



DIP

Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP1: Reasoning

D1.9

D1.9 WSML-Flight Reasoner

Stephan Grimm, Gábor Nagypál

June 27, 2006



EXECUTIVE SUMMARY

Deliverable D1.9 is a prototype of a reasoning engine for the WSML-Flight language variant. It allows for inferencing according to the semantics of WSML-Flight defined in deliverable D1.7. This document serves as a fact sheet for the WSML-Flight Reasoner implementation based on KAON2 as its underlying inference engine. It outlines the basic principles of the reasoning functionality and describes how to use the D1.9 reasoning tool component.

Deliverable D1.9 is of particular interest for developers of components in Workpackages WP2, WP4 and WP5 who want to include reasoning support into their systems. It is also interesting for the use cases for users who want to directly investigate the benefits of reasoning with domain models.

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. 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

IST Project Number	FP6 – 507483	Acronym	DIP
Full Title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://dip.semanticweb.org/		
Document URL			
EU Project Officer	Kai Tullius		

Deliverable	Number	1.9	Title	D1.9 WSML-Flight Reasoner
Work Package	Number	1	Title	Reasoning

Date of Delivery	Contractual	30-Jun-2006	Actual	30-Jun-2006
Status	version 1.0		final	<input checked="" type="checkbox"/>
Nature	prototype <input checked="" type="checkbox"/> report <input type="checkbox"/> dissemination <input type="checkbox"/> ontology <input type="checkbox"/>			
Dissemination Level	public <input type="checkbox"/> consortium <input checked="" type="checkbox"/>			








Authors (Partner)	Stephan Grimm (FZI), Gábor Nagypál (FZI)			
Resp. Author	Stephan Grimm, Gábor Nagypál		E-mail	stephan.grimm@fzi.de, gabor.nagypal@fzi.de
	Partner	FZI	Phone	+49 721/9654714 +49 721/9654816

Abstract (for dissemination)	
Keywords	Automated Reasoning, WSML, Rule-Based Inferencing

Version Log			
Issue Date	Rev No.	Author	Change
06-06-2006	1	Stephan Grimm	v 1.0

Reviewers				
	Livia Prediou		E-mail	livia.predoiu@deri.org
	Partner	UIBK	Phone	++43 (0) 512 507 6111
	Marc Richardson		E-mail	marc.richardson@bt.com
	Partner	BT	Phone	+44 (0) 1473 609589

PROJECT CONSORTIUM INFORMATION

Partner	Acronym	Contact
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 Rowlett, 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		Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG		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.		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		Alexander Wahler Hanival Internet Services GmbH Kirchengasse 13/1a A-1070 Wien E-mail: wahler@hanival.net Tel.: +43 131 95843 11
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		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		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.		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




<p>Unicorn Solution Ltd.</p>	<p>Unicorn </p>	<p>Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111</p>
<p>Vrije Universiteit Brussel</p>	<p>VUB   Vrije Universiteit Brussel</p>	<p>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</p>

TABLE OF CONTENTS

1	INTRODUCTION	1
2	IMPLEMENTED FUNCTIONALITY	2
2.1	Reasoning with WSML-Flight Ontologies	2
2.1.1	Ontology Transformations	2
2.1.2	WSML Reasoning by Datalog Queries	3
2.2	Debugging Support	4
3	API	6
4	INSTALLATION INSTRUCTIONS	7
4.1	System Requirements	7
4.2	System Installation	7

1 INTRODUCTION

The Web Service Modeling Language (WSML) [2] comes in different variants based on the two different knowledge representation paradigms of concept description languages and logic programming. Within Workpackage 1, we develop reasoner components for various WMSL variants, basing on KAON2 as the underlying inference engine delivered with D1.2 in a first version.

KAON2¹ is a hybrid reasoning systems that combines the two knowledge representation paradigms by allowing Datalog-style rules to interact with structural description logics knowledge bases. It supports the SHIQ(D) [5] description logic and disjunctive Datalog [1] programs. The WSML reasoners developed within Workpackage 1 can be technically seen as wrappers around the KAON2 system, which serve as an adapter between KAON2's inferencing capabilities and the varoius WSML variants.

Deliverable D1.9 provides a reasoner prototype for the WSML-Flight language variant, referred to as “WSML-Flight reasoner component”. The component allows for performing the reasoning tasks of ontology consistency, entailment and instance retrieval (query answering), as described in [3]. It has a connection to the WSMO4J² API which it uses to programmatically handle ontology objects in memory and for communication and data exchange with other components and users.

This document serves as a fact sheet and short documentation of the WMSL-Flight reasoner component. The deliverable D1.9 consists of a zip file with the following content:

- the program library for the WSML-Filght reasoner component
- the libraries which the reasoner component depends on, namely those for WSMO4J and KAON2
- the generated JavaDoc HTML documentation for the WSML-Flight reasoner component
- an example of how to use the reasoner component, which includes a WSML ontology taken from a WP8 telecommunication use case scenario together with a test-run that performs some reasoning on this ontology
- this fact sheet document

¹<http://kaon2.semanticweb.org/>

²<http://wsmo4j.sourceforge.net/>

2 IMPLEMENTED FUNCTIONALITY

In this section we give an overview on the implemented functionality of the WSML-Flight reasoner component, which is in detail described in [4]. It allows a user to perform the following reasoning tasks, defined in [3].

- *ontology consistency* – to check whether an ontology is consistent in itself and does not contain contradictory information
- *ground entailment* – to check whether a ground fact¹ is a logical consequence of the knowledge in an ontology
- *instance retrieval* – to query for all tuples of entities that instantiate the free variables in a given query expression. (Instance retrieval is on ground entailment and is not listed separately in [3].)

2.1 Reasoning with WSML-Flight Ontologies

The semantics of rule-based WSML is defined via a mapping to Datalog [1] with (in)equality and integrity constraints, as described in [3]. To make use of KAON2 as a rule engine, the system performs various syntactical transformations to convert an original ontology in WSML syntax into a semantically equivalent Datalog program. The WSML reasoning tasks of knowledge base satisfiability and instance retrieval are then realized by means of Datalog querying via calls to an underlying Datalog inference engine that is fed with the rules contained in this program.

2.1.1 Ontology Transformations

The transformation of a WSML ontology to Datalog rules forms a pipeline of single transformation steps which are subsequently applied, starting from the original ontology.

Axiomatization. In a first step, all conceptual syntax elements, such as concept and attribute definitions or cardinality and type constraints, are converted into appropriate axioms specified by logical expressions. The result is a set of WSML logical expressions semantically equivalent to the original ontology.

Normalization. In a next step, various normalizations are applied to reduce the complexity of the logical expressions according to [3, Section 8.2]. This brings the expressions closer to the simple syntactic form of literals in Datalog rules. The reduction includes conversion to negation and disjunctive normal forms as well as decomposition of complex WSML molecules. After this normalization has been applied, the resulting WSML logical expressions have the form of logic programming rules with no deep nesting of logical connectives.

Lloyd-Topor Transformation. The Lloyd-Topor transformations [6] are applied to flatten the complex WSML logical expressions, producing rules of a simpler form. The resulting WSML expressions have the form of proper Datalog rules with a single head and conjunctive (possibly negated) body literals.

¹A ground fact is an expression that does not contain any free variables.

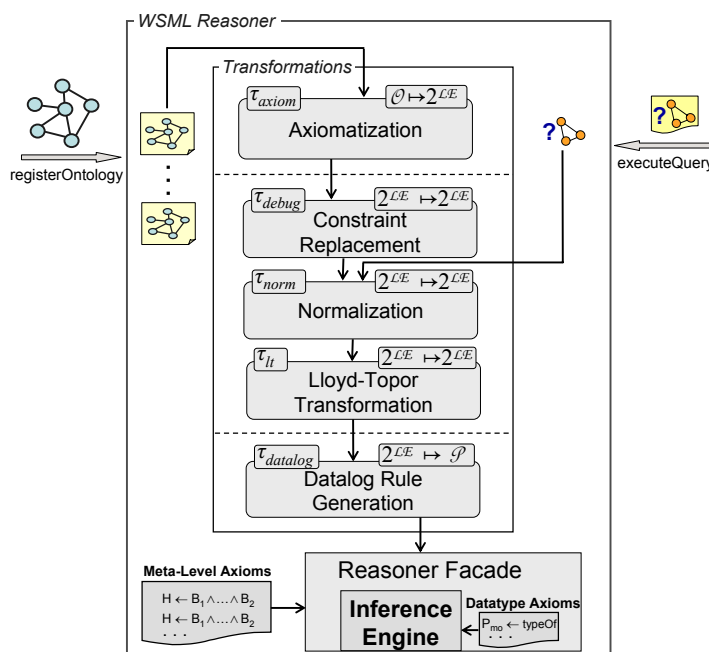


Figure 2.1: Internal framework architecture.

Datalog Rule Generation. In a final step, the WSML logical expressions are turned into generic datalog rules. Rule-style language constructs, such as rules, facts, constraints, conjunction and (default) negation, are mapped to the respective Datalog elements. All remaining WSML-specific language constructs, such as `subConceptOf` or `ofType`, are replaced by special meta-level predicates for which the semantics of the respective language construct is encoded in meta-level axioms as described in [4].

The different transformation steps are depicted in Figure 2.1, which illustrates the highly modular internal architecture of the WSML-Flight reasoner component. They are implemented in such way that the various transformation steps can easily be reused for other tasks, such as validation, and recombined to yield a transformation pipeline with different behaviour. The debugging features described in Section 2.2, for example, can be easily switched on and off by including (or excluding) the constraint replacement module in (/from) the loop.

2.1.2 WSML Reasoning by Datalog Queries

The formerly described transformation pipeline, denoted by τ , transforms a WSML-Flight ontology O into a semantically equivalent Datalog program $P_O = \tau(O)$, when interpreted with respect to meta-level axioms, as discussed in [4]. The different WSML reasoning tasks are then realized by performing Datalog queries on P_O . Posing a query $Q(\vec{x})$ to a Datalog program P is denoted by

$$(P, ? - Q(\vec{x}))$$

and yields a set of tuples that instantiate the vector \vec{x} of variables in the query.

Ontology Consistency – The task of checking a WMSL ontology for consistency is done by querying for the empty clause, as expressed by the following equivalence.

$$O \text{ is satisfiable} \Leftrightarrow (P_O, ? - \square) = \emptyset$$

If the resulting set is empty then the empty clause could not be derived from the program and the original ontology is satisfiable, otherwise it is not.

Entailment – The reasoning task of entailment of ground facts by a WMSL ontology can be done by using queries that contain no variables, as expressed in the following equivalence.

$$O \models \phi \Leftrightarrow (P_O, ? - \tau(\phi)) \neq \emptyset$$

The WMSL ground fact ϕ is transformed to Datalog with a transformation similar to the one that apply to the ontology, and is evaluated together with the Datalog program P_O . If the resulting set is non-empty then ϕ is entailed by the original ontology, otherwise it is not. Since in WMSL-Flight entailment is in general restricted to ground facts, this reasoning task is also called instance checking.

Retrieval – Similarly, instance retrieval can be performed by posing queries that contain variables to the Datalog program P_O , as expressed in the following equivalence.

$$\{\vec{x} : O \models Q(\vec{x})\} = (P_O, ? - \tau(Q(\vec{x})))$$

The query $Q(\vec{x})$, formulated as a WMSL logical expression with free variables \vec{x} , is transformed to Datalog and evaluated together with the program P_O . The resulting set contains all tuples \vec{x} for which an instantiation of the query expression is entailed by the original ontology.

2.2 Debugging Support

During the process of ontology development, an ontology engineer can easily construct an erroneous model containing contradictory information. The WMSL-Flight reasoner prototype supports debugging features that report inconsistencies to the ontology engineer, together with some details about the ontological elements that cause the inconsistency.

In rule-based WMSL, the source for erroneous modelling are primarily constraints, together with a violating situation of concrete instances related via attributes.

For the various types of constraint violations, the information reported by the system is different from case to case.

Attribute Type Violation – An attribute type constraint of the form $C[a \text{ ofType } T]$ is violated whenever an instance of the concept C has value V for the attribute a , and it cannot be inferred that V belongs to the type T . Here, T can be either a concept or a datatype, while V is then an instance or a data value, accordingly. In such a situation, the system reports the instance I in the attribute value V that caused the constraint violation, together with the attribute a and the expected type T which the value V failed to adhere to.

Minimum Cardinality Violation – A minimum cardinality constraint of the form `concept C a (n *)`, is violated whenever the number of distinguished values of the attribute a for some instance I of the concept C is less than the specified cardinality n . In such a situation, the system reports the the instance I that failed to have a sufficient number of attribute values, together with the actual attribute a .

Maximum Cardinality Violation – A maximum cardinality constraint of the form `concept C a (0 n)`, is violated whenever the number of distinguished values of the attribute a for some instance I of the concept C exceeds the specified cardinality n . Again, here the system reports the instance I for which the number of attribute values was exceeded, together with the actual attribute a .

User-Defined Constraint Violation – Not only built-in WSMML constraints, but also user-defined constraints, contained in an axiom definition of the form `axiom AxID definedBy :- B`, can be violated. In this case, the information which helps an ontology engineer to repair an erroneous situation is dependent on the arbitrarily complex body B and cannot be determined in advance. Here, the system at least identifies the violated constraint by reporting the identifier $AxID$ of the axiom.

The error reporting is initiated at the point in time when ontologies are registered with the reasoner component. The API offers functionality to ask for the debugging information about all violated constraints.

3 API

The WSML-Flight reasoner component is part of the WSMX architecture and serves as a reasoner to be used by other WSMX components via an API. However, it can also be used as a stand alone module. The full component API is documented in the generated JavaDoc HTML documents in the folder `doc` within the deployment package. Here, we outline the main entry points for using the component's functionality.

Before reasoning can be performed, the respective ontology has to be registered with the reasoner component. Several ontologies can be registered at the same time, and registering is done by the following method.

```
void registerOntology(Ontology ontology) throws InconsistencyException
```

If the ontology to be registered is inconsistent, the invocation of this method yields an exception object that can be used to query for the problems that cause the inconsistency, as described in Section 2.2. Notice that the reasoner component only performs semantic checks that are related to reasoning and to the WSML-Flight semantics as defined in [3] – rather than performing syntactic validation, it expects the ontology to be in valid WSML-Flight syntax.

Once registered with the reasoner, querying can be performed on the ontology. The following method asks for whether a ground logical expression is entailed by the ontology.

```
boolean entails(IRI ontologyID, LogicalExpression expression)
```

Besides the (variable-free) query expression, the according registered ontology has to be identified with respect to which the query is asked.

Besides asking for whether some ground fact holds, the querying API can be used for retrieving tuples of instances that represent answers to query expressions with free variables. (Sometimes this task is also called query answering and the tuples are called variable bindings.) The following method realizes instance retrieval with a registered ontology and a query expression.

```
Set<Map<Variable,Term>> executeQuery(IRI ontologyID, LogicalExpression query)
```

The set of answer tuples is returned as a map of pairs in any which a variable is bound to a term. The term can be an identifiable entity¹ or a data value. To give an example for querying, the result for the query

```
?x memberOf Student and ?x[enrolledIn hasValue ?y] and ?y memberOf CSCourse
```

to some ontology about students in a university domain could yield the following result.

```
?x = Peter    ?y = CS528
```

```
?x = Peter    ?y = CS439
```

```
?x = Susan    ?y = CS314
```

This result would reflect that there are two students enrolled in computer science courses, Peter and Susan, and for both the result set contains tuples with the variables `?x` and `?y` bound to the person and the course, respectively.

¹Since WSML-Flight allows for metamodeling features, the result of the retrieval can be not only instances but also concepts and attributes. In this way, also schema-like queries can be asked.

4 INSTALLATION INSTRUCTIONS

The content of deliverable D1.9 is available at the DIP BSCW server in form of a deployment package under the path “Work Packages / WP 01 / Deliverables / D1.9 WSML-Flight Reasoner /”¹.

4.1 System Requirements

The WSML reasoner prototype has been written in Java 5². The system should work on any computer under any operating system capable of running Java 5.

The WSML-Flight reasoner component is dependent on the following technologies, which are also (partly) developed within the DIP project:

- **KAON2** – The hybrid reasoning system KAON2 serves as the underlying inference engine of the WSML-Flight reasoner. It is contained in the file `kaon2-2005-12-08.jar`.
- **WSMO4J** – The WSMO API WSMO4J is used within the WSML-Flight reasoner for the programmatic handling of ontologies. It is contained in the files `wsmo4j-20060308.jar` and `wsmlparser-20060210.jar`, and is used for parsing a WSML ontology and for navigating its data model.

All the jar-files required to run the WSML-Flight reasoner are contained in the deployment package.

4.2 System Installation

The core part of the WSML-Flight reasoner prototype is delivered with the Java class libraries of the reasoner component contained in the file `wsml_flight_reasoner.jar`. It is dependent on the WSMO4J API and the KAON2 inference engine, which come in separate libraries as described in Section 4.1.

To use the WSML-Flight reasoner component within other Java projects, its jar-file has to be included in the project’s classpath, together with the libraries it depends on. The reasoner component is used via the interface `WSMLFlightReasoner`. An example of how to instantiate this interface and how to invoke its methods can be looked up in the source code of the class `example.ReasonerUsage`.

This class also contains a test run that invokes some of the reasoning functionality on the ontology in the file `bundles.wsml`. To execute the test run, start the batch-file `runReasonerUsage.bat`.

¹<https://bscw.dip.deri.ie/bscw/bscw.cgi/0/53560>

²<http://www.java.sun.com>

REFERENCES

- [1] Michael Dahr. *Deductive Databases: Theory and Applications*. International Thomson Publishing, December 1996.
- [2] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The Web Service Modeling Language WSML: An Overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC)*, 2006.
- [3] Jos de Bruin. The Web Service Modeling Language (WSML) Specification. Technical report, Digital Enterprise Research Institute (DERI), February 2005. <http://www.wsmo.org/TR/d16/>.
- [4] Stephan Grimm, Holger Lausen, Uwe Keller, and Gábor Nagypál. A reasoning framework for rule-based wsml. Technical report, FZI Research Center for Information Technologies at the University of Karlsruhe, June 2006. http://www.fzi.de/downloads/wim/sgr/WSML_Reasoning.pdf.
- [5] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [6] John Lloyd and Rodney Topor. Making Prolog More Expressive. *Journal of Logic Programming*, 3:225–240, 1984.