



D2v1.0. Web Service Modeling Ontology (WSMO)

WSMO Working Draft 21 July 2004

This version:

<http://www.wsmo.org/2004/d2/v1.0/20040721/>

Latest version:

<http://www.wsmo.org/2004/d2/v1.0/>

Previous version:

<http://www.wsmo.org/2004/d2/v1.0/20040717/>

Editors:

Dumitru Roman
Holger Lausen

Authors:

Dumitru Roman
Holger Lausen
Uwe Keller
Eyal Oren
Christoph Bussler
Michael Kifer
Dieter Fensel

This document is also available in non-normative [PDF](#) version.
The intent of the document is to be submitted to [W3C](#).

Table of contents

- [1. Introduction](#)
- [2. Global Issues](#)
 - [2.1 Namespaces](#)
 - [2.2 Identifier](#)
 - [2.3 Non functional properties](#)
 - [2.3.1 Non functional properties - core properties](#)
 - [2.3.2 Non functional properties - web service specific properties](#)
- [3. Ontologies](#)
 - [3.1 Namespaces](#)
 - [3.2 Non functional properties](#)
 - [3.3 Import Ontologies](#)
 - [3.4 Used mediators](#)

- [3.5 Concepts](#)
 - [3.6 Relations](#)
 - [3.7 Functions](#)
 - [3.8 Instances](#)
 - [3.9 Variables](#)
 - [3.10 Axioms](#)
 - [3.11 Logical Expressions](#)
 - [4. Goals](#)
 - [5. Mediators](#)
 - [6. Web Services](#)
 - [6.1 Capability](#)
 - [6.2 Interfaces](#)
 - [7. Conclusions and further directions](#)
 - [References](#)
 - [Acknowledgement](#)
 - [Appendix A. Conceptual Elements of WSMO](#)
 - [Appendix B. BNF Grammar for WSML](#)
-

Abstract

This document presents an ontology called Web Service Modeling Ontology (WSMO) for describing various aspects related to Semantic Web Service. Having the Web Service Modeling Framework (WSMF) [[Fensel & Bussler, 2002](#)] as a starting point, we refine this framework and develop a formal ontology and language.

1. Introduction

WSMF [[Fensel & Bussler, 2002](#)] consists of four different main elements for describing semantic web services (see [Figure 1](#)): ontologies that provide the terminology used by other elements, goal repositories that define the problems that should be solved by web services, web services descriptions that define various aspects of a web service and mediators which bypass interpretability problem.

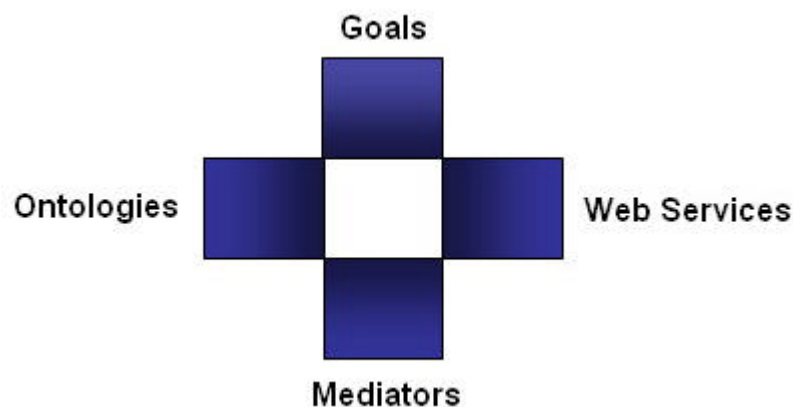


Figure 1. The main elements of WSMF

This document presents an ontology called Web Service Modeling Ontology

(WSMO) for describing various aspects related to Semantic Web Service. Having the Web Service Modeling Framework (WSMF) [[Fensel & Bussler, 2002](#)] as a starting point, we refine this framework and develop a formal ontology and language.

[Section 2](#) presents global issues related to the elements that are considered in the ontology. Following the philosophy of WSMF, we further define in the next sections the ontologies ([Section 3](#)), goals ([Section 4](#)), mediators ([Section 5](#)) and web service ([Section 6](#)). [Section 7](#) presents our conclusions and further directions.

For a brief tutorial on WSMO we refer to the [WSMO Primer \[Arroyo & Stollberg, 2004\]](#) and for a non-trivial use case demonstrating how to use WSMO in a real-world setting we refer to the [WSMO Use Case Modeling and Testing \[Stollberg et al., 2004\]](#).

2. Global Issues

This section addresses issues that concern all subsequent sections, it is especially concerned with compliance to current web standards.

2.1 Namespaces

The vocabulary is fully extensible, being based on URIs with optional fragment identifiers (URI references, or URIrefs) [[Berners-Lee et al, 1998](#)]. URI references are used for naming all kinds of things in WSMO. Subsequent we allow the use of namespace abbreviations and its declaration.

The default namespace for the listings in this document is wsmo: <http://www.wsmo.org/2004/d2/>. Furthermore this document assumes the following namespace declarations:

- dc: <http://purl.org/dc/elements/1.1#>
- xsd: <http://www.w3.org/2001/XMLSchema#>
- foaf: <http://xmlns.com/foaf/0.1/>

In the remainder of this document the above prefixes denote the correspondingly defined URIs.

2.2 Identifier

WSMO distinguishes 3 kinds of identifiers: URI references, Literals and Anonymous Ids; this general idea is taken from the RDF concepts [[Manoler & Miller, 2004](#)]:

URI references

Everything in WSMO is an Identifier denoted by a URI, except it is a Literal or an Anonymous Id. Like RDF, WSMO is based on the idea of identifying things using web identifiers (called Uniform Resource Identifiers). However this limits WSMO not to make statements about things that are not accessible on the web, like with the uri: "urn:isbn:0-520-02356-0" that identifies a certain book. Also the WSMO keywords (defined in Appendix B) are URIs; to be brief the namespace prefix of WSMO keywords may be omitted. URIs can be expressed as follows:

- **full URIs:** e.g. <http://www.wsmo.org/2004/d2>
- **relative URIs** that are resolved against the in-scope base URI.
- **qualified Names (QNames)** that are resolved using namespace declarations.

Literals

Literals are used to identify values such as numbers by means of a lexical representation. Anything represented by a literal could also be represented by a URI, but it is often more convenient or intuitive to use literals. Literals are either plain literals or typed Literals. A Literal can be typed to a data type (e.g. to `xsd:integer`). Formally such a data type is defined by [\[Hayes, 2004\]](#):

- a non-empty set of character strings called the lexical space of `d`; e.g. {"true", "1", "false", "0"}
- a non-empty set called the value space of `d`; e.g. {true, false}
- a mapping from the lexical space of `d` to the value space of `d`, called the lexical-to-value mapping of `d`. e.g. {"true", "1"}->{true}; {"false", "0"}->{false}

Furthermore the data type may introduce facets on its value space, such as ordering (and therefore define the axiomatization for the relations `<`, `>` and function symbols like `+` or `-`). The XML Schema Definition [\[Biron ? Malhotra, 2001\]](#) does this in an informal way, we are currently using this to define the usual arithmetic build-in operators used in the logical expressions (see [Section 3.11](#)).

Anonymous Ids

Anonymous Ids can be numbered (`_#1`, `_#2`, ...) or unnumbered (`_#`). Anonymous Ids represent Identifier. The same numbered Anonymous Id represents the same Identifier within the same scope (`logical-expression`), otherwise Anonymous Ids represent different Identifier [\[Yang & Kifer, 2003\]](#). Anonymous Ids can be used to denote objects that exists, but don't need an explicit identifier (e.g. if someone wants to say that a Person john has a address `_#` which itself has a street name "hitchhikerstreet" and a street number "42"), then the object of the address itself does not need a own URI, but since it must exist as connecting object between john and "hitchhikersstreet", "42" we can denote it with an Anonymous Id). This concept is similar to blank nodes in RDF [\[Hayes, 2004\]](#), however there are some differences as outlined in the afore mentioned paper [\[Yang & Kifer, 2003\]](#).

2.3. Non functional properties

Non functional properties are defined as a set of tuples, where each tuple consists of a property and its value constraint. We classify non functional properties in two categories: core properties and web service specific properties.

2.3.1 Non functional properties - core properties

The core properties can be used for all the modeling elements of WSMO. They consist of the [Dublin Core Metadata Element Set](#) [\[Weibel et al., 1998\]](#) plus the `version` element. We do not enforce restrictions on the range value type (and this also omit the range restriction in the following listings) but in some cases provide additional recommendations:

```

entity non-functional-properties
  dc:title
  dc:creator
  dc:subject
  dc:description
  dc:publisher
  dc:contributor
  dc:date
  dc:type
  dc:format
  dc:identifier
  dc:source
  dc:language
  dc:relation
  dc:coverage
  dc:rights
  version
    
```

Title

A name given to an element. Typically, `dc:title` will be a name by which the element is formally known.

Creator

An entity primarily responsible for creating the content of the element.

Examples of `dc:creator` include a person, an organization, or a service. The Dublin Core specification recommends, that typically, the name of a `dc:creator` should be used to indicate the entity.

WSMO Recommendation: In order to point unambiguously to a specific resource we recommend the use an instance of `foaf:Agent` as value type [[Brickley & Miller, 2004](#)].

Subject

A topic of the content of the element. Typically, `dc:subject` will be expressed as keywords, key phrases or classification codes that describe a topic of the element. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

Description

An account of the content of the element. Examples of `dc:description` include, but are not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.

Publisher

An entity responsible for making the element available. Examples of `dc:publisher` include a person, an organization, or a service. The Dublin Core specification recommends, that typically, the name of a `dc:publisher` should be used to indicate the entity.

WSMO Recommendation: In order to point unambiguously to a specific resource we recommend the use an instance of `foaf:Agent` as value type [[Brickley & Miller, 2004](#)].

Contributor

An entity responsible for making contributions to the content of the element. Examples of `dc:contributor` include a person, an organization, or a service. The Dublin Core specification recommends, that typically, the name of a `dc:contributor` should be used to indicate the entity.

WSMO Recommendation: In order to point unambiguously to a specific resource we recommend the use an instance of `foaf:Agent` as value type

[[Brickley & Miller, 2004](#)].

Date

A date of an event in the life cycle of the element. Typically, `dc:date` will be associated with the creation or availability of the element.

WSMO Recommendation: We recommend to use the an encoding compatible to the XML Schema Definition (YYYY-MM-DD) [[Biron & Malhotra, 2001](#)].

Type

The nature or genre of the content of the element. The `dc:type` includes terms describing general categories, functions, genres, or aggregation levels for content.

WSMO Recommendation: We recommend to use an URI encoding to point to the namespace or document describing the type, e.g. for a domain ontology expressed in WSMO, one would use:
<http://www.wsmo.org/2004/d2/#ontologies>.

Format

A physical or digital manifestation of the element. Typically, `dc:format` may include the media-type or dimensions of the element. Format may be used to identify the software, hardware, or other equipment needed to display or operate the element. Examples of dimensions include size and duration.

WSMO Recommendation: We recommend to use types defined in the list of Internet [Media Types](#) [[IANA, 2002](#)] by the IANA (Internet Assigned Numbers Authority)

Identifier

An unambiguous reference to the element within a given context.

Recommended best practice is to identify the element by means of a string or number conforming to a formal identification system. In Dublin Core formal identification systems include but are not limited to the Uniform element Identifier (URI) (including the Uniform element Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).

WSMO Recommendation: We recommend to use URIs as Identifier, depending on the particular syntax the identity information of an element might already be given, however it might be repeated in `dc:identifier` in order to allow Dublin Core meta data aware applications the processing of that information.

Source

A reference to an element from which the present element is derived. The present element may be derived from the `dc:source` element in whole or in part. Recommended best practice is to identify the referenced element by means of a string or number conforming to a formal identification system.

WSMO Recommendation: We recommend to use URIs as Identifier where possible.

Language

A language of the intellectual content of the element.

WSMO Recommendation: We recommend to use the language tags defined in the ISO Standard 639 [[ISO639, 1988](#)], e.g. "en-GB", in addition the logical language used to express the content should be mentioned, for example this can be [OWL](#).

Relation

A reference to a related element. Recommended best practice is to identify the referenced element by means of a string or number conforming to a formal identification system.

WSMO Recommendation: We recommend to use URIs as Identifier where

possible.

Coverage

The extent or scope of the content of the element. Typically, `dc:coverage` will include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity).

WSMO Recommendation: For more complex applications, consideration should be given to using an encoding scheme that supports appropriate specification of information, such as [DCMI Period](#), [DCMI Box](#) or [DCMI Point](#).

Rights

Information about rights held in and over the element. Typically, `dc:rights` will contain a rights management statement for the element, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions may be made about any rights held in or over the element.

Version

As many properties of an element might change in time, an identifier of the element at a certain moment in time is needed.

WSMO Recommendation: If applicable we recommend to use the revision numbers of a version control system. Such a system can be for example CVS (Concurrent Version System), that automatically keeps track of the different revisions of a document, an example CVS version Tag looks like: "\$Revision: 1.2".

2.3.2. Non functional properties - web service specific properties

Besides the core properties described in the previous section, the web service specific non functional properties also include properties related to the quality aspect of a web service (QoS):

Listing 2. Non functional properties definition for web services

```
entity ws-non-functional-properties subentity-of non-functional-properties
  accuracy
  availability
  financial
  network-related-qoS
  performance
  reliability
  robustness
  scalability
  security
  transactional
  trust
```

Accuracy

It represents the error rate generated by the web service. It can be measured by the numbers of errors generated in a certain time interval.

Availability

It defines whether the Web Service is ready for immediate consumption. This particularly includes spatial as well as temporal aspects of availability.

Financial

It represents the cost-related properties of a web service.

Network-related QoS

They represent the QoS mechanisms operating in the transport network which are independent of the web services. They can be measured by network delay, delay variation and/or message loss.

Performance

It represents how fast a service request can be completed. According to [\[Rajesh & Arulazi, 2003\]](#) performance can be measured in terms of throughput, latency, execution time, and transaction time. The response time of a service can also be a measure of the performance. High quality web services should provide higher throughput, lower latency, lower execution time, faster transaction time and faster response time.

Reliability

It represents the ability of a web service to perform its functions (to maintain its service quality). It can be measured by the number of failures of the service in a certain time interval.

Robustness

It represents the ability of the service to function correctly in the presence of incomplete or invalid inputs. It can be measured by the number of incomplete or invalid inputs for which the service still function correctly.

Scalability

It represents the ability of the service to process more requests in a certain time interval. It can be measured by the number of solved requests in a certain time interval.

Security

It represents the ability of a service to provide authentication (entities - users or other services - who can access service and data should be authenticated), authorization (entities should be authorized so that they only can access the protected services), confidentiality (data should be treated properly so that only authorized entities can access or modify the data), traceability/auditability (it should be possible to trace the history of a service when a request was serviced), data encryption (data should be encrypted), and non-repudiation (an entity cannot deny requesting a service or data after the fact).

Transactional

It represents the transactional properties of the web service.

Trust

It represents the trust worthiness of the service.

Further discussions on service specific non-functional-properties can be used during the service life cycle can be found in [\[O`Sullivan et al., 2002\]](#). In conclusion, the web service specific non functional properties extend the common core properties (in [Section 2.3.1](#)) especially by quality of service aspects. Nonetheless, the model is extensible and more (even application-domain specific) aspects could be added.

3. Ontologies

In WSMO Ontologies are the key to link consensual real world semantics intended by humans with computers. *An ontology is a formal explicit specification of a shared conceptualization* [\[Gruber, 1993\]](#). From this rather conceptual definition we want to extract the essential components which define an ontology. Ontologies define a consensual terminology by providing concepts and relationships among the set of concepts. In order to capture semantic properties of relations and concepts, an ontology generally also provides a set of axioms, which means expressions in some

logical framework. An ontology is defined as follows:

Listing 3. Ontology definition

```
entity ontology
  namespaces oftype set namespace
  non-functional-properties oftype non-functional-properties
  import-ontologies oftype set ontology
  used-mediators oftype set oo-mediator
  concept-definitions oftype set concept-definition
  relation-definitions oftype set relation-definition
  function-definitions oftype set function-definition
  instance-definitions oftype set instance-definition
  variable-definitions oftype set variable-definition
  axiom-definitions oftype set axiom-definition
```

3.1 Namespaces

They define prefixes for elements defined in the ontology.

3.2 Non functional properties

The `non-functional-properties` of an ontology consist of the core properties described in [Section 2.3.1](#).

3.3 Import Ontologies

Building an ontology for some particular problem domain can be a rather cumbersome and complex task. One standard way to deal with the complexity is modularization. `import-ontologies` allow a modular approach for ontology design; this simplified statement can be used as long as no conflicts need to be resolved, otherwise an `oo-mediator` needs to be used.

3.4 Used mediators

When importing ontologies, most likely some steps for aligning, merging and transforming imported ontologies have to be performed. For this reason and in line with the basic design principles underlying the WSMF, ontology mediators (`oo-mediator`) are used when an alignment of the imported ontology is necessary. Mediators are described in [Section 5](#) in more detail.

3.5 Concepts

Concepts constitute the basic elements of the consensual terminology for some problem domain. From a high-level perspective, a concept – described by a concept definition – provides attributes with names and types. Furthermore, a concept can have several (possibly none) direct superconcepts as specified by the "is-a"-relation.

Listing 4. Concept definition

```
entity concept-definition subentity-of axiom-definition
super-concepts oftype set concept-definition
attributes oftype set attribute-definition
```

Superconcepts

There can be a finite number of concepts that serve as a `super-concepts` for some concept. In particular, being a subconcept of some other concept means that a concept inherits the signature of this superconcept and the corresponding constraints. Furthermore all instances of a concept are also instances of its superconcept.

Attributes

Each concept provides a (possibly empty) set of `attributes` that represent named slots for data values for instances that have to be filled at the instance level. An attribute specifies a slot of a concept by fixing the name of the slot as well as a logical constraint on the possible values filling that slot. Hence, this logical expression can be interpreted as a typing constraint.

Listing 5. Attribute definition

```
entity attribute-definition subentity-of axiom-definition
range oftype axiom-definition
```

Range

A logical expression constraining the possible values for filling the slot of any instance of a particular concept. In a simple case this might be a concept, as it is a subentity of logical-expression.

3.6 Relations

`Relations` are used in order to model interdependencies between several concepts (respectively instances of these concepts).

Listing 6. Relation definition

```
entity relation-definition subentity-of axiom-definition
parameters oftype set parameter-definition
```

Parameters

A list of parameters; a parameter is a named placeholder for some value.

Listing 7. Parameter definition

```
entity parameter-definition subentity-of axiom-definition
domain oftype axiom-definition
```

Domain

A logical expression constraining the possible values that the parameter can take.

3.7 Functions

A function symbol is a special relation, with a unary range and a n-ary domain (parameters), where the range specifies the return value.

Listing 8. Function definition

```
entity function-definition subentity-of axiom-definition
range oftype axiom-definition
parameters oftype set parameter-definition
```

Range

A logical expression constraining the possible return values (for filling the slot of any instance of a particular concept).

Parameters

A list of parameters; a parameter is a named placeholder for some value.

3.8 Instances

Instances are defined by a link to an instance store. This instance store (i.e. a database, a flat text file, etc.) should be associated with a namespace and connected via an ooMediator, such that a build-in relation can be used within that namespace to retrieve instances.

A detailed discussion and a concrete proposal on how to integrate large sets of instance data in an ontology model can be found in [DIP Deliverable D2.2 \[Kiryakov et. al., 2004\]](#).

3.9 Variables

Variables itself are identified with URIs, they are declared and typed globally, however their scope is always locally (within one logical-expression). Additional type restrictions (e.g. within the scope of a logical-expression) are interpreted as conjunction, such that a type can be refined but never broadened.

Listing 9. Variable definition

```
entity variable-definition subentity-of axiom-definition
restriction oftype set concept-definition
```

Restriction

`Restriction` limits the possible values of the variable to instances of the concept defined in this restriction, as it is done in sorted logic [\[Enderton, 1972\]](#).

3.10 Axioms

An `axiom-definition` is considered to be a logical expression together with its non functional properties.

Listing 10. Axiom definition

```
entity
non-functional-properties oftype non-functional-properties
```

Non functional properties

The `non-functional-properties` of an axiom consist of the core properties described in [Section 2.3.1](#).

Defined by

The actual statement captured by the axiom is defined by an formula in a logical language as described below in [Section 3.11](#).

3.11 Logical Expressions

Most elements in the WSMO Ontology model are entities derived from `axiom-definition`.

As the major component of `axiom-definition`, logical expressions are used almost everywhere in the WSMO Ontology model to capture specific nuances of meaning of modeling elements or their constituent parts in a formal and unambiguous way. In the following, we give a brief description of the formal language that is used for specifying `logical-expressions`.

The language define here basically is a first-order language, similar to First-order Logics [[Enderton, 1972](#)] and Frame Logic (F-Logic, resp.) [[Kifer et al., 1995](#)]. That means basically, that we allow variables to be used in formal statements that represent individuals of some domain (or universe, resp.) and provide the means to represent computations within the domain by means of terms (using function symbols and variables). In particular, we exploit the advanced object-oriented modeling constructs of F-Logic and reflect these constructs in our language.

Following common practice in the definition of logical languages, in particular First-order Logic and F-Logic, we start with the definition of the basic vocabulary for building logical expression. Then we define the set of terms and the most basic formulas (atomic formulae or F-molecules, resp.).

Let *URI* be the set of all valid uniform resource identifiers. This set will be used for the naming (or identifying, resp.) various entities in a WSMO description.

The **vocabulary V of our language L(V)** consists of the following symbols:

- An infinite set of **variables** *Var* which is a subset of *URI*
- An infinite set of **function symbols** (object constructors, resp.) *FSym* which is a subset of *URI*
- An infinite set of **predicate symbols** *PSym* which is a subset of *URI*
- A finite set of **auxiliary symbols** *AuxSym* including (,), `oftype`, `oftype set`, `memberOf`, `subConceptOf`, `hasValue`, `hasValues`, `false`, `true`.
- A finite set of **logical connectives and quantifiers** including the usual junctors and quantifiers in First-Order Logics: `or`, `and`, `not`, `<-,->`, `<->` , `forall`, `exists`.
- All these sets are assumed to be *mutually distinct*.
- For each symbol *s* in *FSym* or *PSym*, we assume that there is a corresponding **arity** *arity(s)* defined, which is a non-negative integer specifying the number of arguments that are expected by the corresponding symbol when building expressions in our language.

As usual, 0-ary function symbols are called *constants*. 0-ary predicate symbols correspond to propositional variables in classical propositional logic.

Given a vocabulary V , we can define the **set of terms** $\text{Term}(V)$ (over vocabulary V) as follows:

- Any identifier u in URI is a term in $\text{Term}(V)$.
- If f is a function symbol from $FSym$ with $\text{arity}(f) = n$ and t_1, \dots, t_n are terms, then $f(t_1, \dots, t_n)$ is a term in $\text{Term}(V)$.
- Nothing else is a term.

As usual, the set of ground terms $\text{GroundTerm}(V)$ is the subset of terms in $\text{Term}(V)$ which do not contain any variables.

Note: The definition covers the usual case of variables as well since the set of variable symbols Var is a subset of URI .

Terms can be used in general to describe computations (in some domain). One important additional interpretation of terms is that they denote objects in some universe and thus provide names for entities in some domain of discourse.

We extend this definition to the **set of logical-expression** (or formulae, resp.) $L(V)$ (over vocabulary V) as follows:

A logical-expression in $L(V)$ is inductively defined by

- If p is a predicate symbol in $PSym$ with $\text{arity}(p) = n$ and t_1, \dots, t_n are terms, then $p(t_1, \dots, t_n)$ is a logical expression in $L(V)$.
- true and false are logical expression in $L(V)$.
- If P, T are terms in $\text{Term}(V)$, then $P \text{ ofType } T$ is a logical expression in $L(V)$.
- If $P, T_1, \dots, T_n, (1 \leq n)$ are terms in $\text{Term}(V)$, then $P \text{ ofType set } (T_1, \dots, T_n)$ is a logical expression in $L(V)$.
- If O, T are terms in $\text{Term}(V)$, then $O \text{ memberOf } T$ is a logical expression in $L(V)$.
- If C_1, C_2 are terms in $\text{Term}(V)$, then $C_1 \text{ subConceptOf } C_2$ is a logical expression in $L(V)$.
- If O, V are terms in $\text{Term}(V)$, then $O \text{ hasValue } V$ is a logical expression in $L(V)$.
- If $O, V_1, \dots, V_n, (1 \leq n)$ are terms in $\text{Term}(V)$, then $O \text{ hasValues } \{V_1, \dots, V_n\}$ is a logical expression in $L(V)$.
- If L is a logical expression in $L(V)$, then $\text{not } L$ is a logical expression.
- If L_1 and L_2 are logical expressions in $L(V)$ and op is one of the junctors in $\{\text{or}, \text{and}, \text{not}, \text{<-}, \text{->, <->\}$, then $L_1 \text{ op } L_2$ is a logical expression.
- If L is a logical expression in $L(V)$, x is a variable from Var and Q is a quantor in $\{\text{forall}, \text{exists}\}$, then $Qx(L)$ is a logical expression.
- Nothing else is a logical expression (or formula, resp.).

The logical expressions defined in the first part of the bulleted list is commonly called *atomic logical expressions* or *atomic formulae*.

Notational conventions: There is a precedence order defined for the logical junctors as follows, where $op_1 \prec op_2$ means that op_2 binds stronger than op_1 : $\text{<-} \prec \text{->} \prec \text{<->} \prec \text{or} \prec \text{and} \prec \text{not}$.

The precedence order can be exploited when writing logical expressions in order to prevent from extensive use of parantheses. In case that there are ambiguities in

evaluating an expression, parantheses must be used to resolve the ambiguities.

The terms $P \text{ of type set } (T)$ and $O \text{ has values } \{V\}$ (that means for the case $n = 1$ in the respective clauses above) can be written simpler by omitting the parantheses.

Note: Note that this definition for our language $L(V)$ is extensible by extending the basic vocabulary V . In this way, the language for expressing logical expressions can be customized to the needs of some application domain.

4. Goals

In this section, we introduce the notion of `goals` and define the elements that are used in the description of a goal. A Goal is defined as follows:

Listing 11. Goal definition

```
entity goal
  namespaces oftype set namespace
  non-functional-properties oftype non-functional-properties
  import-ontologies oftype set ontology
  used-mediators oftype set (oo-mediator or gg-mediator)
  post-conditions oftype set axiom-definition
  effects oftype set axiom-definition
```

Namespaces

They define prefixes for elements defined in the goal.

Non functional properties

The `non-functional-properties` of a goal consist of the core properties described in the [Section 2.3.1](#).

Import Ontologies

It is used to import ontologies as long as no conflicts are needed to be resolved.

Used mediators

A goal can import ontologies using ontology mediators (oo-mediators) when steps for aligning, merging and transforming imported ontologies are needed. A goal may be defined by reusing one or several already existing goals. This is achieved by using goal mediators (gg-mediators). For a detailed account on mediators we refer to [Section 5](#).

Post-conditions

`Post-conditions` in WSMO describe the state of the information space that is desired.

Effects

`Effects` describe the state of the world that is desired.

5. Mediators

In this section, we introduce the notion of `mediators` and define the elements that are used in the description of a mediator.

We distinguish four different types of mediators :

- `gg-mediators`: mediators that link two goals. This link represents the

- refinement of the source goal into the target goal.
- `oo-mediators`: mediators that import ontologies and resolve possible representation mismatches between ontologies.
 - `wg-mediators`: mediators that link web service to goals. They explicitly may state the difference between the two entities and map different vocabularies (through the use of `oo-mediators`).
 - `ww-mediators`: mediators linking two Web Services.

The `mediator` is defined as follows:

Listing 12. Mediators definition

```

entity mediator
  namespaces oftype set namespace
  non-functional-properties oftype non-functional-properties
  import-ontologies oftype set ontology
  source oftype set (ontology or goal or web-service or mediator)
  target oftype (ontology or goal or web-service or mediator)
  mediation-service oftype (goal or ww-mediator)

entity oo-mediator subentity-of mediator
  source oftype set (ontology or oo-mediator)

entity gg-mediator subentity-of mediator
  used-mediators oftype set oo-mediator
  source oftype set (goal or gg-mediator)
  target oftype (goal or gg-mediator)

entity wg-mediator subentity-of mediator
  used-mediators oftype set oo-mediator
  source oftype (web-service or wg-mediator)
  target oftype (goal or gg-mediator)

entity ww-mediator subentity-of mediator
  used-mediators oftype set oo-mediator
  source oftype (web-service or ww-mediator)
  target oftype (web-service or ww-mediator)

```

Namespace

They define prefixes for elements defined in the mediator.

Non functional properties

The non functional properties of a mediator consist of the core properties described in the [Section 2.3.1](#) (where, in this case, an element in the core properties is equivalent to a mediator). Besides these properties, and taking into account that a mediator uses a mediation service, the non functional properties of a mediator also include aspects related to the quality aspect of the mediation service (see [Section 2.3.2](#) for a description of these properties). The quality aspects of the mediator and the mediation service, taken together, might be enhanced (e.g. improving the robustness) or weakened (e.g. increasing the response time) by the mediator.

Import Ontologies

It is used to import ontologies as long as no conflicts are needed to be resolved.

Source

The source components define entities that are the sources of the mediator.

Target

The target component defines the entity that is the targets of the mediator.

Mediation Service

The `mediation-service` points to a goal that declarative describes the mapping or to a `ww-mediator` that links to a web service that actually implements the mapping.

Used Mediators

Some specific types of mediators, i.e. `gg-mediator`, `wg-mediator` and `ww-mediator`, use a set of `oo-mediators` in order to map between different vocabularies used in the description of goals and webservice capabilities and align different heterogeneous ontologies.

6. Web Services

In this section we identify the concepts needed for describing various aspects of a web service. The following properties of a web service are considered: `namespaces`, `non-functional-properties`, `import-ontologies`, `used-mediators`, `capability` and `interfaces`.

Listing 13. Web service definition

```
entity web-service
  namespaces oftype set namespace
  non-functional-properties oftype ws-non-functional-properties
  import-ontologies oftype set ontology
  used-mediators oftype set oo-mediator
  capability oftype capability
  interfaces oftype set interface
```

Namespaces

They define prefixes for elements defined in the web service.

Non functional properties

The `non-functional-properties` of a web service are described in [Section 2.3.2](#).

Import Ontologies

It is used to import ontologies as long as no conflicts are needed to be resolved.

Used mediators

A web service can import ontologies using ontology mediators (`oo-mediators`) when steps for aligning, merging and transforming imported ontologies are needed.

Capability

The `capability` of a web service is described in [Section 6.1](#).

Interfaces

The `interfaces` of a web service are described in [Section 6.2](#).

6.1 Capability

A `capability` defines the web service by means of its functionality.

```

entity capability
  non-functional-properties oftype non-functional-properties
  import-ontologies oftype set ontology
  used-mediators oftype set (oo-mediator or wg-mediator)
  preconditions oftype set axiom-definition
  assumptions oftype set axiom-definition
  postconditions oftype set axiom-definition
  effects oftype set axiom-definition

```

Non functional properties

The `non-functional-properties` of a capability consist of the core properties described in the [Section 2.3.1](#).

Import Ontologies

It is used to import ontologies as long as no conflicts are needed to be resolved.

Used mediators

A capability can import ontologies using ontology mediators (oo-mediators) when steps for aligning, merging and transforming imported ontologies are needed. It can be linked to a goal using a wg-mediator.

Pre-conditions

`Pre-conditions` in WSMO describe what a web service expects for enabling it to provide its service. In other words, they constrain the set of states of the information space such that each state satisfying these constraints can serve as a valid starting state (in the information space) for executing the service in a defined manner. The constraints refer to the input values of a service.

Post-conditions

`Post-conditions` in WSMO describe the states of the information space that will be reached by executing the service. In particular, they describe what a web service returns in response to its input and define the relation between the input and the output of a service execution.

Assumptions

`Assumptions` in WSMO describe the expectation of the service on the state of the world when starting an execution of the service. The service guarantees the declared functionality only if it is started in such a state. Thus, the assumptions constrain the set of states of the world to the set of valid starting states. In fact, assumptions are similar to pre-conditions, however, they also reference aspects of the state of the world beyond the actual input.

Effects

`Effects` describe the state of the world after that is going to be reached after the execution of the service.

6.2 Interfaces

An `interface` describes how the functionality of the service can be achieved (i.e. how the `capability` of a service can be fulfilled) by providing a twofold view on the operational competence of the service:

- `choreography` decomposes a capability in terms of interaction with the service (service user's view)
- `orchestration` decomposes a capability in terms of functionality required from other services (other service providers' view)

This distinction reflects the difference between communication and cooperation. The `choreography` defines how to communicate with the web service in order to consume its functionality. The `orchestration` defines how the overall functionality is achieved by the cooperation of more elementary service providers.

An `interface` is defined by the following properties:

Listing 15. Interface definition

```
entity interface
  non-functional-properties oftype non-functional-properties
  import-ontologies oftype set ontology
  used-mediators oftype set oo-mediator
  choreography oftype choreography
  orchestration oftype orchestration
```

Non functional parameters

The `non-functional-properties` of an interface consist of the core properties described in the [Section 2.3.1](#).

Import Ontologies

It is used to import ontologies as long as no conflicts are needed to be resolved.

Used mediators

An interface can import ontologies using ontology mediators (oo-mediators) when steps for aligning, merging and transforming imported ontologies are needed.

Choreography

`Choreography` provides the necessary information for the user to communicate with the web service (i.e. it describes how the service works and how to access the service from the user's perspective).

Orchestration

`Orchestration` describes how the service works from the provider's perspective (i.e. how a service makes use of other web service or goals in order to achieve its capability).

The distinction between `choreography` and `orchestration` [1] should be considered in the context of the role the service is playing in a conversation: provider or requester. In case the service acts as a provider, the way of interacting with it is specified in its choreography. If the service acts as a requester, requesting functionalities of different services, then

- the way in which this services are composed and
- the way of interacting with them

are specified in the orchestration of the service.

7. Conclusions and further directions

This document presented the Web Service Modeling Ontology (WSMO) for describing several aspects related to web services, by refining the Web Service Modeling Framework (WSMF). The definition of the missing elements (choreography and orchestration) will be treated separately, in other documents, and consists of work to be done for the future.

References

- [Arroyo & Stollberg, 2004] S. Arroyo and M. Stollberg (Eds.): *WSMO Primer*, WSMO Deliverable D3.1, DERI Working Draft, 2004, latest version available at <http://www.wsmo.org/2004/d3/d3.1/>.
- [Baader et al., 2003] F. Baader, D. Calvanese, and D. McGuinness: *The Description Logic Handbook*, Cambridge University Press, 2003.
- [Berners-Lee et al, 1998] T. Berners-Lee, R. Fielding, and L. Masinter: *RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax*, IETF, August 1998, available at <http://www.isi.edu/in-notes/rfc2396.txt>.
- [Biron & Malhotra, 2001] P. V. Biron and A. Malhotra: *XML Schema Part 2: Datatypes*, W3C Recommendation 02, 2001, available at: <http://www.w3.org/TR/xmlschema-2/>.
- [Brickley & Miller, 2004] D. Brickley and L. Miller: *FOAF Vocabulary Specification*, available at: <http://xmlns.com/foaf/0.1/>.
- [Dean et al., 2004] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein: *OWL Web Ontology Language Reference*, W3C Recommendation, 10 February 2004. Available at <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- [Enderton, 1972] H.B. Enderton: *A Mathematical Introduction to Logic*, Academic Press, 1972.
- [Fensel & Bussler, 2002] D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications, 1(2), 2002.
- [Gruber, 1993] T. Gruber: A translation approach to portable ontology specifications, *Knowledge Acquisition*, 5:199-220, 1993.
- [Hayes, 2004] P. Hayes (ed.): *RDF Semantics*, W3C Recommendation 10 February 2004, 2004.
- [IANA, 2002] Internet Assigned Number Authority: *MIME Media Types*, available at: <http://www.iana.org/assignments/media-types/>, February 2002.
- [ISO639, 1988] International Organization for Standardization (ISO): *ISO 639:1988 (E/F). Code for the Representation of Names of Languages*. First edition, 1988-04-01. Reference number: ISO 639:1988 (E/F). Geneva: International Organization for Standardization, 1988. iii + 17 pages.
- [Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: *Logical foundations of object-oriented and frame-based languages*. *Journal of the ACM*, 42:741-843, July 1995.
- [Kiryakov et. al., 2004] A. Kiryakov, D. Ognyanov, and V. Kirov: *A framework for representing ontologies consisting of several thousand concepts definitions*, Project Deliverable D2.2 of DIP, June 2004.

[Manoler & Miller, 2004] F. Manola and E. Miller: *RDF Primer*, W3C Recommendation 10 February 2004, available at <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.

[O`Sullivan et al., 2002] J. O`Sullivan, D. Edmond, and A. Ter Hofstede: *What is a Service?: Towards Accurate Description of Non-Functional Properties*, Distributed and Parallel Databases, 12:117-133, 2002.

[Pan & Horrocks, 2004] J. Z. Pan and I. Horrocks: *OWL-E: Extending OWL with expressive datatype expressions*. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester, 2004. Available from <http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf>.

[Rajesh & Arulazi, 2003] S. Rajesh and D. Arulazi: *Quality of Service for Web Services-Demystification, Limitations, and Best Practices*, March 2003. (See <http://www.developer.com/services/article.php/2027911>.)

[Stollberg et al., 2004] M. Stollberg, H. Lausen, A. Polleres, R. Lara (Eds.): *WSMO Use Case Modeling and Testing*, WSMO Deliverable D3.2, DERI Working Draft, 2004, latest version available at <http://www.wsmo.org/2004/d3/d3.2/>

[Weibel et al., 1998] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf: *RFC 2413 - Dublin Core Metadata for Resource Discovery*, September 1998.

[Yang & Kifer, 2003] G. Yang and M. Kifer: *Reasoning about Anonymous Resources and Meta Statements on the Semantic Web* J. Data Semantics I 2003: 69-97.

Acknowledgement

The work is funded by the European Commission under the projects [DIP](#), [Knowledge Web](#), [SEKT](#), [SWWS](#), and [Esperanto](#); by [Science Foundation Ireland](#) under the [DERI-Lion](#) project; and by the Vienna city government under the [CoOperate](#) program.

The editors would like to thank to all the members of the [WSMO](#), [WSML](#), and [WSMX](#) working groups for their advice and input into this document.

Appendix A. Conceptual Elements of WSMO

entity non-functional-properties

- dc:title
- dc:creator
- dc:subject
- dc:description
- dc:publisher
- dc:contributor
- dc:date
- dc:type
- dc:format
- dc:identifier
- dc:source
- dc:language
- dc:relation
- dc:coverage

dc:rights
version

entity ws-non-functional-properties

performance
reliability
security
scalability
robustness
accuracy
transactional
trust
financial
network-related-qoS

entity ontology

namespaces **oftype set** namespace
non-functional-properties **oftype** non-functional-properties
import-ontologies **oftype set** ontology
used-mediators **oftype set** oo-mediator
concept-definitions **oftype set** concept-definition
relation-definitions **oftype set** relation-definition
function-definitions **oftype set** function-definition
instance-definitions **oftype set** instance-definition
variable-definitions **oftype set** variable-definition
axiom-definitions **oftype set** axiom-definition

entity concept-definition **subentity-of** axiom-definition

super-concepts **oftype set** concept-definition
attributes **oftype set** attribute-definition

entity attribute-definition **subentity-of** axiom-definition

range **oftype** axiom-definition

entity relation-definition **subentity-of** axiom-definition

parameters **oftype set** parameter-definition

entity relation-definition subentity-of axiom-definition

parameters oftype set parameter-definition

entity function-definition **subentity-of** axiom-definition

range **oftype** axiom-definition
parameters **oftype set** parameter-definition

entity variable-definition **subentity-of** axiom-definition

restriction **oftype set** concept-definition

entity axiom-definition

non-functional-properties **oftype** non-functional-properties
defined-by **oftype** logical-expression

entity goal

namespaces **oftype set** namespace
non-functional-properties **oftype** non-functional-properties
import-ontologies **oftype set** ontology
used-mediators **oftype set** (oo-mediator or gg-mediator)
post-conditions **oftype** goalAxiomDefinition
effects **oftype** goalAxiomDefinition

entity mediator

namespaces **oftype set** namespace
non-functional-properties **oftype** non-functional-properties

import-ontologies **oftype set** ontology
source **oftype set** (ontology or goal or web-service or mediator)
target **oftype set** (ontology or goal or web-service or mediator)
mediation-service **oftype** (goal or ww-mediator)

entity oo-mediator **subentity-of** mediator
source **oftype set** (ontology or oo-mediator)

entity gg-mediator **subentity-of** mediator
used-mediators **oftype set** oo-mediator
source **oftype set** (goal or gg-mediator)
target **oftype** (goal or gg-mediator)

entity wg-mediator **subentity-of** mediator
used-mediators **oftype set** oo-mediator
source **oftype** (web-service or wg-mediator)
target **oftype** (goal or gg-mediator)

entity ww-mediator **subentity-of** mediator
used-mediators **oftype set** oo-mediator
source **oftype** (web-service or ww-mediator)
target **oftype** (web-service or ww-mediator)

entity web-service
namespaces **oftype set** namespace
non-functional-properties **oftype** ws-non-functional-properties
import-ontologies **oftype set** ontology
used-mediators **oftype set** oo-mediator
capability **oftype** capability
interfaces **oftype set** interface

entity capability
non-functional-properties **oftype** non-functional-properties
import-ontologies **oftype set** ontology
used-mediators **oftype set** (oo-mediator or wg-mediator)
preconditions **oftype set** axiom-definition
assumptions **oftype set** axiom-definition
postconditions **oftype set** axiom-definition
effects **oftype set** axiom-definition

entity interface
non-functional-properties **oftype** non-functional-properties
import-ontologies **oftype set** ontology
used-mediators **oftype set** oo-mediator
choreography **oftype** choreography
orchestration **oftype** orchestration

Appendix B. BNF Grammar for WSML

This Appendix describes the grammar for the Web Services Modeling Language (WSML). This language is meant for representing the concepts introduced in the Web Service Modeling Ontology (WSMO) in a human-readable way.

The language to write down this syntax is a variant of Extended Backus Nauer Form; a variant readable by ANTLR ([Parr and Quong 1995](#)). ANTLR is a tool that can construct recognisers, parsers and compilers from a grammar specification, its functionality is (among others) similar to the well-known lex/yacc tools. Visit www.antlr.org for more information.

```

wsmoDefinition:
( nameSpaceDefinition
|
)
( ontologyDefinition
| webServiceDefinition
| goalDefinition
| mediatorDefinition
)* ;

nameSpaceDefinition:
( NAMESPACE
| NAMESPACEURI
)
( prefix )*
( TARGETNAMESPACE QNAME
|
) ;

ontologyDefinition:
ONTOLOGY QNAME
( nameSpaceDefinition
|
)
( nonFunctionalDefinition
|
)
( USEMEDIATOR QNAME ( COMMA QNAME ) *
|
)
( conceptDefinition
| axiomDefinition
| instanceDefinition
| relationDefinition
| variableDefinition
| functionDefinition
)* ;

webServiceDefinition:
WEBSERVICE QNAME
( nameSpaceDefinition
|
)
( nonFunctionalDefinition
|
)
( USEMEDIATOR QNAME ( COMMA QNAME ) *
|
)
( capabilityDefinition
|
)
( interfaceDefinition ) * ;

goalDefinition:
GOAL QNAME
( nameSpaceDefinition
|
)
( nonFunctionalDefinition
|
)
( USEMEDIATOR QNAME ( COMMA QNAME ) *
|
)
( ( POSTCONDITION
| EFFECT
)
axiomDefinition )+ ;

mediatorDefinition:
ooMediatorDefinition
| ggMediatorDefinition
| wgMediatorDefinition
| wwMediatorDefinition ;

prefix:
QNAME EQUAL QNAME ( COMMA QNAME ) * ;

```

nonFunctionalDefinition:

```
NFP  
( nonFunctionalBlock  
| version  
)*  
ENDNFP ;
```

axiomDefinition:

```
AXIOM QNAME  
( nonFunctionalDefinition  
|  
|  
| )  
logicalExpression ;
```

ooMediatorDefinition:

```
OOMEDIATOR QNAME  
( nameSpaceDefinition  
|  
| )  
( nonFunctionalDefinition  
|  
| )  
( SOURCE QNAME ) * ( TARGET QNAME ) *  
( USESERVICE QNAME  
|  
| ) ;
```

ggMediatorDefinition:

```
GMEDIATOR QNAME  
( nameSpaceDefinition  
|  
| )  
( nonFunctionalDefinition  
|  
| )  
( SOURCE QNAME  
|  
| )  
( TARGET QNAME  
|  
| )  
( USEMEDIATOR QNAME ( COMMA QNAME ) *  
|  
| )  
( REDUCTION axiomDefinition  
|  
| ) ;
```

wgMediatorDefinition:

```
WMEDIATOR QNAME  
( nameSpaceDefinition  
|  
| )  
( nonFunctionalDefinition  
|  
| )  
( SOURCE QNAME  
|  
| )  
( TARGET QNAME  
|  
| )  
( USEMEDIATOR QNAME ( COMMA QNAME ) *  
|  
| )  
( REDUCTION axiomDefinition  
|  
| ) ;
```

wwMediatorDefinition:

```
WWMEDIATOR QNAME  
( nameSpaceDefinition  
|  
| )  
( nonFunctionalDefinition  
|  
| )  
( SOURCE QNAME  
|  
| )
```

```

)
( TARGET_QNAME
|
)
( USEMEDIATOR_QNAME ( COMMA_QNAME )*
|
);

capabilityDefinition:
( USECAPABILITY_QNAME )
| ( CAPABILITY_QNAME
  ( nonFunctionalDefinition
  |
  )
  ( USEMEDIATOR_QNAME ( COMMA_QNAME )*
  |
  )
  ( ( PRECONDITION
    | POSTCONDITION
    | ASSUMPTION
    | EFFECT
    )
    axiomDefinition )+ );

interfaceDefinition:
( USEINTERFACE_QNAME )
| ( INTERFACE_QNAME
  ( nonFunctionalDefinition
  |
  )
  ( USEMEDIATOR_QNAME ( COMMA_QNAME )*
  |
  )
  choreographyDefinition
  ( orchestrationDefinition
  |
  )
  ));

choreographyDefinition:
CHOREOGRAPHY_PLACEHOLDER ;

orchestrationDefinition:
ORCHESTRATION_PLACEHOLDER ;

conceptDefinition:
CONCEPT_QNAME
( nonFunctionalDefinition
|
)
( SUBCONCEPT_QNAME
|
)
( attributeDefinition )* ;

relationDefinition:
RELATION_QNAME
( nonFunctionalDefinition
|
)
parameterDefinition ;

variableDefinition:
VARIABLE_QNAME ( COMMA_QNAME )* MEMBEROF_NAME ;

functionDefinition:
FUNCTION_QNAME
( nonFunctionalDefinition
|
)
( parameterDefinition )*
( rangeDefinition
|
)
);

attributeDefinition:
QNAME OFTYPE
( SET
|

```

```

)
QNAME ;

parameterDefinition:
PARAMETER QNAME OFTYPE QNAME ;

rangeDefinition:
RANGE QNAME OFTYPE QNAME ;

valueDefinition:
( HASVALUE value )
| ( HASVALUES value ( COMMA value )* );

value:
literal
| QNAME ;

literal:
String ;

// At present logical expression are only represented as strings in the BNF grammar.
// The concrete grammar rules for the logical expressions are skipped in this version of
// the document for the sake of simplicity; instead we simply use strings as a data format
// for logical expressions. The corresponding rules will be integrated
// in the next version of this deliverable.

logicalExpression:
BEGINLOGIC String ENDLOGIC ;

nonFunctionalBlock:
QNAME
( QNAME
| literal
)
( COMMA
( QNAME
| literal
) ) * ;

version:
VERSION
( QNAME
| literal
) ;

mSLComment : "comment:" ( ( '\n'
| '\r'
) ) *
( '\n'
| '\r'
)
;

protected mNCBegin:
( mLowerCaseLetter
| mUnderscore
| mUpperCaseLetter
) ;

protected mLowerCaseLetter:
'a'..'z';

protected mUnderscore : '_';

protected mUpperCaseLetter:
'A'..'Z';

protected mNCRest:
( mNCBegin
| mPOINT
| mDash
| mHash
| mSlash
| mAmp
| mQuestion
| mDigit
| mTilde
| mSemiColon
) ;

protected mPOINT : '!';
protected mDash : '-';
protected mHash : '#';
protected mSlash : '/';
protected mAmp : '&';
protected mQuestion : '?';

```

```

protected mDigit : '0'..'9';
protected mTilde : '~';
protected mSemiColon : ';';
protected mNCName:
  mNCBegin
  ( mNCRest
    | mCOLON
  )*;
mCOLON : ':';
mQNAME : mNCName;
protected mURICharacters:
  mNCRest
  | mEQUAL
  | mCOLON
  | '%';
mEQUAL : '=';
mEQUALQNAME:
  mEQUAL mNCBegin ( mURICharacters )+;
mNAMESPACEURI:
  mNAMESPACE ' ' mNCBegin ( mURICharacters )+;
mNAMESPACE : "namespace";
mCOMMA : ',';
mString : '"' ( ~'"' )* '"';
mTARGETNAMESPACE : "target-namespace";
mUSEMEDIATOR : "used-mediator";
mOOMEDIATOR : "oo-mediator";
mGGMEDIATOR : "gg-mediator";
mWGMEDIATOR : "wg-mediator";
mWWWMEDIATOR : "ww-mediator";
mSOURCE : "source";
mTARGET : "target";
mUSESERVICE : "use-service";
mREDUCTION : "reduction";
mGOAL : "goal";
mWEBSERVICE : "webservice";
mUSECAPABILITY : "use-capability";
mCAPABILITY : "capability";
mPRECONDITION : "precondition";
mPOSTCONDITION : "postcondition";
mASSUMPTION : "assumption";
mEFFECT : "effect";
mUSEINTERFACE : "use-interface";
mINTERFACE : "interface";
mCHOREOGRAPHY : "choreography";
mORCHESTRATION : "orchestration";
mPLACEHOLDER : "****";
mONTOLOGY : "ontology";
mCONCEPT : "concept";
mSUBCONCEPT : "subconcept-of";
mOFTYPE : "oftype";
mSET : "set";
mINSTANCE : "instance";
mMEMBEROF : "member-of";
mHASVALUE : "hasvalue";
mHASVALUES : "hasvalues";
mRELATION : "relation";
mPARAMETER : "parameter";
mFUNCTION : "function";
mVARIABLE : "variable";
mMETHOD : "method";
mRANGE : "range";
mAXIOM : "axiom";
mBEGINLOGIC : "logical-expression";
mENDLOGIC : "end-logical-expression";
mVERSION : "version";
mNFP : "non-functional-properties";
mENDNFP : "end-non-functional-properties";

```

[1] One could argue that *orchestration* should not be part of a public interface because it refers to how a service is implemented. However, this is a short-term view that does not reflect the nature of fully open and flexible eCommerce. Here companies shrink to their core processes were they are really profitable in. All other processes are sourced out and consumed as eServices. They advertise their services in their capability and choreography description and they advertise their

needs in the orchestration interfaces. This enables on-the-fly creation of virtual enterprises in reaction to demands from the market place. Even in the dinosaurian time of eCommerce where large companies still exist, `orchestration` may be an important aspect. The `orchestration` of a service may not be made public but may be visible to the different departments of a large organization that compete for delivering parts of the overall service. Notice that the actual business intelligence of a service provider is still hidden. It is his capability to provide a certain functionality with a choreography that is very different from the sub services and their orchestration. The ability for a certain type of process management (the overall functionality is decomposed differently in the `choreography` and the `orchestration`) is where it comes in as a Silver bullet in the process. How he manages the difference between the process decomposition at the `choreography` and the `orchestration` level is the business intelligence of the web service provider.



[webmaster](#)

\$Date: Wednesday 21 July 2004 - 00:15:11\$
