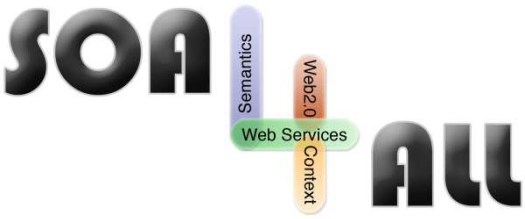

Project Number: **215219**
Project Acronym: **SOA4All**
Project Title: **Service Oriented Architectures for All**
Instrument: **Integrated Project**
Thematic Priority: **Information and Communication Technologies**

D3.1.4 Defining the features of the WSML-Rule v2.0 language

Activity N:	Activity 2 - Core Research and Development
Work Package:	WP3 - Service Annotation and Reasoning
Due Date:	M12
Submission Date:	20/02/2009
Start Date of Project:	01/03/2006
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	UIBK
Revision:	1.0
Author(s):	Ioan Toma UIBK Barry Bishop UIBK Florian Fischer UIBK
Reviewers(s):	Dumitru Roman Rafael Cabero ATOS

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2009-01-21	Initial TOC	Barry Bishop
0.2	2009-01-28	Updated Section 3.3 and 3.4	Ioan Toma
0.3	2009-01-31	Updated Section 1, 2 and 3 based on Florian's input	Ioan Toma, Florian Fischer
0.4	2009-02-03	Updated Section 3.1.2, 3.1.3 and 3.2.3	Barry Bishop
0.5	2009-02-03	Updated Section 3.5	Florian Fischer
0.6	2009-02-04	Updates Section 3.1.1; Fix the references	Ioan Toma
0.7	2009-02-04	Completed section 3	Barry Bishop
1.0	2009-02-23	Implement reviewers comments	Ioan Toma
Final	2009-03-09	Overall format and quality review	Malena Donato

Table of Contents

List of Figures

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
WP	Work Package
HLDD	High Level Design Document
WSML	Web Service Modelling Language
WSMO	Web Service Modelling Ontology
LP	Logic Programming
DL	Description Logic

Executive summary

In order to automate tasks such as location 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, mediators and goals.

This deliverable, along with others, defines an updated version of the WSML language stack, in order to bring it in line with the scalability requirements of reasoning in SOA4All and realign it with new research results and other standards (i.e. W3C Rule Interchange Format - RIF). Thus, this document describes WSML-Flight 2.0 and WSML-Rule 2.0, the Logic Programming variants of the WSML language. It covers limitations placed upon its high-level conceptual syntax, as well as upon the expressivity of its logical expression syntax.

1. Introduction

SOA4All's aim is to facilitate a web where billions of parties are exposing and consuming services via advanced Web technology. The outcome of the project will be a framework and infrastructure "that integrates four complimentary and revolutionary technical advances into a coherent and domain independent service delivery platform":

- Web principles and technology as the underlying infrastructure for the integration of services at a worldwide scale.
- Web 2.0 as a means to structure human-machine cooperation in an efficient and cost-effective manner.
- Semantic Web technology as a means to abstract from syntax to semantics as required for meaningful service discovery.
- Context management as a way to process in a machine understandable way user needs that facilitates the customization of existing services for the needs of users.

Thus, one basic technological building block is Semantic Web technology, which abstracts from pure syntax to semantics. Ontologies are used as a semantic data model, by which means services gain machine-understandable annotations. This information makes the development of high quality techniques for automated selection, construction, etc. possible. Furthermore, precise formal models allow for the expression of context-specific rules and constraints, which can be taken into account during the inference process. The basic building blocks for this are formal languages for describing resources in a clear and unambiguous way.

The Web Service Modelling Language WSML [1] is such a formal language for the specification of ontologies and different aspects of Web services, based on the conceptual model of WSMO [2]. Several different WSML language variants exist, which are founded upon different logical formalisms. The main formalisms exploited for this purpose are Description Logics [3], Logic Programming [4], and First-Order Logic [5]. Furthermore, WSML has been influenced by F-Logic [6] and frame-based representation systems.

This deliverable introduces a revised version of the WSML-Flight and WSML-Rule variants of the WSML language family, in order to align them with the recent progress in W3C Rule Interchange Format (RIF) [19] and as well with the updated version of WSML-Core variant. This deliverable belongs to a set of conceptually related M12 deliverables, namely:

- D3.1.1 Defining the features of the WSML-Quark language
- D3.1.2 Defining the features of the WSML-Core v2.0 language
- D3.1.3 Defining the features of the WSML-DL v2.0 language
- **D3.1.4 Defining the features of the WSML-Rule v2.0 language**

These four deliverables form the foundation for a redefinition of WSML that brings it in line with the tractability requirements of SOA4ALL, which envisions "billions of parties exposing services". Working with and reasoning over the vast datasets that are implied by this vision poses a significant scalability challenge.

1.1 Purpose and Scope

1.1.1 Audience

This document is intended as a reference of the features of the WSML language. 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.

Inside the consortium, this mainly applies to partners involved in technical work packages within Activity cluster A2 – “Core R&D Activities”. For outside parties beyond the consortium it can serve as an introduction to WSML.

1.1.2 Scope

The main purpose of this deliverable is to present the features of the reworked Logic Programming variants of the WSML family of languages, namely WSML-Flight and WSML-Rule. We focus on describing the changes made in regard to the existing WSML-Flight and WSML-Rule specification. These changes have been motivated by recent work in the development of the Rule Interchange Format (RIF) and the need for alignment with the updated version of the WSML-Core language specification.

We describe the modelling elements in WSML-Flight 2.0 and WSML-Rule 2.0, restrictions imposed on the languages, and a motivation for them. Beyond the definition of the conceptual and logical expression syntax of the languages, we also outline the steps involved in a practical implementation and explain the relation with the other language variants within the WSML stack and their respective layering.

1.2 Structure of the document

The remainder of this deliverable is structured as follows: Section 2 clarifies the relation of this document and the WSML language in relation to the SOA4All project and other deliverables. Section 3 defines the WSML-Flight 2.0 and WSML-Rule 2.0 languages by describing the individual language elements and pointing out the particular restrictions placed on them for each language variant. It then proceeds to outline the algorithmization of WSML-Flight 2.0 and WSML-Rule 2.0 on rule-based reasoners. Finally, section 3.6 clarifies the relation between these new versions of the two language variants and the other languages in the WSML specification. Section 3.7 concludes this deliverable and points out the next steps for future work.

2. Technical deliverable remarks

2.1 Deliverable relation with the architecture of the project

The work conducted towards a reworked WSML language stack in WP3 conceptually belongs to the Base Layer of the SOA4All architecture (see ***Figure 1***). In the SOA4All architecture, different elements are distributed in three different layers according to their functional dependencies on each other.

The Base Layer contains elements such as (1) formal languages and ontologies, (2) Reasoner and (3) Semantic spaces as the publication and communication element of the infrastructure.

The Web Enabled Service platform (the second layer), consists of (4) Service Ranking and Selection, (5) Service Location, (6) Service Adaptation and Service, (7) Service Grounding, (8) Service Delivery, (9) Service Monitoring and Management and (10) Service Context.

Finally, in the User Layer are components such as (11) Service Modelling, (12) Service Provisioning and (13) Service Consumption.

The “Semantic Service Bus” ties all these components together and serves as infrastructural backbone. In ***Figure 1*** the Semantic Service Bus is indicated by the outer “envelope” around the other components and shows the possibility of being connected to other buses as an extension.

The changes to the WSML family have direct consequences for the reasoning components to be developed in WP3, which directly process these formal languages.

Furthermore, any component from the second layer, which operates on (i) semantically annotated Web Services or (ii) relies on an ontology for other reasons will make use of WSML, at least indirectly. This most directly applies to WP5 for the purpose of service location, discovery and ranking, as well as to WP6 for the purpose of service composition.

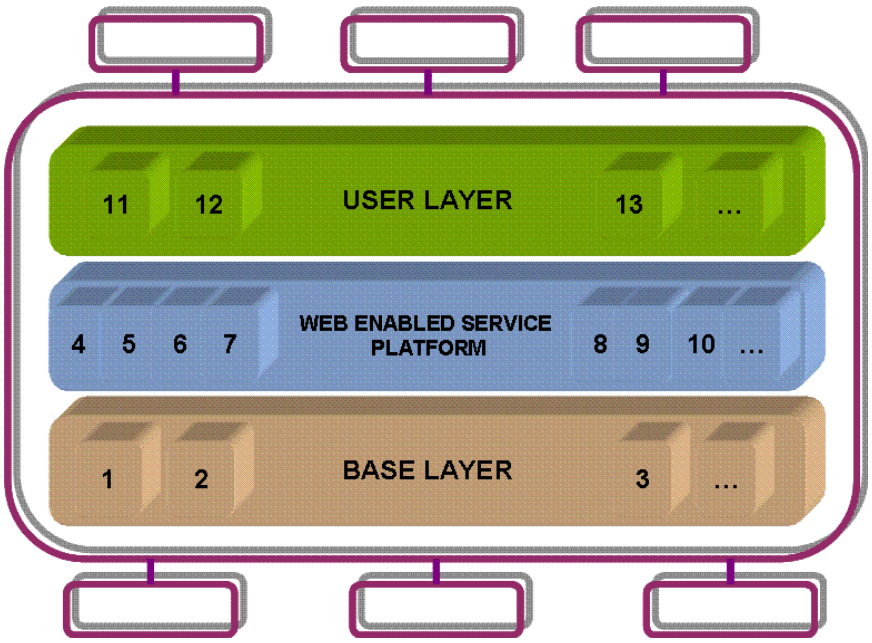

Figure 1 SOA4All Semantic Service Bus

2.2 Deliverable relation with the use-cases

This section clarifies the relation of the WSML language family with the ongoing use-case activities in SOA4All and points out direct applications of WSML as they are apparent at the time of the writing.

2.2.1 End-user Integrated Enterprise Service Delivery Platform

As the End-user Integrated Enterprise Service Delivery Platform case study will fully use service annotation and reasoning about such annotations, it will also make direct use of WSML and the reasoner components associated with it. The End-user Integrated Enterprise Service Delivery Platform use-case is specified as part of the SOA4All deliverables D7.1, D7.2 and D7.3.

This use-case aims for an open, dynamic and lightweight service platform in place of heavyweight existing solutions, which are hard to set up and maintain due their complexity. An envisioned outcome (among several) from the end user's perspective is a tool to compose processes[1] from services and reuse services in a visual tool without requiring an in-depth technical background. Apart from the requirements that stem from **service composition** an envisioned outcome of the use-case is to provide support for publishing, finding and reusing existing processes. In order to find processes in repositories **search** mechanisms based on semantic descriptions (and hence WSML descriptions) are required.

2.2.2 W21C BT Infrastructure

This use-case will create a semantically enhanced and expanded version of BT's Web21c platform [15], which will result in a framework for the delivery of service, both by BT itself and third parties. This requires in-depth technical knowledge and the aim of the case study is to simplify the process of **discovering**, **integrating**, using and sharing BTs capabilities on this platform. Thus, in the BT W21C case study the focus is shifted slightly by using **service location** technologies to discover capabilities within the BT Web21c infrastructure.

The specification of the W21C BT infrastructure use-case is provided as part of SOA4All deliverables D8.1, D8.2 and D8.3

Reasoning with formal service semantics forms the basis for **composition** tools that will enhance and aide the creation of more complex services. Furthermore, unambiguous descriptions of services facilitate the **selection** of services for the end user. WSML will thus be used directly in this work package.

2.2.3 C2C Service eCommerce

One of the focuses of this use-case in WP9 is to investigate the impact and sustainability of future C2C eCommerce applications based on services and to enable eCommerce as a common distribution channel for end-users by means of SOA4All. In this scenario, non-technical end-users can make use of existing services and combine them to build eCommerce applications in order to market and sell their own products. The C2C Service eCommerce use-case is specified as part of the SOA4All deliverables D91, D92 and D9.3.

This use-case again entails several tasks that are based on annotation and (WSML)

[1] In the loose sense of a "business process" composed from various subtasks (services) in order to accomplish a specific goal.

reasoning, among them easy **composition** of services**, service location, ranking** and **selection** in the case of similar services. In this sense the scenario demonstrates almost all parts of the SOA4ALL concept including service discovery, integration, etc. and as such heavily relies on the formal languages work conducted in WP3.

3. WSML-Flight and WSML-Rule Language Definition

This section contains the specification of the updated rule-based WSML languages, namely WSML-Flight2.0 and WSML-Rule2.0. We start with a short motivation section in which we point out why rule languages are important, especially in the context of a Web of pages and services (Section 3.1.1), we describe the differences between WSML-Flight and WSML-Rule (Section 3.1.2) and finally we motivate the need of updating WSML-Flight and WMSL-Rule specification (Section 3.1.3). One of the most important requirements that has motivated the update of the two rule-based languages namely the alignment with the Rule Interchange Format (RIF) is discussed in Section 3.2. The updated specifications of the two languages are provided in Section 3.3 for WSML-Flight v2.0 and Section 3.4 for WSML-Rule v2.0. Details about how the updates in the languages do influence the reasoning algorithms are discussed in Section 3.5. Finally, Section 3.6 describes the relation with the other languages from the WSML family of languages and Section 3.7 concludes this section and identifies possible future research and development directions.

3.1 Motivation

3.1.1 Rule languages

From expert systems to deductive databases, the theory and practice of automating inference based on symbolic knowledge representations has had a rich history and continues to be a key technology driver. Representing knowledge using rules is one of the most important and widely used approaches for knowledge representation. Rule-languages and rule-based systems have played seminal roles in the history of computer science and the evolution of information technology.

Rules are a convenient and intuitive way to add axiomatic information to knowledge-bases. In comparison to other programming approaches, such as conventional programming and scripting languages, rules are more easily understood. This makes it easier for both programmers and non-programmers to specify, modify and merge rules. As a result, increasing attention has been given to rule-based systems from both academia and industry. Nowadays a large amount of commercial rule-based systems are available on the market. SQL (relational databases), Prolog, Production rules (JESS[2], CLIPS[3], etc.) are some of rule-based system categories that have been in growing commercial deployment in the last two decades. Most of these systems are using Logic Programming as the underlying theory. In particular the Datalog (Horn) subset of Logic Programming has proven to have many practical applications (e.g. SQL, relation algebra) due to its computational tractability.

Recently there has been a rebirth of interest in rule-based languages, especially in the context of the Semantic Web and Semantic Web services. In this context, several proposals for rule languages have been developed including: RuleML[4], WSML[1], SWRL[5], and recently as a standardization effort, the Rule Interchange Format (RIF)[19].

The WSML language(s), and in particular the rule-based variants WSML-Flight and WSML-Rule, provide the means to specify aspects of ontologies and semantic Web services using an expressive rule syntax. In this document the WSML-Flight and WSML-Rule languages are aligned with the latest developments in rule-based languages, namely the RIF

[2] http://herzberg.ca.sandia.gov/jess/

[3] http://www.siliconvalleyone.com/clips.htm

[4] http://www.ruleml.org

[5] http://www.w3.org/Submission/SWRL/

standardization effort.

3.1.2 The distinction between WSML-Flight and WSML-Rule

Both WSML-Rule and WSML-Flight are rule-based languages with semantics based ultimately on Datalog.

WSML-Flight is an extension of WSML-Core with such features as meta-modeling, constraints, n-ary relations, cardinality constraints and non-monotonic negation. WSML-Flight is based on a logic programming variant of F-Logic [6] and is semantically equivalent to Datalog with inequality and (locally) stratified negation. WSML-Flight is both syntactically and semantically completely layered on top of WSML-Core and allows variables in place of concepts, instances and attribute identifiers.

WSML-Rule is an extension of WSML-Flight and captures several extensions such as the use of function symbols (constructed terms), unsafe rules (i.e. variables that occur in the head of a rule are not required to occur in the body of the rule), and does not require stratification of negation.

Whereas WSML-Rule is the most flexible and expressive variant of the two, the unrestricted use of function symbols means that it is not always decidable – the minimal model may be infinite due to the recursive instantiation of increasingly more complex constructed ground terms.

WSML-Flight, which does not allow the use of function symbols, is therefore a decidable subset of WSML-Rule.

3.1.3 Required changes to WSML-Flight and WSML-Rule

The changes introduced to WSML-Flight and WSML-Rule with this deliverable are twofold.

Firstly, the languages must layer with each other and with the new WSML-Core v2.0 [23], which introduces instance equivalence, otherwise known as ‘equality in rule heads’. This change allows for the declaration that different instance identifiers (IRIs) refer to the same object.

Secondly, both WSML-Flight and WSML-Rule should be as compatible as possible with the emerging Rule Interchange Format (RIF) standards. The most extensive changes here relate to the missing support for RIF built-in data-types, predicates and functions.

3.2 RIF and WSML

The Rule Interchange Format (RIF)[19] is a W3C working group that develops standards for exchanging rules in the context of modern rule systems and the World Wide Web. Such standards should capture today’s requirements to enable the sharing of information suited to machine processing and also be extensible, so that they can be adapted to evolving rule technology.

The working group has made significant progress towards a number of proposed standards that address both the syntactic and semantic descriptions of rules. This includes a framework for defining logic dialects, several concrete dialects, data-type definitions and built-in predicates.

3.2.1 The importance of RIF

Due to the increasing importance of rule exchange, RIF is expected to become a common standard for rule systems. Each system can be described by the RIF rule format(s) it is capable of understanding, either by defining its own logic dialect or by publishing details of its conformance with well-known or concrete dialects, e.g. the Basic Logic Dialect (RIF-BLD).

WSML-Rule and WSML-Flight are rule-based languages and as such the ability to incorporate or interact with standard rule representations improves their utility. In many scenarios it would be useful to be able to exchange axiomatic knowledge by either extracting rules from a WSML ontology for processing by another system or to inject external rules in to a WSML ontology.

The purpose of this analysis is to identify how WSML can be modified in order to make it more compatible with RIF in terms of what can be syntactically and semantically defined using the two languages.

3.2.2 RIF proposals that are relevant to WSML

The RIF working group provides several proposed standards that are relevant for WSML. These are:

- Data type and Built-ins – RIF-DTB [24]
- Basic Logic Dialect – RIF-BLD [25]
- Core – RIF-Core [26] (a common subset of RIF-BLD, RIF-PRD and RIF-DTB)

What follows is an analysis of these three proposed standards for the purpose of identifying what elements of RIF should be supported in WSML, such that we achieve the best compatibility.

3.2.2.1 RIF-DTB

This section discusses possible alignment of WSML with the RIF-DTB variant in terms of primitive data types, built-in functions and built-in predicates.

3.2.2.1.1 Datatypes

See: http://www.w3.org/2005/rules/wiki/DTB#Primitive_Datatypes

WSML supports all the basic XML schema data types and hence almost all the primitive RIF data types. The exceptions are: rdf:XMLLiteral (which was introduced in the 1.0 specification, although not yet implemented) and rdf:Text. These should be supported in WSML as follows.

rdf:XMLLiteral	Represents any valid XML fragment as a single data value. Such a data-type can be supported in WSML by enclosing in double quotes and providing a new shorthand constructor, e.g. _xmlliteral(“<SomeTag>Has this value</SomeTag>” The enclosed string must have each occurrence of the double quote escaped using the backslash character ‘\’, such that removing the backslashes yields a string in the lexical space of rdf:XMLLiteral. However, it might cause some significant overhead for any of the WSML parsers to validate values for conformance with XML. It is suggested that this validation be optional. A new shortcut constructor is suggested: _xmlLiteral("escaped XML content")
rdf:Text	Internationalised string values that contain a tag indicating their spoken language, e.g. "Padre de familia@es". Support in WSML can be achieved using a new shortcut constructor: _rdfText("Padre de familia", "es")
xs:yearMonthDuration	Derived from xs:duration by restricting its lexical representation to

	contain only the year and month components. New shortcut: _yearMonthDuration(2009, 02)
xs:dayTimeDuration	Derived from xs:duration by restricting its lexical representation to contain only the days, hours, minutes and seconds components. New shortcut: _dayTimeDuration(1, 10, 31, 15.5)

3.2.2.1.2 Built-in functions and predicates

See: http://www.w3.org/2005/rules/wiki/DTB#List_of_RIF_Built-in_Predicates_and_Functions

All the RIF-DTB built-in predicates and functions relate to the manipulating and testing of the standard data types. These built-ins are generally very useful and straightforward to implement, hence the set of WSML built-ins should be extended to cover those RIF built-ins not currently supported.

- WSML does not have the cast functions, that coerce a data-type value in to the representation of another data-type.
- WSML does not have the RIF guard and negative guard predicates for data types. WSML does have the wsml#member-of() predicate that can be used to achieve the same thing, albeit with negation for negative guards, but this is a little clumsy and requires the ontology designer to have an understanding of the WSML meta-model.
- WSML supports all the basic arithmetic functions, except modulus.
- WSML does not support some string functions: compare, concat, substring, upper-case, etc. These should be implemented with names consistent with existing string predicates: wsml#stringCompare, wsml#stringConcat, wsml#stringSubstring, wsml#stringToUpper, wsml#stringToLower
- WSML does not support string predicates: contains, starts-with, ends-with (but does support the others). These should be implemented with names: wsml#stringContains, wsml#string, wsml#stringStartsWith, wsml#stringEndsWith
- WSML does not support functions on Dates, Time and Durations, i.e. those functions for extracting elements from these complex data-types.
- Due to lack of support for rdf:XMLLiteral and rdf:text, the few functions and predicates for these types are also not supported at present.

Summary of new built-in predicates and functions with mapping from RIF symbol to WSML symbol:

WSML predicate/function	RIF predicate/function	Signature	Return type (functions)
wsml#to<Datatype> e.g. wsml#toDouble	xs:<datatype> xs:double	any any	<datatype> _double
wsml#isDatatype	pred:isLiteralOfType	any, IRI	
wsml#isNotDatatype	pred:isLiteralNotOfType	any, IRI	

wsml#hasDataType	pred:hasDatatype	any, IRI	
wsml#numericModulus	func:numeric-mod	numeric, numeric	numeric
wsml#stringCompare	func:compare	string, string	string
wsml#stringConcat	func:concat	any, any	string
wsml#stringJoin	func:string-join<N>	string x N	string
wsml#stringSubstring	func:substring1	string, _integer	string
wsml#stringSubstring	func:substring2	string, _integer; _integer	string
wsml#stringLength	func:string-length	string	_integer
wsml#stringToUpper	func:upper-case	string	string
wsml#stringToLower	func:lower-case	string	string
wsml#stringUriEncode	func:encode-for-uri	string	string
wsml#stringIriToUri	func:iri-to-uri	string	string
wsml#stringEscapeHtmlUri	func:escape-html-uri	string	string
wsml#stringSubstringBefore	func:substring-before1	string, string	string
wsml#stringSubstringBefore	func:substring-before2	string, string, string	string
wsml#stringSubStringAfter	func:substring-after1	string, string	string
wsml#stringSubStringAfter	func:substring-after2	string, string, string	string
wsml#stringReplace	func:replace1	string, string, string	string
wsml#stringReplace	func:replace2	string, string, string, string	string
wsml#stringContains	pred:contains1	string, string	
wsml#stringContains	pred:contains2	string, string, string	
wsml#stringStartsWith	pred:starts-with1	string, string	

wsml#stringStartsWith	pred:starts-with2	string, string, string	
wsml#stringEndsWith	pred:ends-with1	string, string	
wsml#stringEndsWith	pred:ends-with2	string, string, string	
wsml#stringMatches	pred:matches1	string, string	
wsml#stringMatches	pred:matches2	string, string, string	
wsml#yearPart	func:year-from-dateTime func:year-from-date func:years-from-duration	datetime date duration	_integer _integer _integer
wsml#monthPart	(as above, but with month)		_integer
wsml#dayPart	(as above, but with day)		_integer
wsml#hourPart	(as above, but with hour)		_integer
wsml#minutePart	(as above, but with minute)		_integer
wsml#secondPart	(as above, but with second)		_decimal
wsml#timezonePart	(as above, but with time-zone)		_dayTimeDuration
wsml#textFromStringLang	func:text-from-string-lang	string, string	_rdfText
wsml#textFromString	func:text-from-string	string	_rdfText
wsml#stringFromText	func:string-from-text	_rdfText	string
wsml#langFromText	func:lang-from-text	_rdftext	string
wsml#textCompare	func:text-compare	_rdfText, _rdfText	_integer
wsml#textLength	func:text-length	_rdfText	_integer

3.2.3 Mapping between WSML and RIF (RIF-Core and RIF-BLD)

RIF-Core corresponds to the language of definite Horn rules without function symbols and standard first-order semantics. It includes the data-types and built-ins of RIF-DTB and is a strict subset of RIF-BLD. RIF-Core has a number of extensions to support objects and

frames as in F-Logic.

RIF-BLD extends RIF-Core with equality in rule conclusion and function symbols.

From this basic outline, it is clear that there should be a straightforward mapping from WSML-Flight to RIF-Core and from WSML-Rule to RIF-BLD.

WSML-Flight and WSML-Rule are both rule languages. This section describes how elements of WSML ontologies can be defined using RIF-BLD syntax.

3.2.3.1 Terms

Terms are syntactically identical in both WSML and RIF, except that the symbol to associate a namespace with an identifier is '#' in WSML and ':' in RIF.

WSML and RIF support the XML schema data-types and use the same notation for literals.

Variables in both WSML and RIF are prefixed with a question mark, e.g. ?x

Constructed terms (function symbols) are only supported in WSML-Rule and RIF-BLD, not in WSML-Flight and RIF-Core.

3.2.3.2 F-Logic Constructs

WSML is based on F-logic[6] and this has a straightforward correspondence with the RIF F-logic syntax, i.e. for sub-concept, instance and attribute definitions.

Essentially, the following are equivalent:

WSML	**RIF**
memberOf	#
subConceptOf	##
hasValue	->

Examples:

Meaning	**WSML**	**RIF**
b1 is a **member of** Book	bks#b1 memberOf cpt#Book	bks:b1#cpt:Book
Human is a **sub-class of** Animal	cpt#Human subConceptOf cpt#Animal	cpt:Human##cpt:Animal
wd1 has **attribute** 'author' with the value 'rifwg' and the **attribute** 'title' with value 'LeRif'	bks#wd1[cpt#author hasValue auth#rifwg, cpt#title hasValue bks#LeRif]	bks:wd1[cpt:author->auth:rifwg cpt:title->bks:LeRif]
b1 is a **member of** Book	bks#b1 memberOf	bks:b1#cpt:Book

[6] http://www.cs.umbc.edu/771/papers/flogic.pdf

and has **attribute** 'author' with value 'rifwg'	cpt#Book[cpt#author hasValue auth#rifwg]	bks:b1[cpt:author->auth:rifwg]

3.2.3.3 Relations

Relations in WSML can be mapped directly to predicates in RIF, e.g. the WSML expression:

emt#loves(o1#Peter, ?x)

is equivalent to the RIF expression:

emt:loves(o1:Peter, ?x)

3.2.3.4 Attribute Properties

RIF does not support any special attribute properties corresponding to WSML transitive, symmetric, inverseOf, subAttributeOf, reflexive. These properties are modifiers for attribute declarations in the context of a concept, i.e. they are 'local' to the concept being declared. Such attribute properties can be expressed in RIF syntax, but this can only be done axiomatically and will apply globally to all attributes with the given name across all concepts.

The following is an example of how to declare an attribute 'p' to be globally transitive:

Meaning	**WSML**	**RIF**
Globally transitive attribute 'p'	?v1[p hasValue ?v2] impliedBy ?v1[p hasValue ?v3] and ?v3[p hasValue ?v2]	?v1[p->?v2] :- And(?v1[p->?v3] ?v3[p ->?v2])

3.2.3.5 Logical Expressions (Axioms/Rules)

3.2.3.5.1 Quantifiers

Both WSML and RIF support the existential and universal quantifiers, 'Exists' and 'Forall' in RIF syntax, 'exists' and 'forall' in WSML syntax.

Unless otherwise declared, all variables in WSML are implicitly universally quantified. This contrasts with RIF, where all variables must be explicitly quantified.

3.2.3.5.2 Connectives

':-' is used for logic programming implication in both WSML and RIF.

Conjunction and disjunction are supported in both WSML and RIF.

Classical implication (implies, impliedBy, equivalent) are supported in WSML in rule heads and bodies (WSML-Rule only), but are not supported in RIF.

Negation as failure is supported in WSML, but not in RIF.

Equality is supported in both WSML and RIF using the '=' symbol. Equality can be present in rule bodies to test for equality or in rule heads to infer the equivalence of two objects.

Inequality is supported in WSML with the inequality symbol '!=', but not directly in RIF. However, inequality in RIF can be achieved using the built-in predicates '*-not-equal', although this is somewhat clumsy.

3.2.3.5.3 Example

If an item is perishable and it is delivered to John more than 10 days after the scheduled delivery date then the item will be rejected by him.

RIF-BLD:

```
Prefix(ppl  http://example.com/people#)
Prefix(cpt  http://example.com/concepts#)
Prefix(func http://www.w3.org/2007/rif-builtin-function#)
Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)

   Forall ?item ?deliverydate ?scheduledate ?diffduration ?diffdays (
        cpt:reject(ppl:John ?item) :-
            And(cpt:perishable(?item)
                cpt:delivered(?item ?deliverydate ppl:John)
                cpt:scheduled(?item ?scheduledate)
                ?diffduration = External(
                    func:subtract-dateTimes(?deliverydate ?scheduledate))
                ?diffdays = External(
                    func:days-from-duration(?diffduration))
                External(pred:numeric-greater-than(?diffdays 10)))
   )
```

WSML:

```
cpt:reject(ppl:John ?item) :-
      cpt:perishable(?item) and
      cpt:delivered(?item, ?deliverydate, ppl#John) and
      cpt:scheduled(?item, ?scheduledate) and
      wsml#numericSubtract(?diffduration, ?deliverydate, ?scheduledate) and
      wsml#dayPart(?diffdays, ?diffduration)
      ?diffdays > 10.
```

3.2.4 Summary of what isn't supported in RIF that is in WSML

This section indicates what WSML elements cannot be serialized to a RIF document. The intention is to make it clear what would be lost in doing so.

- Negation
- Explicit inequality '!=', but can be achieved with '*-not-equal' predicates
- Classical implication (implies, impliedBy, equivalent)
- Local attribute modifiers (transitive, symmetric, inverseOf, subAttributeOf, reflexive)

3.2.5 Summary of what is supported in RIF, but not in WSML (Flight, Rule)

Function symbols are supported in RIF-BLD, but not in WSML-Flight. (Function symbols are supported in WSML-Rule).

3.3 WSML-Flight v2.0 Syntax

WSML-Flight 2.0 is both syntactically and semantically completely layered on top of WSML-Core 2.0. This means that every valid WSML-Core 2.0 specification is also a valid WSML-Flight 2.0 specification. Furthermore, all consequences inferred from a WSML-Core 2.0 specification are also valid consequences of the same specification in WSML-Flight 2.0. Finally, if a WSML-Flight 2.0 specification falls inside the WSML-Core 2.0 fragment then all consequences with respect to the WSML-Flight 2.0 semantics also hold with respect to the WSML-Core 2.0 semantics.

WSML-Flight 2.0 adds the following features to WSML-Core 2.0:

- N-ary relations with arbitrary parameters
- Constraining attribute definitions for the abstract domain
- Cardinality constraints
- (Locally Stratified) default negation in logical expressions (in the bodies of rules)
- Expressive logical expressions, namely, the full Datalog subset of F-Logic, extended with inequality (in the body) and locally stratified negation
- Meta-modeling. WSML-Flight no longer requires a separation of vocabulary (wrt. concepts, instances, relations)

Default negation means that the negation of a fact is true, unless the fact is *known* to be true. Locally stratified negation means that the definition of a particular predicate does not negatively depend on itself.

The rest of this section is organized as follows. Section 3.3.1 defines the restrictions on goals, web services and mediators in WSML-Flight v2.0. Section 3.3.2 defines the restrictions on logical expressions in WSML-Flight v2.0. Section 3.3.3 outlines the differences between WSML-Core and WSML-Flight v2.0.

3.3.1 Goals, Web services and Mediators in WSML-Flight v2.0

3.3.1.1 Goals in WSML-Flight v2.0

Goals in WSML-Flight follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and 'definition' of a non-functional property are limited to WSML-Flight logical expressions.

3.3.1.2 Web Services in WSML-Flight v2.0

Web Services in WSML-Flight v2.0 follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and 'definition' of a non-functional property are limited to WSML-Flight v2.0 logical expressions.

3.3.1.3 Mediators in WSML-Flight v2.0

Mediators in WSML-Flight v2.0 follow the common WSML syntax.

3.3.2 WSML-Flight v2.0 Logical Expression Syntax

WSML-Flight is a rule language based on the Datalog subset of F-Logic, extended with locally stratified default negation, the inequality symbol '!=' and the unification operator '='. Furthermore, WSML-Flight allows monotonic Lloyd-Topor [20] which means that we allow classical implication and conjunction in the head of a rule and we allow disjunction in the body of a rule.

The head and the body of a rule are separated using the Logic Programming implication symbol '**:-**'. This additional symbol is required because negation-as-failure (**naf**) is not defined for classical implication (**implies**, **impliedBy**). WSML-Flight allows classical implication in the head of the rule. Consequently, every WSML-Core logical expression is a WSML-Flight rule with an empty body.

The syntax for logical expressions of WSML Flight is the same as described in Section 2.8 with the restrictions described in the following. We define the notion of a WSML-Flight vocabulary in Definition 1.

Definition 1 Any WSML vocabulary [1] is a WSML-Flight vocabulary.

Definition 2 defines the set of WSML-Flight terms $Term_{Flight}(V)$ for a given vocabulary V.

Definition 2. Given a vocabulary V, the set of terms $Term_{Flight}(V)$ in WSML-Flight is defined as follows:

- Any $f \in V_O$ is a term.
- Any $v \in V_V$ is a term
- If $d \in V_D$ and $dv_1, ..., dv_n$ are in $V_{DV} \cup V_V$, then $d(dv_1, ..., dv_n)$ is a term.

As usual, the set of ground terms $GroundTerm_{Flight}(V)$ is the maximal subset of $Term_{Flight}(V)$ which does not contain variables.

Definition 3 Given a set of WSML-Flight terms $Term_{Flight}(V)$, an atomic formula in $L(V)$ is defined by:

- If $r \in V_R$ and $t_1, ..., t_n$ are terms, then $r(t_1, ..., t_n)$ is an atomic formula in $L(V)$.
- If $\alpha, \beta \in Term_{Flight}(V)$ then α **=** β, and α **!=** β are atomic formulae in $L(V)$.
- If $\alpha, \beta \in Term_{Flight}(V)$ and $\gamma \in Term(V)$ or γ is of the form $\{ \gamma_1,...,\gamma_n \}$ with $\gamma_1,...,\gamma_n \in Term_{Flight}(V)$, then:
  - α **subConceptOf** γ is an atomic formula in $L(V)$
  - α **memberOf** γ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **ofType** $\gamma]$ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **impliesType** $\gamma]$ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **hasValue** $\gamma]$ is an atomic formula in $L(V)$

A ground atomic formula is an atomic formula with no variables.

Definition 4. Given a WSML-Flight vocabulary V, the set of formulae in $L(V)$ is recursively defined as follows:

- We define the set of admissible head formulae $Head(V)$ as follows:
  - Any atomic formula α which does not contain the inequality symbol (!=) is in $Head(V)$.
  - Let $\alpha,\beta \in Head(V)$, then α **and** β is in $Head(V)$.
  - Given two formulae α, β such that α, β do not contain { **implies**, **impliedBy**, **equivalent**}, the following formulae are in $Head(V)$:

    - *α* **implies** *β*, if *β* ∈ *Head(V)* and *α* ∈ *Head(V)* or *α* ∈ *Body(V)*
    - *α* **impliedBy** *β*, if *α* ∈ *Head(V)* and *β* ∈ *Head(V)* or *β* ∈ *Body(V)*
    - *α* **equivalent** *β* if *α* ∈ *Head(V)* or *α* ∈ *Body(V)* and *β* ∈ *Head(V)* or *β* ∈ *Body(V)*
- Any variable-free admissible head formula in *Head(V)* is a formula in *L(V)*.
- We define the set of admissible body formulae *Body(V)* as follows:
    - Any atomic formula *α* is in *Body(V)*
    - For any atomic formula *α*, **naf** *α* is in *Body(V)*.
    - For *α,β* ∈ *Body(V)*, *α* **and** *β* is in *Body(V)*.
    - For *α,β* ∈ *Body(V)*, *α* **or** *β* is in *Body(V)*.
- Given a head-formula *β* ∈ *Head(V)* and a body-formula *α* ∈ *Body(V)*, *β* **:-** *α* is a formula. Here we call *α* the *body* and *β* the *head* of the formula. The formula is admissible if (1) *α* is an admissible body formula, (2) *β* is an admissible head formula, and (3) the safety condition holds.
- Any formula of the form !- *α* with *α* ∈ *Body(V)* is an admissible formula and is called a *constraint*.
- The Logic Programming implication symbol !- is not absolutely needed. If it is missing, a formula is in Head.

As with the general WSML logical expression syntax, <-, -> and <-> can be seen as synonyms of the keywords **implies**, **impliedBy** and **equivalent**, respectively.

In order to check the *safety condition* for a WSML-Flight rule, the following transformations should be applied until no transformation rule is applicable:

- Rules of the form A_1 **and** ... **and** A_n **:-** *B* are split into *n* different rules:
    - A_1 **:-** *B*
    - ...
    - A_n **:-** *B*
- Rules of the form A_1 **equivalent** A_2 **:-** *B* are split into 2 rules:
    - A_1 **implies** A_2 **:-** *B*
    - A_1 **impliedBy** A_2 **:-** *B*
- Rules of the form A_1 **impliedBy** A_2 **:-** *B* are transformed to:
    - A_1 **:-** A_2 **and** *B*
- Rules of the form A_1 **implies** A_2 **:-** *B* are transformed to:
    - A_2 **:-** A_1 **and** *B*
- Rules of the form *A* **:-** B_1 **and** (*F* **or** *G*) **and** B_2 are split into two different rules:
    - *A* **:-** B_1 **and** *F* **and** B_2
    - *A* **:-** B_1 **and** *G* **and** B_2
- Rules of the form *A* **:-** B_1 **and naf** (*F* **and** *G*) **and** B_2 are split into two different rules:
    - *A* **:-** B_1 **and naf** *F* **and** B_2
    - *A* **:-** B_1 **and naf** *G* **and** B_2
- Rules of the form *A* **:-** B_1 **and naf** (*F* **or** *G*) **and** B_2 are transformed to:
    - *A* **:-** B_1 **and naf** *F* **and naf** *G* **and** B_2
- Rules of the form *A* **:-** B_1 **and naf naf** *F* **and** B_2 are transformed to:
    - *A* **:-** B_1 **and** *F* **and** B_2

Application of these transformation rules yields a set of WSML-Flight rules with only one atomic formula in the head and a conjunction of literals in the body.

The safety condition holds for a WSML-Flight rule if every variable which occurs in the rule occurs in a positive body literal which does not correspond to a built-in predicate. For example, the following rules are not safe and thus not allowed in WSML-Flight:

```
p(?x) :- q(?y).
a[b hasValue ?x] :- ?x > 25.
?x[gender hasValue male] :- naf ?x[gender hasValue female].
```

We require each WSML-Flight knowledge base to be *locally stratified*. For more details on local stratification please refer to [21]. The following are examples of WSML-Flight logical expressions (note that variables are implicitly universally quantified):

No human can be both male and female:

```
!- ?x[gender hasValue {?y, ?z}] memberOf Human and ?y = Male and ?z =
Female.
```

The brother of a parent is an uncle:

```
 ?x[uncle hasValue ?z] impliedBy ?x[parent hasValue ?y] and ?y[brother
hasValue ?z].
```

Do not trust strangers:

```
 ?x[distrust hasValue ?y] :- naf ?x[knows hasValue ?y] and ?x memberOf
Human and ?y memberOf Human.
```

3.3.3 Differences between WSML-Core and WSML-Flight

The features added by WSML-Flight compared with WSML-Core are the following: Allows n-ary relations with arbitrary parameters, constraining attribute definitions for the abstract domain, cardinality constraints, (locally stratified) default negation in logical expressions, (in)equality in the logical language (in the body of the rule), Full-fledged rule language (based on the Datalog subset of F-Logic).

3.3.4 Differences between WSML-Flight v1.0 and WSML-Flight v2.0

The features added by WSML-Flight v2.0, that are not available in WSML-Flight v1.0 are those introduced for alignment with WSML-Core v2.0. More precisely WSML-Flight v2.0 introduces instance equivalence, otherwise known as ‘equality in rule heads’. This change allows for the declaration that different instance identifiers (IRIs) refer to the same object.

Secondly, WSML-Flight v2.0 is aligned with the emerging Rule Interchange Format (RIF) standards. The most extensive changes here relate to the missing support for RIF built-in data-types, predicates and functions.

3.4 WSML Rule v2.0 Syntax

WSML-Rule is an extension of WSML-Flight in the direction of Logic Programming. WSML-Rule no longer requires safety of rules and allows the use of function symbols. The only differences between WSML-Rule and WSML-Flight are in the logical expression syntax.

WSML-Rule is both syntactically and semantically layered on top of WSML-Flight and thus each valid WSML-Flight specification is a valid WSML-Rule specification. Because the only differences between WSML-Flight and WSML-Rule are in the logical expression syntax, we do not explain the conceptual syntax for WSML-Rule.

The rest of this section is organized as follows. Section 3.4.1 defines the restrictions on goals, web services and mediators in WSML-Rule v2.0. Section 3.4.2 defines the restrictions on logical expressions in WSML-Rule v2.0. Section 3.4.3 outlines the differences between WSML-Core and WSML-Flight v2.0.

3.4.1 Goals, Web services and Mediators in WSML-Rule v2.0

3.4.1.1 Goals in WSML-Rule v2.0

Goals in WSML-Rule follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and 'definition' of a non-functional property are limited to WSML-Rule logical expressions.

3.4.1.2 Web Services in WSML-Rule v2.0

Web Services in WSML-Rule v2.0 follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and 'definition' of a non-functional property are limited to WSML-Flight v2.0 logical expressions.

3.4.1.3 Mediators in WSML-Flight v2.0

Mediators in WSML-Flight v2.0 follow the common WSML syntax.

3.4.2 WSML-Rule Logical Expression Syntax

WSML-Rule is a simple extension of WSML-Flight. WSML-Rule allows the unrestricted use of function symbols and no longer requires the safety condition, i.e., variables which occur in the head are not required to occur in the body of the rule.

The syntax for logical expressions of WSML Rule is the same as described in Section 2.8 with the restrictions which are described in the following: we define the notion of a WSML-Rule vocabulary in Definition 5.

Definition 5 Any WSML vocabulary (see [22]) is a WSML-Rule vocabulary.

Definition 6 defines the set of terms *Term(V)* for a given vocabulary *V*.

Definition 6. Any WSML term (see Definition 2.2) is a WSML Rule term.

As usual, the set of ground terms *GroundTerm(V)* is the maximal subset of *Term(V)* which does not contain variables.

Definition 7. Given a set of WSML-Rule terms $Term_{Rule}(V)$, an atomic formula in $L(V)$ is defined by:

- If $r \in V_R$ and $t_1, ..., t_n$ are terms, then $r(t_1, ..., t_n)$ is an atomic formula in $L(V)$.
- If $\alpha, \beta \in Term_{Rule}(V)$ then α **=** β, and α **!=** β are atomic formulae in $L(V)$.
- If $\alpha, \beta \in Term_{Rule}(V)$ and $\gamma \in Term(V)$ or γ is of the form $\{ \gamma_1,...,\gamma_n \}$ with $\gamma_1,...,\gamma_n \in Term_{Rule}(V)$, then:
  - α **subConceptOf** γ is an atomic formula in $L(V)$
  - α **memberOf** γ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **ofType** $\gamma]$ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **impliesType** $\gamma]$ is an atomic formula in $L(V)$
  - $\alpha[\beta$ **hasValue** $\gamma]$ is an atomic formula in $L(V)$

A ground atomic formula is an atomic formula with no variables.

Definition 8. Given a WSML-Rule vocabulary *V*, the set of formulae in *L(V)* is recursively defined as follows:

- We define the set of admissible head formulae *Head(V)* as follows:
  - Any atomic formula α which does not contain the inequality symbol (!=)is in *Head(V)*.
  - Let $\alpha,\beta \in Head(V)$, then α **and** β is in *Head(V)*.

  - Given two formulae *α*, *β* such that *α*, *β* do not contain { **implies**, **impliedBy**, **equivalent,**}, the following formulae are in *Head(V)*:
    - *α* **implies** *β*, if *β* ∈ *Head(V)* and *α* ∈ *Head(V)* or *α* ∈ *Body(V)*
    - *α* **impliedBy** *β*, if *α* ∈ *Head(V)* and *β* ∈ *Head(V)* or *β* ∈ *Body(V)*
    - *α* **equivalent** *β* if *α* ∈ *Head(V)* or *α* ∈ *Body(V)* and *β* ∈ *Head(V)* or *β* ∈ *Body(V)*

- Any admissible head formula in *Head(V)* is a formula in *L(V)*.
- We define the set of admissible body formulae *Body(V)* as follows:
  - Any atomic formula *α* is in *Body(V)*
  - For *α* ∈ *Body(V)*, **naf** *α* is in *Body(V)*.
  - For *α,β* ∈ *Body(V)*, *α* **and** *β* is in *Body(V)*.
  - For *α,β* ∈ *Body(V)*, *α* **or** *β* is in *Body(V)*.
  - For *α,β* ∈ *Body(V)*, *α* **implies** *β* is in *Body(V)*.
  - For *α,β* ∈ *Body(V)*, *α* **impliedBy** *β* is in *Body(V)*.
  - For *α,β* ∈ *Body(V)*, *α* **equivalent** *β* is in *Body(V)*.
  - For variables $?x_1,...,?x_n$ and *α* ∈ *Body(V)*, **forall** $?x_1,...,?x_n$ (*α*) is in *Body(V)*.
  - For variables $?x_1,...,?x_n$ and *α* ∈ *Body(V)*, **exists** $?x_1,...,?x_n$ (*α*) is in *Body(V)*.
- Given a head-formula *β* ∈ *Head(V)* and a body-formula *α* ∈ *Body(V)*, *β* **:-** *α* is a formula. Here we call *α* the *body* and *β* the *head* of the formula. The formula is admissible if (1) *α* is an admissible body formula, (2) *β* is an admissible head formula.
- Any formula of the form !- *α* with *α* ∈ *Body(V)* is an admissible formula and is called a *constraint*.
- The Logic Programming implication symbol !- is not absolutely needed. If it is missing, a formula is in Head.

As with the general WSML logical expression syntax, <-, -> and <-> can be seen as synonyms of the keywords **implies**, **impliedBy** and **equivalent**, respectively.

The following are examples of WSML-Rule logical expressions:

Both the father and the mother are parents:

```
?x[parent hasValue ?y] :- ?x[father hasValue ?y] or ?x[mother hasValue ?y].
```

Every person has a father:

```
?x[father hasValue f(?x)] :- ?x memberOf Person.
```

There may only be one distance between two locations, and the distance between locations *A* and *B* is the same as the distance between *B* and *A*:

```
!- distance(?location1,?location2,?distance1) and
  distance(?location1,?location2,?distance2) and ?distance1 != distance2.

distance(?B,?A,?distance) :-
  distance(?A,?B,?distance).
```

3.4.3 Differences between WSML-Flight and WSML-Rule

WSML-Rule allows unsafe rules and the use of function symbols (constructed terms) in logical expressions.

Furthermore, due to the meta-modeling of WSML constructs (classes, objects and attributes), unlimited use of WSML-Rule logical expressions can lead to Datalog representations containing unstratified rules, i.e. a rule set that contain recursive dependencies through negation. The semantics applied to negation in this context is that of the Well-founded semantics [17].

3.4.4 Differences between WSML-Rule v1.0 and WSML-Rule v2.0

The features added by WSML-Rule v2.0, that are not available in WSML-Rule v1.0 are those introduced for alignment with WSML-Core v2.0. More precisely WSML-Rule v2.0 introduces instance equivalence, otherwise known as 'equality in rule heads'. This change allows for the declaration that different instance identifiers (IRIs) refer to the same object.

Secondly, WSML-Rule v2.0 is aligned with the emerging Rule Interchange Format (RIF) standards. The most extensive changes here relate to the missing support for RIF built-in data-types, predicates and functions.

3.5 Algorithmisation

Reasoning for the rule based WSML variants in principle can be achieved in just the same way as for WSML-Core 2.0 and involves the same conversion steps that syntactically transform a WSML ontology to the corresponding Datalog program. The fundamental difference is that due to the added expressivity in WSML-Flight 2.0, and beyond that in WSML-Rule 2.0, the underlying Datalog engine has to cover more features.

The conversion can still be expressed as a series of transformation steps in a pipeline, which can roughly be divided into (i) Axiomatization, (ii) Normalization, and (iii) Datalog rules generation, as described in D3.2.1.

The overall reasoning process is presented in ***Figure 2***. It requires as input the ontology on which the reasoning is performed as well as the specification of the reasoning task. The reasoning task results are provided as outputs. The overall communication is realized through the WSML Reasoner API. The API provides the means to register data / ontologies, to query and to retrieve the reasoning results. The algorithmization contains a set of transformation/normalization steps performed in a pipeline manner. It includes the transformation of the conceptual syntax to logical expressions and further on into generalized clauses / rules. The generalized clauses/ rules are provided as input for external reasoning tools, or native reasoning tools accessible via an adapter interface.

Additional features that need to be mapped to corresponding elements in the Datalog program come in the form of function symbols, (default) negation, LP implication, and integrity constraints. However, these features do not fundamentally complicate the translation.

However, meta-modelling features are introduced in WSML-Flight 2.0, which no longer requires a separation of the vocabularies of concepts, instances and relations. This meta-modelling requires special consideration and these features can be realized by an indirect mapping to Datalog through meta-level predicates for WSML language constructs. This means, for example, that concept membership of an instance (denoted via **memberOf**) is not directly translated to a unary predicate, but rather to a special binary "memberOf" predicate.

For example,

```
instance Mary memberOf Human
```

would not be translated to `Human(Mary)`, but rather to `member-of(Mary, Human)`. This extra level of indirection allows an entity to be treated as a concept and an instance at the

same time.

In order to capture the intended semantics of these meta-level predicates (e.g. transitivity of **subConceptOf**) a set of associated (meta-level) axioms is required.

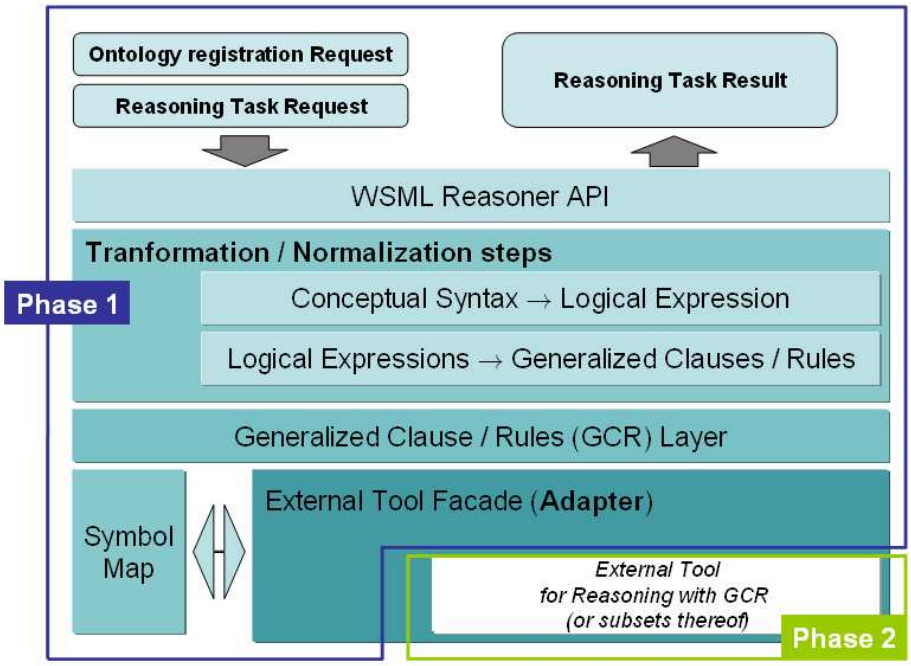

Figure 2 Conceptual Layering of Transformations

3.6 Relation with other WSML Variants and Language Layering

As mentioned earlier, WSML actually consists of distinctly different language variants, identified for their particular properties in terms of modelling and performance of reasoning tasks. They differ in expressiveness as well as in their underlying logical formalism. This allows users of the language to decide on (i) the level of expressivity and thus also on (ii) the associated complexity, as well as (iii) the style of modelling which they want to use, on a case by case basis – depending on the requirements of a specific application.

The relation between the different WSML variants is depicted in ***Figure 3***. As can be seen, WSML-Quark and WSML-Core 2.0 form a common, lightweight, yet increasingly expressive foundation for extensions towards the paradigms of both Description Logic (in the form of WSML-DL 2.0) and Logic Programming (in the form of WSML-Flight 2.0 and WSML-Rule 2.0). Consequently, WSML-DL 2.0 and WSML-Flight/Rule 2.0 are both layered on WSML-Core 2.0, which defines a common subset. WSML-Core v2.0 is in turn layered upon WSML-Quark.

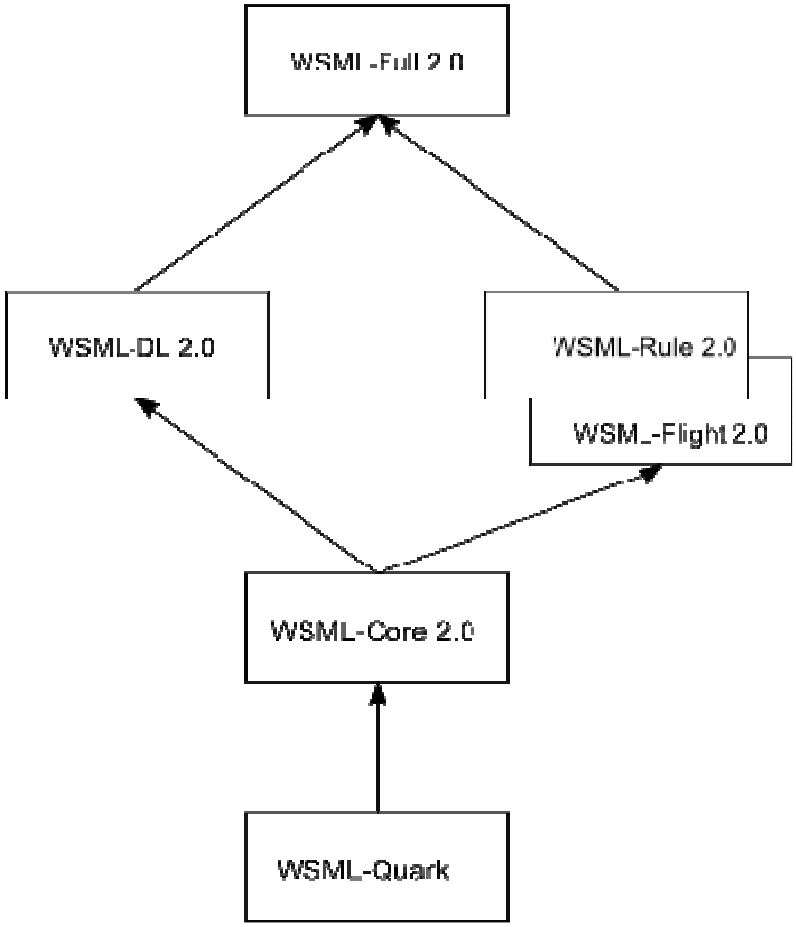

Figure 3 WSML Language Layering

WSML-Quark is a very lightweight and intuitive language variant that allows for the simple organization of concepts in to a hierarchical classification system. WSML-Quark can be used as a very efficient stepping stone towards more formal and complex WSML language variants.

WSML-Core 2.0 inherits many features from the first version of WSML-Core, which was based on DLP [11] - formed by the intersection of the Description Logic SHIQ and Horn Logic. It has been adjusted to align results of ongoing standardization efforts, most notably OWL 2 RL [14], as well research results such as the L2 language [13], which has similar language features, albeit specified directly at the level of RDF. Furthermore, WSML-Core 2.0 forms the common subset between the DL and LP based variants of WSML.

WSML-DL 2.0
WSML-DL 2.0 is the Description Logic variant of WSML, based on ELP [16], which is based on the tractable DL EL++ [10], and covers OWL 2 RL, OWL 2 EL and OWL 2 QL, while at the same time retaining polynomial combined complexity.

WSML-Flight 2.0 is the least expressive of the two LP-based variants. Compared with WSML-Core, it adds features such as meta-modeling, constraints, and non-monotonic (stratified) negation. WSML-Flight is semantically equivalent to Datalog with equality and integrity constraints.

WSML-Rule 2.0 extends WSML-Flight 2.0 with further features from Logic Programming, namely the use of function symbols, unsafe rules, and unstratified negation. Due to the intended tractability goals, WSML-Rule 2.0 relies on the Well-Founded Semantics [17] in place of the more general Stable Model Semantics for the purpose of query answering.

WSML-Full 2.0 finally reconciles the DL and LP variants of WSML in a more expressive superset. While the specification of WSML-Full is still open at this stage, the use of hybrid MKNF knowledge bases forms a possible option. [18] defines the well-founded semantics for this approach, which still preserves tractable data complexity.

3.7 Conclusions and Future Work

The new versions of WSML-Flight and WSML-Rule as described in this document contain modifications that maintain the consistency of both syntactic and semantic layering of the entire WSML family. Further, increased support for the emerging RIF standards has been improved with the introduction of many missing built-in predicates and functions.

The first prototype implementations of WSML-Flight 2.0 and WSML-Rule 2.0 reasoners will be developed in SOA4All and reported in deliverable: D3.2.3 First Prototype Rule Repository Reasoner for WSML-Rule v2.0 (Month 18).

Later implementations will address scalability issues and it is planned to re-use software components developed in European project LarKC to achieve scalability using a combination of parallel, distributed and approximate reasoning algorithms.

It may be desirable at some stage to define new RIF dialects that completely capture the semantics of WSML-Flight and WSML-Rule. The essential difference of such new dialects would be the inclusion of non-monotonic negation.

4. References

[1] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel, "The Web Service Modeling Language WSML," WSML Final Draft D16.1, 2005.

[2] D. Roman, H. Lausen, and U. Keller, "Web Service Modeling Ontology (WSMO)," *WSMO Working Draft*, 2004.

[3] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, "The Description Logic Handbook," 2007.

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

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

[6] 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.

[7] I. Horrocks, U. Sattler, and S. Tobies, "Practical reasoning for very expressive description logics," *Logic Journal of IGPL*, vol. 8, 2000, pp. 239-263.

[8] U. Hustadt, B. Motik, and U. Sattler, "Data Complexity of Reasoning in Very Expressive Description Logics."

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "DL-Lite: Tractable Description Logics for Ontologies," *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 602.

[10] F. Baader, S. Brandt, and C. Lutz, "Pushing the EL Envelope," *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, LAWRENCE ERLBAUM ASSOCIATES LTD, 2005, p. 364.

[11] B. GROSOF, I. HORROCKS, R. VOLZ, and S. DECKER, "Description Logic Programs: Combining Logic Programs with Description Logic."

[12] H.J. ter Horst, "Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity," *Proc. of ISWC*, Springer, 2005, pp. 6-10.

[13] F. Fischer, U. Keller, A. Kiryakov, Z. Huang, V. Momtchev, E. Simperl, D. Fensel, and R. Dumitru, "D1.1.3 Initial Knowledge Representation Formalism," *LarKC Deliverable*, 2004.

[14] B. Motik, C. Bernardo, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, "OWL 2 Web Ontology Language: Profiles," *W3C Working Draft*, Dec. 2008.

[15] "Web21C SDK."

[16] M. Krötzsch, S. Rudolph, and P. Hitzler, "ELP: Tractable rules for OWL 2," *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, Springer, 2008.

[17] A. Van Gelder, K.A. Ross, and J.S. Schlipf, "The well-founded semantics for general logic programs," *Journal of the ACM (JACM)*, vol. 38, 1991, pp. 619-649.

[18] M. Knorr, J.J. Alferes, and P. Hitzler, "A Coherent Well-founded Model for Hybrid MKNF Knowledge Bases," *ECAI 2008: Proceedings, 18th European Conference on Artificial Intelligence, July 21-25, 2008, Patras, Greece: Including Prestigious Applications of Intelligent*, IOS Press, 2008, p. 99.

[19] Rule Interchange Format (RIF) Working Group,

http://www.w3.org/2005/rules/wiki/RIF_Working_Group

[20] John W. Lloyd and Rodney W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 1(3):225{240, 1984

[21] J. de Bruijn (Ed). *WSML Abstract Syntax*. WSML Working Draft D16.3v0.3, 2008. Available from http://www.wsmo.org/TR/d16.3/v0.3/.

[22] *The Web Service Modeling Language WSML*, WSML v1.0, WSML deliverable D16.1, 2009.

[23] Florian Fischer (Ed). *Defining the features of the WSML-Core v2.0 language*, SOA4All deliverable D3.1.2.

[24] Rule Interchange Format Data Type and Built-ins (RIF-DTB) - http://www.w3.org/2005/rules/wiki/DTB

[25] Rule Interchange Format Basic Logic Dialect (RIF-BLD) - http://www.w3.org/2005/rules/wiki/BLD

[26] Rule Interchange Format Core (RIF-Core) - http://www.w3.org/2005/rules/wiki/Core