



# D13.3v0.2 WSMX Data Mediation

WSMO Working Draft 28 January 2005

**This version:**

<http://www.wsmo.org/2005/d13/d13.3/v0.2/20050128>

**Latest version:**

<http://www.wsmo.org/2004/d13/d13.3/v0.2/>

**Previous version:**

<http://www.wsmo.org/2005/d13/d13.3/v0.2/20050107>

**Editor:**

Adrian Mocan

**Authors:**

Adrian Mocan  
Emilia Cimpian

**Reviewer:**

Jos de Bruijn

This document is also available in non-normative [PDF](#) version.

---

## Table of Contents

### **1. Introduction**

### **2. Problem Definition and Scope of Ontology Mediation in WSMX**

2.1. Problem Definition

2.2. Addressed Ontologies

2.3. Scope of Ontology Mediation in WSMX

### **3. Requirement Analysis**

3.1. Ontology Mediation in WSMX

3.2. Ontology to Ontology Mediation Problems - Ontology Mismatches

### **4. WSMX Mediation Approach**

### **5. Mappings**

5.1. Abstract Mappings

5.2. Mappings Creation

5.3. Built-in (Conversion) Functions

5.4. Built-in (Conversion) Function Creation

5.5. Mapping Rules Generation

5.6. Execution

### **6. Addressed Ontology Mismatches**

6.1. Different composition of concepts

6.2. Incomplete/over-complete concept representation

6.3. Data-type mismatch

6.4. Cardinality

### **7. Flora2 - A Reasoner for WSMX Mediator**

- 8. From WSML to Flora2**
- 9. Expressing Mapping Rules in Flora2**
- 10. Prototype Implementation**
  - 10.1. Graphical User Interface
  - 10.2. Mapping Rules Generator Module
  - 10.3. Execution Environment
- 11. Related Work**
  - 11.1. PROMPT
  - 11.2. MAFRA
- 12. Conclusions and Further Directions**
- References**
- Acknowledgements**
- Appendix A. WSML-Core to Flora2 Translation**
- Appendix B. Changelog**

---

## 1. Introduction

This deliverable presents our approach to mediation in the Web Service Execution Environment [WSMX, 2004], an environment that is designed to allow dynamic mediation, selection and invocation of web services. WSMX has as final scope the domain defined by Web Service Modeling Ontology (WSMO) [WSMO, 2004], in order to provide a testbed and a proof for the ideas presented in WSMO. The WSMX work includes the creation of a conceptual model [WSMX O, 2004], the description of the execution semantics [WSMX Execution Semantics, 2004], and an architectural and software design [WSMX Architecture, 2004], together with a working implementation [WSMX Implementation, 2004]. As part of this work, this document describes the mediation problems addressed in this context, together with the appropriate solutions and a software implementation. In the first phase, the scope of WSMX is WSMO-Lite [WSMO-Lite, 2004], which offers a minimal, yet meaningful, subset of WSMO (also referred as WSMO-Standard). In the future, WSMX will extend its scope to cover both WSMO-Standard and WSMO-Full [WSMO-Full, 2004]. As a consequence, the mediation addressed in this document is the only form of mediation referred by WSMO-Lite: ontology-to-ontology mediation.

It is well known now that models or ontologies describing the same or related problem domains are created by different entities around the world. This is why more and more systems and applications require mediation in order to be able to integrate and use heterogeneous data sources. The mapping between models is required in several classes of application, as *Information Integration and Semantic Web*, *Data Migration* or *Ontology Merging* [Madhavan et al., 2002].

Unfortunately, it is impossible for a mapping tool to automatically generate mapping rules. Some of the best results of research projects in this area are mapping tools that are able to validate or to suggest possible mappings, but at some points of the mapping process, domain expert intervention still remains a necessity. Efforts to build such tools concentrate on two main areas [Madhavan et al., 2002]: the first is the designing of heuristics based on structure, on non-functional properties and on names of the concepts involved, in order to generate the most plausible mappings [Noy & Musen, 2000]. Sometimes, domain independent heuristics augmented by more specific and domain related heuristics

can be used [Mitra et al., 2000]. The second approach is to use machine learning techniques to obtain the required mappings semi-automatically. For example, [Doan et al., 2002] use the instances of one ontology to learn a classifier for it, and then apply this classifier to classify the instances of the second ontology, and vice versa. In this way, the probability that two concepts are similar can be approximated using the fraction of instances that belong to both of them.

The mediation solution proposed by us is based on the first approach described above: we use decomposition methods for exploiting the internal structure characteristics of the concepts involved in the mediation process. In addition, strategies for suggesting the proper decompositions as well as the proper mappings between un-decomposable concepts will be provided. As for the second approach, the machine-learning-based strategies are not suited to our needs because these methods require the usage of large sets of training data (in our case instances) for obtaining a high accuracy of the predicted mappings. But the aim of the execution environment mediation is, for a given instance (obtained at the runtime), from the source ontology, to provide the correspondent instance (or instances) from the target ontology. Additionally, one has to keep in mind that each instance that has to be mediated contains a piece of information about internal business processes and behaviours, and almost no company or enterprise will agree to grant access to its business information (i.e. all its available instances), not even for the laudable purpose of allowing the learning of useful mappings for its future cooperations. That is, no training data-set is available, but only the instance that has to be mediated.

The entire process of mediation described here consists of three main steps: the creation of mappings, the creation of appropriate mapping rules and the execution of the mapping rules. The first two parts are carried out at the schema level: all the mappings are created considering only the schema information, without using any instance data. Furthermore, the fact that the execution environment scope is the WSMO domain implies that the mediated ontologies conform to the WSMO conceptual model for ontologies, having the same meta-level definition. The third step, the execution, involves also the instance data considering the given source instance and creating the target instance(s).

Although this deliverable describes the execution environment mediation, the first of the three steps presented above, takes place outside the execution environment. Its implementation consists of a graphical environment that allows the domain expert to place his or her inputs, and the implementation of the required processes that assure the semi-automatic mediation behaviour of the module (e.g. the similarity computing process). The implementations of the second and the third steps are included in the WSMX environment and provide means of translating the given source instances into target instances, based on the previously created mappings.

This document is structured as follows: Section 2 presents the problem definition and scope of ontology mediation in WSMX. In this section the input ontologies are also described, introducing the conceptual model for ontologies of Web Service Modeling Ontology (WSMO). Section 3 presents the requirements that have to be achieved by WSMX mediation together with an exhaustive classification of potential ontology mediation mismatches. Section 4 introduces the terminology adopted in our approach and describes the solutions proposed for ontology mediation and the set of addressed problems. Section 5 briefly presents the prototype that implements the ideas presented in the previous sections. Finally,

Section 6 offers an overview of two related works in this area and Section 7 draws some conclusions and refers to the further directions of WSMX mediation.

## 2. Problem Definition and Scope of Ontology Mediation in WSMX

This section provides an overview of the domain problem and of the scope of the mediation. The first subsection presents the problem definition; the second subsection offers a short description of the ontologies used in our mediation process, and finally the last subsection outlines the scope of the ontology mediation in WSMX.

### 2.1. Problem Definition

In recent years, one of the solutions adopted for dealing with the heterogeneity over the World Wide Web has been to add semantic information to the data. This approach will lead in the near future to the development of a high number of ontologies, modeling aspects of the real world. But unfortunately a new problem has arisen: how to integrate applications that are using different conceptualizations of the same domain. Furthermore, the web services enriched with the semantics offered by the ontologies aim to enable a dynamic and a strongly decoupled communication and cooperation between different entities.

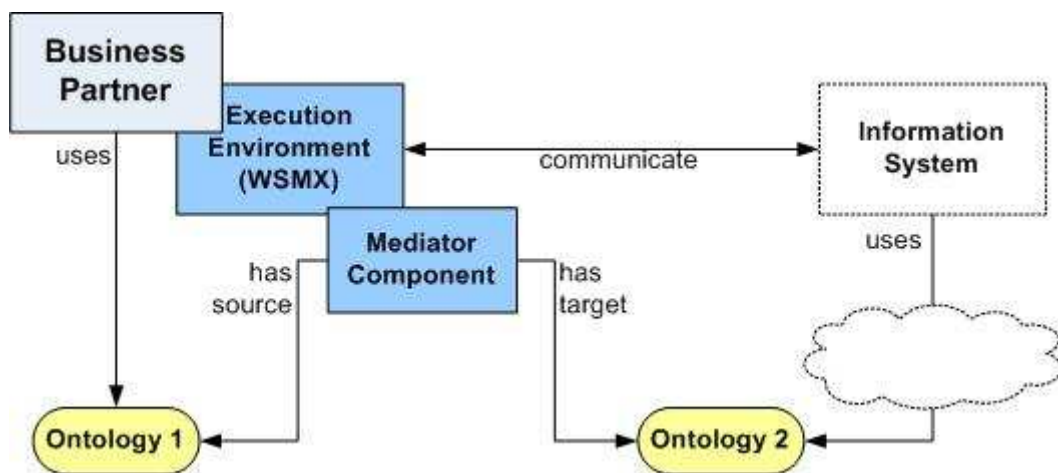


Figure 1. WSMX Mediation Scenario

As a consequence, it is very easy to imagine the following scenario (see Figure 1): two enterprises decide to become partners in a business process. One of the enterprises wants to buy something that the other enterprise offers, and to achieve this, a sequence of messages has to be exchanged (e.g. purchase orders and purchase order acknowledgements). Each of these messages is represented in terms of the sender's ontology, and each of the business partners (e.g. enterprises) understands only messages expressed in terms of its own ontology. One of the roles of the execution environment (by mean of mediation), is to transform, if necessary, the received message from the terms of sender's ontology into the terms of the receiver's ontology, before sending it further. From the perspective of the ontologies, each message contains instances of the source ontology that have to be transformed into instances of the target ontology.

## 2.2. Addressed Ontologies

Web Service Modeling Ontology [WSMO, 2004] is a meta-ontology for describing several aspects related to Semantic Web Services. Its aim is to facilitate the usage of Web Services, providing the means for semi-automatic discovery, invocation, composition, execution, monitoring, mediation and compensation. For achieving this ambitious goal, WSMO defines four main modeling elements: Web Services, Goals, Ontologies and Mediators. The Web Services represent the functional part: each Web Service offers a functionality which has to be semantically described, to enable its (semi) -automatic use. A Goal specifies the objective of a client: a client consults a Web Service only if this Web Service may satisfy his goal. The Ontologies have the role of providing semantics to the information used by all the other components, and the Mediators provide interoperability among the other components.

The purpose of this deliverable is to present a possible approach for constructing a mediation system able to mediate between two ontologies (called *ooMediator* in WSMO). Our component is able to mediate between ontologies which conform to WSMO-Lite Specification [WSMO-Lite, 2004]. WSMO-Lite represents a minimal, yet meaningful, subset of WSMO (also referred as WSMO Standard). We chose WSMO-Lite as reference for the first version of this deliverable and for the resulting software module, but in the future we plan to extend our approach to WSMO-Standard conceptual model for ontologies. In what follows, we present the definitions (adopted from WSMO-Lite) of the elements that play an important role in this approach.

The first definition we have to consider is, of course, the ontology definition. As presented in *Listing 1*, WSMO ontologies consist of non-functional properties, used mediators, concepts, relations and instances.

Listing 1: Ontology definition

```
ontology[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> ooMediators
conceptDefinitions =>> conceptDefinition
relationDefinitions =>> relationDefinition
instances =>> instance
]
```

In *Listing 2* you can see that WSMO-Standard nonFunctionalProperties definition WSMO-Lite has only two elements:

Listing 2: Non-functional (core properties) definition

```
nonFunctionalProperties[
title => title
identifier => identifier
]
```

The *title* is actually the name by which the ontology is known and the identifier represents an unambiguous reference to the element within a given context. Every component of the WSMO ontology has the same non functional properties: *title* and *identifier*.

Regarding the *usedMediators*, as previously stated, the aim of this deliverable is to describe strategies and mechanisms for the construction of an *ooMediator* together with a software module that implements them.

The next element, *conceptDefinitions*, plays a major role, the concepts and their attributes being the endpoints of the links that represent the basis of the mediation process. The concept definition is presented below:

Listing 3: Concept Definition

```
conceptDefintion[
nonFunctionalProperties => nonFunctionalProperties
superConcepts =>> simpleLogicalExpression
attributes =>> attributeDefintion
]
```

As previously stated, the concepts' non functional properties are as defined in *Listing 2*. For each concept a finite number of concepts that represent its direct super concepts and a set (possible null) of attributes are defined. The attribute definition is presented in *Listing 4*:

Listing 4: Attribute Definition

```
attributeDefinition[
nonFunctionalProperties => nonFunctionalProperties
rangeDefinition => simpleLogicalExpression
]
```

The simple logical expression that stands for *rangeDefinition* is a type of constraint that can represent, in the simplest case, the extension of a concept in the ontology, or it can represent more sophisticated constraints if necessary.

The relations are not yet addressed by our mediation approach. In the future we will extend it to allow relation mediation also.

The instances are defined as follows:

Listing 5. Instance definition

```
instance[
nonFunctionalProperties => nonFunctionalProperties
instanceOf =>> logicaExpression
attributeValues =>> attributeValueDefinition
]
```

The instances are actually the subject of our mediation component. We define mappings between concepts of ontologies, which state, for example, that concept A from the first ontology corresponds to concept B from the second ontology, only for being able to construct at runtime instances of B that are equivalent to given instances of A.

## 2.3. Scope of Ontology Mediation in WSMX

The general scope of mediation in WSMX is similar to the scope of some other components part of WSMX architecture: to provide a default implementation of the desired functionality, in our case ontology-to-ontology mediation. The future

intention is to provide support for integration of external components, providing the same functionality, but another (or better) realization of it.

As mentioned already, the WSMX mediation addresses (for now) only ontology-to-ontology mediation. The intention is to provide support for transforming instances referring to one ontology into instances specified in terms of the other ontology. The transformations applied to the source ontology affect both their structure and their values - the rules that drive these transformations may be seen as a mix of *integration* and *context* rules as described in [Wache, 1999]. The rules built based on the similar entities from the two ontologies (i.e. concepts and attributes) identified before the runtime and saved in a storage internal to WSMX. The similar entities are defined and made known to WSMX by using a graphical user interface which allows the user to place his inputs and guides him/her through the entire process. Though the similar entities are identified based on human inputs (choices, validations etc.), the rest of the process (rules generation and instances transformations) is carried out automatically, without human intervention.

### 3. Requirement Analysis

This section defines the main requirements for the mediation component in WSMX, and presents the main problems that need to be addressed during the ontology-to-ontology mediation process. In the first subsection, the functionality offered by the mediation component is analyzed in more detail, while in the second subsection a set of ontology mismatches is presented (as identified by [Visser et al., 1997]) relating them to our approach.

#### 3.1. Ontology Mediation in WSMX

The ontology-to-ontology mediation in WSMX should provide three main functionalities: the first should facilitate the identification of the potential similarities between the entities (i.e. concepts and attributes) of two given ontologies; the second should provide the means of exploiting these similarities by creating appropriate rules to express the links between the two ontologies; the third functionality should offer an environment in which the existing links could drive the transformation process from the source to the target. In the following sections we will analyze the requirements for each of this functionalities.

##### Identifying similarities

This functionality can be seen as the first stage of the mediation process between two ontologies. Even if the two ontologies are modeling the same aspects of the real world, they could be developed by different parties, so different terminology might be used, or different modeling strategies might be chosen. As a consequence, we can infer the first requirement:

- <sup>1</sup> The mediation process has to provide means to identify the similar entities from the two conceptualizations.

In fact, the process of finding similarities involves the identification of entities with the same semantic and, unfortunately, even if different heuristics and techniques were developed for addressing this issue, the human user inputs are still necessary. Considering this:

- 1 The mediation component has to offer an easy way to incorporate the domain expert inputs: a graphical interface that offers a visual representation of his or her actions and decisions.

Furthermore:

- 1 This interface should include a set of the above-mentioned heuristics and techniques.

in order to reduce the human effort from a laborious and error-prone work to simple choices and validation decisions.

The graphical interface should provide facilities for:

- 1 browsing the input ontologies;
- 1 guiding the user through the whole process;
- 1 hiding irrelevant information from the user;
- 1 offering suggestions, based on the already-discovered similarities and other appropriate heuristics.

The user should always have access to the decisions he/she has already made for potential updates and refinements. After the process ends

- 1 the discovered similarities have to be saved in a persistent storage for further usage

either for continuing later the similarity discover between the two ontologies or for applying them to the next stage of the mediation process. The way of storing these similarities should be independent of the language used for expressing the ontologies or for expressing the links in the next stage.

## Creating the links

The next logical step that should be taken is to exploit the already identified similarities and to actually

- 1 create specific links between the two ontologies.

Each link should serve a well-defined purpose, which is to specify all the transformations that have to be performed for transforming a set of instances from the terms of the first ontology into the terms of the second ontology. Such a link may contain specifications of the structural transformations of the source instance and/or of the values referred by that instance - transformations dictated by the structural differences between the ontologies and the differences between the contexts in which the instances are created [Wache, 1999].

These links may or may not be expressed in the same language as the ontologies, the decision of what language to use being determined by the complexity of the transformations required, and of the type of entities to be transformed. This stage of the mediation process has to be completely automatic:

- 1 the links have to be created without any user or domain expert intervention.

## Translating instances

The last step of the mediation process implies the actual transformations, and has to be performed based on the links created in the previous stage. That is, this stage (we may call it the execution engine) should provide

- 1 means for executing the transformations specified in links

and for each given source instance a set of target instances has to be created. The execution engine is directly dependent on the language chosen in the previous stage for expressing the links and should be able to

- 1 execute the transformations in a completely automatic way without any human intervention.

Based on the degree of involvement from the user side, we can say that the first functionality is to be implemented as part of the *design-time* phase, while the last two functionalities are to be made available in the *run-time* phase.

## 3.2. Ontology-to-Ontology Mediation Problems - Ontology Mismatches

In this section we present an exhaustive set of ontology mapping mismatches identified by [Visser et al., 1997] and analyze which of them should be addressed in our approach. The possible mismatches between ontologies are classified from two points of view: from the conceptualization point of view (*conceptualization mismatches*) and from the explications point of view (*explication mismatches*).

### Conceptualization mismatches

The conceptualization mismatches refer to differences between the ontological concepts and between the relations among these concepts. In the first case they consist of the *categorization mismatches* and the *aggregation level mismatches*, which are described in the following paragraphs.

*Categorization mismatch* occurs when the two conceptualizations distinguish the same concepts but organize them in different sub-concepts. A common example could be if one conceptualization is organized around the *mammals* and *birds* concepts and the other conceptualization around *carnivores* and *herbivores*. Our approach, considering the fact that our aim is to create for the given source instance a set of target instances, is not concerned with the mediation of sub-concept relationship between concepts, but only with the identification of the most similar concept from the target, for a given source concept.

The *aggregation-level mismatches* appear because of the usage of different levels of abstraction for defining similar concepts. As an example, one ontology can use the concept of *person* and another ontology can use the concepts of *male* and *female* without having a concept similar to *person* as super-concept. As in the previous case the scope of our mediation process seems to reduce the dimension of these types of mismatches.

Another conceptualization mismatch is the *relation mismatch* which is determined by the differences in the hierarchical relation between concepts and in the assignment of the attributes to the classes. There are three types of such mismatches identified: *structure mismatches*, *attribute-assignment mismatches* and *attribute-type mismatches*.

The *structure mismatches* are determined by the way the similar concepts are structured by means of relations. This version of mediation in WSMX doesn't address the mappings between the relations defined in the source and target ontology, even though the special case of binary relations is addressed in this approach by considering them attributes. As a consequence, these mismatches, together with the *attribute-assignment mismatches* (generated by the different assignation of attributes to concepts) and *attribute-type mismatches* (occurring when two concepts have the same assigned attribute but differ by their assumed instantiation), are covered by the WSMX mediation both at the stage of finding similarities and in the links creation process.

## Explication mismatches

[Visser et al., 1997] classify these mismatches based on the assumption that an ontology is formed by a set of definitions and each definition has the following form: Def=<T, D, C> where T stands for the term used, D for the definiens and C is the ontological concept that is explained. For two given definitions (one from the source ontology and one from the target ontology) there are eight ( $2^3$ ) possible combination of mismatches, but there are two that are not relevant for this discussion: when all three terms are different (if there is no correspondence, we cannot talk about mismatch) or when all three terms are identical (there is no mismatch). All six remaining mismatches should be addressed by the WSMX mediation at the similarity identifying stage, by being able to discriminate between cases where the two analyzed definitions refer to the same concept, or not. In fact, the task of identifying similarities involves the analysis of the terms (T), and of the definiens (D), and based on this analysis a conclusion has to be drawn: the explained concepts (C) are similar, or they are not similar. The six possible combinations of mismatches are discussed in more details below.

- 1 **CT mismatch** - the modeled concepts and terms used are different but the definitions are the same. In terms of our approach, this means that we have the same definitions (the same attributes) for two concepts (with different names) that model different aspects of the domain. Even if the definitions are identical the concepts modeled are different - in this case the domain expert input should be considered for taking the correct decision.
- 1 **CD mismatch** - the modeled concepts and the definitions are different but the terms used coincide; in our context, two different concepts are modeled using different attributes. Our assumption is that all concepts from the input ontologies are different until it is specified otherwise, so the system should be able to use the appropriate heuristics for inferring the correct suggestions (based on this suggestion the domain expert should be able to make the final decisions and validations). The terms used in this case are called homonyms.
- 1 **C mismatch** - the modeled concepts are different but the definitions and terms used are identical. Here we have the same problem as in the CT mismatch. Domain expert intervention in this case is a must. Note that for domain-related ontologies this case should be very rare. As in the previous case the terms used are called homonyms.

- 1 **TD mismatch** - the modeled concepts are similar but the terms used and the definitions are different. This means that two similar concepts are denoted by different terms and modeled using different attributes. At first sight, one may conclude that having a different definition for the two concepts the system will fail to make accurate suggestions. In reality, if the two input ontologies are modeling related domains, it is very likely that the definitions will converge on the same primitive elements (see Section 4.1 for more details related to the approach to this requirement). The terms used in this case are called synonyms.
- 1 **T mismatch** - the modeled concepts and their definitions are similar but the terms used are different. The offered heuristics should be able to suggest the two modeled concepts as similar (in this case the terms are also called synonyms). Based on this suggestions the domain expert should be able to make final decisions and validations.
- 1 **D mismatch** - the modeled concepts and the terms used are the same but their definitions are different. The situation is very similar to the TD mismatch but the identical terms used should improve the quality of the returned suggestions.

The WSMX mediation has to be able to respond to these ontology-to-ontology mediation problems. Some of these problems are eliminated by the nature of the problem we intend to solve, some of them should be addressed by means of the heuristics provided (heuristics results may be validated by the human expert) and others only by the explicit intervention of the human user in the similarity identification process.

## 4.1. WSMX Mediation Approach

[Wiederhold, 1994] identifies three main approaches that deal with ontology integration. The first of these approaches implies the use of domain experts for creating definitions for the terms in the used ontologies, definitions that must be accepted by all parties involved. When these definitions are completely documented and released, all the parties have to conform and adjust to these definitions. The second approach is to assume that the terms from the subject ontologies are matching and further discovered mismatches are resolved by annotating the corresponding terms with some source or domain identifiers. The last approach, in contrast, assumes that all the terms from the given ontologies are distinct and denote different concepts when not otherwise specified. This kind of information is explicitly stated using mapping rules.

The first approach is used in [Dou et al., 2003] in the design of the OntoMerge system used for ontology merging and automated reasoning. The system uses a merged ontology (containing symbols and facts from the source and target ontologies, together with a set of bridging axioms) for performing translations between two ontologies. But this approach is not suited to our needs, our aim being to create a tool that provides the mapping rules in a semiautomatic manner and then executes them.

The second approach could be useful for cases where the subject ontologies use very similar terminology for modeling the problem domain. Unfortunately, the ontologies are usually created independently, so there could be major differences even in representing the same domain. So, the best choice for our mediation component remains the third one: all the concepts from the source and the target

ontology are different until mapping rules specify otherwise.

Following the requirements, our solution proposes three main steps for the WSMX mediation: the *mapping creation* step for dealing with similarity identification, *the mapping rules generation* step for creating the links and the *execution* step for performing the instance translation. Each of these steps is presented in detail in the following sections.

## 5. Mappings

The mappings have the role of expressing the links between the source and the target ontology. Each of these links should express the similarity degree existing between the linked entities. In addition, the mappings represent the connection between the design-time and the run-time phases of the mediation process. As a consequence, we can state that the mappings represent the critical point of any mediator system, directly influencing the quality, effectiveness and efficiency of the mediation process.

In this section we discuss a way of representing this mappings as well as a methodology for their creation. We investigate then how the mappings are acting on the data to be transformed and provide a classification of the helper functions (known as conversion or built-in functions) together with strategies that can be used for their creation. This section ends with a description of the run-time phase of the mediation process, mainly of the rules generated from the mappings and their execution.

### 5.1. Abstract Mappings

As mentioned before, in our view, mappings represent the critical factor in any mediation process. Furthermore, the mappings could be the connecting point for different mediator systems that might share ontology mappings between them in order to cooperate in solving more difficult tasks. As a consequence, the mappings should be expressed in such a way that offer maximum reusability as well as great expressivity in order to accommodate a large number of mediation scenarios.

We chose for expressing our mappings to use a subset of the mapping language developed in SEKT project [de Bruijn, 2004]. Even if this language borrows some of the features of WSML-Flight [de Bruijn et al., 2004] and OWL [Dean & Schreiber], it is not committed to any ontology representation formalism. Only an abstract syntax is provided and depending of its specific usage a formal semantics has to be associated to it. In our case, Section 9 provides a grounding of this abstract mapping language to Flora2 [Flora-2]. Listing 6 presents the abstract syntax of the language subset used in our approach, written in EBNF. For more information about this mapping language please refer to [de Bruijn, 2004].

Listing 5. The abstract syntax of the mapping language subset (from [de Bruijn, 2004])

```
expression ::= 'classMapping('['one-way'|'two-way']
                classExpr classExpr
                {attributeMapping} {classCondition}
                ['{'logicalExpression'}]')'
```

```

attributeMapping ::= 'attributeMapping(['one-way'|'two-way']
                        attributeExpr attributeExpr
                        {attributeCondition}')'

classExpr ::= classID
            | 'and('classExpr classExpr {classExpr}')'
            | 'or('classExpr classExpr {classExpr}')'
            | 'not('classExpr')'
            | 'join('classExpr classExpr {classExpr}
                    ['{logicalExpression}'])'

attributeExpr ::= attributeID
               | 'and('attributeExpr attributeEx
{attributeExpr}')'
               | 'or('attributeExpr attributeEx
{attributeExpr}')'
               | 'not('attributeExpr')'
               | 'inverse('attributeExpr')'
               | 'symmetric('attributeExpr')'
               | 'reflexive('attributeExpr')'
               | 'trans('attributeExpr')'
               | 'join('attributeExpr attributeEx
{attributeExpr}
                        ['{logicalExpression}'])'

classCondition ::= 'attributeValueCondition('attributeID
                  (individualID|dataLiteral|classExpr)')'
classCondition ::= 'attributeOccurrenceCondition('attributeID')

attributeCondition ::= 'valueCondi
('individualID|dataLiteral|classExpr)')'
attributeCondition ::= 'expressionCondition('attributeExpr)''

classID ::= URIReference
attributeID ::= URIReference
individualID ::= URIReference

dataLiteral ::= typedLiteral|plainLiteral
typedLiteral ::= lexicalForm^^URIReference
plainLiteral ::= lexicalForm['@'languageTag]

```

Our approach adds another level of abstractions in order to guarantee the reusability and composition of the work done during the design-time phase. This additional abstraction level involves breaking the mapping rules in atomic entities that are to be discovered and stored as such in a persistent storage during design-time and retrieved and compose in rules during run-time. This atomic entities are classified in the following categories:

- 1 **Concepts and attributes mappings.** They correspond to the simplest cases of *classMapping* and *attributeMapping*, respectively: 'classMapping(one-way classExpr classExpr)'' and

'attributeMapping(one-way' attributeExpr  
attributeExpr')'. [TO DO: discuss the possibility of defining two-way mappings]. We consider in the next sections for the sake of simplicity the **classExpr** and **attributeExpr** as being simple *classIds* and *attributeIds* (in the future versions of this deliverable the possibility of having more complex expressions for classes and attributes will be considered). Section 5.2 describes how these mappings are derived during design-time.

- 1 **Concept and Attribute Conditions.** They correspond to `classCondition` and `attributeCondition` respectively. They can be seen as a special case of built-in functions and they will be described in more details in Section 5.3. and Section 5.4.
- 1 **Conversion functions.** The conversion functions describe how the attributes value from the instances to be mediated are actually transformed. They correspond in the mapping language to logical expressions and in Section 5.3 and Section 5.4 you can find more details about their description and creation.

In the followings, we propose a way to create these atomic mapping entities, to store them in an efficient way and to compose them in complex mapping rules required by complex data transformations.

## 5.2. Mappings Creation

The first step is the most important as well as the most challenging. This step is concerned with the identifications of the possible links between the elements from the source and target ontologies, considering only the schema level of the input ontologies. These links are called *mappings* and each of these mappings connects either a pair of concepts or a pair of attributes. The semantic of such a concept pairs (*mapped concepts*) is that the involved concepts are model similar or related concepts in the source and target ontology, respectively. Each attributes pair (*mapped attributes*) denotes a way of realizing the mapping between the concepts that provide them (see Section 2.2. or [WSMO, 2004] for more details about concept and attribute definitions). Even though it has been the subject of intense research activity in the last ten years, the process of determining the possible mappings between two ontologies cannot be totally automated. That is, the human user input is still required at least in some of the key points of the mapping process. The challenge is how to minimize the user's (usually a domain expert's) effort and to reduce it to simple choices and validations.

The approach adopted in this direction aims to make full use of the advantages offered by the assumption that all input ontologies conform to the WSMO-Lite conceptual model for ontologies (see Section 2.2). This assumption allows the defining of strategies and mechanisms applicable to all concepts and to all attributes from both the source and the target ontologies.

Most of the strategies used in this step are based on the so-called *decomposition* mechanism. But in order to properly describe this mechanism it is necessary to introduce several other notions. We define the *source context* (*target context*) as a set of elements (only attributes or only concepts) from the source (target) ontology that could be chosen to be part of a mapping at a certain point of the mapping creation process. That is, at each stage, a source and a target context are defined and all the new mappings can be created only by using a pair from these contexts.

Further, we call a *compound concept* a concept that refers to other concepts through its attributes or relations. At this stage of our research, we consider only the attributes, so compound concepts are considered to be those concepts that have at least one attribute defined. The remaining concepts are called *primitive concepts* - concepts that are no further defined and don't contain any attributes (e.g. data types like string or integer). Accordingly, a *compound attribute* is an attribute that refers through its range to a compound concept and a *primitive attribute* is an attribute that refers in the same way to a primitive concept. As a consequence, we have *direct mappings* when both participants are primitive concepts or primitive attributes and *indirect mappings* otherwise. In fact, the last type of mapping is determined by zero or more other indirect mappings and one or more direct mappings (we'll see further in which way). In this document, all the mappings are considered in this document to be one-way mappings, even if in some particular cases they are bi-directional.

The decomposition mechanism (implemented by a recursive process) is applied each time a potential mapping has to be done. In each recursion step the contexts are updated accordingly and the decomposition process is further applied for other pair of elements chosen from the new context (the way in which the new pair is chosen is discussed later in this section). The recursive calls end when a direct mapping, or a mapping between two attributes with already mapped concept ranges, is done or when no other mappings are possible (or desired) in a certain step. When returning from recursion after at least one successful mapping, an indirect mapping between the chosen pair of elements is registered. In Table 1 the possible combinations of elements to be mapped are presented.

Table 1. Possible Mapping Situations

Source element	Target Element	Action
<b><i>Primitive Concept</i></b>	<b><i>Primitive Concept</i></b>	A direct mapping can be done and the decomposition process returns to the upper level (to the previous contexts) if there is one.
<b><i>Primitive Concept</i></b>	<b><i>Compound Concept</i></b>	This is an invalid combination because the source and the target concepts are considered to be part of different mapping levels. This combination would introduce artificial mappings (with no semantics) between primitive concepts and all the compound concepts that refer by means of their attributes to these primitive concepts. For example, <i>String</i> could be mapped with any compound concept A that has an attribute with the range <i>String</i> and also, in a recursive manner, with any other compound concept that has at least one attribute with the range mappable with <i>String</i> .
<b><i>Compound Concept</i></b>	<b><i>Primitive Concept</i></b>	This is an invalid combination (identical with the previously described one) because the source and the target concepts are considered to be part of different mapping levels. These two combinations are covered by the <i>Compound Concept to Compound Concept</i> mappings, our <i>Primitive Concept</i> being the range of an attribute owned by one of the compound concepts.
		The decomposition must be applied both on the source and on the target element; as a consequence, the

<b>Compound Concept</b>	<b>Compound Concept</b>	contexts will be updated in the following way: the source (target) context will be formed from the attributes owned by the source (target) concept. The decomposition process is applied further on a new pair of elements from the new source and target contexts respectively.
<b>Primitive Attribute</b>	<b>Primitive Attribute</b>	A direct mapping between these two elements is possible. In fact, in this case two steps are performed: the source and target context are updated to contain the range concept of the source attribute and the range concept of the target attribute respectively, and then primitive concept to primitive concept mapping is required. Then the decomposition process returns to the upper level (to the previous contexts) if there is one.
<b>Primitive Attribute</b>	<b>Compound Attribute</b>	The decomposition has to be applied to the target element. In this case the source context is updated to contain only the source attribute and the target context to contain all the attributes of the range concept of the target attribute. The decomposition process is applied further to a new pair of elements from the new source and target contexts respectively.
<b>Compound Attribute</b>	<b>Primitive Attribute</b>	This situation is similar to the one presented above, only the roles of the source and target elements are exchanged.
<b>Compound Attribute</b>	<b>Compound Attribute</b>	The decomposition process is applied to both source and target elements. The new contexts will contain the attributes owned by the ranges of the two attributes respectively. The decomposition process is applied further to a new pair of elements from the new source and target contexts respectively.

The pair of elements that is further considered in each decomposition step can be chosen in different ways. The simplest approach is to ask for the domain expert's inputs, and in this case the decomposition process has the role of offering guidance through the mapping definition process. Another possibility is to use the same decomposition process for computing suggestions based on the internal structure of decomposable elements and on lexical similarities of the names and non-functional properties. That is, the decomposition can be applied for any two concepts from the source and target contexts respectively, and, based on the number of direct and indirect mappings together with the lexical similarities, a global similarity measure can be computed. The pair with the higher similarity measure will be picked for the next step of the original decomposition process.

One has to keep in mind that the computed suggestion could represent in some cases not exactly what was desired, so the human expert intervention is still necessary (at least for validations). Even if the last solution is used, there are some cases when no valid suggestions are made (a valid suggestion is one that offers a similarity degree greater than a given threshold), and the human user intervention still remains a must. For this second approach, the decomposition process together with the computed similarities measures aim to reduce the human effort, transforming a laborious and error-prone process to simple choices

and validations. The general decomposition algorithm is outlined in Listing 7.

Listing 7. Decomposition Algorithm

```
decomposeConcepts(Concept A, Concept B){
    if (primitive(A) and primitive(B) then
        return;

    if ((primitive(A) and not(primitive(B)) or (not
(primitive(A)) and primitive(B)) then
        throw IllegalCombination(A,B);

    ContextA = getContext(A);
    ContextB = getContext(B);
    PairsCollection = choosePairsForDecomposition
(ContextA,ContextB);
    for each P in PairsCollection do
        decomposeAttributes(P.getSourceAttribute,
P.getTargetAttribute);
}

decomposeAttributes(Attribute A, Attribute B){
    if (primitive(A) and primitive(B) then
        decomposeConcepts(A.getRange; B.getRange);

    if (primitive(A) and not(primitive(B)) then{
        PairsCollection = choosePairsForDecomposition
({A},B.getRange.getContext);
        for each P in PairsCollection do
            decomposeAttributes(P.getSourceAttribute.
P.getTargetAttribute);
        return;
    }

    if (not(primitive(A)) and primitive(B)) then{
        PairsCollection = choosePairsForDecomposition
(A.getRange.getContext, {B});
        for each P in PairsCollection do
            decomposeAttributes(P.getSourceAttribute.
P.getTargetAttribute);
        return;
    }

    PairsCollection = choosePairsForDecomposition
(A.getContext, B.getContext);
    for each P in PairsCollection do
        decomposeConcepts(P.getSourceConcepts.
P.getTargetConcepts);
}
```

Note that in the case of a recursive definition of a concept (e.g. when the concept contains an attribute that has as range the concept itself) this algorithm will not terminate. A simple solution that can be adopted is to consider the attribute that introduces recurrence only a finite number of times in the decomposition process.

This solution is also adopted in the system that generates the source instances to be mediated; the problem remains to determine this number and exactly which of the attributes of the initial concept to consider for the last recursive step, in order to match the source instance. For now, the solution adopted is to allow only one recursive step, eliminating from the second step the attribute that introduces the recursion.

All the mappings, direct or indirect, are saved in an external storage in order to be retrieved later, either for the suggestion mapping process or for the mapping rules creation step.

### 5.3. Builtin (Conversion) Functions

### 5.4. Builtin (Conversion) Functions Creation

### 5.5. Mapping Rules Generation

After the schema level mappings between the two ontologies are done, these mappings are used to create the *mapping rules*. A mapping rule specifies the way in which the values in the first instance are extracted and used for the creation of the target instance(s). When there is a demand for converting a source instance into the corresponding target instance(s), the owner of the source instance is identified and then a search in the storage is performed to determine if there are some available mappings. If no mappings are available, the mapping rule cannot be created and in this case a return to the previous step is required. If for the given source concept one or more mapped concepts are found the similarity measures are used again for determining the most probable mapping. And finally, if there is an indirect mapping between the chosen pair of concepts, a simplified decomposition process is applied to determine all the indirect and direct mappings that lead to this mapping. In fact, all the involved indirect mappings are used for specifying the structure of the target instance(s) that has (have) to be created and the direct mappings deal with specifying the passing of the actual data to the target instance(s).

The language in which the mapping rules are generated is usually determined by the language in which the input ontologies and source instance are expressed; also the new target instance will be expressed in the same language. Some translators may be used to allow the usage of different languages (e.g. for expressing the target instance in another language).

Once a mapping rule is created, it can be saved for other potential usages or it can be redone each time this is necessary. If the used ontologies are unlikely to change, then it may be more efficient to store the mapping rules in a persistent storage and just reuse them when they are needed. If the ontologies are evolving, and they are frequently changing, the second approach would be more appropriate: consistency must be maintained only between the mappings in order to have consistent mapping rules.

### 5.6. Execution

This step deals with the execution of the existing mapping rules using an environment that understands the language in which the source instance and the

rules are represented. The result of the execution could be one or more instances of concept(s) from the target ontology. This step may also include the checking of the potential constraints defined in the source or in the target ontology and which affect the mapped elements. That is, using the execution environment one can check the correctness and the consistency of the mapping rules.

## 6. Addressed Ontology Mismatches

In this section, the set of problems that are addressed in our approach is presented. In the future, this set of problems will be extended and the next versions of this deliverable and of the provided prototype will offer solutions for a greater number of possible mismatches that can appear during the mapping process.

### 6.1. Different Composition of Concepts

This is the most common problem that can appear during the mapping process and it is caused by the fact that concepts that share the same semantic meaning are modeled in different ways, e.g. using different numbers and types of attributes and relations. That is, we can have different levels of conceptualizations used for modeling the same aspect of the domain, even if in the end, at the lowest level, we have the same primitive concepts. This problem is also described as granularity of match in or simply as granularity [Chalupsky, 2000].

This problem is addressed by applying the decomposition process, revealing the internal structure of the concepts, and recursively moving from one level to another till the primitive elements are encountered. What remains to be done is to create the direct mappings between these primitive elements; we will see later that this could be in some case a non-trivial task at all.

### 6.2. Incomplete/Over-complete Concept Representation

Another problem that may appear during the mapping process is that a required concept (either from the source or from the target ontology), does not have a similar concept in the other ontology. In this case a decision has to be made regarding the missing concept and this decision mostly depends on whether the missing concept appears in the source or in the target ontology. In the first case, the data will be mediated without any content loss (even if from the target point of view the data will be incomplete), but in the second case some of the content will be lost in the mediation process (but from the target point of view the data will be complete).

In general, for this kind of problem several solutions could be adopted, depending on the requirements of the two business partners. When the missing concept is on the source side the mapping rule must assure that in the created instance (target instance) a default value is filled in. Also there are some cases when simply ignoring the missing concept could be enough, the responsibility for dealing with the missing value being delegated to the application that is using the new created instances. When the missing concept is on the target side, the problem is more complicated. Normally, the target doesn't know what to do with an instance of a concept that it cannot understand, but there may be some situations in which the target will have to return back that instance, for example, as a participation proof

in the previous communication. So, a solution could be to simply discard, during the mapping rule execution, all the instances of an unmapped concept (assuming that there will be no need for these instances in the future) or mapping them as instances of a bag concept whose only role is to gather all unmapped instances.

The simplest solution is adopted for the first version of this deliverable: when the missing concept is in the source ontology no instance of the target concept is created; when the missing concept is in the target the corresponding source instance (or value) is discarded.

### 6.3. Data-Type Mismatch

Another common problem for the mediation process appears when concepts denoting the same aspects are defined using different data types. The most common example is when the data-type for an attribute is *string* in one of the ontologies, and for the similar one in the other ontology a more specific data type is used, like *integer*.

We could have different dimensions of this problem, depending of the casting compatibility of the two data types involved. In the above example the solution is very simple when the attribute having as data type an *integer* is part of the source ontology (*integer* can always be converted to a *string*) but if we consider it the other way around, the solution is not trivial at all, since not any *string* can be converted to an *integer*.

Considering our approach, this problem can appear only in the final phase of the mediation process, when mapping between primitive concepts, after all the required decompositions have already been done. Between the attributes, the mappings can be simple assignments from the source attributes to the target attributes, but also more complicated formulas that make the explicit conversion from one data format to the other. The checking of whether the mapping rule has specified the correct operation for converting from one data format to the other is actually done only when the rule is executed. The operations performing the required transformations can be seen as the context rules, described in [Wache, 1999].

### 6.4. Cardinality

When creating the mappings between ontologies, the following situation may also occur: some concepts from one of the ontologies are subsumed by one concept from the other ontology. In [Omelayenko & Fensel, 2001] this class of problems is referred as being 1:n, n:1 and n:m mapping cases and in [Rahm & Bernstein, 2001] it is considered to be part of the match cardinality process as set-oriented mapping case.

It is important to mention here that this problem appears only when mapping primitive elements, so no other concept (and implicitly instance) details are available. The cardinality problem complexity is dependent on where the subsumed concepts are placed in the source ontology or in the target ontology. In the first case, one can get from the subsumed concepts to the target concept by applying a certain operation (e.g. concatenation). In the second case it is not so trivial to create a mapping rule that splits a given instance (which does not have an explicitly defined structure) in a set of subsumed concepts instances. The only

solution for this situation is to find out details about the source instance's internal structure from other information sources (for example the instance provider). The most general case of subsumption is the n:m mapping, but it can be seen as m cases of 1:n mappings.

The problem presented in the above section is a particular case of the cardinality problem, from the point of view of the adopted solution. That is, for this case, we have to provide operations that are able to accept as input one or more source instances and to return one or more target instances. But one has to consider the fact that each of the owners might create their instances in their own way and it is the role of the mediation process to make the required transformations.

## 7. Flora2 - A reasoner for WSMX Mediation

## 8. From WSMML to Flora2

## 9. Expressing Mapping Rules in Flora2

## 10. Prototype Implementation

As described above, the mediation process we propose consists of three main steps: *mapping generation*, *mapping rules creation* and *execution*. Accordingly, the mediation module will consist of three main sub-modules or components: a graphical user interface for defining the mappings, a component that reads the mappings from the storage and generates the appropriate mapping rules and an environment that provides the means for executing the mapping rules. Figure 2 shows how these components interact.

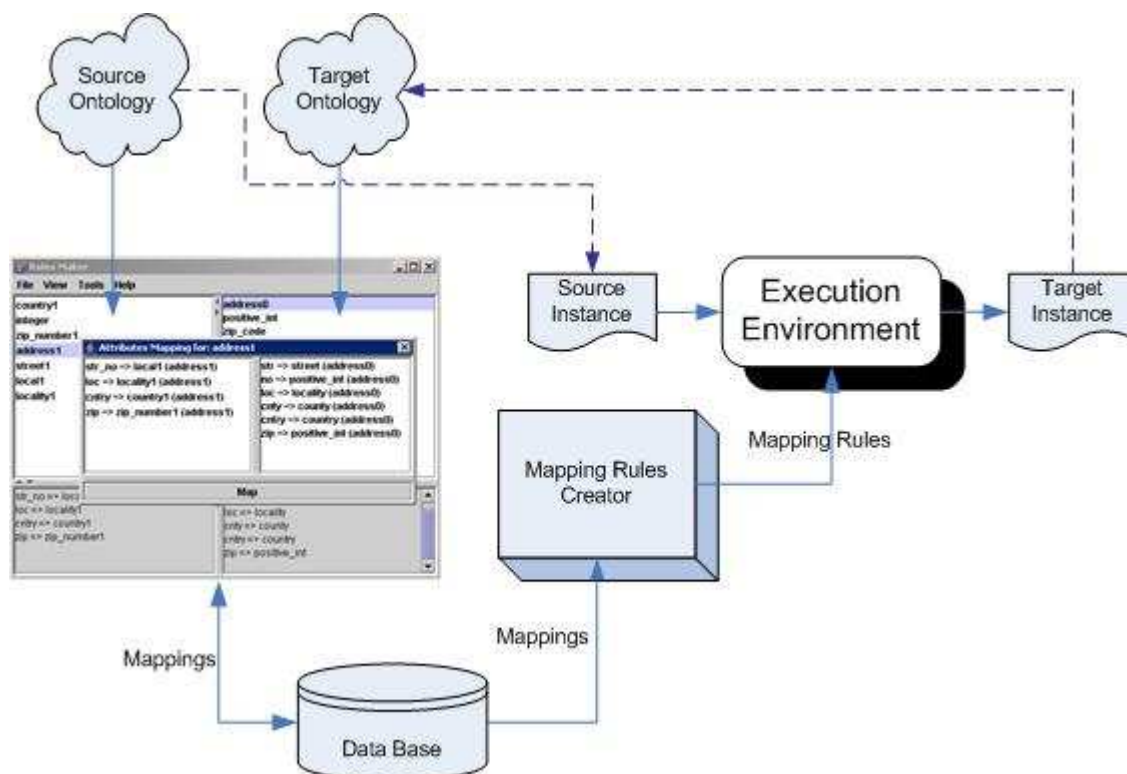


Figure 2. Overview of the Data Mediation Module

## 10.1. Graphical User Interface

The mapping creation process cannot be performed 100% automatically and as a consequence, the domain expert has to provide inputs at various stages of this process. Of course, the mappings could be done entirely manually, but this has proven to be a very laborious, error-prone and time-consuming process. Considering this, the mapping tools offer graphical interfaces in order to assist the user in the mapping creation and to reduce his or her effort to simple choices and validations. These kinds of graphical mapping tools are also described in [Maedche et al., 2002] and [Noy & Musen, 2000].

The graphical component described here has two main roles: firstly to guide the user to the mapping creation process, and secondly, to compute the suggestions based on already done direct and/or indirect mappings. The first task is accomplished by presenting the user the current source and target contexts containing the eligible elements for mapping. In fact, in this top-down approach, the user knows what elements s/he wants to map and through the decomposition process the contexts are updated in each step guiding the user to the primitive elements that will be mapped in the last step. This top-down approach may be very easily combined with a bottom-up approach (this being the recommended solution), in order to enable the suggestion mechanism to return better results. That is, the user can address in the beginning the mapping of some of the basic elements (including primitive elements), in this way constructing a basic vocabulary which will be referred in the next top-down mappings.

The second task is achieved by computing a similarity factor between the elements to be mapped. This similarity measure is based on two similarity factors: the first one is a heuristic function that computes an eligibility factor based on the already done mappings (direct or indirect mappings), and the second one is a lexical similarity factor based on the syntactical similitude between the name of the elements and their nonfunctional properties. These two factors are used together with the decomposition process in order to compute the similarities between compound elements as well as between primitive elements. It is important to notice that without establishing a basic set of mappings between primitive and basic elements of the two ontologies, the results returned by the first factor, the eligibility measure, cannot be accurate. For this, the mappings between primitive concepts could be determined by using the lexical factor, together with a thesaurus (e.g. WordNet).

The graphical interface is built in Java and is able to load Flora-2 [Flora-2] ontologies. The created mappings are stored in an external storage, in this case a relational database, from where they can be loaded by the mapping rules generator module. Also, by means of this graphical interface, the user can load already existing mappings from the external storage for further refinements or as support in computing the suggestions.

## 10.2. Mapping Rules Generator Module

This module deals with the creation of the mapping rules and expresses them in a format that can be later executed. It takes as input a concept from the source ontology and an identifier of the target ontology and searches the storage to find mappings between the given concept and concepts from the target ontology. There are three situations that can occur: no mappings to target concepts are

found, exactly one mapping to a concept from the target is found, or several mappings to different concepts from the target ontology are identified.

In the first case the creation of rule fails and an error message is returned. This means that a return to the previous step is required in order to provide suitable mappings. The second case, when only one mapping exists, is the easiest one to solve: the decomposition process is applied in order to consider all involved elements (attributes and concepts). But the third case is a little bit more complicated. Having more than one mapping could mean that there are several concepts in the target that are similar to the concept in the source ontology. Going further, and assuming that the mappings are correct, this means that in the target, some equivalent concepts are defined which usually is not a good modeling choice, unless this is explicitly specified (e.g. by inheritance relationships, or by subsumptions and aggregations). We will consider that the input ontologies are correctly modeled, so the source concept can be mapped to more equivalent concepts with a different degree of generality. As a consequence, a decomposition process is applied and a similarity factor is computed based on the indirect and direct mappings implied by the initial ones. Note that even if the target ontology is badly modeled and there are equivalent concepts without an explicit relation between them, the above described process will pick one, and due to these equivalences the choice will be as good as any other.

The generated mapping rules will be taken by the execution environment and optionally they could be stored for performance improvements or caching facilities. In our case the mapping rules are represented in Flora2 and no storing actions are performed.

### 10.3. Execution Environment

As mentioned above, the mapping rules are expressed in Flora so the execution environment is a Flora2 environment for XSB [Flora2]. This environment is wrapped by a Java interface and it is ready to take the source instance and the mapping rule and to return the target instance(s) by executing the rule. The target instance is also expressed in Flora but an XML description for it may be provided. For this version no other computations are done in this environment, but it could be used also for testing the validity of the mappings and to perform constraints checking.

The last two sub-modules are used by WSMX at runtime to provide and to execute the mapping rules. For each source instance provided by WSMX, the available mappings are retrieved from the storage and a mapping rule is created. The newly created mapping rule is forwarded to the execution module, executed, and the result (i.e. the new target instance) is returned to WSMX. The graphical user interface was used earlier on, before runtime, for defining and storing the mappings between similar entities.

The first version of this prototype allows the user (more or less a domain expert) to create a set of mappings between concepts and attributes of two ontologies. The types of mismatches that can be addressed and solved using this prototype were described in sections 4.2.1, 4.2.2, 4.2.3, and 4.2.4. The process of creating mappings could be seen as an iterative one: the user should analyze the suggestion offered, implement some of them and then re-explore the new set of suggestions offered (the suggestions are strongly affected by the created

mappings: better suggestions are obtained based on a higher number of mappings). Once the mapping process has ended, the mappings can be dropped in the external storage to be picked by the run-time mediation component, transformed in mapping rules and executed in order to obtain the target instances.

## 11. Related Work

In this section we present two related approaches to ontology to ontology mediation. We chose to present PROMPT [Noy & Musen, 2000] and MAFRA [Maedche et al., 2002], because of their similarities to our approach both in the conceptual framework proposed and in the software implementation they offer.

### 11.1. PROMPT

PROMPT is an algorithm and a tool proposed by Noy and Musen [Noy & Musen, 2000] which allows semi-automated ontology merging and alignment. It takes as inputs two ontologies and guides the user through an iterative process for obtaining a merged ontology as an output. This process starts with the identification of the classes with similar names and provides a list with initial matches. Then the following steps are repeated several times: the user selects an action (by choosing a suggestion or by editing the merged ontology directly) and the tool computes new suggestions and determines the eventual conflicts. This iterative process is presented in Figure 3.

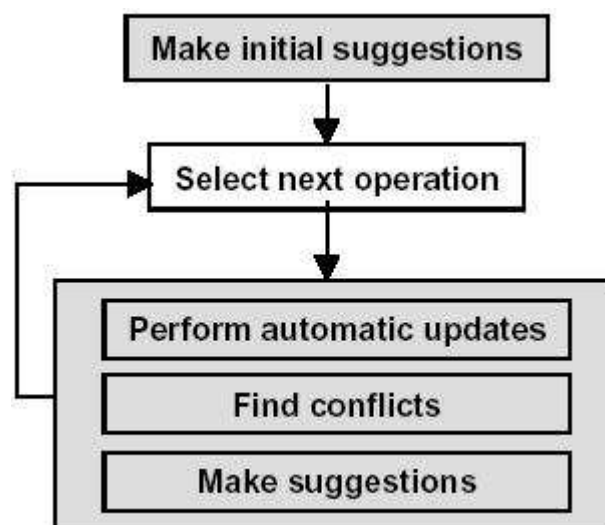


Figure 3. The PROMPT Algorithm

PROMPT is able to handle ontologies conforming to OKBC [Chaudri et al., 1998] format having at the top level classes, slots, facets, and instances. The operations allowed include merge classes, merge slots, merge binding between a slot and a class, perform deep/shallow copy of a class from one ontology to another (including/not including all the parents of the class up to the root of hierarchy and all the classes and slots it refers to). Some of the conflicts identified by PROMPT are name conflicts, dangling references (a frame refers to another frame that doesn't exist), redundancy in the class hierarchy (more than one path from a class to a parent other than root) or slot-value restriction that violates class inheritance.

### Implementation

PROMPT was implemented as a plugin in Protege-2000 - a knowledge base acquisition tool, and by this PROMPT can take advantage of all the ontology engineering capabilities of this tool. Some of the features offered by this implementation are: setting the preferred ontology (for automatic conflict-solving in favor of one ontology), maintaining the user's focus (suggestions the user sees first should relate with the frame in the same area), providing explanations for the suggestions made, logging and reapplying operations.

## 11.2. MAFRA

MAFRA [Maedche et al., 2002] is a Mapping Framework for Distributed Ontologies, designed to offer support at all stages of the ontology mapping life-cycle. Also, a partial implementation is offered for this framework as part of the Kaon Ontology and Semantic Web Framework.

The architecture of the MAFRA conceptual framework is presented in Figure 4.

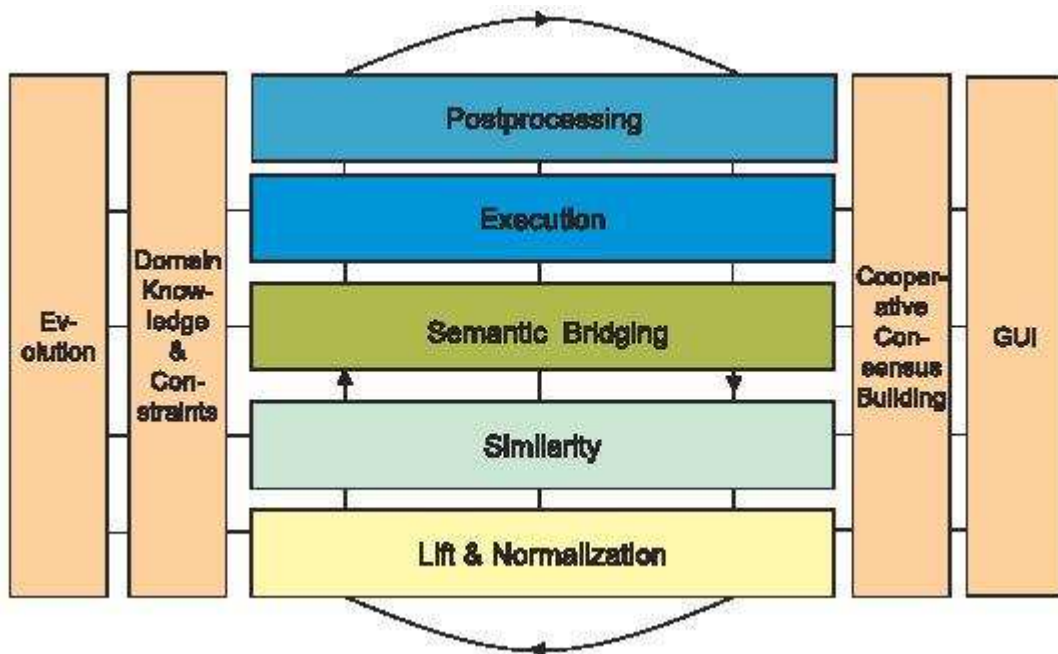


Figure 4. Conceptual Architecture

The framework is organized in two dimensions: it contains horizontal and vertical modules. The horizontal modules (*Lift & Normalization*, *Similarity*, *Semantic Bridging*, *Execution*, *Post-processing*) describe fundamental and distinct phases in the mapping process, while the vertical modules (*Evolution*, *Domain Constraints & Background Knowledge*, *Cooperative Consensus Building*, *Graphical User Interface*) run along the entire mapping process interacting with the horizontal modules.

### Horizontal Dimension

The *Lift & Normalization* module has the role of coping with syntactical, structural and language heterogeneity, raising all the data to be mapped to the same representation language. Both ontologies are normalized to a uniform representation, in this case RDF(S). The *Similarity* module implements strategies for discovering the similarities between entities. It determines lexical similarity, as

well as so called property similarities - similarities between concepts based on their properties, either attributes or relations. Also it defines bottom-up/top-down similarities for propagating the similarities or dissimilarities from lower/upper parts of the taxonomy to the upper/lower concepts. *Semantic bridging*, based on the similarities previously determined, establishes correspondences between entities from the source and target ontologies. It specifies bridges between entities in a way that each instance represented according to the source ontology is translated into the most similar instance described according to the target ontology. The *Execution* module actually transforms instances from the source ontology to the target ontology, by evaluating the semantic bridges defined in the previous step, and the *Post-processing* module takes the results to check and improve the quality of the transformations.

### Vertical Dimension

*Evolution* focuses on keeping the bridges obtained by the Semantic Bridging module up to date, consistent with the transformation and the evolutions that may take place in the source and target ontologies. *Cooperative Consensus Building* is responsible for establishing a consensus between two communities relative to the semantic bridges used. The need for this module comes from the multiple bridges available for performing transformations and from the aim of reducing the human contribution in the mapping process. *Domain Constraints & Background Knowledge* may improve substantially the quality of bridges by using background knowledge and domain constraints, as glossaries, lexical ontologies or thesauri. Finally, the *Graphical User Interface* is required due to the fact that the mapping creation is a difficult and a time consuming process, thus extensive graphical support must be provided.

## 12. Conclusions and Further Directions

This deliverable provides solutions for ontology-to-ontology mediation problems, as a starting-point for the more complex problem of mediation in the context of Semantic Web Services. Our intentions are to provide conceptual foundations for a semi-automatic approach to ontology-to-ontology mediation, together with a software implementation as a testbed and a proof of their validity. Also, the software implementation is integrated in Web Service Execution Environment (WSMX) as a default implementation of the mediation component.

In our future work, the focus will be on improving the solutions proposed for ontology mediation, including the extension of the problem set we address, together with the creation of better heuristics for computing suggestions. Another aim is to further explore the mediation problems in the context of WSMO, offering solutions for the other types of mediators which were not addressed in this document and integrating their implementation into the WSMX mediation component.

## References

[Chalupsky, 2000] H. Chalupsky: OntoMorph: A Translation System for Symbolic Knowledge. In *Proceedings of 7th International Conference on Knowledge Representation and Reasoning (KR), Breckenridge, (CO US), pages 471-482, 2000.*

**[Chaudhri et al., 1998]** V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, J. P. Rice: OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* (pp. 600-607). Madison, Wisconsin, USA: MIT Press, 1998.

**[de Bruijn, 2004]** J. de Bruijn: Ontology mediation patterns. SEKT Project Deliverable D4.3.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.

**[de Bruijn et al., 2004]** J. de Bruijn, D. Foxvog, H. Lausen, E. Oren, D. Roman, and D. Fensel: The WSML Family of Representation Languages. Deliverable D16v0.2, WSML, <http://www.wsmo.org/wsml/>, 2004. Available from <http://www.wsmo.org/2004/d16/v0.2/>

**[Dean & Schreiber]** M. Dean and G. Schreiber (eds.): OWL Web Ontology Language Reference. 2004. W3C Recommendation 10 February 2004.

**[Doan et al., 2002]** A. Doan, J. Madhavan, P. Domingos, A. Halevy: Learning to Map Between Ontologies on the Semantic Web. In *WWW2002*, 2002.

**[Dou et al., 2003]** D. Dou, D. McDermott, P. Qi: Ontology Translation on the Semantic Web. In *ODBASE'03*, 2003.

**[Fensel & Bussler, 2002]** D. Fensel, C. Bussler: The Web Service Modeling Framework (WSMF). In *White Paper and Internal Report Vrije Universiteit Amsterdam*, 2002.

**[Flora-2]** Available at <http://flora.sourceforge.net/>

**[Lin et al., 2001]** H. Lin, T. Risch T. Katchaounov: Adaptive Data Mediation Over XML Data. In *Journal of Applied System Studies (JASS)*, Cambridge International Science Publishing, 2001.

**[Madhavan et al., 2002]** J. Madhavan, P. A. Bernstein, P. Domingos, A. Y. Halevy: Representing and Reasoning About Mappings Between Domain Models. *Eighteenth National Conference on Artificial Intelligence*, p.80-86, Edmonton, Alberta, Canada, July 28-August 01, 2002.

**[Maedche et al., 2002]** A. Maedche, B. Motik, N. Silva, R. Volz: MAFRA - A Mapping Framework for Distributed Ontologies. In *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW*, Madrid, Spain, September 2002.

**[Mitra et al., 2000]** P. Mitra, G. Wiederhold, M. Kersten: A Graph-Oriented Model for Articulation of Ontology Interdependencies. In *Proceeding Extending DataBase Technologies, Lecture Notes in Computer Science*, vol. 1777, pp. 86-100, Springer, Berlin Heidelberg New York, 2000.

**[Noy & Musen, 2000]** N. Noy, M. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

**[Omelayenko & Fensel, 2001]** B. Omelayenko, D. Fensel: An Analysis of

Integration Problems of XML-Based Catalogues for B2B E-commerce. In *Proceedings of the 9th IFIP 2.6 Working Conference on Database (DS-9), Semantic Issues in e-commerce Systems*, Hong Kong, April 2001.

**[Rahm & Bernstein, 2001]** E. Rahm, P. Bernstein: A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal* 10: 334-350, 2001.

**[Visser et al., 1997]** P. R. S. Visser, D. M. Jones, T. J. M. Bench-Capon, M. J. R. Shave: An Analysis of Ontological Mismatches: Heterogeneity Versus Interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.

**[Wache, 1999]** H. Wache: Towards Rule-based Context Transformation in Mediators. In *S. Conrad, W. Hasselbring, and G. Saake, editors, International Workshop on Engineering Federated Information Systems (EFIS 99)*, Kuhlungsborn, Germany, 1999.

**[Wache et al., 2001]** H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hubner: Ontology-Based Integration of Information - A Survey of Existing Approaches. In *Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing*, Vol. pp. 108-117, Seattle, WA, 2001.

**[Wiederhold, 1994]** G. Wiederhold: Interoperation, Mediation, and Ontologies, In *Proceedings International Symposium on Fifth Generation Computer Systems (FGCS94), Workshop on heterogeneous Cooperative Knowledge-Bases*, Vol. W3, pages 33-48, Tokyo, Japan, Dec. 1994.

**[WSMO-Lite, 2004]** D. Roman, U. Keller, H. Lausen (eds.): Web Service Modeling Ontology - Lite (WSMO-Lite), version 0.2, 2004, available at <http://www.wsmo.org/2004/d11/v0.2/>

**[WSMO-Full, 2004]** C. Preist, D. Roman (eds.): Web Service Modeling Ontology - Full (WSMO-Full), WSMO Working Draft v01, 2004, available at <http://www.wsmo.org/2004/d12/v0.1/20040624/>

**[WSMO, 2004]** D. Roman, U. Keller, H. Lausen (eds.): Web Service Modeling Ontology - Standard (WSMO - Standard), version 0.2, 2004, available at <http://www.wsmo.org/2004/d2/v1.0/>

**[WSMX, 2004]** E. Oren, M. Zaremba, M. Moran: Overview and Scope of WSMX, WSMO Working Draft v01, 2004, available at <http://www.wsmo.org/2004/d13/d13.0/v0.1/>

**[WSMX\_O, 2004]** E. Cimpian, A. Mocan, M. Moran, E. Oren, M. Zaremba: Web Service Modeling Execution Environment – Conceptual Model (WSMX\_O), v 0.1 available at <http://www.wsmo.org/2004/d13/d13.1/v0.1>

**[WSMX Execution Semantics, 2004]** E. Oren: WSMX Execution Semantics, WSMO Working Draft v01, 2004, available at <http://www.wsmo.org/2004/d13/d13.2/v0.1/20040531/index.pdf>

**[WSMX Architecture, 2004]** M. Moran, M. Zaremba: WSMX Architecture, WSMO Working Draft v01, 2004, available at

**[WSMX Implementation, 2004]** M.Moran, E. Cimpian, A. Mocan, E. Oren: WSMX Implemenattion, WSMO Working Draft v01, 2004, available at <http://www.wsmo.org/2004/d13/d13.5/v0.1/20040719/>

## Acknowledgments

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, and SWWS; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the CoOperate program.

The editors would like to thank all the members of the WSMO working group for their advice and inputs to this document.

## Appendix A. WSML-Core to Flora2 Translation

*The corresponding mappings between WSML-Core and Flora2*

WSML-Core conceptual syntax	Flora2 syntax	Remarks
<i>Logical Declarations</i>		
<b>instance</b> <i>o</i> <b>memberOf</b> <i>C1, ..., Cn</i> <i>A1</i> <b>hasValue</b> <i>V1</i> ... <i>Am</i> <b>hasValue</b> <i>Vm</i>	<i>o:C1. ... o:Cn</i>  <i>o[A1 -&gt; V1,</i> ... <i>An -&gt; Vn]</i>	

Table 2: Mapping between WSML-Core and Flora2 syntax

*Translating WSML-Core abstract syntax into Flora2 syntax by using a recursive translation function*

### General

```
//id
//id - iri
t('<" iri_reference ">') →
  " iri_reference "

t(ncname1 ':' ncname2) →
  t(ncname2)
```

```

t('_' nnamechar1 ... nnamecharn) →
    '_' nnamechar1 ... nnamecharn

t([0x0041 ... 0x005A] nnamechar1 ... nnamecharn) →
    "[0x0041 ... 0x005A] nnamechar1 ... nnamecharn"

t([0x0061 ... 0x007A] nnamechar1 ... nnamecharn) →
    [0x0061 ... 0x007A] nnamechar1 ... nnamecharn

```

```

//id - anonymous
t(anonymous) →
    anonymous

```

```

//id - literal
t(plainliteral '^' iri) →
    t(plainliteral)

```

```

t(" literal_content ") →
    "literal_content"

```

```

t(number) →
    number

```

```

t(string) →
    string

```

```

t('true') →
    'true'

```

```

t('false') →
    'false'

```

## NonFunctionalProperties

```

//nfp
t('nfp' attributevalue1 ... attributevaluen 'endnfp') →
    'nonFunctionalProperties' '->' '_#' '[' t(attributevalue1) '!' ... '!' t(attributevaluen)
    ']'

```

```

t('nonFunctionalProperties' attributevalue1 ... attributevaluen
'endNonFunctionalProperties') →
    'nonFunctionalProperties' '->' '_#' '[' t(attributevalue1) '!' ... '!' t(attributevaluen)
    ']'

```

## Attributes

```

//attribute value
t(id 'hasValue' id) →
    t(id) '->' t(id)

```

$t(\text{id 'hasValue' '{' id ',' id}_1 \text{' ... ',' id}_n \text{'}}) \rightarrow$   
 $t(\text{id}) \text{'->>' '{' } t(\text{id}) \text{' , ' } t(\text{id}_1) \text{' , ' } \dots \text{' , ' } t(\text{id}_n) \text{' }'$

## Instances

//instances

$t(\text{'instance' id memberof nfp attributevalue}_1 \dots \text{attributvalue}_n) \rightarrow$   
 $t(\text{memberof, id}) \text{'.' } t(\text{'instance' id nfp attributevalue}_1 \dots \text{attributvalue}_n)$

$t(\text{'instance' id memberof nfp}) \rightarrow$   
 $t(\text{memberof, id}) \text{'.' } t(\text{'instance' id nfp})$

$t(\text{'instance' id memberof attributevalue}_1 \dots \text{attributvalue}_n) \rightarrow$   
 $t(\text{memberof, id}) \text{'.' } t(\text{'instance' id attributevalue}_1 \dots \text{attributvalue}_n)$

$t(\text{'instance' id memberof}) \rightarrow$   
 $t(\text{memberof, id})$

$t(\text{'instance' id nfp attributevalue}_1 \dots \text{attributvalue}_n) \rightarrow$   
 $t(\text{id}) \text{'[' 'nonFunctionalProperties' '->' } t(\text{nfp}) \text{' , '}$   
 $\quad \quad \quad t(\text{attributevalue}_1) \text{' , ' } \dots \text{' , '}$   
 $\quad \quad \quad t(\text{attributevalue}_n) \text{' ]'}$

$t(\text{'instance' id nfp}) \rightarrow$   
 $t(\text{id}) \text{'[' 'nonFunctionalProperties' '->' } t(\text{nfp}) \text{' ]'}$

$t(\text{'instance' id attributevalue}_1 \dots \text{attributvalue}_n) \rightarrow$   
 $t(\text{id}) \text{'[' } t(\text{attributevalue}_1) \text{' , ' } \dots \text{' , '}$   
 $\quad \quad \quad t(\text{attributevalue}_n) \text{' ]'}$

$t(\text{'instance' id}) \rightarrow$   
 $t(\text{id})$

//memberof

$t(\text{memberof, X}) \rightarrow$   
 $t(\text{'memberOf' idlist, X})$

$t(\text{'memberOf' '{' id ',' id}_1 \text{' ... ',' id}_n \text{'}, X) \rightarrow$   
 $t(\text{'memberOf' id, X}) \text{'.' } t(\text{'memberOf' id}_1, X) \text{'.' } \dots \text{'.' } t(\text{'memberOf' id}_n, X)$

$t(\text{'memeberOf' id, X}) \rightarrow$   
 $X \text{'.' } t(\text{id})$

## Appendix B. Changelog

2005.01.28

- 1 This changelog was added
  - 1 The Appendix A, containing an initial version of WSML-Core to Flora2 translation was added. This first version covers only the instance transformations from WSML to Flora2.
  - 1 The introduction in section 5 as well as subsection 5.1
- 



webmaster