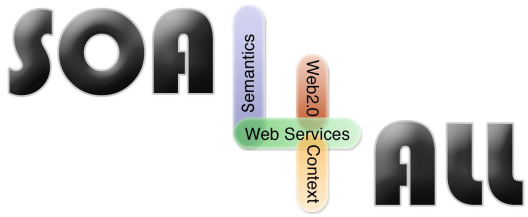

Project Number: **215219**

Project Acronym: **SOA4All**

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

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D3.2.3 First Prototype Rule Reasoner for WSML-Rule v2.0

Activity N:	Activity 2 – Core R&D Activities
Work Package:	WP3 - Service Annotation and Reasoning
Due Date:	M18
Submission Date:	11/09/2009
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	UIBK
Revision:	1.0
Author(s):	Adrian Marte 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	Adrian Marte (UIBK)
0.3	2009-08-24	Added comments	Barry Norton (UIBK)
0.4	2009-08-25	Updated draft	Adrian Marte (UIBK)
0.5	2009-08-28	Review changes applied	Adrian Marte (UIBK)
0.5	2009-09-10	Final Editing	Malena Donato (ATOS)

Table of Contents

EXECUTIVE SUMMARY 5

1. INTRODUCTION 6

1.1 PURPOSE AND SCOPE 6
1.2 STRUCTURE OF THE DOCUMENT 6

2. REFLECTION ON THE SPECIFICATION 7

3. INSTALLATION AND CONFIGURATION 8

4. SOFTWARE DESCRIPTION 11

4.1 RIF DATA TYPES AND BUILT-IN PREDICATES 11

5. CONCLUSIONS 13

6. REFERENCES 14

7. APPENDIX 15

7.1 EXAMPLE ONTOLOGY 15

List of Figures

Figure 1: Reasoning example 9

Glossary of Acronyms

Acronym	Definition
D	Deliverable
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 [1].

This deliverable, along with others, describes the first prototype reasoner for WSML-Flight 2.0 and WSML-Rule 2.0 with a focus on the required changes done to the underlying Datalog reasoner *IRIS*. The *WSML2Reasoner* reasoning framework can be downloaded from the project website[1] (optionally already includes the *IRIS* libraries) and the Datalog reasoner *IRIS* can be downloaded from the *IRIS* project website[2].

The implementation presented in this deliverable yields the realization of the concepts and specifications that were released with deliverable *D3.2.1 Framework and APIs for integrated reasoning support* with respect to reasoning for WSML-Rule 2.0.

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

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

1. Introduction

This report describes a first prototype of a rule reasoner for WSML-Flight 2.0 and WSML-Rule 2.0. WSML-Flight 2.0 is the least expressive of the two LP-based WSML variants. Compared to WSML-Core, it adds features such as meta modelling, constraints and non-monotonic (stratified) negation. WSML-Flight is semantically equivalent to Datalog with equality and integrity constraints. WSML-Rule 2.0 is an extension of WSML-Flight 2.0. It adds features from Logic Programming, such as the use of function symbols, unsafe rules and unstratified negation [3].

Reasoning for these two WSML variants is realized by converting a WSML ontology to the corresponding Datalog program and then perform reasoning on this Datalog program using a Datalog reasoner. Therefore, the Datalog reasoner has to cover the features required by WSML-Flight 2.0 and WSML-Rule 2.0. In this deliverable, we focus on describing the necessary changes done to the Datalog reasoner *IRIS*.

The implementation presented in this deliverable follows the concepts and specifications that were released with deliverable *D3.2.1 Framework and APIs for integrated reasoning support* with respect to reasoning for WSML-Rule 2.0.

1.1 Purpose and Scope

This document is intended as a documentation for the first prototype reasoner for WSML-Flight 2.0 and WSML-Rule 2.0. In turn, its main audience are users who want to model Web services and ontologies using WSML, as well as technical staff building tools (i.e. reasoners) that use the WSML language.

Reasoning for WSML-Flight 2.0 and WSML-Rule 2.0 can be achieved in the same way as for WSML-Core 2.0 by performing the conversion steps that transform a WSML ontology to the corresponding Datalog program. In order to support the added expressivity in WSML-Flight 2.0 and WSML-Rule 2.0 the underlying Datalog reasoner needs to provide support for the required features. In particular, the Datalog reasoner needs to provide support for Rule Interchange Format (RIF) built-in data types, predicates and functions [3] and for *instance equivalence* (or in other words *equality in rule heads*) [2].

The main purpose of this deliverable is to present the first prototype of a rule reasoner for WSML-Flight 2.0 and WSML-Rule 2.0. In this deliverable, we focus on describing the changes made to the Datalog reasoner *IRIS* in order to support the new features introduced in WSML-Flight 2.0 and WSML-Rule 2.0.

Reasoning is of interest to various components in the *Web Enabled Service Platform* layer and more particularly for developers of components in work packages *WP5 Service Location* and *WP6 Service Construction*, for which reasoning is basic infrastructure in the process of service discovery and composition. Furthermore, all use-cases (WP7, WP8 and WP9) have certain dependencies on the reasoning component.

1.2 Structure of the document

The remainder of this deliverable is structured as follows: Section 2 provides a reflection on the original specification of this deliverable. The installation and configuration of the software and an example execution for reasoning is described in section 3, whereas the software itself is described in section 4. Finally, in section 5 a conclusion of the deliverable is given, which also provides information on where to download the prototype reasoner.

2. Reflection on the Specification

The implementation of a prototype reasoner for the rule-based language variants WSML-Flight 2.0 and WSML-Rule 2.0 has been realized without any changes from the original specification. However, the comparison of the performance of the developed prototype reasoner with the theoretical complexity results for the reasoning tasks for the language has not been done yet.

3. Installation and Configuration

In order to install and configure the reasoning framework, a Java Virtual Machine (version > 1.5) is required. The *WSML2Reasoner* is provided as a Java implementation and can be downloaded from the WSML2Reasoner website. 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[3] 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[4] 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.

In order to use the *WSML2Reasoner* framework both the wsml2reasoner-x.x.x.jar and the libraries found in the lib folder have to be on the class path. Figure 1 shows an example execution for reasoning on an ontology defined in the file specified in Appendix 1. The file name “example.wsml” will be used to address this content. In the example components of the WSMO4J object model are used, see the WSMO4J programmers guide for more information[5]. For the sake of simplicity, exceptions are not handled in the example.

[3] Pellet: The Open Source OWL Reasoner, http://clarkparsia.com/pellet/ [28.08.2009]

[4] KAON2, http://kaon2.semanticweb.org/ [28.08.2009]

[5] WSMO4J programmers guide, http://wsmo4j.sourceforge.net/doc/wsmo4j-prog-guide.pdf [28.09.2009]

```
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);
        }
}
```

Figure 1: Reasoning example

Apart from its integrated use in the *WSML2Reasoner* framework the *IRIS* reasoner can also be used as a standalone Datalog reasoner. *IRIS* is provided as a Java implementation. It is available under the GNU lesser general public licence (LGPL) and can be downloaded separately from the *IRIS* project website. The download package consists of the following components:

- *JavaDoc:* The JavaDoc of the reasoner API is contained in the “build” folder.
- *Source code:* The source code of the *IRIS* reasoner is contained in the “build” folder.
- *Optional libraries:* Some optional libraries are contained in the “lib” folder
- *User guide:* A user guide describing the basic notions, the architecture and the usage of the *IRIS* reasoner is contained in the “doc” folder.
- *iris-x.xx.jar:* The *IRIS* reasoner library (where *x.xx* stands for the version of the *IRIS* reasoner). This library has to be on the class path in order to use the *IRIS* reasoning engine.
- *iris-parser-x.xx.jar:* The parsers which parses Datalog programs in string or file form into the representation required by the *IRIS* reasoner. Note that the *iris-x.xx.jar* file has to be on the class path in order to parse a Datalog program.
- *iris-app-x.xx.jar:* Contains demos and performance stress tests.

IRIS does not need any configuration to be runnable, the predefined parameters allow for out of the box reasoning. However, a dedicated configuration object can be used to configure the reasoner. For instance, the configuration object allows to define which evaluation strategy to use or how to deal with *equality in rule heads*.

4. Software Description

The implementation of the prototype reasoner for WSML-Flight 2.0 and WSML-Rule 2.0 follows the guidelines specified in *D3.2.1 Framework and APIs for integrated reasoning support*. The reasoner uses the Java programming language and is integrated in the reasoning framework *WSML2Reasoner.* The reasoner can be used with various underlying Datalog reasoning engines. In order to fit into the existing reasoner framework, the reasoner provides implementations for the following interfaces:

- *DatalogBasedWSMLReasoner:* An implementation of this interface takes care of axiomatization, normalization and generation of Datalog rules of WSML expressions. The Datalog rules are represented by a generic object model. These rules are then passed on to an external reasoning engine represented by a concrete implementation of a *DatalogReasonerFacade*.
- *DatalogReasonerFacade:* An implementation of this interface converts the generic Datalog object model into the representation required by the underlying Datalog reasoning engine. The prototype reasoner provides such facades for the following Datalog reasoners: *IRIS, Kaon2, Mins and XBS*.

WSML-Flight 2.0, respectively WSML-Rule 2.0, differ from WSML-Flight 1.0, respectively WSML-Rule 1.0, by supporting the new features inherited from WSML-Core 2.0. In particular, WSML-Flight 2.0 and WSML-Rule 2.0 now support *instance equivalence* (or in other words *equality in rule heads)*, which allows the declaration that different instance identifiers (IRIs) refer to the same object. Additionally, WSML-Flight 2.0 and WSML-Rule 2.0 have been aligned with the emerging Rule Interchange Format (RIF) standards to support the missing built-in data types, functions and predicates[6] in order to provide better interoperability.

The Datalog reasoner *IRIS,* which is also written in the Java programming language, has been modified in order to support the new features introduced and required by WSML-Flight 2.0 and WSML-Rule 2.0. This is particularly important, since both WSML-Flight 2.0 and WSML-Rule 2.0 are translated to Datalog programs and therefore rely on the Datalog reasoner to cover the required features. Note that *instance equivalence* is inherited from WSML-Core 2.0, therefore, the corresponding changes done to the Datalog reasoner are described in D3.2.2.

4.1 RIF data types and built-in predicates

The Rule Interchange Format (RIF) is a W3C working group[7] that develops standards for exchanging rules in the context of modern rule systems and the World Wide Web. RIF enables the semantic and syntactic description of rule systems, which can be further used to exchange axiomatic knowledge between other systems. RIF includes a framework for defining logic dialects, several concrete dialects, data type definitions and built-in predicates.

Both WSML-Flight 2.0 and WSML-Rule 2.0 are translated to Datalog programs in order to perform reasoning. Consequently, the Datalog reasoner *IRIS* has been updated to support a variety of RIF built-in data types, predicates and functions that have been identified as being relevant for WSML [2]. For the built-in data types, predicates and functions, *IRIS* provides new constructors respectively predicate names to "instantiate" them.

[6] RIF Datatypes and Built-Ins, http://www.w3.org/2005/rules/wiki/DTB [28.08.2009]

[7] RIF working group, http://www.w3.org/2005/rules/wiki/RIF_Working_Group [28.08.2009]

The new supported data types are:

- *rdf:text:* An internationalized text consisting of a string value and a language tag indicating its spoken language, e.g. "Family Guy@en" denoting an English text, "Padre de familia@es" denoting a Spanish text.
- *rdf:XMLLiteral:* Represents any valid XML fragment, e.g. "<tag>text</tag>".
- *xs:yearMonthDuration:* Derived from xs:duration by restricting its values to contain only the year and month components.
- *xs:dayTimeDuration:* Derived from xs:duration by restricting its values to contain only the day, hours, minute, seconds components.

The new supported built-in functions and predicates are:

- *Data type conversion functions:* Various functions to convert from one data type to another.
- *Data type guarding predicates:* Predicates to check if a specified term is of the specified data type or not.
- *String functions:* Various predicates representing functions to manipulate strings, such as functions to compare or concatenate strings.
- *String predicates:* Various predicates to check if a specified strings fulfills a specified requirement, such as to check if a string ends with a given string or to check if a given string contains a given string.
- *Date, time and duration functions:* Various functions to extract elements from the complex data types date, time and duration.

The WSML 2.0 reasoning framework *WSML2Reasoner*, particularly the corresponding *IRIS* facade, has been updated to support the new RIF features described above. The transformation from Datalog programs to the *IRIS* object model has been aligned to the new required features provided by this specific Datalog reasoner. Particularly, the *WSML2Reasoner* correctly instantiates the new components representing the RIF data types, predicates and functions. For example, the WSML function `wsml#stringCompare` is mapped to the RIF function `func:compare` by creating the corresponding element in the *IRIS* object model. Note that the conversion from WSML to Datalog programs has not been effected.

5. Conclusions

In this deliverable, we presented a first prototype implementation of a reasoner for the rule-based WSML language variants WSML-Flight 2.0 and WSML-Rule 2.0. The prototype reasoner is able to use various underlying Datalog reasoning engines, such as *IRIS, Kaon2* or *XBS*. Due to added expressivity in WSML-Core 2.0, WSML-Flight 2.0 and WSML-Rule 2.0 the underlying Datalog reasoner needs to support additional features, particularly, *instance equivalence* (or in other words equality in rule heads) and the RIF built-in data types, predicates and functions. Therefore, the Datalog reasoner *IRIS* has been updated in order to support the mentioned features.

The prototype reasoner is ready for use and can be downloaded from the *WSML2Reasoner* project website. The Datalog reasoner *IRIS* can be downloaded separately from the *IRIS* project website. For both projects, there are also nightly-builds available for download.

6. References

[1] B. Bishop, F. Fischer, P. Hitzler, M. Krötzsch, S. Rudolph, Y. Trimponias, G. Unel: *Defining the features of the WSML-DL v2.0 language*, SOA4ALL deliverable D3.1.3.

[2] F. Fischer, B. Bishop: *Defining the features of the WSML-Core v2.0 language*, SOA4ALL deliverable D3.1.2.

[3] F. Fischer: *Defining the features of the WSML-Rule v2.0 language*, SOA4ALL deliverable D3.1.4.

7. Appendix

7.1 Example Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
namespace { _"http://www.example.org/example#",
      wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }

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 wsml#stringEquals(?name, ?nomen).
            p(?x) :- ?x memberOf C2.
```