Project Number:	**215219**
Project Acronym:	**SOA4ALL**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D3.2.1 Framework and APIs for integrated reasoning support

Activity N:	A2 Core R&D	
Work Package:	WP3 Service Annotation and Reasoning	
Due Date:		31/08/2008
Submission Date:		26/08/2008
Start Date of Project:		01/03/2006
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Author(s):	Florian Fischer Barry Bishop	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
	Dissemination Level	
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	25/06/2008	Initial Content	Florian Fischer
0.2	05/07/2008	Architectural section and use-cases	Florian Fischer
0.3	10/07/2008	Requirements and reasoning tasks	Florian Fischer
0.4	15/07/2008	API specification	Florian Fischer
0.5	28/07/2008	Revised requirements section	Florian Fischer
1.0	22/08/2008	Review	Sudhir Agarwarl Mihail Konstantinov
	22/08/2008	Changes according to reviewer comments	Florian Fischer

Table of Contents

EXECUTIVE SUMMARY **5**
1. INTRODUCTION **6**
1.1 PURPOSE AND SCOPE 7
1.1.1 Audience *7*
1.1.2 Scope *7*
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 *10*
3. API SPECIFICATION **12**
3.1 REQUIREMENT ANALYSIS 12
3.1.1 Possible Use-Cases *12*
3.1.2 Reasoning Tasks *13*
3.2 FRAMEWORK COMPONENTS 14
3.3 API DEFINITION 16
3.3.1 General Interface *18*
3.3.2 Language specific methods: *20*
4. CONCLUSIONS **22**
5. REFERENCES **23**

Table of Figures

Figure 1 SOA4All Semantic Service Bus 9
Figure 2 Conceptual Layering 15
Figure 3 High Level Components 16
Figure 5 General Methods 17
Figure 6 DL-Reasoner specific Methods 21
Figure 4 Logic Programming specific Methods 21

© SOA4ALL consortium

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

This deliverable has two main purposes: First, it is supposed to serve as a requirements document for the integrated reasoner component in SOA4ALL. Secondly, and based on this requirements analysis, it will define a high-level API for the required infrastructure to support reasoning with service annotations by taking the existing WSML2Reasoner tool as a baseline to evolve from.

The result of this deliverable is the definition of an high-level API for the reasoning component, which facilitates relevant reasoning tasks in an optimized manner for the underlying formalisms WSML-Core, WSML-Rule and WSML-DL in order to meet the scalability goals of SOA4LL.

Deliverable D3.2.1, situated at the base-layer of the SOA4ALL architecture, is of interest to various other components in the *Web Enabled Service Platform* layer and more particularly for developers of components in work packages WP5 *Service Location and WP6 Service Construction*, for which reasoning is basic infrastructure in the process of service discovery and composition. Furthermore, all use-cases (WP7, WP8 and WP9) have certain dependencies on the reasoning component.

1. Introduction

SOA4All's aim is to facilitate a web on 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 world wide scale.

  Web 2.0 as a means to structure human-machine cooperation in an efficient and costeffective 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 and is for example needed to support meaning based service location and discovery, context for direct reasoning over services and to state relationships between services. Based on this, by using ontologies as semantic data model, services gain machine-understandable annotations, which can be further processed by logical inference. This combination makes the development of high quality techniques for automated selection, construction, etc. possible. Furthermore, precise formal models allow expressing context-specific rules and constraints which can be taken into account during the inference process.

Existing solutions such as OWL-S [1] and WSMO [2] can be considered as rather heavy-weight solutions and follow a top-down approach to modelling of services. This means that they imply a initial modelling of the semantics of a service in terms of ontologies, funcitional and non-functional descriptions and behavioural semantics, which are then grounded for example in WSDL and SOAP. This approach is difficult to apply to existing services (in a bottom up fashion).

Alternatively, WSML-Lite [3] as an outcome of SOA4ALL will provide a lightweight solution that is in line with the goal to realize a web of services that encompasses billions of services and can be applied to enhance existing services easily. This approach is layered on SAWSDL [4], which defines extensions for WSDL and XML Schema that can be used to link WSDL components to semantic description. In this way, SAWSDL forms the foundation for bottom-up modelling and is fundamentally independent of a particular semantic formalism.

SAWSDL allows WSDL components to be annotated semantically with three extension attributes:

- modelReference allows to denote concepts that describe a particular component.
- loweringSchemaMapping / liftingSchemaMapping denote the mappings between XML and a specific semantic formalism.

Based on this, WSMO-Lite is a concrete service ontology that fills the semantic layer on top of SAWSDL annotations and that allows a formal and precise semantic definition.

For the purpose of reasoning about these formal definitions this deliverable forms a basic building block in the SOA4ALL architecture by defining a high-level API for the reasoning framework that will be realized in deliverables D3.2.2, D3.2.3 and D3.2.4 as prototype reasoners, that support WSML-Core, WSML-Rule and WSML-DL. WSML [5] is a family of languages taking Description Logics [6], Logic Programming [7] and First-Order Logic [8] as its semantic basis. Furthermore, it has been influenced by F-Logic [9] and frame-based

representation systems. While the respective dialects will be described in more detail in the respective prototype deliverables (D3.2.2 – D3.2.4), it is clear that particularities of the different variants have to be taken into account even within the scope of defining an initial high-level API.

Furthermore, we note that the new version of the WSML specification[1] is currently being developed which is to a certain degree incompatible with the current version. Therefore, we anticipate minor changes to the API at later stages in the project.

1.1 Purpose and Scope

1.1.1 Audience

This document is intended to give an introduction to using the reasoner component and its application programming interface (API) and serve as a high level design document (HLDD).

Thus its main audience are consortium partners involved in technical work packages that

- rely on the functionality of the reasoning component (Activity cluster A2 - Core R&D Activities), and A1,
- provide conceptual fundamental work and infrastructure (A1 - Fundamentals and Integration Activities),

as well as use-case partners that work on practical showcases (A3 - Use Case Activities). Furthermore, while it does not replace in-depth software documentation, it can serve as a “non-technical” guide to the basic usage of the reasoner API for developers in general.

1.1.2 Scope

This deliverable has two main purposes: First, it is supposed to serve as a requirements document for the integrated reasoner component in SOA4ALL. Secondly, and based on this requirements analysis, it will define a high-level API for the required infrastructure to support reasoning with service annotations by taking the existing WSML2Reasoner[2] framework as a baseline. Reasoning over a precise logical formalism is a fundamental requirement in order to make use of the implicit information available in service annotations. Thus ontology reasoning is required to realize a multitude of higher-level tasks.

The result of this deliverable is the definition of an high-level API for the reasoning component, which facilitates relevant reasoning tasks in an optimized manner for the underlying formalisms WSML-Core, WSML-Rule and WSML-DL in order to meet the scalability goals of SOA4LL.

Deliverable D3.2.1, situated at the base-layer of the SOA4ALL architecture, is of interest to various other components in the *Web Enabled Service Platform* layer and more particularly for developers of components in work packages WP5 *Service Location* and WP6 *Service Construction,* for which reasoning is basic infrastructure in the process of service discovery and composition.*Furthermore* all the use-cases (WP7, WP8 and WP9) have certain dependencies on the reasoning component.

[1] http://www.wsmo.org/TR/d16/d16.1/v0.3/

[2] http://tools.sti-innsbruck.at/wsml2reasoner/

© SOA4ALL consortium

1.2 Structure of the document

The remainder of this deliverable is structures as follows: Chapter 2 clarifies the relation of this document and the reasoner component in relation to the SOA4All infrastructure and other deliverables. Chapter 3 defines the reasoner API by first analysing requirements and different reasoning tasks in Section 3.1. It then proceeds with a conceptual overview of the framework in Section 3.2 and specifying the supported functionality of the API in Section 3.3. Finally Chapter 4concludes this document and outlines the next steps towards reasoner prototype implementations in subsequent deliverables

2. Technical deliverable remarks

2.1 Deliverable relation with the architecture of the project

The reasoner component that will result from efforts in WP3 is situated at 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, 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 as purple "envelop" of the other components and also shows the possibility to be connected with other buses as extension.

Obviously, the reasoner has to be integrated with the infrastructural backbone. Since a number of components in the Web Enabled Service Platform stemming from other work packages (For example in the scope WP5 and more particularly D5.3.1, which aims at the of fundamental reasoning techniques for Web Service discovery) depend on the reasoner is is consequently positioned in the Base Layer of the SOA4All architecture.

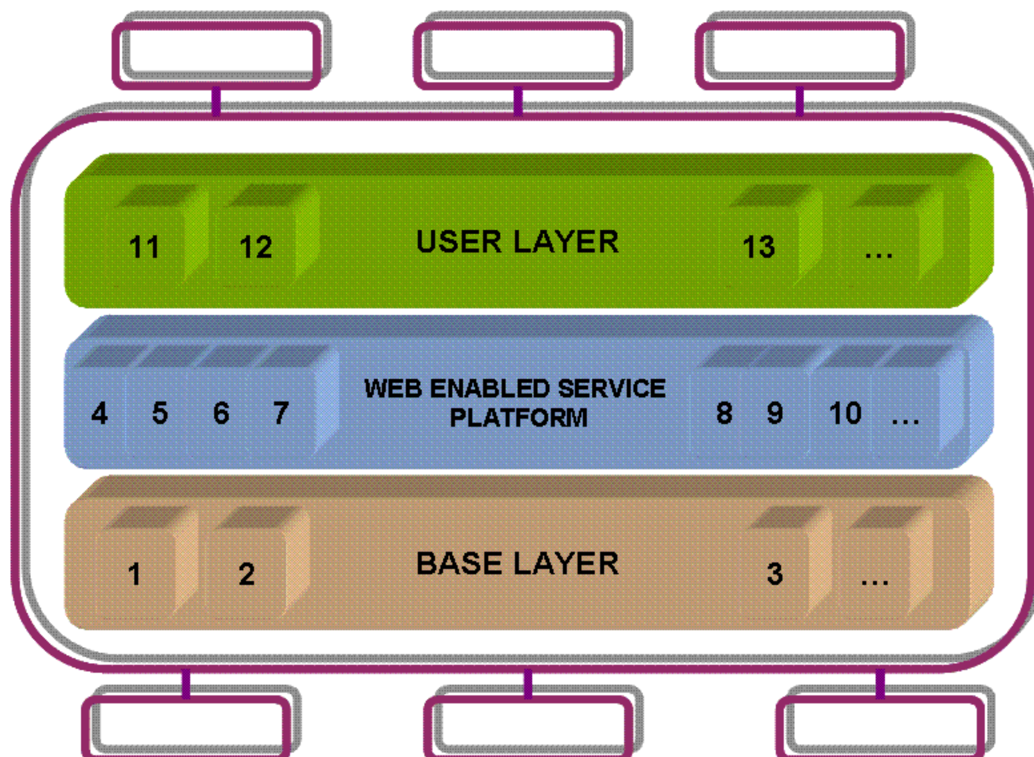

Figure 1 SOA4All Semantic Service Bus

2.2 Deliverable relation with the use-cases

This section serves two distinct purposes. Primarily it clarifies the relation of the reasoner component with the ongoing use-case activities in SOA4ALL. Furthermore, it also identifies technical challenges in the use-cases that result in relevant tasks for the requirements analysis in Section 3.1.

2.2.1 End-user Integrated Enterprise Service Delivery Platform

D3.2 is of basic interest for WP7, the End-user Integrated Enterprise Service Delivery Platform case study, as this use case will fully use service annotation and reasoning about annotations.

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[3] from services and reuse services in a visual tool without 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 annotation specifications are helpful, since they can consider the **user's** context, which is possibly also specified in a formal ontology. This facilitates a **personalized search**. These requirements also entail dependencies on the work in WP5 on **service discovery** and thus also on the reasoner component. Additionally, a further issue addressed in this use-case is the **integration** of different service interface descriptions.

2.2.2 W21C BT Infrastructure

This use-case will create a semantically enhanced and expanded version of BT's Web21c platform [10], which will result in a framework for the delivery of service, both by BT itself and third parties. Now 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.

Based on that, formal semantics and thus reasoning can enable new **composition** tools that will enhance and aide in the creation of more complex services. Composition will also take **contextual knowledge** from various sources into account, which will result in better suggestions for a user over time and thus simplify the **selection** of services for him.

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.

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

© SOA4ALL consortium

This use-case again entails several tasks that are based on annotation and reasoning. Among them easy **composition** of services, service **context, service location, ranking** and **selection** in the case of similar services and in this sense the scenario demonstrates almost all parts of the SOA4ALL concept including service discovery, integration, etc.

3. API Specification

3.1 Requirement Analysis

In this section, we first outline possible use-cases for the reasoning component within SOA4ALL based on the previous section. Furthermore, we identify how these use-cases map to basic reasoning task within the WSML language family. Consequently, this requirement analysis forms the foundation to identify the required functionality in terms of an initial high-level API.

3.1.1 Possible Use-Cases

This section characterizes several high level use-cases for the reasoner in a semi-formal and very general way independent of a particular logical formalism, although with a slight emphasize on the common Description Logic style of modeling knowledge, which is a cornerstone of the Semantic Web. In this sense we will also use the terms *terminology*, *ontology*, *concept* and *instance* as follows: a terminology is set of concept-definitions organized in a subsumption hierarchy; Furthermore, *roles* denote relations between objects (instances or concepts). The latter can also be considered as attributes attached to objects. However, very roughly the same operations that apply to concept hierarchies also carry over to role hierarchies.

Instances are member of such concepts. Ontologies consist of terminologies and instance sets.

At a lower logical level concepts are designated by unary predicate symbols and represent classes of objects sharing some common characteristics, that means they can be interpreted as a set of objects. Roles are designated by binary predicate symbols and are interpreted as (binary) relations between objects. Furthermore, in a sense also subsumption is a special binary relation between concepts. Based on these primitives it is usually possible to build more complex concept descriptions from various sub-expressions, using specific allowed constructors. A detailed treatment of this style of knowledge representation is available in [11] and [6].

Search

Search is a mapping which assigns a set of instances for each concept c and each ontology O=(T,A) where T is a terminology and A is a set of instances.

Browse

Browsing is a variant of searching with the difference that its output can either be a set of instances (as in search), or a set of concepts that allows for further browsing. In turn browsing is a mapping which assigns a set of individuals or a set of concepts for each concept c and each ontology O=(T,A) .

Web Service Selection

Rather than only searching for static material such as text and images, the aim of semantic web services is to allow searching for active components, using semantic descriptions of web-services. This is similar to search in general with the difference that the instance set consists of services and that a query is concerned with a particular functionality rather than content.

Web Service Composition

Going beyond selection is to compose from a set of service a single new service with a specific functionality. Again the query describes this desired functionality.

Integration

The goal of integration is to take multiple instance sets, each organized in their own ontology, and to construct a single, merged instance set, organized in a single, merged ontology. This implies a mapping which assigns a (new) ontology to a set of ontologies.

Personalization

Personalization consists of taking a (large) data set plus a personal profile, and to return a (smaller) data set based on this user profile. The profile which characterizes the interests of the user can be in the form of a set of concepts, or a set of instances (e.g. typical recommend services at on-line shops use previously bought items, which are instances, while news-casting sites typically use general categories of interest, which are concepts).

3.1.2 Reasoning Tasks

In this section we show a limited number of basic reasoning tasks that can be used, either individually or as a combination, to facilitate the typical higher level use-cases from the previous section. These reasoning tasks are by no means the only possible combination and can often be reduced to each other. For example classification could be reduced to repeated subsumption checks. A more in-depth treatment of relevant reasoning tasks, with an emphasize on DL reasoning, is available in [6] as well.

Realization

Realization is the task of determining of which concepts a given instance is a member.

Subsumption

Subsumption is the task of determining whether one concept is a subset of another:

Classification

Classification is the task of determining where a given class should be placed in a subsumption hierarchy.

Retrieval

Retrieval is the inverse of realization.

Entailment

In general, entailment means to check whether some formula is a logical consequence of the knowledge in ontology.

Consistency Checking

This reasoning tasks means to check whether ontology is consistent in itself and does not contain contradictory information. This task can also be applied to concepts and logical formulas at a lower level. In general this is facilitated by checking for satisfiability.

Based on these tasks we can identify a set of methods that need to be supported by the API of the reasoner component in order to provide the desired level of functionality:

- Check Ontology, Concept or Logical Expression consistency.
- Get all concepts, instances or attributes from the ontology.
- Get all constraint or inferring attributes from the ontology.
- Get all subconcepts or superconcepts of a specified concept.
- Get all direct subconcepts or superconcepts of a specified concept.

- Check if a specified concept is a subconcept of another specified concept.
- Check if a specified instance is a member of a specified concept.
- Get all instances of a specified concept.
- Get all direct concepts a specified instance is member of.
- Get all (direct and indirect) concepts a specified instance is member of.
- Checking entailment of a logical formula or a set of logical formulas.
- Get all instance or datavalue ranges from a specified inferring or constraint attribute.
- Get all inferring or constraint attributes with the corresponding values or datavalues.
- Get all sub- or superrelations (-attributes) of a specified relation(attribute).
- Get all direct sub- or superrelations (-attributes) of a specified relation (attribute).
- Get all relations(attributes) inverse to a specified relation(attribute).
- Execution of arbitrary legal (depending on language variant) queries based on logical expression.

3.2 Framework Components

Figure 3 depicts the most high-level components of the reasoner. The top-level interface `WSMLReasoner` contains functionality common to all covered WSML language variants and acts as an umbrella API for all methods where this is effectively possible. Methods specific to language variants are situated in the `DLReasoner` (for WSML-DL) and `LPReasoner` (for WSML-Rule) interfaces. As WSML-Core is situated at the intersection of the two former language dialects it does not require a specific interface on its own.

Instead of implementing new reasoners, existing reasoner implementations can be used for WSML through a wrapper that translates WSML expressions into the appropriate syntax for the reasoner. This wrapper contains various validation, normalization and transformation functionalities that are reusable across different WSML variants. The advantages of this approach over standalone reasoners (e.g. Pellet or Kaon2) are 1.) that it is possible to use multiple implementations in a plug-able way, 2.) that it is possible to combine LP and DL specific reasoning tasks in one common method and thus abstracting from the underlying formalism.

Required transformations for Description Logic or Logic Programming (based on a transformation to Datalog [12] rules) then take place in the `DLBasedWSMLReasoner` and `DatalogBasedWSMLReasoner` implementations.

Conversions into the representation required by a concrete reasoning engine are then implemented in a special facade per external engine. In this way, the resulting framework is highly modular and multiple engines can used for specific tasks. This conceptual layering is depicted in **Figure 2**.

These conversion steps will be covered in more detail in each of the subsequent reasoner prototype deliverables for WSML-Core, WSML-Rule and WSML-DL.

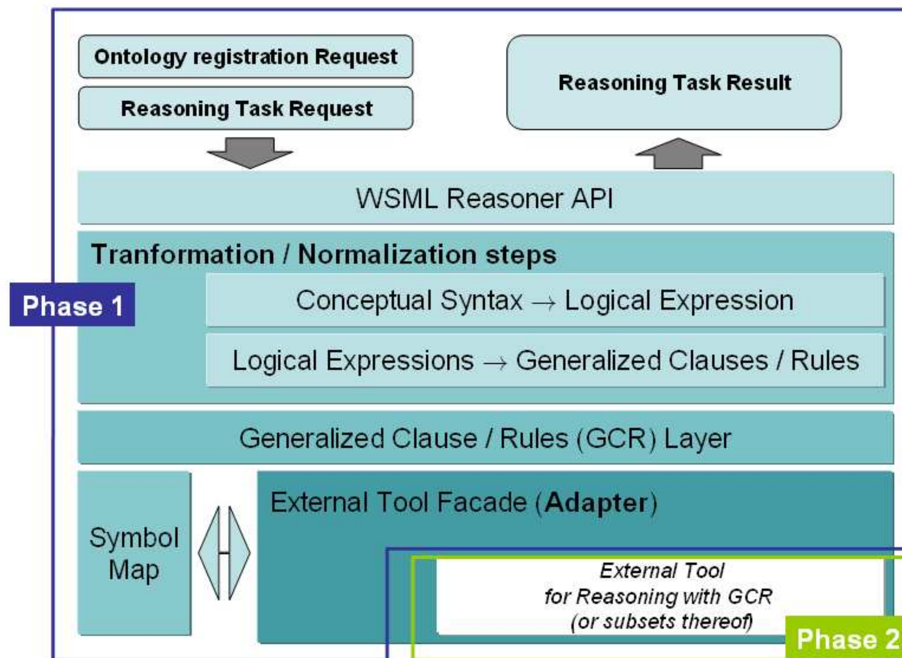

Figure 2 Conceptual Layering

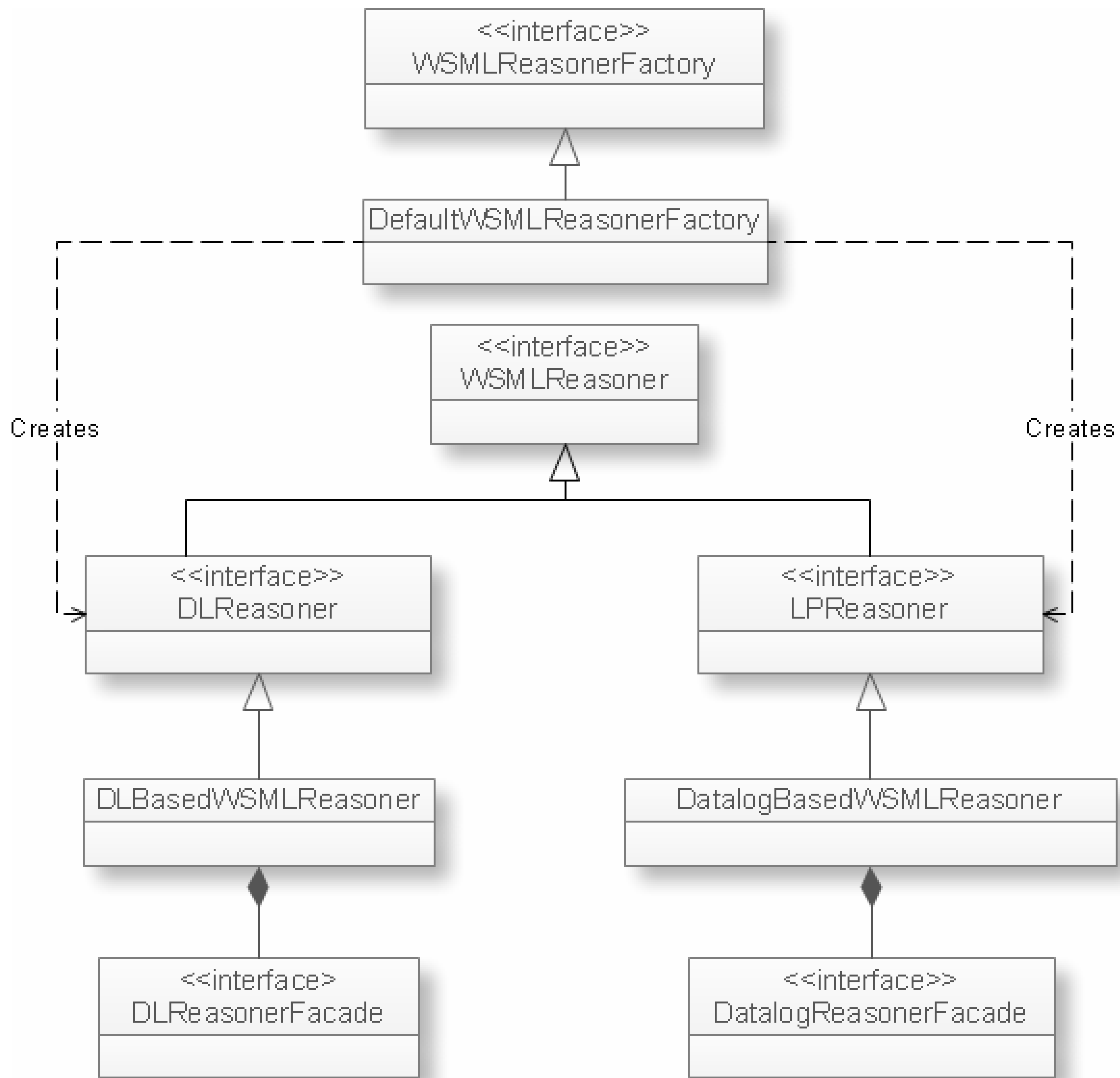

Figure 3 High Level Components

3.3 API Definition

The purpose of this section is to provide a high-level and implementation language independent definition of the functionality the reasoner component provides in the scope of the SOA4All architecture. These methods are split according to their generality and in order to point out differences according to the used WSML variant. Where possible anticipated changed according to the WSML 1.0 (draft at the time of writing) are already taken into account, however in general the stable version of the WSML specification is used as main reference and thus minor changes are possible

The reasoner API has a dependency on wsmo4j[4], which it uses to programmatically handle ontology objects in memory, parsing and syntactic validation of ontology documents etc.

[4] http://wsmo4j.sourceforge.net/

Figure 4 General Methods

© SOA4ALL consortium

3.3.1 General Interface

Figure 4 depicts general-purpose methods that are not tied to a specific language variant. It is possible to separate them roughly according to their functionality:

1. **Ontology Handling:** Before reasoning can be performed, the respective ontology has to be registered with the reasoner component. If the ontology is already registered, updates the ontology content. Registering an ontology with a reasoner instance implies several axiomatization (e.g. all conceptual syntax elements, such as concept and attribute definitions or cardinality and type constraints, are to logical expressions), normalization and further language specific transformation steps depending on the underlying WSML variant. Moreover, the result of this “pipeline” process is converted to the required format for the underlying reasoning engine (IRIS, Pellet, etc.) and the original parameter object is not altered. A closer look at these transformations (for rule based WSML variants) is for example presented in [13].

   In turn updating can either mean: 1.) Updating an ontology incrementally if the underlying reasoning engine permits this, or 2.) Resetting the underlying reasoning engine and registering the ontology from scratch again if necessary.

   Furthermore, unless indicated by the method name, registration also implies to check whether an ontology is consistent in itself and does not contain contradictory information. This is a semantic check on top of the syntactic validation performed within wsmo4j.

   Several ontologies can be registered at the same time, and reasoning is performed with all the registered ontologies as once. This resembles the **importsOntology** keyword. The following methods can be used for handling ontologies:

   ```
   registerOntologies(Set_of_Ontologies)
   registerOntology(Ontology)
   registerOntologyNoVerification(Ontology)
   registerEntitiesNoVerification(Set_of_Entities)
   registerEntities(Set_of_Entities)
   ```

   In these entities can be arbitrary ontology elements (see the wsmo4j programmer’s guide[5] for more details).

   ```
   deRegister()
   ```

   The above method consequently drops all registered information and resets the reasoner.

2. **Low-level reasoning tasks:**

   The following methods check for the entailment of a WSML logical expression which is in turn equivalent to a ground (variable free) query (a WSML logical expression). Executing a ground query can also be regarded as entailment checking.

   ```
   boolean entails(LogicalExpression)
   boolean entails(Set_of_LogicaExpressions)
   boolean executeGroundQuery(LogicalExpression)
   ```

[5] http://wsmo4j.sourceforge.net/doc/wsmo4j-prog-guide.pdf

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

```
Result executeQuery(LogicalExpression)
```

The following two methods are also correlated as Ontology consistency corresponds to satisfiability. However checking for consistency returns more detailed debugging information about the source of inconsistency (cardinality violations, attribute type violations, etc).

```
boolean isSatisfiable()
Set_of_Violations checkConsistency()
```

3. **Higher-level methods:**

The remaining methods realize conceptually higher level tasks that are reduced to lower level methods internally but i) are used often ii) more convenient to use as distinct methods. They each cover special reasoning tasks to some degree, sometimes even multiple tasks. Thus their separation into groups is by no means absolute and only serves readability as first goal

The following methods cover subsumption and classification.

```
boolean isSubConceptOf(Concept1, Concept2)
Set_of_Concepts getSubConcepts(Concept)
Set_of_Concepts getDirectSubConcepts(Concept)
Set_of_Concepts getSuperConcepts(Concept)
Set_of_concepts getDirectSuperConcepts(Concept)
```

Instance retrieval is possible with the following methods:

```
Set_of_Instances getInstances(Concept)
Set_of_Instances getAllInstances()
```

On the other hand the following methods cover the reasoning task of realization:

```
boolean isMemberOf(Instance, Concept)
Set_of_Concepts getConcepts(Instance)
Set_of_Concepts getDirectConcepts(Instance)
Set_of_Concepts getAllConcepts()
```

The remaining methods deal with attributes in a similar way and often involve several sub queries which would be highly inconvenient to perform with the more basic methods.

```
Set_of_Concepts getConceptsOf(Attribute)
Set_of_Attributes getAllAttributes()
Set_of_Attributes getAllConstraintAttributes()
Set_of_Attributes getAllInferenceAttributes()
```

Please note that WSML allows instances which are not members of a particular concept. Therefore the attribute range does not need to contain a concept identifier or

datatype identifier but might be empty. This especially concerns the following two methods:

```
Set_of_Types getRangesOfInferingAttribute(Attribute)

Set_of_Types getRangesOfConstraintAttribute(Attribute)
```

Several more methods deal with attributes in more complex fashion (see ***Figure 4*** for their complete signature) and thus warrant a detailed description.

- `getInferingAttributeValues` returns a map containing all inferring attributes of a specified instance and for each a set containing all its values.
- `getConstraintAttributeValues` returns a map containing all constraining attributes of a specified instance and for each a set containing all its values.
- `getInferingAttributeInstances` returns a map containing all instances that have values for a specified inferring attribute and for each a set containing all its values.
- `getConstraintAttributeInstances` returns a map containing all instances that have values for a specified constraining attribute and for each a set containing all The remaining two methods return a set containing instance values for the given instance and the infering or respectively constraining attribute.
- `getInferingAttributeValues`
- `getConstraintAttributeValues`

3.3.2 Language specific methods:

A number of methods are specific to a certain language dialect.

Most notably, WSML-Core as it does not allow for the direct specification of the attribute features reflexive, transitive, symmetric, inverseOf and subAttributeOf. This restriction stems from the fact that reflexivity, transitivity, symmetricity and inverse of attributes are defined locally to a concept in WSML as opposed to Description Logics. It is however possible to specify these properties globally via an appropriate axiom if desired. This restriction is also carried over to WSML-Rule.

Furthermore, the restrictions imposed by the logical formalism underlying WSML-Core (the intersection of Datalog and Description Logic [14]) also limit the expressivity of its logical expression syntax. WSML-DL's logical expression syntax is more powerful as it is based on the Description Logic SHIQ(D) [15]. Thus, WSML-Core and WSML-Rule also only support a limited form of equivalence. The methods specific to the DLReasoner are depicted in

```
<<interface>>
DLReasoner
Set_of_Concepts getEquivalentConcepts(Concept)
boolean isEquivalentConcept(Concept1, Concept2)
Set_of_Attributes getEquivalentAttributes(Attribute)
Set_of_Attributes getInverseAttribute(Attribute)
Set_of_Attributes getSuperAttributes(Attribute)
Set_of_Attributes getSubAttributes(Attribute)
```

Figure 5.

Furthermore, a number of methods are specific to the Logic Programming based WSML variants (WSML-Flight and WSML-Rule). Most notably the upcoming WSML

© SOA4ALL consortium

1.0 specification disallows the usage of relations in WSML-Core and WSML-DL[6]. Thus, methods that are specific to dealing with arbitrary relations are only present in the LPReasoner interface.

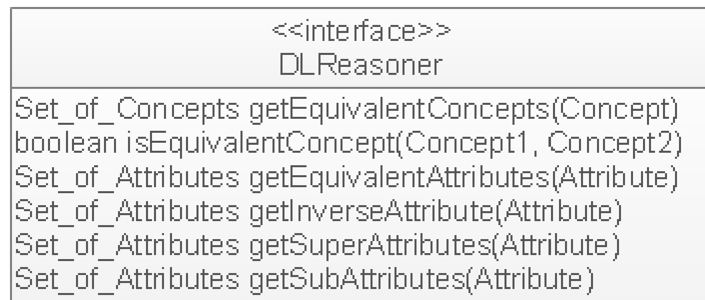

Figure 5 DL-Reasoner specific Methods

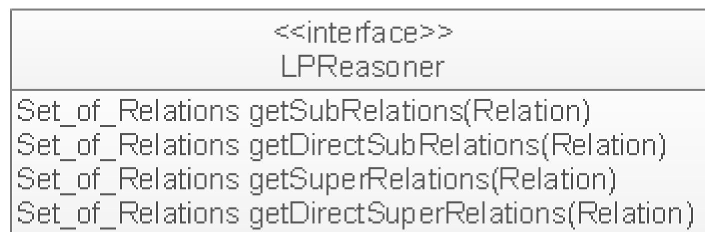

Figure 6 Logic Programming specific Methods

[6] http://www.wsmo.org/TR/d16/d16.1/v1.0/#cha:changelog

4. Conclusions

This deliverable presented an high-level view of the reasoner component. Based on an overview of relevant use-case we derive basic reasoning tasks that need to be supported and define a set of methods that will form the initial interfaces for future steps.

In subsequent deliverables D3.2.2, D3.2.3 and D3.2.4 which will be WSML-Core, WSML-Rule and WSML-Rule reasoner prototypes respectively. These deliverables will also discuss the particularities and semantics of the individual language fragments in more detail and specify the individual APIs in more detail.

Chronologically the next steps lead to D3.2.2 (First Prototype Repository reasoner for WSML-Core), which will be based on a reworked version of the WSML2Reasoner[7] framework supported by the IRIS[8] inference engine. In addition to WSML-Core IRIS can also be used as an RDFS reasoner and thus this initial prototype will further support the idea of a **light-weight** solution for retrieving implicit knowledge out of a repository.

[7] http://tools.sti-innsbruck.at/wsml2reasoner/

[8] http://www.iris-reasoner.org/

© SOA4ALL consortium

5. References

1. D. Martin et al., “OWL-S: Semantic Markup for Web Services,” W3C Member Submission, vol. 22, 2004.
2. D. Roman, H. Lausen, and U. Keller, “Web Service Modeling Ontology (WSMO),” WSMO Working Draft, 2004.
3. T. Vitvar et al., “WSMO-Lite Annotations for Web Services,” The Semantic Web: Research and Applications, 2008, pp. 674-689; http://dx.doi.org/10.1007/978-3-540-68234-9_49.
4. J. Farrell and H. Lausen, “Semantic Annotations for WSDL and XML Schema,” W3C Proposed Recommendation, vol. 5, 2007.
5. J. de Bruijn et al., “The Web Service Modeling Language WSML,” WSML Final Draft D, vol. 16, 2005.
6. F. Baader et al., “The Description Logic Handbook,” 2007.
7. J.W. Lloyd, Foundations of logic programming, Springer-Verlag New York, Inc. New York, NY, USA, 1987.
8. M. Fitting, First-Order Logic and Automated Theorem Proving, Springer, 1996.
9. 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.
10. “Web21C SDK”; http://web21c.bt.com/.
11. V.H. Frank, L. Vladimir, and Bruce Porter, eds., Handbook of Knowledge Representation (Foundations of Artificial Intelligence), Elsevier Science, 2007.
12. J.D. Ullman, Principles of Database Systems, WH Freeman & Co. New York, NY, USA, 1983.
13. S. Grimm et al., “A Reasoning Framework for Rule-Based WSML,” Semantic Web Research and Applications, 2007.
14. B. GROSOF et al., “Description Logic Programs: Combining Logic Programs with Description Logic.”
15. A. Borgida, “On the relative expressiveness of description logics and predicate logics,” Artificial Intelligence, vol. 82, 1996, pp. 353-367.