: 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  
E-mail: rbenjamins@isoco.com  
Tel. +34 913 349 797 |
| Hanival Internet Services GmbH | Hanival | Alexander Wahler  
Hanival Internet Services GmbH  
Kirchengasse 13/1a  
A-1070 Wien  
E-mail: wahler@niwa.at  
Tel.: +43 131 95843 11 |
| MDR Partners | MDR Partners | Rob Davies  
MDR Partners  
8 St. Andrew Street & Hertford, Herts,  
United Kingdom, SG14 1JA,  
E-mail: rob.davies@mdrpartners.com  
Tel.: +44 (0)208 8763121 |
| The Open University | OU | Dr. John Domingue  
Knowledge Media Institute,  
The Open University, Walton Hall,  
Milton Keynes, MK7 6AA, UK  
E-mail: 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  
E-mail: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma AI Ltd. | Sirma | Atanas Kiryakov,  
Ontotext Lab, - Sirma AI EAD,  
Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse,  
Sofia 1784, Bulgaria  
E-mail: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
| Unicorn Solution Ltd. | Unicorn | Jeff Eisenberg  
Unicorn Solutions Ltd,  
Malcha Technology Park 1  
Jerusalem 96951,  
Israel  
E-mail: Jeff.Eisenberg@unicorn.com  
Tel.: +972 2 6491111 |
|----------------------|---------|-----------------|
| Vrije Universiteit Brussel | VUB | Pieter De Leenheer,  
Starlab- VUB  
Vrije Universiteit Brussel  
Pleinlaan 2, G-10  
1050 Brussel, Belgium  
E-mail: Pieter.De.Leenheer@vub.ac.be  
Tel.: +32 (0) 2 629 3749 |
LIST OF KEYWORDS/ABBREVIATIONS

API – Application Programming Interface
ASM – Abstract State Machine
IRI – Internationalised Resource Identifier
NFP – Non-Functional Property
OWL – Web Ontology Language
RDF – Resource Description Framework
UML – Unified Modeling Language
XML – eXtensible Markup Language
WSDL – Web Services Description Language
WSML – Web Service Modeling Language
WSMO – Web Service Modeling Ontology
CONTENTS

1 INTRODUCTION 4

2 WSMO API 6
   2.1 Introduction 6
   2.2 Common interfaces 6
      2.2.1 Identifier 6
      2.2.2 Entity and TopEntity 6
      2.2.3 Namespaces, WSML variants and NFPs 8
   2.3 Factories 9
   2.4 Ontologies 12
      2.4.1 Ontology 12
      2.4.2 Types and Values 12
      2.4.3 Axioms 14
      2.4.4 Concepts, Instances and Attributes 15
      2.4.5 Relations, Relation Instances and Parameters 17
   2.5 Logical Expressions 18
   2.6 Services & Goals 25
   2.7 Mediators 27
   2.8 Parsers and Serializers 28
   2.9 Datastores and Repositories 29
   2.10 Validators 29

3 GROUNDING API 33
   3.1 Introduction 33
   3.2 Grounding Factory 33
   3.3 Grounding 33
   3.4 Model References 34
3.5 Assertions .............................................. 35
3.6 Category .................................................. 35
3.7 Parsers and Serializers ............................... 35

4 CHOREOGRAPHY API ................................. 37
  4.1 Introduction ........................................... 37
  4.2 Choreography ......................................... 37
  4.3 State signature ....................................... 37
  4.4 Transition rules ...................................... 40
  4.5 Choreography factory .............................. 40

5 WSMO4J .................................................. 42
  5.1 Introduction .......................................... 42
  5.2 List of changes since M12 ......................... 42

6 CONCLUSION ........................................... 44

BIBLIOGRAPHY ............................................ 44

7 APPENDIX A – EXAMPLES ............................ 47
LIST OF FIGURES

2.1 Identifier hierarchy ........................................... 7
2.2 Entity .......................................................... 7
2.3 Top Level hierarchy ............................................. 8
2.4 Namespace interface ........................................... 9
2.5 Factory class ................................................... 9
2.6 WsmoFactory interface ....................................... 10
2.7 DataFactory interface ........................................... 11
2.8 Ontology interface ............................................. 13
2.9 Type hierarchy ................................................ 14
2.10 ComplexType interface ....................................... 14
2.11 Data values ................................................... 15
2.12 Axiom interface ............................................... 16
2.13 Concept interface ............................................. 16
2.14 Attribute interface ........................................... 17
2.15 Instance interface ............................................. 18
2.16 Relation interface ............................................ 19
2.17 Parameter interface ......................................... 19
2.18 RelationInstance interface ................................. 20
2.19 Logical expression Terms ................................... 21
2.20 Atomic logical expressions ................................. 22
2.21 Compound logical expressions .............................. 23
2.22 LogicalExpressionFactory interface ...................... 24
2.23 WebService and Goal interfaces ......................... 25
2.24 Capability interface ......................................... 26
2.25 Interface, Orchestration and Choreography interfaces 27
2.26 Mediator interfaces ......................................... 28
2.27 Parser interface .................................................. 29
2.28 Serializer interface .................................................. 29
2.29 DataStore and Repository interfaces ................................. 31
2.30 Validator interface .................................................... 32
2.31 Validation warnings and errors .......................................... 32

3.1 GroundingFactory interface ........................................... 33
3.2 Grounding interface ..................................................... 34
3.3 ModelReference interface .............................................. 34
3.4 Precondition and Effect interfaces ................................. 35
3.5 Category interfaces .................................................... 36
3.6 Parser and Serializer interfaces for Grounding ..................... 36

4.1 Choreography interface ............................................... 38
4.2 StateSignature interface .............................................. 38
4.3 Choreography modes .................................................. 39
4.4 Choreography rules .................................................... 41
LIST OF TABLES
1 INTRODUCTION

This deliverable presents the second version of a Java API (WSMO API) and reference implementation (wsmo4j) for building Semantic Web Service applications based on the Web Service Modelling Ontology (WSMO).

The first version of the API has already been presented in [5] but since the API was substantially refactored since the first release, no prior knowledge of [5] is necessary for reading this document.

The results presented in this deliverable directly contribute to two of the main DIP goals, namely:

- open source semantic Web services architecture, by providing the WSMO API, standardising the interface definitions of low-level infrastructure components (e.g. parsers, serializers, validators, datastores, factories, etc.)

- exploitable tools, by providing an implementation of the WSMO API in the form of a common infrastructure layer, and thus application level components (such as SWS editors or SWS runtime environments) do not need to spend time and effort on these tasks, but instead can focus on their specific added value.

The WSMO API / wsmo4j are available under an open source licence, specifically LGPL\(^1\).

This deliverable is directly relevant to the following DIP work packages:

- WP2, where the ontology tools (DOME\(^2\)) use the ontology related functionality of WSMO API
- WP4, where the tools (WSMO Studio\(^3\)) use the API for creating and parsing the descriptions of WSMO ontologies, services and mediators as well as service choreographies.
- WP6, where the runtime environment (WSMX\(^4\)) uses the API for parsing WSMO descriptions and service choreographies.

The target audience of this deliverable is comprised of software architects and software developers providing end-user applications (such as editors) or infrastructure components (such as execution environments).

The document is structured as follows:

- chapter 2 introduces the refactored and extended version of the WSMO API.

\(^1\)http://www.opensource.org/licenses/lgpl-license.php
\(^2\)http://dome.sourceforge.net/
\(^3\)http://www.wsmostudio.org/
\(^4\)http://www.wsmx.org/
• **chapter 4** introduces an extension of the WSMO API that provides functionality for describing WSMO centric service choreographies [11].

• **chapter 3** describes an extension of the WSMO API that provides functionality for describing grounding of WSMO descriptions according to [1] and [7].

• Finally, **chapter 5** provides a summary on the most important changes in the reference implementation (**wsmo4j**), i.e. changes that are not necessarily reflected by the APIs introduced in the previous chapters.
2 WSMO API

2.1 Introduction

This chapter introduces the core WSMO API interfaces. The chapter is organised as follows:

- section 2.4, section 2.6 and section 2.7 introduces the interfaces that correspond to various entities defined in the WSMO specification [10]
- section 2.5 introduces the Logical Expressions API\(^1\) which is not part of the core WSMO specification but is elaborated in the WSML specification [4]
- section 2.3, section 2.8, section 2.9 and section 2.10 introduce various ”helper” interfaces that are not part of the WSMO domain specification, but instead provide infrastructure functionality (e.g. parsers, validators, factories, etc.).

2.2 Common interfaces

Core interfaces and classes such as *Entity, Identifier, Namespace* and *NFP* are part of the *org.wsmo.common* package.

2.2.1 Identifier

*Identifier*\(^2\) (see Figure 2.1) is the base interface for identifiers and all WSMO entities have an identifier. There are two defined sub-interfaces of Identifier:

- *UnnumberedAnonymousID*, representing anonymous unnumbered identifiers, as defined in [4]

Identifiers are created by the *WsmoFactory* (see section 2.3).

2.2.2 Entity and TopEntity

The base interface for all *WEntity* entities is *Entity*. All WSMO objects that can be identified are entities:

\(^1\)The work on Logical Expressions is *not* part of DIP D6.14 funded effort and is briefly presented here only for the purpose of completeness, since parts of the WSMO API and the Choreography API refer to it.

\(^2\)More information on identifiers is available in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#ids](http://www.wsmo.org/TR/d2/v1.2/#ids)
• Attributes, axioms, concepts, instances, relations (see section 2.4)
• Capabilities, goals, services and interfaces (see section 2.6)
• Mediators (oo-mediator, ww-mediator, wg-mediator, gg-mediator) (see section 2.7)

The Entity interface is quite generic and provides methods for accessing / modifying the identifier and the non-functional properties associated with the entity (see Figure 2.2). It is important to note that the identifier of an Entity is an immutable property, i.e. once specified, the Entity identifier cannot be changed\(^3\), in order to reduce the possibility that referential integrity is broken. Other Semantic Web frameworks, such as Jena\(^4\), employ a similar approach toward identifier immutability.

The TopEntity interface (which extends Entity) represents the four building blocks in WSMO: services, goals, ontologies and mediators (see Figure 2.3). It provides common functionality for these four entity types:

• accessing and modifying the list of imported ontologies
• accessing and modifying the list of mediators referenced by the entity

\(^3\)If such a change is required, then a new Entity, with the desired identifier, should be created and the old entity should be removed

\(^4\)http://jena.sourceforge.net/
• accessing and modifying the list of namespaces defined by the entity
• specifying the WSML variant of the entity description (see subsection 2.2.3)

![Figure 2.3: Top Level hierarchy](image)

### 2.2.3 Namespaces, WSML variants and NFPs

The `org.wsmo.common` package contains three additional classes:

- **Namespace** – represents a namespace binding (i.e. a (prefix, IRI) pair, see Figure 2.4)
- **NFP** – a placeholder for the non-functional property keys as defined by the Dublin Core set\(^5\) [15].

\(^5\)Note that NFP keys in WSMO / WSML are not restricted to the Dublin Core set and may be extended in an arbitrary way by the user. Check out [14] for an overview of the application of non-functional properties in Web Services
• WSML – which contains the identifiers for the five WSML variants (WSML_CORE, WSML_DL, WSML_FLIGHT, WSML_FULL and WSML_RULE). Each TopEntity has a WSML variant specified.

![Diagram of Namespace interface](image)

Figure 2.4: Namespace interface

2.3 Factories

The WSMO API makes heavy use of the Factory design pattern (see [8] for details).

There are five factories defined at present:

- **Factory** – this is a **meta-factory** responsible for creating other factories and service objects such as *Parsers*, *Serializers* (explained in section 2.8), *Datastores*, *Locators* (explained in section 2.9) and *Validators* (explained in section 2.10). See Figure 2.5 for the UML representation of Factory.

![Diagram of Factory class](image)

Figure 2.5: Factory class

Note that the Factory class is a **Singleton**, i.e. there is only one existing instance of the class at any time.

- **WsmoFactory** – this factory is responsible for:
  - creating identifiers and namespaces (see subsection 2.2.1 and subsection 2.2.3)

---

6 Note that new factories may be added in the future as the API is extended with new functionality.
7 More details on the Singleton pattern are available in [8]
– creating WSMO entities (see subsection 2.2.2)

See Figure 2.6 for details.

**Note:** Note that the WsmoFactory provides both create* and get* methods for each WSMO element. The purpose of the former is to create new element descriptions, while the latter return references to existing elements. Note that get* will still create a new element if no existing element was found. In the future the create* methods will most probably be removed from the API.

![Figure 2.6: WsmoFactory interface](image)

- **DataFactory** – this factory is responsible for creating instances of the built-in WSML datatypes. There are 19 built-in datatypes in WSML (see [4] for details).  

8Also available at [http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-built-in-datatypes](http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-built-in-datatypes)
The datatypes in the WSMO API are represented by the `SimpleDataType` and `ComplexDataType`, which will be presented in subsection 2.4.2.

Figure 2.7 presents the `DataFactory` interface.

![DataFactory interface](image)

- **LogicalExpressionFactory** – this factory creates the various logical expression instances (`Atoms`, `Molecules`, `Rules`, etc., which are presented in section 2.5).

- **ChoreographyFactory** – this factory creates elements related to the choreography interface of a service (check out [11] for a detailed overview of WSMO choreographies). The choreography elements such as `Modes`, `Rules` and `Containers` will be presented in chapter 4.
The general purpose factories are part of the `org.wsmo.factory` package, though some specific factories (such as the `ChoreographyFactory`) are part of the respective extension packages.

2.4 Ontologies

The `org.omwg.ontology` package is the core package in the WSMO API, that contains interfaces related to ontology modelling.

2.4.1 Ontology

This is the central interface in the package (see Figure 2.8). An `Ontology` contains a set of related `Concepts`, `Relations`, `Instances` and `Axioms`.

Since an `Ontology` extends `TopEntity` (subsection 2.2.2), it can also define `Namespaces` and import `Mediators` or other `Ontologies`.

2.4.2 Types and Values

In the WSMO API there is a distinction between data types and concepts. `Type` is the root interface of the type hierarchy, with `Concept` and `WsmlDataType` being the only possible specialisations (see Figure 2.9).

`WsmlDataType` is the top level interface representing the built-in WSML types, as defined by [4], which are equivalent to the data types defined in the XML Schema specification [2]: String, Decimal, Integer, Float, Double, IRI, QName, Boolean, Duration, DateTime, Time, Date, GYearMonth, GYear, GMonthDay, GDay, GMonth, HexBinary, Base64Binary.

The `SimpleDataType` interface represents types such as Boolean, String, Integer, Float, etc., while `ComplexDataType` (see Figure 2.10) represents composite types such as `Date(year, month, day)`, `Time(hour, minute, second)`, etc.

Data types are created by the `DataFactory`, which was already presented in section 2.3.

The `Value` interface provides a common abstraction over instances of concepts and values of the built-in WSML types. Figure 2.11 presents the data value hierarchy. `SimpleDataValues` and `ComplexDataValues` correspond to `SimpleDataTypes` and `ComplexDataTypes` respectively.

`DataValues` are created by the `DataFactory` (section 2.3)

---

9See also the definition of Ontology in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#ontologies](http://www.wsmo.org/TR/d2/v1.2/#ontologies)
Figure 2.8: *Ontology* interface
2.4.3 Axioms

The *Axiom*\(^{10}\) interface represents a WSML logical expression together with its non-functional properties (Figure 2.12)

Complex logical expressions can be created with the help of the interfaces and classes in the `org.omwg.logicalexpression` package (described in section 2.5)

\(^{10}\)See also the definition of Axiom in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#axioms](http://www.wsmo.org/TR/d2/v1.2/#axioms)
2.4.4 Concepts, Instances and Attributes

The Concept interface (Figure 2.13) represents a concept\(^{11}\) in a WSMO ontology. A concept may define attributes and may relate to several other super-concepts by an IS-A relation. Since a Concept is also an Entity, it has an Identifier and may define several non-functional properties.

Concepts are created by the WsmoFactory (section 2.3). The Concept interface provides methods for navigating the concept hierarchy (i.e. listing super-concepts and sub-concepts) as well as listing the set of instances of the concept. Concept also serves as a factory for Attributes, i.e. attributes are created from the defining concept.

A Concept may define one or more Attributes (Figure 2.14) representing named slots for data values (for the concept instances). Attributes\(^{12}\) may be reflexive (e.g. partOf), transitive (e.g. hasAncestor) or symmetric (e.g. marriedTo). Attributes may be associated with an inverse attribute (e.g. hasParent is the inverse of hasChild), and also specify cardinality constraints.

Note that Attributes are local, and thus are not created by the WsmoFactory. Instead, they are created by the defining Concept (see the createAttribute(Identifier) method).

\(^{11}\)See also the definition of Concept in the WSMO specification at http://www.wsmo.org/TR/d2/v1.2/#concepts

\(^{12}\)See also the definition of Attribute in the WSMO specification at http://www.wsmo.org/TR/d2/v1.2/#concepts
Figure 2.12: Axiom interface

Figure 2.13: Concept interface
The Instance\(^{13}\) interface (Figure 2.15) represents an instance of a concept defined in an ontology. An instance may be associated with more than one concept (or with no concept at all). Instances may specify values for the attributes defined by the respective concepts.

**Instances** are created by the WsmoFactory (section 2.3).

### 2.4.5 Relations, Relation Instances and Parameters

The Relation\(^{14}\) interface (Figure 2.16) represents a relation definition in a WSMO ontology. Relations are used to model interdependencies between several concepts. A relation may be a specialisation of one or more super-relations.

A Relation may define zero or more Parameters\(^{15}\) (Figure 2.17).

**Relations** are created by the WsmoFactory (section 2.3).

\(^{13}\)See also the definition of Instance in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#instances](http://www.wsmo.org/TR/d2/v1.2/#instances)

\(^{14}\)See also the definition of Relation in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#relations](http://www.wsmo.org/TR/d2/v1.2/#relations)

\(^{15}\)See also the definition of Parameter in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#relations](http://www.wsmo.org/TR/d2/v1.2/#relations)
Note that Parameters are local, and thus are not created by the WsmoFactory. Instead, they are created by the defining Relation (see the createParameter(byte) method).

The RelationInstance\(^\text{16}\) interface (Figure 2.18) represents instances of relations. A relation instance is associated with a single relation and may specify values for the parameters defined by the respective relation.

RelationInstances are created by the WsmoFactory (section 2.3).

### 2.5 Logical Expressions

**Note:** The work on Logical Expressions is not part of DIP D6.14 and is briefly presented here only for the purpose of completeness, since parts of the WSMO API and the Choreography API refer to it.

The packages org.omwg.logicalexpression and org.omwg.logicalexpression.term are used as object-oriented constructs of the WSML logical expressions. They are

\(^{16}\)See also the definition of RelationInstance in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#instances](http://www.wsmo.org/TR/d2/v1.2/#instances)
Figure 2.16: Relation interface

Figure 2.17: Parameter interface
Figure 2.18: RelationInstance interface

used within the Axiom, Capability, part of WSMO API, or TransitionRule, part of the Choreography API (chapter 4) to refine the WSMO elements using a logic language.

Each of the WSML syntaxes imposes different limitation over the following connectivity types: and, or, implies, impliedBy, equivalent, neg, naf, forall, exists, (, ), [], ,, =, !=, :=, memberOf, hasValue, subConceptOf, ofType and impliesType, as well as the symbols for Logical Programming Rules and database-style constraints: :-, !- (see [4] for details).

The basic construct of logical expressions are the Terms (Figure 2.19), which could be:

- **Constructed term** (function symbol) i.e. john\[age(2005) hasValue 25]\]
- Logical expression **Identifier** is less restrictive then the **Identifiers** (introduced in subsection 2.2.1) may be also **NumberedAnonymousID**.
- **DataValue** john\[age(2005) hasValue 12]\] (see also subsection 2.4.2)
- Free or shared **Variable** john\[age(2005) hasValue ?currentage]\]

The **LogicalExpression** interface is the super-interface to all logical expression connectivity types. It introduces methods to apply the **Visitor** design pattern (see [8]) and to serialize its content to a **String**. The implementations of toString(TopEntity) has to guarantee the correct usage of the namespace context when serializing.
Atom is a predicate symbol with number of parameters (terms) as arguments. Molecule is a special type of Atom, used to describe information from the conceptual model of the ontology. There are several types of Molecules:

- SubConceptMolecule $\pi(X_1 \ subConceptOf X_2)$
- MembershipMolecule $\pi(X_1 \ memberOf X_2)$
- AttributeConstrainMolecule $\pi(X_1 [X_2 \ ofType X_3])$
- AttributeValueMolecule $\pi(X_1 [X_2 \ hasValue X_3])$
- AttributeInferenceMolecule $\pi(X_1 [X_2 \ impliesType X_3])$

Figure 2.20 presents the hierarchy of atomic expressions.

The CompoundExpression interface (Figure 2.21) is a used to connect multiple Atom and/or Molecule expressions.

- CompoundMolecule – aggregates several Molecules (i.e. \textit{john[age hasValue 25] memberOf Man})
- Unary – defines unary logical expression operators
- Binary – defines binary logical expression operators

Logical expressions are created by the \textit{LogicalExpressionFactory} (Figure 2.22).
Figure 2.20: Atomic logical expressions
Figure 2.21: Compound logical expressions
Figure 2.22: `LogicalExpressionFactory` interface
2.6 Services & Goals

The org.wsmo.service package contains the interfaces related to service description in a WSMO centric way - goals, capabilities, services and service interfaces.

The ServiceDescription interface (Figure 2.23) is the common super-interface of WebService\(^{17}\) and Goal\(^{18}\). A Web Service or a Goal in WSMO may be associated with a single Capability and zero or more Interfaces.

![Figure 2.23: WebService and Goal interfaces](image)

The Capability\(^{19}\) interface (Figure 2.24) represents a WSMO capability definition. Capabilities in WSMO are used to formally define the functionality provided by a Web Service by means of the pre-conditions, assumptions, post-conditions and effects of the service (expressed as Axioms).

The Interface\(^{20}\) interface (Figure 2.25) represents a WSMO interface (a description of the web service / goal orchestration and choreography). Each Interface is associated with at most one Choreography and at most one Orchestration.

---

\(^{17}\)See also the definition of Web Service in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#services](http://www.wsmo.org/TR/d2/v1.2/#services)

\(^{18}\)See also the definition of Goal in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#goals](http://www.wsmo.org/TR/d2/v1.2/#goals)

\(^{19}\)See also the definition of Capability in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#capability](http://www.wsmo.org/TR/d2/v1.2/#capability)

\(^{20}\)See also the definition of Interface in the WSMO specification at [http://www.wsmo.org/TR/d2/v1.2/#interface](http://www.wsmo.org/TR/d2/v1.2/#interface)
Note that in the WSMO API both *Capability* and *Interface* are top-level entities that can be reused by several Web Services or Goals. This is a slight deviation from the WSMO Specification [10] where capabilities and interfaces cannot be shared and reused. In our opinion, making capabilities and interfaces reusable is very important, since it is most likely that several services will provide functionality satisfying the same capability according to the same interface definition, and such a restriction in the WSMO specification induces unnecessary duplication of capability and interface definitions for each particular service.
2.7 Mediators

Mediators\(^{21}\) in WSMO provide an abstraction for components that provide interoperability on the data, protocol or process level [10].

There are four types of mediators defined at present:

- **ggMeditors** that link two goals (i.e. state equivalence between goals, or refine source goal into the target goal)
- **ooMeditors** that provide interoperability between two ontologies
- **wgMeditors** that link services to goals (i.e. specify that the source service fully or partially fulfils the target goal)
- **wwMeditors** that mediate between two services

The corresponding Java interfaces, part of the WSMO API, are presented on Figure 2.26.

\(^{21}\)See also the definitions of mediators in the WSMO specification at http://www.wsmo.org/TR/d2/v1.2/#mediators
2.8 Parsers and Serializers

The `org.wsmo.wsml` package contains interfaces related to import and export of WSML definitions from and into various formats.

At present parsers / serializers for the following languages and formats are available:

- WSML
- WSML-XML, the XML representation of WSML\(^{22}\)
- a subset of OWL-DL\(^{23}\) (import only)
- RDF\(^{24}\) (import only)

Figure 2.27 and Figure 2.28 present the *Parser* and *Serializer* interfaces respectively.

*Parsers* and *Serializers* are created by the *Factory* (section 2.3).

---

\(^{22}\)See [http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-xml](http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-xml) for details

\(^{23}\)See [http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-owl-mapping](http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-owl-mapping) for details

\(^{24}\)See [http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-rdf](http://www.wsmo.org/TR/d16/d16.1/v0.3/#sec:wsml-rdf) for details
2.9 Datastores and Repositories

The `org.wsmo.datastore` package contains interfaces related to interacting with datastores and repositories for storing WSMO descriptions of ontologies, services, goals and mediators.

The `DataStore` interface (Figure 2.29) provides a simple abstraction of a persistent storage that can be used to store and load WSMO descriptions. The `WsmoRepository` interface further refines `DataStore` by providing specific methods for each top-entity (`Ontology`, `WebService`, `Goal` and `Mediator`).

DataStores and WsmoRepositories are created by the Factory (section 2.3).

2.10 Validators

The `org.wsmo.validator` package contains interfaces that assist validation of the WSML descriptions created by the WSMO API.

The need for some validation mechanism emerges from the fact that WSMO provides several variants, namely WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full (see [4] for details), and since the variants are based on different log-
ical formalisms a description of a WSMO element created with the WSMO API (e.g. a TopEntity) may be valid in certain variant but not valid in another.

The Validator interface (Figure 2.30) provides means for checking the validity of a TopEntity according to its specified WSML variant. The validation process produces a list of warnings (i.e. non-critical problems) and errors (critical problems) identified. [12] presents more details about the validation process.

Validation warnings and errors are facilitated by the ValidationWarning and ValidationError interfaces (Figure 2.31), which are further subclasses into AttributeError and LogicalExpressionError.

Validators are created by the Factory (section 2.3).
Figure 2.29: *DataStore* and *Repository* interfaces
Figure 2.30: Validator interface

Figure 2.31: Validation warnings and errors
3 GROUNDING API

3.1 Introduction

The Grounding API is an extension of the WSMO API, which provides functionality for creating grounding of WSMO descriptions according to the WSDL-S [1] and SA-WSDL [7] specifications.

The grounding related interfaces are part of the org.wsmo.grounding package.

3.2 Grounding Factory

The GroundingFactory interface (see Figure 3.1) is provides the factory\(^1\) for creating grounding related elements (e.g. groundings, model references, assertions, categories, etc.). The GroundingFactory itself is created by the Factory (see section 2.3).

![GroundingFactory interface](image)

**Figure 3.1:** GroundingFactory interface

3.3 Grounding

The Grounding interface (Figure 3.2) is the main grounding description, i.e. it contains the mappings between WSDL elements (such as operations and XML types) and WSMO elements (such as concepts and axioms). A grounding description is comprised of:

- zero or more Categories (section 3.4)
- zero or more Effects and Preconditions (section 3.5)
- zero or more Model References (section 3.6)

\(^1\)See [8] for details on the Factory pattern
3.4 Model References

ModelReferences (Figure 3.3) represent mappings between elements in the two domains (WSDL and WSMO), for example a correspondence between an XML type in the WSDL file and a concept from a WSMO ontology may be specified.

ModelReferences are further divided into OperationModelReference (for mappings to WSDL operations), MessageModelReference (for mappings to WSDL messages), FaultModelReference (for mappings to WSDL faults) and TypeModelReference (for mappings to XML types) since there are specific restrictions on the different types of mappings.
3.5 Assertions

Preconditions or Effects (Figure 3.4) may be associated with certain WSDL operations in order to specify assertions that must/will hold before/after a web service operation is invoked. The assertions are either described by means of a ModelReference or by means of a logical expression.

![Diagram of Assertion, Effect, and Precondition interfaces]

Figure 3.4: Precondition and Effect interfaces

3.6 Category

A grounding may be associated with zero or more Categories (Figure 3.5), which refer to a specific taxonomy.

3.7 Parsers and Serializers

The Parser and Serializer interfaces provide ways to import and export the grounding descriptions into the formats specified by WSDL-S [1] and SA-WSDL [7].
Figure 3.5: Category interfaces

Figure 3.6: Parser and Serializer interfaces for Grounding
4 CHOREOGRAPHY API

4.1 Introduction

The Choreography API is an optional extension of the WSMO API, that provides the java interfaces for modelling WSMO centric choreographies based on Abstract State Machines, as specified by [11].

The Choreography API is still evolving and thus is not integrated with the main WSMO API. Besides, keeping the choreography extension separate from the WSMO API core, makes it easier to plug into the WSMO API other choreography modelling approaches such as Cashew ([9]) or ADO ([9]).

The Choreography API interfaces are part of the `org.wsmo.service.choreography` package.

The Choreography API provides a core conceptual model to deal with the description of the service invocation. A state-based approach is used and is inspired from the Abstract State Machine methodology is used. A key extension to the traditional ASM is that the definition of the machine signature is defined in the terms of WSMO ontologies and logical language to dynamically modify the underlying ontologies (see [11] for details). The Choreography is composed of a state signature and transition rules.

4.2 Choreography

There are two choreography interfaces: `org.wsmo.service.Choreography` (Figure 2.25) is part of the WSMO API and provides no additional methods beside the methods of `Entity` interface. The `org.wsmo.service.choreography.Chorepgraphy` (Figure 4.1), referred in this chapter simply as `Choreography`, is part of the Choreography API and is extended with the support of the state signature and transition rules.

4.3 State signature

The `StateSignature`\(^1\) (Figure 4.2) is a container for the `Mode` objects (which define how the ontology instances are interchanged between the client and the web service interface) and the imported ontologies (see [13]).

There are five different mode types, as defined by [11] (see Figure 4.3):

- **Static** – extension of the concept cannot be changed (default mode).

\(^1\)See also the definition of State Signature in the WSMO specification at http://www.wsmo.org/TR/d14/v0.4/#chorSig
Figure 4.1: *Choreography* interface

Figure 4.2: *StateSignature* interface
• In – extension of the concept or relation can only be changed by the environment and read by the choreography execution; a grounding mechanism for this item, that implements write access for the environment, must be provided.

• Out – extension of the concept or relation can only be changed by the choreography execution and read by the environment; a grounding mechanism for this item, that implements read access for the environment, must be provided.

• Shared – the extension of the concept or relation can be changed and read by the choreography execution and the environment; a grounding mechanism for this item, that implements read/write access for the environment and the service, may be provided.

• Controlled – the extension of the concept is changed and read only by the choreography execution.

Figure 4.3: Choreography modes

Grounding is upper interface for any grounding descriptions. The current Choreography API supports only WSDLGrounding, which defines methods for getting information about IRI pointing to the input/output parameter of some WSDL ([3]).
4.4 Transition rules

Transition rules\(^2\) define a formal algebra to model the changes of the state in the ASM. The web service interface execution is described by a finite set of transition rules, which are executed in parallel by the ASM agent (i.e. their order is not important). Rules express the changes of the state by modifying set of instances (adding, removing and updating instances to the signature ontology).

The available rules are:

- **if** condition **then** rules **endIf**
- **forall** variables with condition **do** rules **endForall**
- **choose** variables with condition **do** rules **endChoose**
- **add**( fact )
- **delete**( fact )
- **update**( fact\(_{old} \rightarrow \text{fact}_{new} \)), or **update**( fact\(_{new} \))
- Rule\(_1 \) | Rule\(_2 \) | Rule\(_3 \)

The corresponding Java interfaces (**IfThen**, **ForAll**, **Choose**, **Add**, **Delete**, **Update** and **PipedRules** respectively) are presented on Figure 4.4

4.5 Choreography factory

The choreography modelling elements are created by the **ChoreographyFactory**, which implements the **Factory** design pattern [8].

\(^2\)See also the definition of Transition rules in the WSMO specification at [http://www.wsmo.org/TR/d14/v0.4/#chorGt](http://www.wsmo.org/TR/d14/v0.4/#chorGt)
Figure 4.4: Choreography rules
5  WSMO4J

5.1  Introduction

In addition to specifying a Java API for WSMO, our work in WP6 (and specifically D6.4 and D6.14) has been focused on providing a reference implementation for the WSMO API.

The reference implementation, called wsmo4j, is freely available at http://wsmo4j.sourceforge.net and is already used by other tools developed within DIP (such as DOME\(^1\) in WP2, WSMO Studio\(^2\) in WP4 and WSMX\(^3\) in WP6) as well as other EU IST projects (InfraWebs\(^4\) and SemanticGov\(^5\)).

5.2  List of changes since M12

Deliverable D6.4 (due M12) already presented the first version of the WSMO API and the reference implementation (wsmo4j). Our work on the second version (D6.14) may be summarised as follows:

1. Conformance to the latest specifications – the WSMO API and wsmo4j have been updated to the latest WSMO [10] and WSML [4] specifications

2. Major refactoring of API – based on our experience with the first version, and based on the feedback from the wsmo4j community at SourceForge, and the WSMO/WSML working groups, we have performed major refactoring of the API in order to achieve:

   - better compliance with the WSMO and WSML specifications
   - improved robustness of the software
   - improved usability of the API

The result of the refactored API has been presented in chapter 2

3. Bugfixes – more than 100 bugs have been fixed since the M12 release of wsmo4j. A detailed log of the changes (incl. bugfixes) is available at http://wsmo4j.sourceforge.net/changes-report.html.

4. Performance improvements in various parts of the implementation (parsers, class loading)

\(^1\)http://dome.sourceforge.net/
\(^2\)http://www.wsmostudio.org/
\(^3\)http://www.wsmx.org/
\(^4\)IST project FP6-511723, http://www.infrawebs.org/
\(^5\)IST project FP6-4-027517, http://www.semantic-gov.org/
5. *Grounding API extension*, which was already presented in chapter 3

6. *Choreography API extension*, which was already presented in chapter 4

7. *New parsers* for WSML-XML and OWL-DL\(^6\)

8. *Support for metamodelling*\(^7\), as recommended by [4].

\(^6\)Note that the RDF parser was not created as part of a DIP WP6 effort.

\(^7\)Metamodelling allows for example an entity to be *both* a concept and an instance. Note that not all WSML variants allow metamodelling.
6 Conclusion

This deliverable presented the second version of a Java API (WSMO API) and reference implementation (wsmo4j) for building Semantic Web Service applications based on WSMO.

The WSMO API and its reference implementation (wsmo4j) have already been successfully used in various tools and projects\(^1\)

Although D6.14 is the last WSMO API / wsmo4j deliverable in DIP, we already have a developer and user community around the open source project, which will continue to enhance and extend the component in the future.

\(^1\)At present wsmo4j is used in DOMÉ (http://dome.sourceforge.net/), WSMO Studio (http://www.wsmostudio.org/), WSMX (http://www.wsmx.org/) and IRS-III (http://kmi.open.ac.uk/projects/irs/). In the last 18 months since its first release, wsmo4j has been downloaded more than 600 times from SourceForge (statistics available at http://sourceforge.net/project/showfiles.php?group_id=113501)
BIBLIOGRAPHY


Appendix A – Examples

Examples demonstrating the usage of WSMO API are available online at http://wsmo4j.sourceforge.net/examples.html