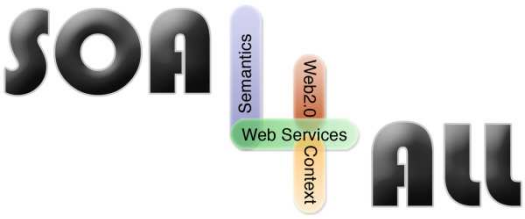

Project Number: **215219**
Project Acronym: **SOA4All**
Project Title: **Service Oriented Architectures for All**
Instrument: **Integrated Project**
Thematic Priority: **Information and Communication Technologies**

D3.1.1 Defining the features of the WSML-Quark language

Activity N:	Activity 2 - Core Research and Development	
Work Package:	WP3 - Service Annotation and Reasoning	
Due Date:		M12
Submission Date:		10/09/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Author(s):	Gulay Unel UIBK Uwe Keller UIBK Florian Fischer UIBK Barry Bishop UIBK	
Reviewers(s):	Jacek Kopecky UIBK Marin Dimitrov UIBK	

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	5/12/2008	Initial outline	Uwe Keller (UIBK)
0.2	30/01/2009	First complete draft with Algorithmisation	Gulay Unel, Florian Fischer (UIBK)
0.3	02/02/2009	Editing and minor corrections	Barry Bishop (UIBK)
0.4	04/03/2009	Corrections from reviewers	Gulay Unel (UIBK)
0.5	05/03/2009	Fixed errors in first 2 sections.	Florian Fischer (UIBK)
1.0	06/03/2009	Updates to section 3	Florian Fischer (UIBK)
Final	09/03/2009	Overall format and quality revision	Malena Donato (ATOS)

Table of Contents

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

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

This deliverable, along with others, 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-Quark, an ultra-lightweight 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":

- 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 [22] 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 based 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 completely new WSML variant called WSML-Quark, an ultra-lightweight language specifically designed for modelling classification systems only. As such it has the lowest expressivity of all the WSML variants. 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 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.

A lot of 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.

Section 3.4 describes a variety of reasoning techniques for WSML-Quark that all have sub-polynomial query time.

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 WSML-Quark language variant.

We describe the modelling elements in WSML-Quark, 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 reasoner 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 relationship of this document and the WSML language described to the SOA4All project and other deliverables. Section 3 defines the WSML-Quark 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 algorithmization of WSML-Quark, and clarifies the relationship of WSML-Quark to the other WSML language variants, and their layering. Section 3 concludes with some ideas for future work relating to implementations of reasoning algorithms and their re-use in other situations.

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***.

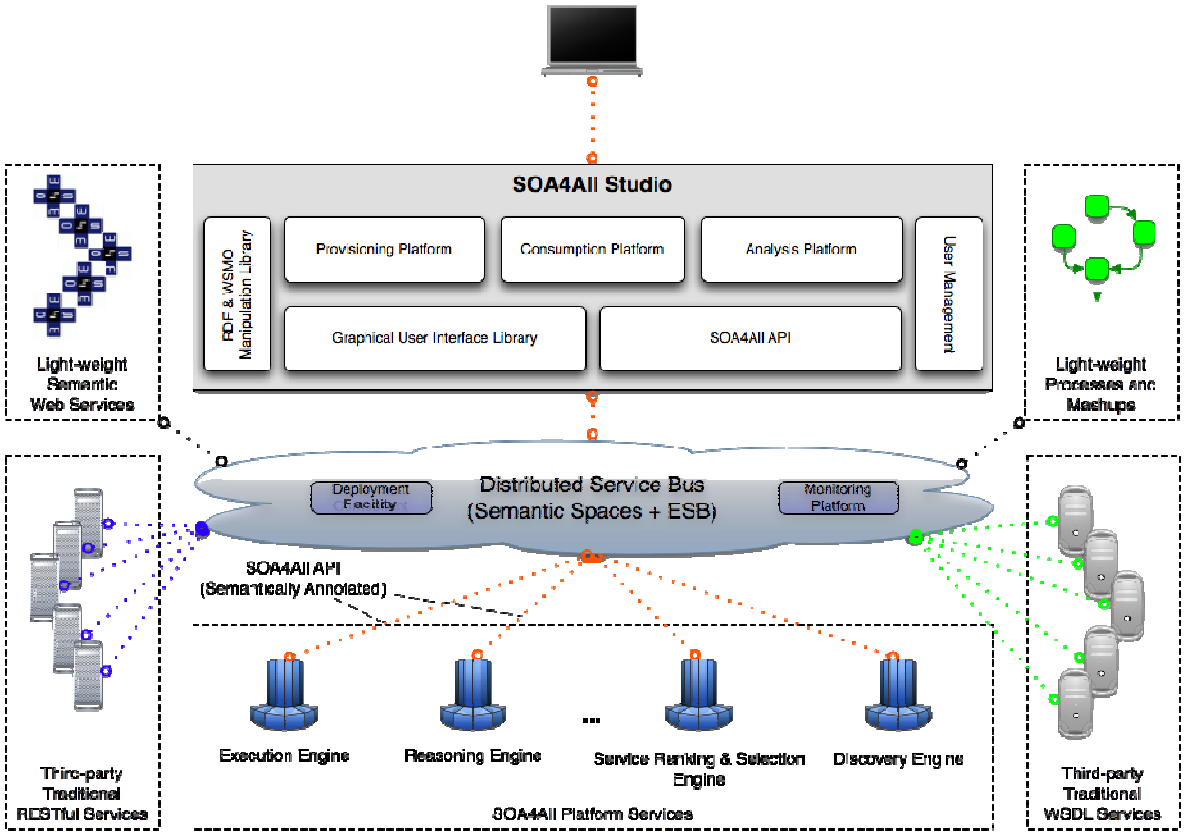

Figure 1 SOA4All Overall Architecture

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[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

[1] 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 Quark Language Definition

In this section, we define WSML Quark. We start with discussing the motivation for introducing such a lightweight language. Then we outline the related work in Section 3.2, define the syntax of WSML Quark in Section 3.3, explain the reasoning algorithms in Section 3.4, discuss the relation with other WSML variants and language layering in Section 3.5, and finally provide the conclusions and future work in Section 3.6.

3.1 Motivation

WSML-Quark embodies the most lightweight representation language in the WSML-family of languages. WSML-Quark is specifically designed to meet the representational needs of very lightweight knowledge-based systems typically considered in the context of the Web 2.0: applications which focus on the sharing of a large number of resources (such as documents, images, media files etc.) based on lightweight annotation with controlled vocabularies. As such WSML-Quark can be considered as a bridging technology between informal, unstructured metadata, which is usually the result of community-driven social process, i.e. so-called tagging, and more formal and rigours knowledge representation systems. These applications do not demand fully-fledged, expressive knowledge representation languages that are able to represent very detailed semantic interdependencies between the various concepts that are relevant to the problem domain. Instead, these applications are typically based on simple knowledge organization systems (SKOS) as described in [18]. Such simple knowledge organization systems essentially allow (a) the definition of concepts that can be used for classifying resources into groups (by means of a community-based tagging process) and (b) the organisation of these concepts into generalization / specialization structures [18]. At the same time, such applications are data-intensive, i.e. they deal with very large numbers of resources that are shared and hence retrievable within a community.

WSML-Quark strives for optimal support of these applications by maximally restricting the full WSML language to a minimal core of modelling primitives that (a) cover the expressive needs of these data-intensive applications and (b) allow for extremely scalable reasoning. All other WSML language variants extend WSML-Quark syntactically and semantically. Adding expressiveness allows support of applications with more advanced representational needs, but comes at the cost of more limited scalability. We further want to point out that a dedicated WSML-Quark inference system can be reused as a special-purpose subsystem in an inference engine for the more expressive WSML language variants to implement reasoning with a specific part of an ontology, the concept hierarchy, in the most efficient way possible.

A concept represents a meaningful unit of thought in a certain problem domain [18]. It can be used to organize items with similar features into classes. A particularly important class of resources to be considered in applications of WSML are Web services.

A concept is described by a unique identifier (IRI) and various types of meta-information (such as labels to be used in applications, a natural language specification, author information, version and so forth). A default set of description elements for concepts is defined in [18]. The WSML annotation mechanism can be used to integrate meta-data for concepts into semantic descriptions to be processed by applications. A particularly relevant example for such a meta-data systems is SKOS [18].

Concepts (or tags) in controlled vocabularies can be organized into hierarchical structures, so-called taxonomies, where concepts can specialize or generalize other concepts in a vocabulary. For two concepts C, D we can declare that C is a sub-concept of D, meaning that C specializes D. Semantically, the specialization relation is considered to be transitive, i.e. if C specializes D and D specializes E then it holds as well that C specializes E.

Specialization between concepts is expressed by the **subConceptOf** relation. This roughly allows the same modelling as skos:broader and skos:narrower in SKOS.

Sometimes, concepts cannot be organized into purely tree-like structures (such as taxonomies) in a natural way – the concept specialization / generalization relation over concepts may form a directed, acyclic graph (DAG) and some concepts might be connected by more than one specialization / generalization path. Tree-like structures are considered the most important special cases and implementations are free to implement special-purpose inference algorithms that take advantage of tree structures.

We do not consider the classification of resources as part of the WSML-Quark language, but leave this aspect to applications that use WSML-Quark. Consequently, we do not include any language primitives to define instances and relate them to concepts in WSML-Quark.

WSML-Quark therefore focuses on the key inference service required for any of the Web 2.0 applications mentioned in the beginning: terminological reasoning in simple knowledge organization systems. Applications can check if some concept specializes another one (within large taxonomic structures) or retrieve all sub- or super-concepts of a given concept. Using this key inference service, applications can subsequently retrieve all relevant instances for given concepts in a straightforward way using standard data management techniques or systems.

The rationale for leaving instance classifications out of the WSML-Quark language is as follows: It has been pointed out in [20] that when translating large existing taxonomies in different real-world domains (such as eClass [28] or UNSPSC[29]) to ontologies expressed in standard ontology languages such as OWL, the interaction between instance-level classification of instances (i.e. assigning a resource as a member of a specific concept) and the standard semantics of concept specialization relations can easily give undesired consequences to applications.

The main reason is the semantic interaction between instances that are classified as instances of certain concepts and the predefined transitive and reflexive semantics of sub-concept relations. The hierarchical structures in many real world taxonomies are weaker than sub-concept relations, i.e. for the intended applications of taxonomies concept specialization is not necessarily reflexive and in particular does not automatically cause instance sets for sub-concepts to be propagated to instance sets of super-concepts. By leaving the instance classification outside of WSML-Quark, applications have full freedom over how to propagate instance sets across the taxonomy. Similarly, [18] considers specialization / generalization relations in over controlled vocabularies that are more general than the sub-concept relations in typical ontology languages; here again, hierarchical relations focus on the terminological level and have no side effects at the instance level.

Similarly, SKOS limits this interaction as well. It does not cover the modelling of different types of hierarchical relation: for example, instance-class and part-whole relationships. Technically every SKOS concept is an instance of skos:concept, which is an OWL class. As such SKOS concepts are in OWL terms always individuals, which are connected by a special skos:broader relation which is not transitive in order to achieve the desired behaviour. So at a fundamental level, SKOS also limits itself to basically one kind of reasoning over hierarchies of related individuals, whereas WSML-Quark limits itself to reasoning over hierarchies of related concepts (classes in OWL's terms). Both leave extensions open in order to allow a degree of flexibility for specific application scenarios.

Note that in all WSML variants extending WSML-Quark, instance definitions are possible and have the standard semantic interaction with the sub-concept hierarchies.

3.2 Related Work and Background

As mentioned in the previous section, WSML-Quark supports applications based on simple knowledge organization systems that perform classification of entities into groups by defining concepts and generating generalization / specification structures over these concepts.
One of the most prominent examples of such systems is SKOS[18], an upcoming W3C standard.

Simple Knowledge Organisation Systems (**SKOS**) is a family of formal languages designed for representation of structured controlled vocabulary such as thesauri, classification schemes, taxonomies, subject-heading systems etc. The SKOS vocabulary is built upon RDF and RDFS and can be used to express the content and structure of a concept scheme as an RDF graph. The main objective of this representation is to enable easy publication of structured controlled vocabularies for the Semantic Web.

A usable mapping between WSML-Quark to SKOS, for the sake of basic interoperability, can trivially be established in the same way as i.e. for RDFS. SKOS does not define `skos:broader` as transitive, but neither as intransitive. So just as it is legitimate to re-interprete `rdfs:subClassOf` as `skos:broader` and RDFS classes as SKOS concepts, the same correspondence can be established between WSML-Quark concepts and their corresponding SKOS counterpart. Since the hierarchical relation holding between WSML-Quark concepts also denotes that each concept is a sub-concept of itself (it is reflexive), this also means that the resulting SKOS concept is bound to be broader than itself.

Apart from the systems designed exclusively for knowledge organization, reasoning on the organization of concepts can be performed on any available knowledge base that supports class and sub/super class relationship definitions. In addition, the WSML-Quark language can be used to model many real-world domains (such as eClass[28] or UNSPSC[29]).

3.3 WSML Quark Syntax Definition

WSML documents can be explicitly identified as being described using WSML-Quark by using the following WSML variant identifier:

http://www.wsmo.org/wsml/wsml-syntax/wsml-quark

3.3.1 WSML- Quark Syntax Basics

WSML-Quark inherits the basics of the WSML syntax specified in Section 2.1 of [21]. In this section we describe restrictions that WSML-Quark puts on the syntax basics.

WSML-Quark inherits the namespace mechanism of WSML.

WSML-Quark restricts the use of identifiers. The vocabulary of WSML-Quark consists only of concept identifiers and annotation identifiers. All other terms (i.e. datatype values or attribute identifiers) cannot be used in WSML-Quark.

Definition. A WSML-Quark vocabulary *V* has the following restrictions:

- V_C and V_{ANN} are the sets of concept identifiers and annotation identifiers. These sets are all subsets of the set of IRIs and are pairwise disjoint
- The set of identifiers V_{ID} is partitioned into V_C and V_{ANN}

The second property means that the sets of datatype wrappers V_D, datatype values V_{DV}, relation indentifiers V_R, instance identifiers V_I, anonymous identifiers V_A, object constructors V_O, function symbols V_F and variable identifiers V_V, are all empty and therefore can not be used in a WSML-Quark ontology.

3.3.2 WSML-Quark Ontologies

In this section we explain the restrictions on the WSML ontology modelling elements imposed by WSML-Quark. The restrictions posed on the conceptual syntax for ontologies are necessary because of the focus on representing concept hierarchies only.

3.3.2.1 Concepts

WSML-Quark allows the definition of concepts by means of an identifier, their super-concepts and annotation properties. It does not allow for the specification of attributes (and consequently disallows the specification of attribute features **reflexive**, **transitive**, **symmetric**, **inverseOf** and **subAttributeOf**, range restrictions and range constraints, and cardinality constraints).

Further, WSML-Quark does not allow cycles in concept hierarchies, i.e. that for two different concept identifiers C, D it holds that both C is a sub-concept of D and D is a sub-concept of C. This restriction stems from the intention of capturing concept hierarchies.

Please note, that we do not require the concept hierarchy to form a tree or forest, in other words to represent a taxonomy or a collection of taxonomies over the same concepts However, these two cases are considered as important special cases, and implementations are encouraged to take advantage of the specific interconnection structure to improve the scalability of inference over concept structures of this specific form even further.

Further, we encourage the use of SKOS vocabulary [18] in annotations of a given concept, e.g. to define multi-lingual labels and preferred labels for concepts. This enables SKOS aware tools to interpret certain meta-data elements about concepts in an ontology, for instance when rendering an ontology in some ontology editor.

3.3.2.2 Instances

WSML-Quark does not allow for the specification of instances

3.3.2.3 Relations

WSML-Quark does not allow for the specification of relations.

3.3.2.4 Relation Instances

WSML-Quark does not allow the specification of relation instances, as the use of relations is disallowed.

3.3.2.5 Axioms

WSML-Quark does not impose restrictions on the specification of axioms, apart from the fact that WSML-Quark only allows the use of a restricted form of the WSML logical expression syntax. These restrictions are specified below.

3.3.3 WSML-Quark Goals

Goals in WSML-Quark restrict the common WSML syntax: 'assumptions', 'preconditions', 'effects', 'postconditions' and ‘shared variables’ parts of a capability can not be used and definitions of non-functional properties are limited to WSML-Quark logical expressions.

3.3.4 WSML-Quark Web Services

Web services in WSML-Quark restrict the common WSML syntax: 'assumptions',

'preconditions', 'effects', 'postconditions' and 'shared variables' parts of a capability can not be used and the definition non-functional properties are limited to WSML-Quark logical expressions.

3.3.5 WSML-Quark Mediators

Mediators in WSML-Quark follow the common WSML syntax.

3.3.6 WSML-Quark Logical Expressions

WSML-Quark allows only a restricted form of logical expressions. The restrictions reflect the fact that WSML-Quark ontologies are intended to capture concept hierarchies only.

Let *V* be a WSML-Quark vocabulary. Let further $\gamma \in V_C$, Γ be either an identifier in V_C or a list of identifiers in V_C,

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

γ **subConceptOf** Γ is an atomic formula in *L(V)*

Definition: The set of WSML-Quark formulae is defined as follows:

Any atomic formula is a formula in *L(V)*.

If $F_1,...F_n$ are atomic formulae in *L(V)*, then F_1 **and** ... **and** F_n is a formula in *L(V)*.

3.4 Algorithmisation

Reasoning on hierarchical structures that use sub/super concept relationships based on the standard semantics comprises 1) checking if one concept is a sub-concept of another or 2) finding all the sub/super-concepts of a given concept. Hence, reasoning can be reduced to the graph reachability problem. Knowledge organization systems for Semantic Web data involve large graphs and require fast answering of reachability queries. In this section, we present an analysis of the algorithms proposed for solving the graph reachability problem.

Given a graph *G=(V, E)* where *V* is the set of vertices, *E* is the set of edges and $|V| = n$, $|E| = m$, there are two naïve approaches for answering reachability queries. One is to use the shortest path algorithm with *O(m)* query time. Another naïve approach to this problem is to precompute the reachability between every pair of vertices of a graph, so that reachability queries over this graph can be answered in constant time and requires $O(n^2)$ space. As can be seen from their time or space requirements these approaches are impractical for large graphs and in turn for Semantic Web data. Efficient solutions of this problem on large sparse graphs involve reachability labeling methods. Several approaches have been proposed to encode graph reachability information using labeling schemes [22][23][24][25][26]. A labeling scheme assigns labels to vertices of the graph and answers a reachability query over two vertices by comparing the labels of the vertices. Interval-based labeling is used for tree structures that can answer reachability queries in constant time. However, the time complexity of this method is *O(m)* for graphs. Cohen et al [23] proposed a 2-hop labeling scheme which uses $O(nm^{1/2})$ storage and $O(m^{1/2})$ time. Indexing (labeling) time for this method is $O(n^4)$ which then reduced to $O(n^3)$ by the HOPI algorithm proposed by Schenkel et. Al [24][25]. The last method is called dual labeling by Wang et al. [26] which is based on representing a graph with two components: a spanning tree and a set of *t* non-tree edges. For sparse, tree-like graphs, it is assumed that $t<<n$. The two components together contain the complete information needed to answer a reachability query over the original graph. The dual labeling method integrates interval-based labeling, which encode reachability in the spanning tree and non-tree labeling to complete the reachability information of the graph. This method consists of two schemes Dual-I and Dual-II. The Dual-I scheme has constant query time, whereas it is *O(log t)* for Dual-II. Both schemes have $O(n + t^2)$ space complexity, however Dual-II uses less space in practice. Table 1 summarizes the complexity results for

the methods mentioned in this section.

Based on a comparison of graph reachability algorithms we can conclude that choosing the best algorithm for a particular application depends on the properties of the graph structures used. In addition, the trade off between time and space complexities should also be considered. Dual-I method can be used in applications supporting large semantic datasets for fast query answering. It is also possible to design algorithms tailored for the specific properties of each graph structure.

	Query Time	Index Time	Index Size
Shortest Path	*O(m)*	*0*	*0*
Transitive Closure	*O(1)*	$O(n^3)$	$O(n^2)$
Interval	*O(n)*	*O(n)*	$O(n^2)$
2-Hop	$O(m^{1/2})$	$O(n^4)$	$O(nm^{1/2})$
HOPI	$O(m^{1/2})$	$O(n^3)$	$O(nm^{1/2})$
Dual-I	*O(1)*	$O(n+m+t^3)$	$O(n+t^2)$
Dual-II	*O(log t)*	$O(n+m+t^3)$	$O(n+t^2)$

Table 1: Complexity comparison

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 2***. 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.

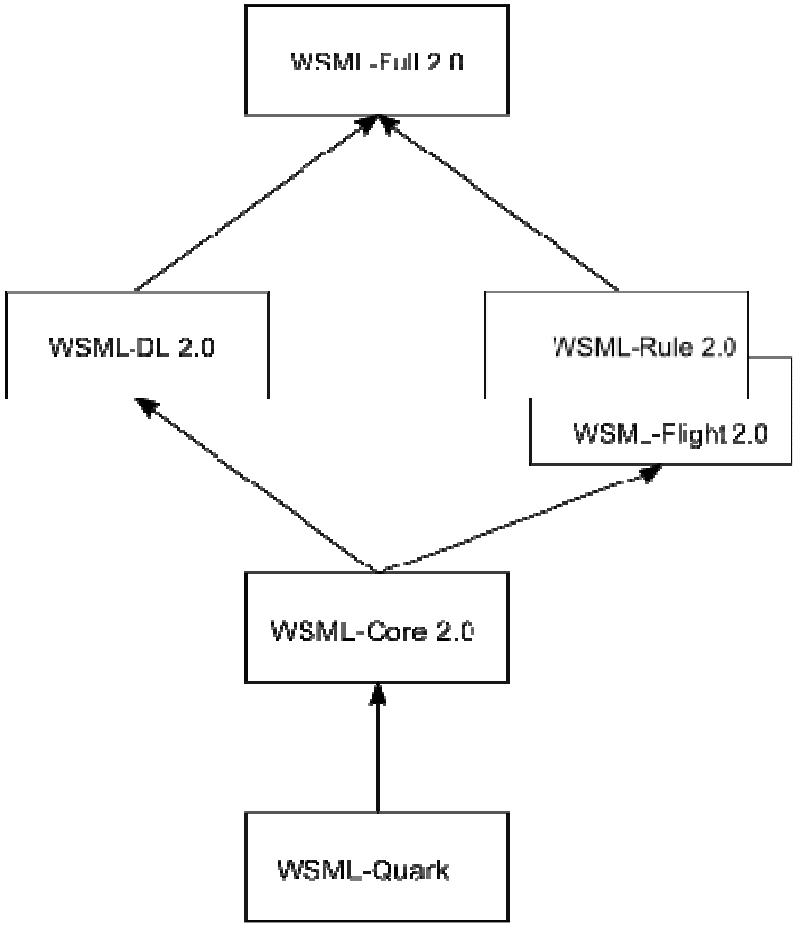

Figure 2 WSML Language Layering

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 is the Description Logic variant of WSML, based on ELP [16], 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 [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.

3.6 Conclusions and Future Work

In this document we presented WSML-Quark which is a representation language designed to meet the needs of lightweight knowledge-based systems. The typical constructs used in such systems are concepts and the sub/super-concept relationships between them. Reasoning is typically based on representing the hierarchical structure of concepts as a graph and performing traversal and answering reachability queries on it.

As future work, we plan to investigate reasoning techniques based on the properties of graphs representing hierarchical structures. We plan to propose methods for a reasoner component that uses different (and possibly hybrid) reasoning strategies that are dependent on the input graph and time/space requirements. We believe that building such an optimized reasoner component can support reasoning on Semantic Web data organization schemes even better than the proposed algorithms.

4. References

[1] D. Roman, H. Lausen, and U. Keller, “Web Service Modeling Ontology (WSMO),” *WSMO Working Draft*, 2004. Available at http://www.wsmo.org/TR/d2/v1.3/

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

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

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

[5] 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.

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

[7] U. Hustadt, B. Motik, and U. Sattler, “Data Complexity of Reasoning in Very Expressive Description Logics.”

[8] 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.

[9] F. Baader, S. Brandt, and C. Lutz, “Pushing the EL Envelope”, *International Joint Conference on Artificial Intelligence, Lawrence Erlbaum Associates Ltd*, 2005, p. 364.

[10] B. Grosof, I. Horrocks, R. Volz, and S. Decker, “Description Logic Programs: Combining Logic Programs with Description Logic”.

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

[12] 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.

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

[14] British Telecom, “Web21C SDK.” http://web21c.bt.com/

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

[16] 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.

[17] 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.

[18] A. Miles and S. Bechhofer, “SKOS Simple Knowledge Organization System Reference”, *World Wide Web Consortium, Working Draft WD-skos-reference-20080829*, 2008. Available at: http://www.w3.org/TR/2008/WD-skos-reference-

20080829/

[19] K. Watanabe, “Introduction of Dublin Core metadata”, *Journal of Information Processing and Management,* 43, 2001.

[20] M. Hepp, “Products and Services Ontologies: A Methodology for Deriving OWL Ontologies from Industrial Categorization Standards”, *in: Int'l Journal on Semantic Web & Information Systems (IJSWIS)*, Vol. 2, No. 1, pp. 72-99, January-March 2006.

[21] J. de Bruijn, Editor, “WSML Language Reference, Deliverable D16.1v1.0”, WSML Final Draft 2008-08-08, Available at http://www.wsmo.org/TR/d16/d16.1/v1.0/20080808/.

[22] R. Agrawal, A. Borgida, and H. V. Jagadish, “Efficient management of transitive relationships in large data and knowledge bases”, In *SIGMOD*, pages 253–262, 1989.

[23] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick, “Reachability and distance queries via 2-hop labels”, In *Proceedings of the 13th annual ACM-SIAM Symposium on Discrete algorithms*, pages 937–946, 2002.

[24] R. Schenkel, A. Theobald, and G. Weikum, “HOPI: An efficient connection index for complex XML document collections”, In *EDBT*, 2004.

[25] R. Schenkel, A. Theobald, and G.Weikum, “Efficient creation and incremental maintenance of the HOPI index for complex XML document collections”, In *ICDE*, 2005.

[26] Haixun Wang, Hao He, Jun Yang, Philip S. Yu, and Jeffrey Xu Yu, “Dual labeling: Answering graph reachability queries in constant time”, In *ICDE ’06: Proceedings of the 22nd International Conference on Data Engineering (ICDE’06)*, page 75, 2006.

[27] International Standard for the Classification and Description of Products and Services, “eClass”, http://www.eclass-online.com/

[28] The United Nations Standard Products and Services Code (UNSPSC), http://www.unspsc.org/