Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D3.2.7 Second Prototype for Description Logic Reasoner for WSML DL v2.0

Report on Installation and Configuration

Activity N: 2	Fundamental and Integration Activities	
Work Package: 3	Service Annotation and Reasoning	
Due Date:		31/08/2010
Submission Date:		31/08/2010
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Report:		UIBK
Revision:		1.0
Author(s):	Daniel Winkler, UIBK Matthias Pressnig, UIBK	
Reviewers:	Barry Norton, KIT Martin Junghans, KIT Mateusz Radzimski, ATOS Yosu Gorroñogoitia, ATOS	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
	Dissemination Level	
PU	Public	**x**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	22/07/2010	First draft	Daniel Winkler
0.2	27/07/2010	Added content	Matthias Pressnig
0.3	28/07/2010	Added content, peer review	Daniel Winkler
0.4	29/07/2010	Internal review	Matthias Pressnig
0.5	30/07/2010	Peer review	Martin Junghans
0.6	03/08/2010	Corrections after peer review	Daniel Winkler
0.7	30/08/2010	Format revision for submission	Julia Wells (ATOS)
1.0	25/08/2010	Final version	Daniel Winkler

Table of Contents

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
WP	Work Package
DL	Description Logic
LP	Logic Programming
OWL	Web Ontology Language
WSML	Web Service Modelling Language
WSMO	Web Service Modelling Ontology
KB	Knowledge Base
RB	Rule Base

Executive summary

This document serves as software description of the WSML2Reasoner reasoning framework, which integrates Logic Programming (LP) and Description Logic (DL) reasoners together with a common object-model. The object-model is implemented in the WSMO4J library, with which all *static operations* on WSML ontologies are carried out (by the user as well as by the reasoning framework). Facades are used to hide the actual reasoner implementations (IRIS, ELLY) from the user, thus those parts of the framework do not need to be configured or managed while performing reasoning.

The main task of WSML2Reasoner is to build a bridge between the WSMO4J object model and the reasoners, which carry out the computations. This means that *dynamic operations* which corresponds to reasoning has to be done by means of the WSML2Reasoner framework.

1. Installation

This section explains how WSML reasoning support can be added to a project by adding the WSML2Reasoner framework as dependency to a project. Since SOA4All agreed on using Maven for distributed project development, this tutorial will mainly focus on how reasoner support can be added to maven projects.

1.1 Hosting

In order to install and configure the reasoning framework, a Java Virtual Machine (version 1.5 or later) is required. The WSML2Reasoner binary distribution can be obtained from the sourceforge project page[1] or via the WSML2Reasoner homepage[2].

Additionally, to ease the integration of the framework including all dependencies, WSML2Reasoner is developed as Apache Maven[3] project and distributed via the STI maven repository (http://maven.sti2.at/archiva/repository/external/). To get releases and snapshots of wsml2reasoner and dependent components, the following repositories have to be added to the project object model (POM) files:

```
<repositories>
      <repository>
            <id>sti2-archiva-external</id>
            <url>http://maven.sti2.at/archiva/repository/external</url>
      </repository>
      <repository>
            <id>sti2-archiva-snapshots</id>
            <url>http://maven.sti2.at/archiva/repository/snapshots</url>
      </repository>
</repositories>
```

However, the general SOA4All project setup should have the SOA4All NEXUS repository[4] hosted by TIE in its configuration, which mirrors both STI repositories, thus the STI repositories do not need to be added explicitly. The repositories that should be used in the configuration for mirroring are:

- `http://coconut.tie.nl:8080/nexus-webapp-1.3.1/content/groups/public/`
- `http://coconut.tie.nl:8080/nexus-webapp-1.3.1/content/groups/public-snapshots/`

[1] https://sourceforge.net/projects/wsml2reasoner/

[2] http://tools.sti-innsbruck.at/wsml2reasoner/

[3] http://maven.apache.org/

[4] http://coconut.tie.nl:8080/nexus-webapp-1.3.1

The software was released on 23/07/2010 in its latest version:

- WSML2Reasoner: version 0.8.1
- WSMO4J: version 2.1.1
- ELLY: version 0.2.1
- IRIS: version 0.7.1

Ongoing work, e.g. bug fixes are released on a weekly basis, the corresponding versions are:

- WSML2Reasoner: version 0.8.2-SNAPSHOT
- WSMO4J: version 0.2.2-SNAPSHOT
- ELLY: version 0.2.2-SNAPSHOT
- IRIS: version 0.7.2-SNAPSHOT

All reasoning related projects are hosted on Sourceforge for WSML2Reasoner and WSMO4J the development is hosted in "WSML2.0" branches on the subversion repository.

As opposed to earlier versions of WSML2Reasoner, support for the OWL reasoners Pellet and KAON2 as well as for the Datalog reasoner MINS has been dropped. This allows distributing the reasoning framework using one single license, namely the GNU Lesser GPL (LGPL).

The package of the WSML2Reasoner framework consists of the following components:

- wsml2reasoner-0.8.1-sources.zip: The source code of the reasoning framework.
- wsml2reasoner-0.8.1-javadoc.zip: The JavaDoc of the reasoning framework API.
- wsml2reasoner-0.8.1.jar: The main executable containing all required jars.

Note that all components can be downloaded from the respective project sites separately. To use the WSML2Reasoner framework, the project user has to use the binaries located in the WSML2Reasoner Java archive file or compile the source and add the needed components to the classpath.

2. Usage

This section will give an overview of the WSML2Reasoner framework and describe how it can be used to perform ontology related tasks.

2.1 Input / Output

The WSML2Reasoner framework supports various WSML syntaxes to be parsed and serialized. All implementations are unified by implementing shared interfaces. Like stated in the introduction, the following tasks are all realized by means of the used WSMO4J library.

2.1.1 Parsing

The parsing interface is

- `org.wsmo.wsml.Parser`

The parser provides a method `parse`, which returns an array of top enities[5]. Top entities may generally be any WSMO top entity; however in the scope of the SOA4All project top entity merely represents the Ontology concept.

The parsing implementations are

- WSML syntax: `com.ontotext.wsmo4j.parser.wsml.WsmlParser`
- RDF syntax: `org.deri.wsmo4j.io.parser.rdf.RDFParser`
- XML syntax: `com.ontotext.wsmo4j.parser.xml.XmlParser`

Be aware that the RDF parser is actually not a parser for the WSML RDF Syntax[6] but rather an interpreter that translates arbitrary RDF(S) to WSML.

A parser that does not implement the above interface is the logical expression (LE) parser, used to parse WSML logical expressions. While being integrated in the above parsers, the LE parser may be used stand-alone to parse any WSML human readable logical expressions e.g. queries or axioms. The implementation of the `LogicalExpressionParser` provides a constructor that allows to pass a WSMO top entity, which serves as “NamespaceHolder” (i.e., the parser extracts the namespaces and the according prefixes from the top entity). This means in particular that a logical expression can be parsed with namespace qualified IRIs, which must be defined in the provided top entity to be interpreted. Otherwise, all IRIs have to be fully qualified.

The specific implementation is

- WSML Syntax:
  `org.deri.wsmo4j.io.parser.wsml.WsmlLogicalExpressionParser`

[5] `org.wsmo.common.TopEntity`

[6] http://www.wsmo.org/TR/d32/v1.0/

Since parsing involves creation of objects, the parser constructors require an instance of `org.wsmo.factory.FactoryContainer`. The factory container is a container that holds instances of all object factories that are required for creating WSML ontologies. These are:

- `org.wsmo.factory.WsmoFactory`
- `org.wsmo.factory.DataFactory`
- `org.wsmo.factory.LogicalExpressionFactory`
- `org.wsmo.factory.ChoreographyFactory`

In the case that no factory container is provided to the constructor, the parser implementations create fresh factories unified in a fresh factory container.

The factory container concept was not present in previous versions of WSMO4J and introduced for the reason that one may want to cope with several separated ontologies in a single JVM. In previous versions all objects were registered in single static factories, such that parsed ontologies were merged semi-automatically if their entities (concepts, instances, relations, attributes, axioms...) shared identifiers. This behaviour can be replicated by creating just one factory container and providing it to all classes that are used to create objects (e.g. parsers or synthetic ontology factories). To create separated ontologies one can either reset the container or create a new one and provide it to the respective classes.

2.1.2 Serialization

As with parsing, the serialization uses a single interface to unify the serialization in different syntaxes:

- `org.wsmo.wsml.Serializer`

As opposed to the `parse` method, the serializers implement a `serialize` method that takes an array of TopEntities as input.

The implementations are

- WSML Syntax[7]: `org.deri.wsmo4j.io.serializer.wsml.WSMLSerializerImpl`
- RDF Syntax[8]: `org.deri.wsmo4j.io.serializer.rdf.WsmlRdfSerializer`
- XML Syntax[9]: `com.ontotext.wsmo4j.serializer.xml.WsmlXmlSerializer`

2.2 Validation

Within the parsing of input files in the WSML2Reasoner framework a basic validation check is automatically done. This check tries to find errors in the parsed WSML file that causes the parser to stop. To check for errors in a WSML file the validator can also be created manually[10], whose `isValid` method can be used to specify a WSML variant against which the ontology will be checked to be valid. Additionally lists for errors and warnings have to be passed, which are used during the validation process to log any warnings and errors.

---

[7] WSML human readable syntax defined at http://www.wsmo.org/TR/d16/d16.1/v1.0/

[8] RDF/XML translation defined in JavaDoc of the class implementation

[9] WSML/XML syntax defined at http://www.wsmo.org/TR/d36/v1.0/

[10] `org.deri.wsmo4j.validator.WsmlValidatorImpl`

3. Reasoning

WSML2Reasoner allows to reason over Logic Programming as well as Description Logic ontologies. Due to the different underlying formalisms, both paradigms need to be represented by different reasoner APIs, and are thus discussed separately.

The creation of the reasoners is unified in the reasoner factory

`org.wsml.reasoner.api.WSMLReasonerFactory`

which features methods of type `createXXXReasoner(HashMap<String,Object> params)`. `XXX` can be replaced by

- `DL`
- `Core`
- `Flight`
- `Rule`
- `WSML`

The fifth option (`WSML`) determines the type of the ontology and creates an appropriate reasoner, based on the used language features.

The `params` argument may be null, which creates a correctly pre-configured reasoner for the respective variant, but may also be configured by the user manually. This parameter is a remnant from the old WSML2Reasoner API that was not changed for compatibility reasons. It’s main purpose in the current implementation would be to pass a factory container, by putting it in the map with the according key[11]. Consult the JavaDoc for further information.

Both reasoner facades share a common interface

`org.wsml.reasoner.api.WSMLReasoner`

which handles the registering and de-registering of ontologies. The actual reasoning interface is defined separately.

3.1 Interface

3.1.1 Logic Programming (LP)

For the Logic Programming paradigm the interface

`org.wsml.reasoner.api.LPReasoner`

defines the functionality. The most important methods for SOA4All use-cases are `checkConsistency` for checking the consistency of the registered ontologies and `executeQuery`, which runs a query against the currently used reasoner and returns a variable binding. A listing of all available methods and detailed information can be found in the JavaDoc.

[11] `WSMLReasonerFactory.PARAM_FACTORY_CONTAINER`

3.1.2 Description Logic (DL)

For the Description Logic paradigm the interface

```
org.wsml.reasoner.api.DLReasoner
```

defines the functionality. This interface differs significantly from the LP interface on a conceptual level, in a sense that there is not a single query method, but every supported query is already exposed as method in the interface, e.g.

```
boolean isSubConceptOf(Concept subConcept, Concept superConcept)
```

which tests whether one WSML concept is sub-concept of another one. There are also corresponding methods

```
Set<Concept> getSubConcepts(Concept concept)
```

that allow to retrieve a set of results. A listing of all available methods and detailed information can be found in the JavaDoc.

3.2 Ontology merging

As mentioned earlier, the concept of a factory container allows to create ontologies in separate object models. For the case that these ontologies are intended to share entities, the class

```
org.sti2.wsmo4j.merger.Merger
```

allows to merge a collection of ontologies into a single one, thereby automatically merging all contained entities (e.g. concepts, instances, relations). This merging is done automatically when registering a bunch of ontologies to a reasoner, since the reasoner does not care about where logically relevant entities are defined (in which ontology).

3.3 Reasoning Example

Listing 1 shows an example Java program that executes a query against an LP reasoner ontology with a WSML-Core ontology registered. The example ontology features equality in rule heads and is outlined in Listing 2. For the sake of simplicity, exceptions are not handled in the example. Listing 3 illustrates the query result.

4. Utility Applications

4.1 Web Reasoning Tools

Online reasoning Tools are available for WSML Rule and DL reasoning.

WSML-Rule v2.0 Reasoner:

http://tools.sti-innsbruck.at/wsml/rule-reasoner/v0.2/

WSML DL v2.0 Reasoner:

http://tools.sti-innsbruck.at/wsml/dl-reasoner/v0.2/

4.2 Web RDF Translation

It is possible to translate RDF(S) to WSML online – the tool is available at the following address:

http://soa4all-runtime.sti2.at:8180/rdftranslation-webservice/

The RDF translation is also available as RESTful web service available. To use the service follow the instructions on this site:

http://soa4all-runtime.sti2.at:8180/rdftranslation-webservice/rs/translate

4.3 Demo Widget

The class `DemoW` – a Demo Widget for reasoning - can be found in the package

`org.wsml.reasoner.gui` in the WSML2Reasoner implementation project.

It provides a GUI for quick reasoning tests.

Annex A. Full Reasoning Example

Listing 1: Reasoning example

```
public class Example {
  public static void main(String[] args) {
    // Create a parser and parse the example ontology file.
    // For simplicity we do not take care of exceptions at the moment.
    Parser wsmlParser = new WsmlParser();
    InputStream is = wsmlParser.getClass().getClassLoader()
        .getResourceAsStream("example/instance-equality.wsml");
    TopEntity[] identifiable = wsmlParser.parse(new InputStreamReader(is));

    // We can be sure here, that we only parse a single ontology.
    Ontology ontology = (Ontology) identifiable[0];

    // Create a query, that should bind x to both instances A and B.
    String queryString = "p(?x)";

    // Define the desired reasoner by setting the corresponding values in
    // the parameters. Here IRIS reasoner with well-founded semantics is
    // used.
    Map<String, Object> params = new HashMap<String, Object>();
    params.put(WSMLReasonerFactory.PARAM_BUILT_IN_REASONER,
        WSMLReasonerFactory.BuiltInReasoner.IRIS_STRATIFIED);

    // Instantiate the desired reasoner using the default reasoner factory.
    LPReasoner reasoner = DefaultWSMLReasonerFactory.getFactory()
        .createFlightReasoner(params);

    // Register the ontology.
    reasoner.registerOntology(ontology);

    // Transform the query in string form to a logical expression object.
    LogicalExpression query = new WsmlLogicalExpressionParser(ontology)
        .parse(queryString);

    // Execute query request and assign the result to 'bindings'.
    Set<Map<Variable, Term>> bindings = reasoner.executeQuery(query);
  }
}
```

Listing 2: Example WSML file

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.example.org/example#",
      wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
      rif  _"http://www.w3.org/2007/rif-builtin-function#"
}
ontology exampleOntology
      concept C1
      concept C2

      instance A memberOf C1
            name hasValue _string("Gordon Freeman")

      instance B memberOf C2
            nomen hasValue _string("Gordon Freeman")

      axiom exampleAxiom definedBy
            ?x = ?y :- ?x[name hasValue ?name]
                  and ?y[nomen hasValue ?nomen]
                  and rif#compare(0, ?name, ?nomen).
            p(?x) :- ?x memberOf C2.
```

Listing 3: Reasoning results

```
2 results to the query:
   (1)      - ?x=http://www.example.org/example#A
   (2)      - ?x=http://www.example.org/example#B
```