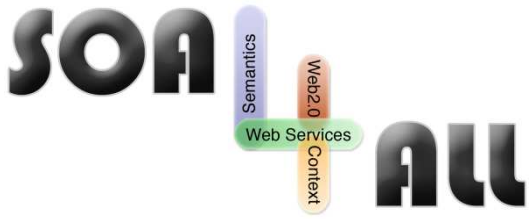

Project Number: **215219**
Project Acronym: **SOA4All**
Project Title: **Service Oriented Architectures for All**
Instrument: **Integrated Project**
Thematic Priority: **Information and Communication Technologies**

D3.1.2 Defining the features of the WSML-Core v2.0 language

Activity N:	A2 Core R&D	
Work Package:	WP3 Service Annotation and Reasoning	
Due Date:		M12
Submission Date:		10/03/2009
Start Date of Project:		01/03/2006
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Author(s):	Florian Fischer UIBK Barry Bishop UIBK	
Reviewers(s):	Rafael González-Cabero ATOS Jean-Pierre Lorre EBM	

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	15/01/2009	Creaton	Florian Fischer (UIBK)
0.2	20/01/2009	Initial Content	Florian Fischer
0.3	22/01/2009	Section 1, 2	Florian Fischer
0.4	25/01/2009	Draft	Florian Fischer
0.5	02/02/2009	Minor corrections	Barry Bishop (UIBK)
0.6		Peer review	Rafael González-Cabero (ATOS) Jean-Pierre Lorre (EBM)
1.0	02/27/2009	Changes to accommodate reviewer comments	Florian Fischer
Final	09/03/2009	Overall format and quality revision	Malena Donato (ATOS)

Table of Contents

EXECUTIVE SUMMARY 6

1. INTRODUCTION 7

1.1 PURPOSE AND SCOPE 8
1.1.1 Audience 8
1.1.2 Scope 8
1.2 STRUCTURE OF THE DOCUMENT 8

2. TECHNICAL DELIVERABLE REMARKS 9

2.1 DELIVERABLE RELATION WITH THE ARCHITECTURE OF THE PROJECT 9
2.2 DELIVERABLE RELATION WITH THE USE-CASES 10
2.2.1 End-user Integrated Enterprise Service Delivery Platform 10
2.2.2 W21C BT Infrastructure 10
2.2.3 C2C Service eCommerce 11

3. WSML CORE V2.0 LANGUAGE DEFINITION 12

3.1 MOTIVATION 12
3.2 RELATED WORK AND BACKGROUND 12
3.3 WSML CORE V2.0 SYNTAX DEFINITION 13
3.3.1 Ontologies 14
3.3.2 Goals in WSML-Core 2.0 15
3.3.3 Web Services in WSML-Core 2.0 15
3.3.4 Mediators in WSML-Core 2.0 15
3.3.5 WSML-Core 2.0 Logical Expression Syntax 15
3.4 ALGORITHMISATION 17
3.5 RELATION WITH OTHER WSML VARIANTS AND LANGUAGE LAYERING 18

4. CONCLUSIONS 21

5. REFERENCES 22

Table of Figures

Figure 1 SOA4All Overall Architecture 9
Figure 2 Conceptual Layering of Transformations 17
Figure 3 WSML Language Layering 19

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
IRI	International Resource Identifier
TBox	Terminological Box
ABox	Assertional Box

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, along with a concrete syntax for modelling ontologies.

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. Thus, this document describes WSML-Core 2.0, a light-weight WSML language variant serving as a common foundation for more expressive variants. 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" (see D1.1.1):

- 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-Core variant of the WSML language family, in order to bring it up to date with recent research in both the Description Logic and Logic Programming paradigm. It 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 of 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.

Many current standards and knowledge representation formalisms for the Web feature very high worst-case complexity results, ranging from EXPTime-complete to NEXPTIME-complete. For example, such worst-case results apply to the OWL language family as well as for WSML-DL, which is a notational variant of the Description Logic SHIQ(D) [7].

In general, tableaux-based methods for Description Logics behave very efficiently in regard to TBox (schema) reasoning, however they do not scale very well when faced with a large ABox (a large instance set) [8].

In order to support tractable inference at a Web-scale there have been proposals for more lightweight representation formalisms such as the DL-Lite family of languages [9], EL++ [10], as well as tractable fragments of OWL like DLP[11] OWL-Horst [12], or L2 [13]. Several of these proposals are in the process of being adopted in the upcoming OWL 2 standard as so called profiles [14]. This deliverable is thus part of an effort to align WSML with these research and standardization efforts.

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, and require a precise specification. Even more so it targets 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 WSML-Core 2.0 language variant, particularly to describe the changes made in regard to the existing WSML-Core language[1].

We describe the modelling elements in WSML-Core 2.0, restrictions imposed on the language, and a motivation for them. Beyond the definition of the conceptual and the logical expression syntax of the language itself, 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 **Error! Reference source not found.** defines the WSML-Core 2.0 language by describing the individual language elements and pointing out the particular restrictions placed on them for this language variant. It then proceeds to outline the algorithmisation of WSML-Core 2.0 on rule-based reasoners. Finally, section 3 clarifies the relation of WSML-Core 2.0 to the other WSML language variants, and their layering. Section 4 concludes this deliverable and points out the next steps for future work.

[1] http://www.wsmo.org/wsml/wsml-syntax

2. Technical deliverable remarks

2.1 Deliverable relation with the architecture of the project

The overall architecture of SOA4All can be structured into four distinct parts: SOA4All Studio, Distributed Service Bus, SOA4All Platform Services, and Business Services (Web services). An overview of SOA4All's overall architecture is depicted in ***Figure 1***.

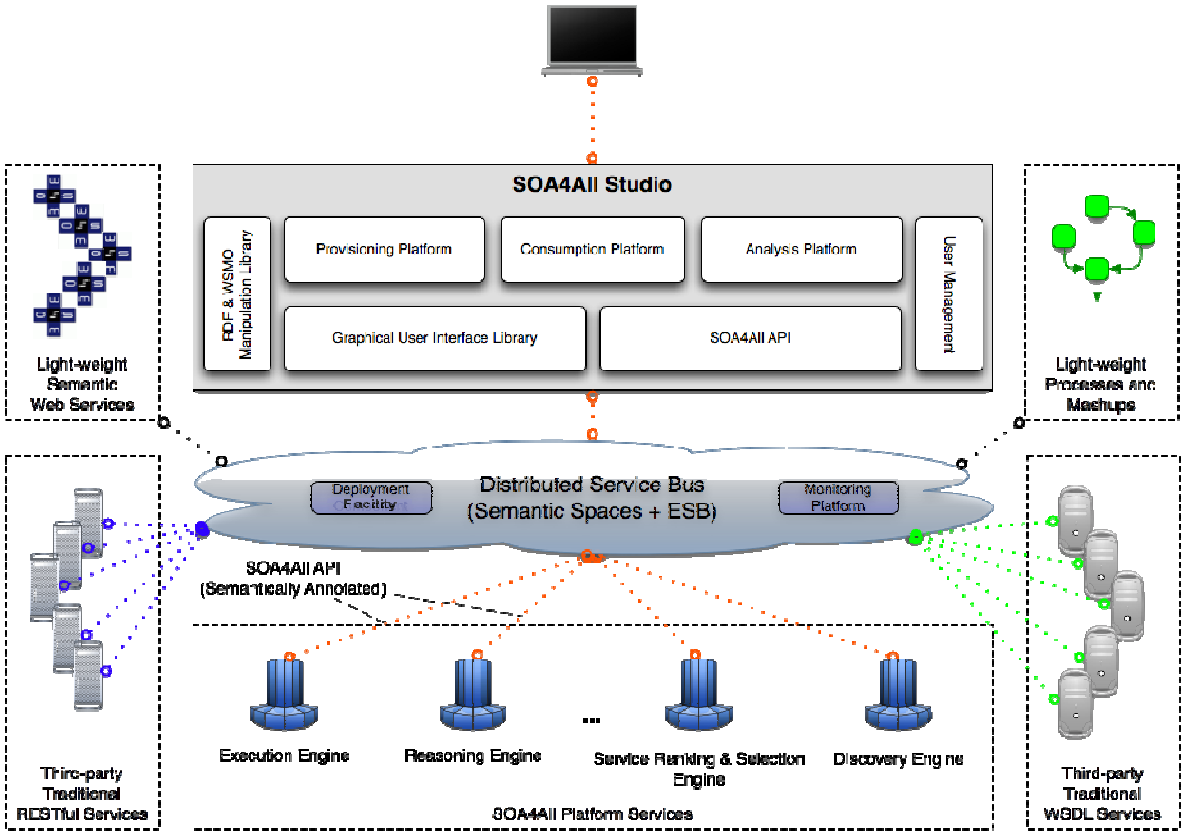

Figure 1 SOA4All Overall Architecture

At the very core of the architecture there is the SOA4All Distributed Service Bus, which serves as infrastructure to tie other components together, and thus forms the central integration platform. In addition the Deployment platform provides uniform support for the management and deployment of all software composing the whole SOA4All service computing environment. The Monitoring Platform collects monitoring data about the usage of SOA4All platform services and traditional third-party services.

Built around the Distributed Service Bus as integration platform, there are at the top the SOA4All Studio and at the bottom the SOA4All Platform services, which are the components delivered by the various research and development work packages.

The SOA4All Studio delivers the user front-ends that enable the creation, provisioning, consumption and analysis of the platform services and various third party business services that are published to SOA4All.

Platform Services deliver the various functionalities needed for service discovery, ranking and selection, composition and invocation. These components are exposed to the SOA4All Distributed Service Bus as Web services and hence consumable as any other published

service.

Business Services are the artefacts that are actually created and manipulated by means of the SOA4All infrastructure. First of all, there are the (publicly) available Web services that are exposed either as traditional RESTful services, or as traditional WSDL-based services. These are invokable third-party business services that SOA4All seeks to fully enable in terms of automation, composition and invocation. Additionally, to the top-left the figure depicts the semantic annotations of the business services, facilitating so-called Semantic Web services. The semantic descriptions are published in the service repository that is part of the Distributed Service Bus, and used for reasoning with service capabilities (functionality) and interfaces.

The conceptual work conducted towards a reworked WSML language stack in WP3 has immediate consequences for the reasoning components to be developed in WP3, which directly process these formal languages and are part of the Platform Services.

Furthermore, additional platform services, as i.e. the Service Ranking & Selection Engine or the Discovery Engine, which (i) operate on semantically annotated Web Services, or (ii) rely on an ontology for other reasons will make use of WSML, at least indirectly.

2.2 Deliverable relation with the use-cases

This section clarifies the relation of this deliverable, and the WSML language family in general, with the 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.

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[2] 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

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

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.

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.

This use-case again entails several tasks that are based on annotation and (WSML) 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 Core v2.0 Language Definition

3.1 Motivation

One of the main limitations of Description Logics (such as OWL) are their intractability and lack of algorithms which allow reasoning with non-trivial ABox sizes. Methods, such as ABox summarization [16], have been developed to alleviate this problem.

Furthermore, as [17] and others have pointed out, semantic data on the Web will usually use only very limited expressivity at the schema level, but with very large instance sets. However, reasoning with very large instance sets, which is also a requirement in SOA4ALL and in a "Web context" in general, is still a very challenging problem.

This has been acknowledged by work on more lightweight knowledge representation methods, which have tractability in regard to specific applications as one of their main concerns, as well by their adoption in standardization efforts. For example, the W3C OWL 2 working group is working towards standardizing several tractable "profiles" of OWL, each chosen for their particular computational properties with respect to reasoning and specific applications.

Reconciling these efforts with the added expressivity and different modelling style of rule-based formalisms is one of the main efforts of this deliverable and the associated work towards a rework of the WSML language family. We argue that different applications on the Semantic Web require these different styles of modelling and thus both types of languages (DL as well as LP based) are needed.

The second main motivation is to rework the WSML language stack in a way that preserves the tractability of the WSML-Core foundation layer when the more expressive variants (WSML-Rule, WSML-DL) are layered on top. This goes hand in hand with the desire to make use of optimized reasoning algorithms that stem from years of deductive database research and are especially well equipped to handle large ABoxes.

In summary, the main motivation for this work is to facilitate a framework that integrates both modelling paradigms, and meets the tractability requirements faced on the Web. In the following section we briefly point out related work that we consider relevant and take this into account.

3.2 Related Work and Background

DLP

The initial version of WSML-Core was based on Description Logic Programs (DLP) [11], which can be roughly described as the logical fragment formed by the intersection of Description Logics and Logic Programming. As [17] points out, there are actually a number of different variants of DLP, depending on the particular target logics of the underlying Logic Programming engine. The practical use of DLP is actually twofold: First of all it is a tractable formalism that captures many of the ontologies found on the Web, and secondly it can serve as a basic interoperability layer between Description Logic and Logic Programming based formalisms.

The OWL 2 RL profile [14], which is part of the forthcoming OWL 2 standard, can be considered to be roughly based on DLP.

L2

L2 [13] is a slight extension of RDFS towards often used OWL language primitives. It can be considered a pragmatically oriented, even lighter subset of OWL-Horst [12]. OWL-Horst is a

non-standard, tractable fragment of OWL, implemented by many scalable inference engines such as Oracle 11g and OWLIM, that defines its semantics in the form of entailment rules that operate directly on RDF triples. In turn L2 can also be considered an OWL fragment (or "profile"') in the sense that it allows an efficient sub-set of inferences to be made. Among the OWL 2 profiles being standardized currently it is closest to OWL 2 RL and in turn also DLP.

OWL 2 RL

The OWL 2 RL is a fragment, which is customized to support reasoning with rule-based engines. It is a profile that is intended to form a proper extension of RDFS while still being computationally tractable. As such it realizes a weakened form of the OWL 2 Full semantics and is very similar in spirit to DLP and OWL-Horst. The OWL 2 RL semantics are provided as a partial axiomatization in the form of additional entailment rules directly on the RDF serialization of OWL 2 in a similar fashion as for OWL-Horst.

OWL 2 EL

OWL 2 EL, which is based on the Description Logic EL++ [10] is a fragment that is very tractable in regard to several key inference tasks such as consistency checking, subsumption checking, instance checking or classification -- they can be decided in polynomial time. On the other hand it is still expressive enough to adequately model many real world problems. The most prominent constructs from OWL 2 that are disallowed in OWL 2 EL are disjunction, negation, enumerations of multiple elements, inverse and irreflexive object properties, functional object properties, symmetric and asymmetric object properties as well as several constructs involving universal quantification.

OWL 2 QL

OWL 2 QL is based on DL-Lite [9] (which is actually not a single language fragment but rather a family of languages with several slight variations) and tailored for reasoning over large instance sets combined with a relatively inexpressive TBox. More specifically, many important reasoning tasks can be performed in logarithmic space with respect to the size of the ABox. The variant picked up in OWL 2 is DL-Lite$_R$ and supports property inclusion axioms. Other variants instead support (inverse) functionality on object properties. A notable feature goes hand in hand with DL-Lite's complexity results: Since query answering over knowledge bases has polytime data complexity and since it is possible to separate TBox and ABox reasoning for the evaluation of a query, it is possible to delegate the ABox reasoning to a standard SQL database engine. Moreover, increasing the expressiveness of the DL-Lite languages also increases space complexity to at least NLogSpace. S languages from the DL-Lite family, and thus also OWL 2 QL, are the maximally expressive Description Logics having the feature of being able to use database engines (along with all the optimizations employed in them) for query evaluation, and thus also support very efficient query answering for large instance sets. As such OWL 2 QL is optimized for data complexity.

3.3 WSML Core v2.0 Syntax Definition

WSML-Core 2.0 inherits the basics of the WSML syntax defined in [1] but restricts it in order to (i) facilitate a layering of more complex variants on top of it and (ii) ensure the tractability of the core language.

It adds the possibility of stating the equality of two individuals, which is required for many practical purposes and present in a lot of Web ontologies.

WSML-Core 2.0 inherits the namespace mechanism of WSML.

WSML-Core 2.0 restricts the use of identifiers and consequently the vocabulary of WSML-Core 2.0 is *separated* in distinct subsets.

A **WSML-Core 2.0 vocabulary** *V* in general has the following restrictions:

- V_C, V_D, V_R, V_I and V_{ANN} are the sets of concept, datatype, relation, instance and annotation identifiers. These sets are all subsets of the set of IRIs and are pairwise disjoint.
- The set of attribute names is equivalent to V_R
- The set of relation identifiers V_R is split into two disjoint sets, V_{RA} and V_{RC}, which correspond to relations with an abstract and relations with a concrete range, respectively.

The arguments of a datatype wrapper in WSML-Core 2.0 can only be strings, integers or decimals. From this arguments it is possible to compose more complex data types (i.e. dates) as specified in [1]. However, no other arguments, e.g. variables, are allowed for such data terms, and neither is it allowed to use complex data terms themselves as arguments (i.e. it is not possible to construct a date from several other complex time types).

3.3.1 Ontologies

In the following sections WSML-Core 2.0 elements are explained, many of which do need require fundamental changes from the current WSML-Core specification. In particular the basic syntax, usage of namespaces, identifiers, goals, Web services and Mediators remains the same. We explain the restrictions in regard to expressivity imposed by WSML-Core 2.0 on

- the conceptual syntax for ontologies,
- and the logical expression syntax,

which is required in order to ensure the compatibility with WSML-DL 2.0 and WSML-Rule 2.0 in the language stack.

3.3.1.1 Concepts

WSML-Core 2.0 poses a number of restrictions on attribute definitions. Most of these restrictions stem from the fact that it is not possible to express constraints in WSML-Core, other than for datatypes.

WSML-Core 2.0 does not allow for the specification of the attribute features **reflexive**, **transitive**, **symmetric**, **inverseOf** and **subAttributeOf**. This restriction stems from the fact that reflexivity, transitivity, symmetry and inverse of attributes are defined locally to a concept in WSML. It is however possible to express such properties by specifying global transitivity, symmetry and inverses of attributes (such as in most DLs) by defining appropriate axioms (see below).

3.3.1.2 Instances

In WSML-Core 2.0, allowed attribute values are restricted in order to disallow constructed data values (function symbols). In contrast to WSML-Core it is now possible to specify individual equality

3.3.1.3 Relations

WSML-Core 2.0 does not allow for the specification of relations. It is however possible to use

attributes, which from an expressivity point of view correspond to the subset of relations with an arity of two.

3.3.1.4 RelationInstances

Since WSML-Core 2.0 does not allow specifying relations, the use of relation instances is also forbidden.

3.3.1.5 Axioms

WSML-Core 2.0 does not impose restrictions on the specification of axioms, apart from the fact that WSML-Core 2.0 only allows the use of a restricted form of the WSML logical expression syntax specified later in this deliverable.

3.3.2 Goals in WSML-Core 2.0

Goals in WSML-Core 2.0 follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and definitions of non-functional properties are limited to WSML-Core 2.0 logical expressions.

3.3.3 Web Services in WSML-Core 2.0

Web Services in WSML-Core 2.0 follow the common WSML syntax. The logical expressions in the 'assumptions', 'preconditions', 'effects' and 'postconditions' of a capability and definitions of non-functional properties are limited to WSML-Core 2.0 logical expressions.

3.3.4 Mediators in WSML-Core 2.0

Mediators in WSML-Core 2.0 follow the common WSML syntax.

3.3.5 WSML-Core 2.0 Logical Expression Syntax

WSML-Core 2.0 allows only a restricted form of logical expressions. There are two sources for these restrictions. Namely, the restriction of the language to a subset of Description Logics restricts the kind of formulas which can be written down to the two-variable fragment of first-order logic. Furthermore, it disallows the use of function symbols and restricts the arity of predicates to unary and binary and chaining variables over predicates. The restriction of the language to a subset of Datalog with equality disallows the use of disjunction in the head of a rule and existentially quantified variables in the head of a rule.

Let *V* be a WSML-Core 2.0 vocabulary. Let further $\gamma \in V_C$, Γ be either an identifier in V_C or a list of identifiers in V_C, Δ be either an identifier in V_D or a list of identifiers in V_D, $\varphi \in V_I$, ψ be either an identifier in V_I or a list of identifiers in V_I, $p,q \in V_{RA}$, $s,t \in V_{RC}$, and *Val* be either a data value or a list of data values.

The set of atomic formulae in *L(V)* is defined as follows:

- γ **subConceptOf** Γ is an atomic formula in *L(V)*
- φ **memberOf** Γ is an atomic formula in *L(V)*
- γ[*s* **ofType** Δ] is an atomic formula in *L(V)*
- γ[*s* **impliesType** Δ] is an atomic formula in *L(V)*
- γ[*p* **impliesType** Γ] is an atomic formula in *L(V)*
- φ[*p* **hasValue** ψ] is an atomic formula in *L(V)*
- φ[*s* **hasValue** *Val*] is an atomic formula in *L(V)*
- $\varphi = \psi$ is an atomic formula in L(V)

This last atomic formula allows equality/unification in the head of a rule, which essentially requires an equality theory in the logic. This addition does not allow equalities to be derived in a very unintuitive way (i.e. based on functional attributes) but only allows it to be denoted explicitly. See [17] for a more in-depth discussion of this extension.

Let Var_1, Var_2, ... be arbitrary WSML variables. We call molecules of the form Var_i **memberOf** Γ *a-molecules*, and molecules of the forms, Var_i[*p* **hasValue** Var_k] and Var_i[*p* **hasValue** {Var_{k1}, Var_{kl}}] *b-molecules*, respectively.

In the following, *F* stands for an lhs-formula (i.e., a formula allowed in the antecedent, or *left-hand side*, of an implication), with the set of lhs-formulae defined as follows:

- Any b-molecule is an lhs-formula
- if F_1 and F_2 are lhs-formulae, then F_1 **and** F_2 is an lhs-formula
- if F_1 and F_2 are lhs-formulae, then F_1 **or** F_2 is an lhs-formula

In the following, *G,H* stand for rhs-formulae (i.e., formulae allowed in the consequent, or *right-hand side*, of an implication), with the set of rhs-formulae defined as follows:

- Any a-molecule is an rhs-formula
- if *G* and *H* are rhs-formulae, then *G* **and** *H* is an rhs-formula

Based on this, the set of WSML-Core 2.0 formulae can be defined as follows:

- Any atomic formula is a formula in *L(V)*.
- If F_1,...F_n are atomic formulae, then F_1 **and** ... **and** F_n is a formula in *L(V)*.
- Var_1[*p* **hasValue** Var_2] **impliedBy** Var_1[*p* hasValue Var_3] **and** Var_3[*p* hasValue Var_2] (globally transitive attribute/relation) is a formula in *L(V)*.
- Var_1[*p* **hasValue** Var_2] **impliedBy** Var_2[*p* hasValue Var_1] (globally symmetric attribute/relation) is a formula in *L(V)*.
- Var_1[*p* **hasValue** Var_2] **impliedBy** Var_1[*q* hasValue Var_2] (globally sub-attribute/relation) is a formula in *L(V)*.
- Var_1[*p* **hasValue** Var_2] **impliedBy** Var_2[*q* hasValue Var_1] (globally inverse attribute/relation) is a formula in *L(V)*.
- *G* **equivalent** *H* is a formula in *L(V)* if it contains only one WSML variable.
- *H* **impliedBy** *F* (and *F* **implies** *H*) is a formula in *L(V)* if all the WSML variables occurring in H occur in F as well and the *variable graph* of *F* is connected and acyclic.
- *H* **impliedBy** *G* (and *G* **implies** *H*) is a formula in *L(V)* if all the WSML variables occurring in H occur in G as well.

Here, the *variable graph* of a logical expression E is defined as the undirected graph having all WSML variables in E as nodes and an edge between Var_1 and Var_2 for every molecule Var_1[*p* **hasValue** Var_2].

Note that wherever an a-molecule (or b-molecule) is allowed in a WSML-Core 2.0 clause, compound molecules abbreviating conjunctions of a-molecules (or b-molecules, respectively) are also allowed.

3.4 Algorithmisation

In the following, we briefly sketch the translation of WSML-Core 2.0 to a suitable variant of Datalog, in order to facilitate common reasoning tasks, like i.e. consistency checking or entailment, which are of importance for applications on the Semantic Web and for dealing with Semantic Web Services.

Reasoning with WSML-Core can be done via a mapping to Datalog with (in)equality, as shown in [17]. This mapping defines a series of syntactical transformation steps in order to convert a WSML-Core 2.0 ontology into a Datalog program which is semantically equivalent. Different reasoning tasks are then mapped to queries by the underlying Datalog engine.

The following describes the steps required to transform a WSML-Core 2.0 ontology to Datalog. The conceptual layering of these steps is shown in ***Figure 2***. A complete definition of each of the steps is available in [18]

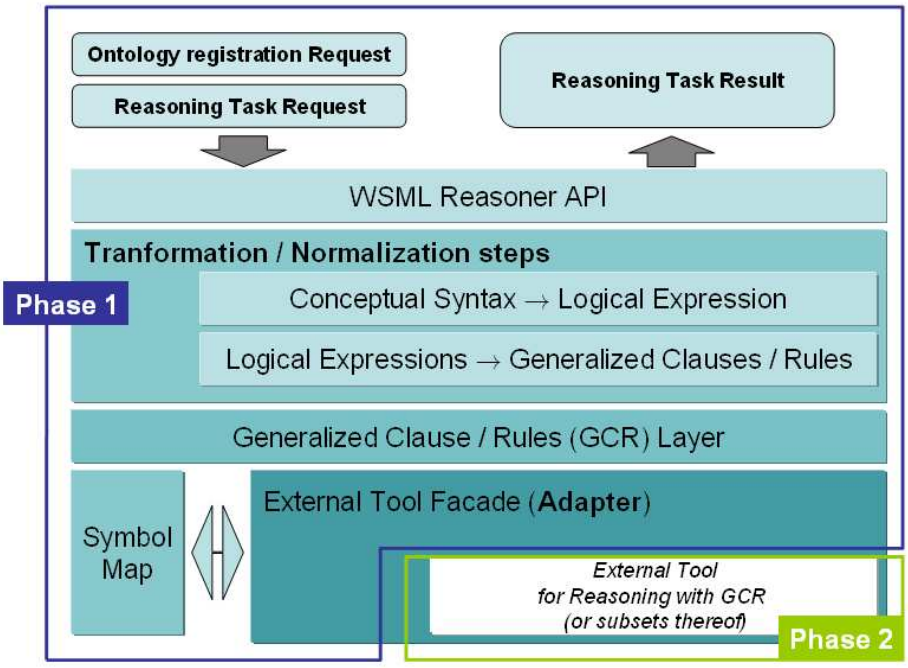

Figure 2 Conceptual Layering of Transformations

Axiomatization

In this initial step WSML ontologies are transformed to logical expressions conforming to the restrictions described in the previous section. Practically this means that elements specified in the high-level conceptual syntax are rewritten to corresponding axioms, as defined in the current version of WSML [1.]

Normalization.

Normalization is a mapping of the WSML logical expressions resulting from axiomatization, to a simplified set of formulae that is closer to the basic form of Datalog rules by

decomposinjg complex WSML molecules. After normalization, the resulting logical expressions follow the form of simple logic programming rules with no deep nesting of logical connectives.

Datalog rules generation

Finally, we convert the resulting logical expressions to a semantically equivalent Datalog program. Lloyd-Topor transformations, as defined in [19], flatten complex WSML logical expressions, producing simple rules accordingly. After this step, the resulting WSML expressions have the form of proper Datalog rules with a single head and conjunctive body literals.

As WSML-Core 2.0 does not include meta-modelling features, which are only covered by WSML-Flight 2.0 and WSML-Rule 2.0, a direct mapping from the WSML-specific language constructs, such as **subConceptOf**, **ofType, memberOf, ...** to Datalog is possible. In this case instances map to unary predicates, attributes mapping to binary predicates, and instances map to constants for the underlying rule engine.

This mapping is simpler and more efficient than the indirect mapping through special meta-level predicates (accompanied with meta-level axioms that capture the intended semantics for these predicates), which is required for the rule based WSML variants due to the meta-modelling features available there.

3.5 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.

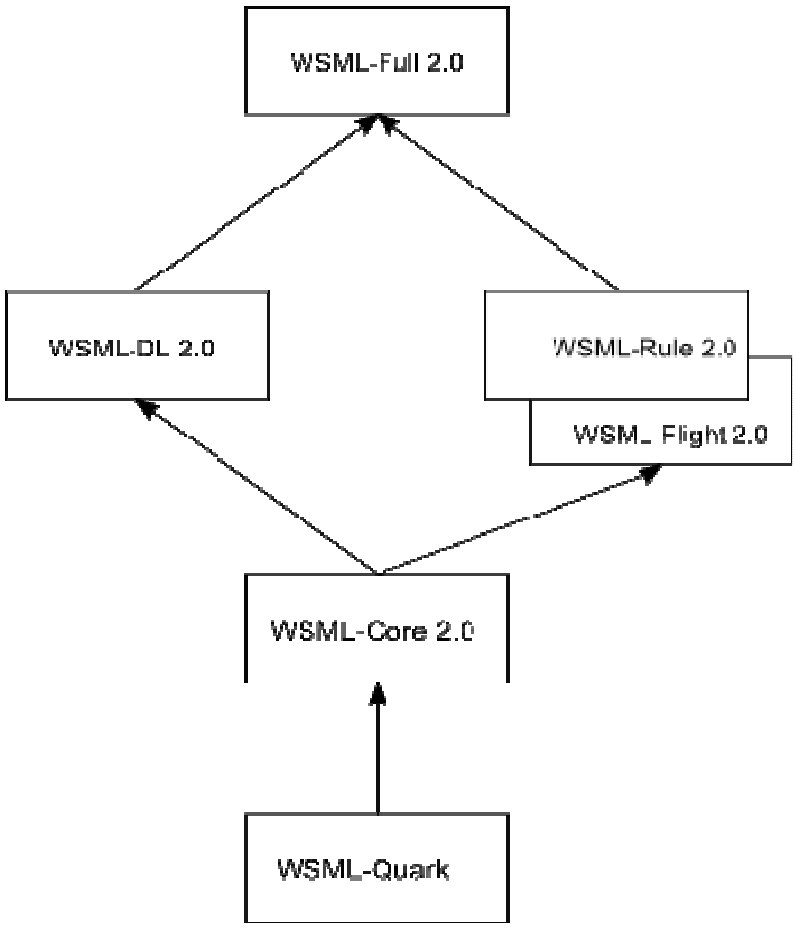

Figure 3 WSML Language Layering

WSML-Quark is a very lightweight and intuitive language variant that allows for the simple organization of concepts into 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 is the Description Logic variant of WSML, based on ELP [20], which is based on the tractable DL EL++ [10], and also 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 [21] 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. [22] defines the well-founded semantics for this approach, which still preserves tractable data complexity.

4. Conclusions

In this deliverable we presented a reworked version of WSML-Core, which is slightly more expressive in the sense that it allows individuals to be denoted as equal. This is motivated both by practical purposes (to state that two resources are in fact identical) and further due to the layering of WSML-DL 2.0 on top of WSML-Core 2.0, which requires this functionality. Thus, a reasoner implementation for the WSML-Core 2.0 language must implement this language feature, which does not increase the worst-case complexity of the language.

Subsequent deliverables D3.2.2, D3.2.3 and D3.2.4 will develop a prototype reasoning framework that is able to deal with all the WSML language variants via one common underlying inference engine. D3.2.2 is specifically for WSML-Core 2.0.

5. References

[1] J. de Bruijn, "WSML Language Reference, Deliverable D16.1v1.0," *WSML Final Draft 2008-08-08*, 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] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas, "The Summary Abox: Cutting Ontologies Down to Size," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4273, 2006, p. 343.

[17] V. Raphael, "Web Ontology Reasoning with Logic Databases ," Universität Karlsruhe (TH), 2004.

[18] S. Grimm, U. Keller, H. Lausen, and G. Nagypal, "A Reasoning Framework for Rule-Based WSML," *Semantic Web Research and Applications*, 2007.

[19] J.W. Lloyd and R.W. Topor, "Making PROLOG more expressive.," *Journal of Logic Programming*, vol. 1, 1984, pp. 225-240.

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

[21] 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.

[22] 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.