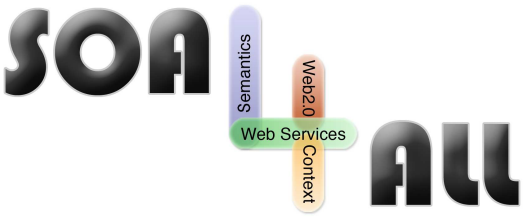

Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D3.2.2 First Prototype Reasoner for WSML-Core v2.0

Activity N:	Activity 2 – Core R&D Activities	
Work Package:	WP3 - Service Annotation and Reasoning	
Due Date:		M18
Submission Date:		10/09/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Author(s):	Matthias Pressnig UIBK	
Reviewers:	Alex Simov UIBK	

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	2009-08-10	First TOC	Barry Norton (UIBK)
0.2	2009-08-24	First draft	Matthias Pressnig (UIBK)
0.3	2009-08-25	Updated draft	Matthias Pressnig (UIBK)
0.4	2009-08-31	Review changes applied	Matthias Pressnig (UIBK)
0.5	2009-09-10	Final Editing	Malena Donato (ATOS)

Table of Contents

Glossary of Acronyms

Acronym	Definition
D	Deliverable
DL	Description Logic
EC	European Commission
WP	Work Package
WSML	Web Service Modeling Language
RIF	Rule Interchange Format

Executive summary

In order to automate tasks such as discovery and composition, Semantic Web Services must be described in a well-defined formal language. The Web Services Modelling Language (WSML) is based on the conceptual model of the Web Service Modelling Ontology (WSMO) and as such can be used for modelling Web services, ontologies, and related aspects.

WSML is actually a family of several language variants, each of which is based upon a different logical formalism. The family of languages are unified under one syntactic umbrella, with a concrete syntax for modelling ontologies, web services, goals and mediators.

This deliverable, along with others, describes the first prototype reasoner for WSML-Core 2.0, in particular the new feature *instance equivalence* also know as *equality in rule heads*[1].

1. Introduction

The Web Service Modelling Language WSML[2] is such a formal language for the specification of ontologies and different aspects of Web services, based on the conceptual model of WSMO [3]. Several different WSML language variants exist, which are based upon different logical formalisms. The main formalisms exploited for this purpose are Description Logics [4], Logic Programming [5], and First-Order Logic [6]. Furthermore, WSML has been influenced by F-Logic [7] and frame-based representation systems.

This deliverable discusses the implementation of the first prototype reasoner for WSML-core 2.0. The WSML-Core 2.0 language aims to provide a minimal but useful expressivity and is inspired by minimal representation from project LarKC[1]. It belongs to a set of related deliverables, which discuss the prototype implementation of several WSML 2.0 variants, namely:

- **D3.2.2. First Prototype Reasoner for WSML-Core v2.0**
- D3.2.3 First Prototype Reasoner for WSML-Rule v2.0
- D3.2.4 First Prototype Reasoner for WSML-DL v2.0

1.1 Purpose and Scope

1.1.1 Audience

This Document is intended to inform about the prototype reasoner for WSML-Core 2.0. In turn, its main audience are users and developers who are intend to use the prototype.

1.1.2 Scope

Reasoning for WSML-Core 2.0 can be performed by transforming the WSML data to adequate Datalog terms, which are reasoned by an underlying reasoning engine. This engine and the transforming process have also to provide the features of the WSML-Core 2.0 implementation.

This deliverable has the objective to provide information about the features of the first prototype reasoner for WSML-Core 2.0. In particular, it targets the changes for *equality in rule heads* feature (or *instance equivalence*), which has been implemented in the Datalog reasoner *IRIS*[2] and the framework *WSML2Reasoner*[3].

1.2 Structure of the document

The structure of this deliverable is formed as follows: Section 2 discusses the actual implementation and its changes according to the Specification. In Section 3 there it is described how to install and use the prototype reasoner for WSML-Core 2.0. The main implementation of the prototype is described in Section 4. Section 5 concludes with a short Summary of the deliverable.

---

[1] LarKC – European Project http://www.larkc.eu/ [24.08.2009]

[2] IRIS Reasoner http://www.iris-reasoner.org [27.07.2009]

[3] WSML2Reasoner http://tools.sti-innsbruck.at/wsml2reasoner/ [24.08.2009]

2. Reflection on the Specification

The WSML-Core 2.0 was designed to provide minimal expressivity and thus improve reasoning scalability. To provide instance equality the IRIS implementation and the WSML2Reasoner had to be modified to perform new transformation and reasoning. Besides of that it was not necessary to change something from the SOA4All Annex I - “Description of Work”.

3. Installation and Configuration

In order to install and configure the reasoning framework, a Java Virtual Machine (version > 1.5) is required. The WSML2Reasoner package with provides the new features can be downloaded on the WSML2Reasoner website[4].

The *WSML2Reasoner* software is licensed under the GNU lesser GPL (LGPL). However, there are three release variants in accordance with the license agreements for the bundled reasoning engine libraries:

- *LGPL:* This release includes all the LGPL libraries used by WSML2Reasoner, including the IRIS and PELLET reasoning engines.
- *GPL:* In addition to the LGPL libraries and packages, this release includes the MINS reasoning engine, which is licensed under the GNU GPL.
- *Proprietary:* This release version does not include any further libraries or reasoning engines. However, it does include wrapper classes that allow the WSML2Reasoner framework to use the KAON2 reasoning engine.

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

- wsml2reasoner-src-x.x.x.zip: The source code of the reasoning framework.
- wsml2reasoner-javadoc-x.x.x.zip: The JavaDoc of the reasoning framework API.
- wsml2reasoner-x.x.x.jar:
- lib folder: The required and optional libraries.

To use the WSML2Reasoner you have to use the binaries located in the wsml2reasoner jar file or compile the source and add the libraries found in the lib-folder to the classpath.

The following example shows a main class, which loads an ontology and does reasoning with the IRIS datalog reasoner[5]:

[4] WSML2Reasoner downloads: http://tools.sti-innsbruck.at/wsml2reasoner/download [24.08.2009]

[5] WSMO4J Programmers guide: http://wsmo4j.sourceforge.net/doc/wsmo4j-prog-guide.pdf [31.08.2009]

```
public class Example {
        public static void main(String[] args) throws IOException, ParserException,
                        InvalidModelException, InconsistencyException {
                // Create a parser and parse the example.wsml file. For simplicity we do
                // not take care of exceptions at the moment.
                Parser parser = Factory.createParser(null);
                TopEntity[] identifiables = parser
                                .parse(new FileReader("example.wsml"));

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

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

                // Create the reasoner using the previously defined parameters and the
                // default reasoner factory.
                LPReasoner reasoner = DefaultWSMLReasonerFactory.getFactory()
                                .createRuleReasoner(params);

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

                // Create the logical expression factory.
                LogicalExpressionFactory factory = Factory
                                .createLogicalExpressionFactory(null);

                // Transform the query in string form to a logical expression object.
                LogicalExpression expression = factory.createLogicalExpression(query,
                                ontology);

                // Execute the query and do something with the result of the query.
                Set<Map<Variable, Term>> bindings = reasoner.executeQuery(expression);
        }
}
```

This example wsml file (example.wsml) can be used to show the instance equality feature. An ontology defines two concepts with two instances; one of them has an attribute:

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"
namespace { _"http://simple#"
}
ontology simple

concept C1
concept C2

instance a memberOf C1
        name hasValue aName

instance b memberOf C2

axiom equalInHead definedBy
        a = b :- true.
        p(?x) :- ?x memberOf C2 and ?x[name hasValue aName].
```

The query “p(?x)” returns both instances since they are set equal in the axiom:

```
2 results to the query:
(1) - {?x=http://simple#b}
(2) - {?x=http://simple#a}
```

4. Software Description

WSML-Quark[11] is a lightweight and intuitive language variant that enables the hierarchical organization of concepts. It forms the most basic layer of the WSML language variants hierarchy. WSML-Core 2.0 is an extension of WSML-Core and is layered upon WSML Quark. It has been updated to align results of ongoing standardization efforts (e.g. OWL 2 RL) as well as research results such as the L2 language, which has similar language features.

The Datalog reasoner *IRIS* has been modified in order to support the new features introduced and required by the new versions of the language variants. This is particularly important, since various WSML variants are translated to Datalog programs and therefore rely on the Datalog reasoner to cover the required features.

4.1 Equality in rule heads

WSML-Core 2.0 introduces instance equivalence, otherwise known as *equality in rule heads*. In WSML, this allows the declaration that different instance identifiers (IRIs) refer to the same object. In Datalog, *equality in rule heads* allows the declaration of equivalence between constant terms, such as strings or integers. *Equality in rule heads* has been integrated into the Datalog reasoner *IRIS*. Two approaches have been implemented to realize this feature, a rewriting technique and integrated support for equivalence in rule heads:

- *Rewriting:* For a given Datalog program containing rules with equality in the head, this technique creates new rules to provide support for *equivalence in rule heads*. First, all occurrences of equality in the head of a rule are replaced by a special predicate (in the following examples denoted by *equivalent*). Then, new rules are created to ensure the correct evaluation of rule head equality. Note that rule (1) and (2) are unsafe rules, since the property *"each variable in the rule head appears in a non-negated, ordinary relation"* is violated:

  (1) equivalent(?X, ?X) :- .
  (2) equivalent(?X, ?Y) :- ?X = ?Y
  (3) equivalent(?X, ?Y) :- equivalent(?Y, ?X).
  (4) equivalent(?X, ?Y) :- equivalent(?X, ?Z), equivalent(?Z, ?Y).

  The rewriting algorithm then creates new rules for each predicate occurring in the program, this includes predicates in rules and the predicates of facts. The number of new rules depends on the arity of the predicates. For each predicate *p* this technique creates *n* new rules, where *n* is the arity of *p*. Assume a predicate *hasName(?X, ?Y, ?Z)* with arity 3. For this predicate the following three rules are created:

  (1) hasName(?U, ?Y, ?Z) :- hasName(?X, ?Y, ?Z), equivalent(?X, ?U).
  (2) hasName(?X, ?U, ?Z) :- hasName(?X, ?Y, ?Z), equivalent(?Y, ?U).
  (3) hasName(?X, ?Y, ?U) :- hasName(?X, ?Y, ?Z), equivalent(?Z, ?U).

  Obviously, this may create an extraordinary amount of additional rules. Furthermore, it is required, that unsafe rules are created. However, an advantage of this approach is, that the resulting program can be evaluated using any Datalog reasoner that supports unsafe rules, regardless of whether the reasoner explicitly supports

equivalence in rule heads or not. This brings support for equality in rule heads for reasoners which do not support it.

- *Integration:* Due to the disadvantages of the approach described above, the support for equivalence in rule heads has been integrated into *IRIS* by modifying the way rules are evaluated. During evaluation of a Datalog program the reasoner keeps track of all the terms that have been identified as being equivalent according to the rules with equality in the head of a rule. When evaluating rules, the reasoner also takes into account the equivalence relations that have been established. For instance, when using a view *p(?X, 'a')* over a relation, the evaluation returns all tuples *(x, y)* where *x* is some term and *y* is any term equivalent to *'a'* (note that *'a'* is equivalent to itself).

5. Conclusions

This deliverable accompanies the fist prototype reasoner for WSML-Core 2.0, which provides the *instance equivalence*.

The software prototype can be downloaded from http://tools.sti-innsbruck.at/wsml2reasoner/download and uses the IRIS reasoner[6] and the WSMO4J Framework[7] to perform the reasoning.

[6] IRIS Reasoner http://www.iris-reasoner.org/ [31.08.2009]

[7] WSMO4J Framework http://wsmo4j.sourceforge.net/ [31.08.2009]

6. References

[1] G. Unel, U. Keller, F. Fischer and B. Bishop: “D3.1.2 Defining the Features of the WSML-Core v2.0 Language”, 2009.

[2] J. de Bruijn, “WSML Language Reference, Deliverable D16.1v1.0,” WSML Final Draft 2008-08-08, 2005.

[3] D. Roman, H. Lausen and U. Keller, “Web Service Modeling Ontology (WSMO)” WSMO Working Draft, 2004.

[4] F. Baader et al., “The Description Logic Handbook,” 2007.

[5] J.W. Lloyd, Foundations of logic programming, Springer-Verlag New York, Inc. New York, NY, USA, 1987.

[6] M. Fitting, First-Order Logic and Automated Theorem Proving, Springer, 1996.

[7] M. Kifer and G. Lausen, “F-logic: a higher-order language for reasoning about objects, inheritance, and scheme,” Proceedings of the 1989 ACM SIGMOD international conference on Management of data, 1989, pp. 134-146.

[8] T. Vitvar et al., “WSMO-Lite Annotations for Web Services,” The Semantic Web: Research and Applications, 2008, pp. 674-689; http://dx.doi.org/10.1007/978-3-540-68234-9_49.

[9] J. Farrell and H. Lausen, “Semantic Annotations for WSDL and XML Schema,” W3C Proposed Recommendation, vol. 5, 2007.

[10] J. de Bruijn et al., “The Web Service Modeling Language WSML,” WSML Final Draft D, vol. 16, 2005.

[11] G. Unel, U. Keller, F. Fisher and B. Bishop, “Defining the features of the WSML-Quark language.”, 2009.

