D2.4.12 Data Mediation in Semantic Web Services

Coordinator: Adrian Mocan (University of Innsbruck)

with contributions from:
François Scharffe (University of Innsbruck)
Emilia Cimpian (University of Innsbruck)
Tomas Vitvar (National University of Ireland, Galway)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.4.12 (WP2.4)

The deliverable presents an ontology-based data mediation framework, that can be used to perform data transformation during Web services' operation in a semantic environment. The framework includes a formal model which bridges the design-time and the run-time phase of the mediation process.
Keyword list: data mediation, ontology mapping, alignment format, mapping language, instance transforation, formal model

Document Identifier	KWEB/2006/D2.4.12/v1.0
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	December 31, 2006
State	final
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

University of Aberdeen (UNIABDN)
Kings College
AB24 3FX Aberdeen
United Kingdom
Contact person: Jeff Pan
E-mail address: jpan@csd.abdn.ac.uk

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Freie Universität Berlin
National University of Ireland Galway
Universidad Politécnica de Madrid
Universidad Politécnica de Catalunya
University of Economics, Prague
University of Innsbruck
University of Liverpool
University of Manchester
University of Trento
École Polytechnique Fédérale de Lausanne

Changes

Version	Date	Author	Changes
0.2	12.06.06	Adrian Mocan	creation
0.3	18.08.06	Adrian Mocan	first draft completed
0.4	14.12.06	François Scharffe	adding the updates on the alignment format
0.5	19.12.06	Adrian Mocan	Small updates and fixes
0.9	20.01.07	Adrian Mocan	Addressing the reviewer comments
1.0	01.02.07	Adrian Mocan	Creating the final version

Executive Summary

In a semantic environment data is described by ontologies and heterogeneity problems have to be solved at the ontological level. This means that alignments between ontologies have to be created, most probably during design-time, and used in various run-time processes. Such alignments describe a set of mappings between the source and target ontologies, where the mappings show how instance data from one ontology can be expressed in terms of another ontology. In this deliverable we propose a formal model for creating mappings and we explore how such a model maps onto a design-time graphical tool that can be used in creating alignments between ontologies. In the other direction, we investigate how such a model helps in expressing the mappings in a logical language, based on the semantic relationships identified using the graphical tool.

Contents

Chapter 1

Introduction

Semantic Web technologies have become more and more popular during the last decade. Based on simple and appealing ideas (common understanding of data, common formats for representing these data), the Semantic Web aims at providing a framework that would allow information sharing across the Web in a manner understandable not only by humans, but also by machine.

The agreed-upon format for representing this information are ontologies, but the representation of data using ontologies cannot guarantee the homogeneity and consistency of information. Even if the ontologies are supposed to be ”formal explicit specification of a shared conceptualization” [Gru93], they are usually developed in isolation, and shared within well defined boundaries. This leads to the development of different conceptualizations of the same domain, different ontologies modeling the same aspects but in different manners.

In this context, ontology mapping is becoming a crucial aspect in solving heterogeneity problems with semantically described data. The benefits of using ontologies, especially in heterogeneous environments where more than one ontology is used, can only be realized if this process is both correct and efficient. The trend is to provide graphical tools capable of creating alignments at design-time in a (semi-)automatic manner [ESS05, MC05, NM03, SR03]. These alignments consist of mapping rules, frequently described as *statements* in a logical language, and can be executed for performing the actual mapping when needed. One of the main challenges is to fully isolate the domain expert (who is indispensable if 100% accuracy is required) from the burdens of dealing with formal logics by using a graphical tool, and at the same time to be able to create complex, complete and correct mappings between the ontologies.

We consider it absolutely necessary to formally describe the mapping creation process and to link it with the instruments available in the graphical tool and with a mapping representation formalism that can be used later at run-time. This allows the capturing of the actions performed by the human user in a meaningful way with respect to the visualized ontology structure and then to associate the results of these actions (mappings)

with concrete statements in a mapping language (mapping rules).

The deliverable structure is as follows: the next section presents the context and motivation for the work. Section 3 introduces the model we propose expressed using First-Order Logic [GN88]. Section 4 describes how this model can be applied to WSMO [FPD+05] ontologies, while Section 5 presents the creation of mapping rules; the prototype that implements and applies the proposed formal model is described in Section 6. Following this related work and conclusions are presented.

Chapter 2

Context and Motivation

The work described in this deliverable has been carried out in relation with the Web Service Execution Environment (WSMX) working group, whose scope is to build a framework that enables discovery, selection, mediation, invocation and interoperation of Semantic Web Services [MMCZ06]. Web Services are semantically described using ontologies, but as they are generally developed in isolation, heterogeneity problems appear between the underlying ontologies. Without resolving these problems the communication (data exchange) between different Semantic Web Services cannot take place.

The data mediation process in WSMX requires two phases: a *design-time* and a *run-time* phase. The mismatches between the ontologies are identified at design time[1], while the semantic relationships found are expressed as mapping rules. These mapping rules are used at run-time to transform the data passing through the system. The run-time phase can be completely automated, while the design-time phase remains semi-automatic, requiring the inputs of a domain expert.

For the design-time phase a semi-automatic ontology mapping tool has been developed that allows the user to create alignments between ontologies and to make these alignments available for the run-time process. There has been much research in the area of graphical mapping tools, e.g. [NM03, SR03], however we believe there are many challenges still to be addressed. In particular, our focus has been on providing the user with proper support (e.g. suggestions and guidance), and in defining strategies that hide the burden of logical languages that are generally used to express ontology alignments, from the domain expert.

The suggestion of correct mappings is accomplished by using a set of algorithms for both lexical and structural analysis of the concepts. A brief description of these algorithms is provided in Section 6.1.3. Additionally, the guidance is offered by decomposition and context updates (as described in Section 3.3). The second aspect, to better isolate the domain expert from the burden of logical languages, is achieved by the use of several

[1] In this work we use *design time* to denote the time when the mapping is created and not the time when the ontologies have been engineered.

perspectives, or views[2], on the ontology that help in identifying and capturing various mismatches only by graphical means (more details are offered in Section 4). All these features are formalized in a model that creates a bridge between the graphical mapping tool and the result of the mapping process (the ontology alignment). In the following section we will describe this model, together with the main principles that support the graphical instruments and how they fit with the underlying logical mechanism.

As described in [MMCZ06], the graphical point of view adopted to visualize the source and target ontologies makes it easier to identify certain types of mappings. We call this viewpoints *perspectives* and argue that only by switching between combinations of these perspectives on the source and target ontologies, can certain types of mappings be created using only one simple operation, *map*, combined with mechanisms for ontology traversal and contextualized visualization strategies. The following sections describes various types of perspectives and provides examples of equivalences (mappings) that can be easily identified and described in a certain perspective but difficult or impossible in another ones.

A formal model that describes the general principles of the perspectives allows for a better understanding of the human user actions in the graphical tool and of the effects of these actions on the ontology alignment (i.e. the set of mapping rules) that is being created. This model defines the main principles that support the graphical instruments (e.g. perspectives) and how they fit with the underlying logical mechanisms (e.g. decomposition, context updates). The same model is also used to describe how the inputs placed through these graphical instruments by the domain expert affect the generated mappings. Having this formal model as a link between the graphical elements and the mappings, defines precisely the process of hiding from the domain expert the complexity of the underlying logical languages. It also allows some of the mapping properties such as (in)completeness or (in)consistency to be reflected back into the graphical tool. Additionally, such a model allows experts to become more familiar with the tools and to create extensions that are more suited for capturing certain types of mismatches.

[2]In [MMCZ06] the perspectives are called views. In order to avoid suggesting a straight forward connection with the data base views (even if they share indeed some similarities) to the readers from the data base community, the term perspective is used from now on in this deliverable to denote this concept.

Chapter 3

A Model for Mappings Creation

This section defines a model and all its elements to be used in the creation of mappings between ontologies. The roles that appear in the graphical user interface, and which will be later associated with the ontological entities, are defined here. First-Order Logic [GN88] is used as a formalism to represent this model.

3.1 Running Example

This section introduces the examples that will be further used to illustrate the concepts introduced in the rest of the deliverable. Elements of the the fragment ontology O_1[1] (see Listing 3.1) need to be mapped to elements described in the ontology fragment O_2 (depicted in Listing 3.2).

Listing 3.1: Ontology Fragment O_1

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://see.deri.org/adrian/ontologies/O1#" ,
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
     dc _"http://purl.org/dc/elements/1.1/" }

ontology _"http://see.deri.org/thesis/ontologies/O1"
     nonFunctionalProperties
          dc#description hasValue "A simple ontology fragment
                                   modeling the concept of person"
     endNonFunctionalProperties

concept person
    name ofType _string
    age ofType _integer
    hasGender ofType gender
    hasChild ofType person
    marriedTo ofType person
```

[1]The fragments of ontologies analyzed in this section are represented using the Human Readable Syntax of Web Service Modeling Language (WSML) [HLF05].

```
concept gender
    value ofType _string

instance male memberOf gender
    value hasValue "male"
instance female memberOf gender
    value hasValue "female"
```

Listing 3.1 presents an example of concepts and their attributes, and some instances of these concepts. The concept *person* is modeled as having 5 attributes, each of them having a type (i.e. a range) that is either another concept or a data type. For the concept *gender* there are two instances defined (i.e. *male* and *female*) that have attributes pointing to values of the corresponding types.

Listing 3.2: Ontology Fragment O_2

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://see.deri.org/adrian/ontologies/O2#" ,
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
     dc _"http://purl.org/dc/elements/1.1/" }

ontology _"http://see.deri.org/thesis/ontologies/O2"
     nonFunctionalProperties
          dc#description hasValue "A simple ontology fragment
                                   modeling the concept of human"
     endNonFunctionalProperties

concept human
    name ofType _string
    age ofType _integer
    noOfChildren ofType _integer

concept man subConceptOf human

concept woman subConceptOf human

concept marriage
    hasParticipant ofType human
    date ofType _date
```

The ontology part in Listing 3.2 defines the concept of *human* having three attributes, *name*, *age* and *noOfChildren*, of type *string*, *integer* and *integer* respectively. The concepts of *man* and *woman* are subclasses of the concept *human*. Additionally, a third concept called *marriage* is defined having two attributes *hasParticipant* and *date* of type *human* and *date* respectively.

In WSML all data types names are prefixed with "_" - in this example *_integer*, *_string* and *_date* are datatypes. Actually, WSML defines a number of built-in functions [dBLK+05] for the use of XML Schema datatypes as they are described in [BM04]. As such, is important to note that in WSML, attributes can have as types either datatypes or concepts.

Assuming that we want to map the concept *person* from O_1 to the concept *human* of O_2 the easiest way to start is to visualize the ontologies using a frame based approach

where concepts are considered to be the main elements of the visualization, and the attributes and their range (value type) can only be displayed as part of the concept; the concepts' instances are not displayed in this perspective. This visualization mode is called the *PartOf* perspective. The two fragments of ontologies presented using this perspective, are displayed as in Figure 3.1.

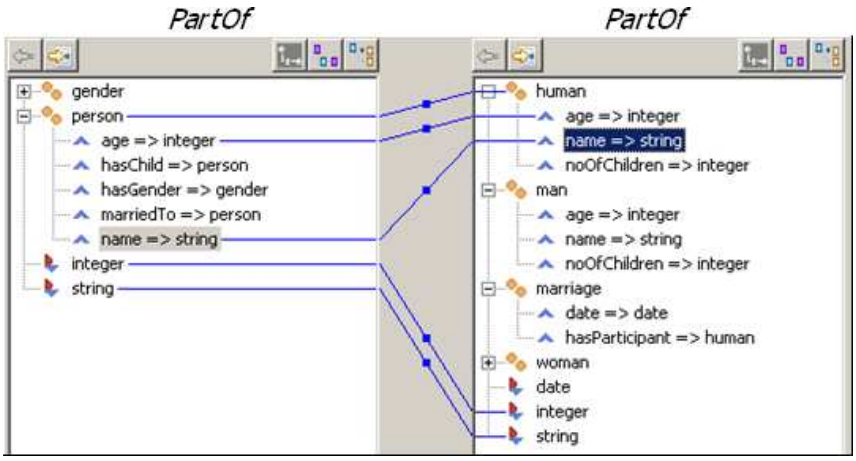

Figure 3.1: *PartOf* Perspective for O_1 and O_2

Using these perspectives it will be very easy for a domain expert to identify, for example, all the elements that need to be mapped from the *person* concept from O_1 to the *human* concept from O_2. However, several of *person*'s attributes can not be directly mapped (at least not in a *natural* way[2]) using this perspective, but by changing the perspectives the mappings become possible.

For example, the *marriedTo* attribute from O_1 should be mapped to the *marriage* concept from O_2. For this we need to change the perspective in the first ontology, so that *marriedTo* is displayed as a main entity, and the concept it belongs to and also its range (value type) are displayed as part of the attribute. This perspective is called *RelatedBy* (Figure 3.2).

Another problem occurs when trying to map the gender attribute from O_1. By simple looking at this short fragments of ontologies, a human user can easily determine that if a person's gender attribute has the value *male* it should be mapped to the *man* concept, and if its value is *female* it should be mapped to the *woman* concept. The graphical user interface should offer support in modeling this kind of mappings conditioned by the values of the attributes. For this a third perspective, *InstanceOf* is introduced; in this perspective the focus is on the actual instances that can fill the attributes of concepts in the ontology (Figure 3.3).

[2]As further described in this deliverable what we informally call here "a natural way" of creating mappings is actually reflected in the formal model as a consistent and uniform methodology for creating ontology mappings.

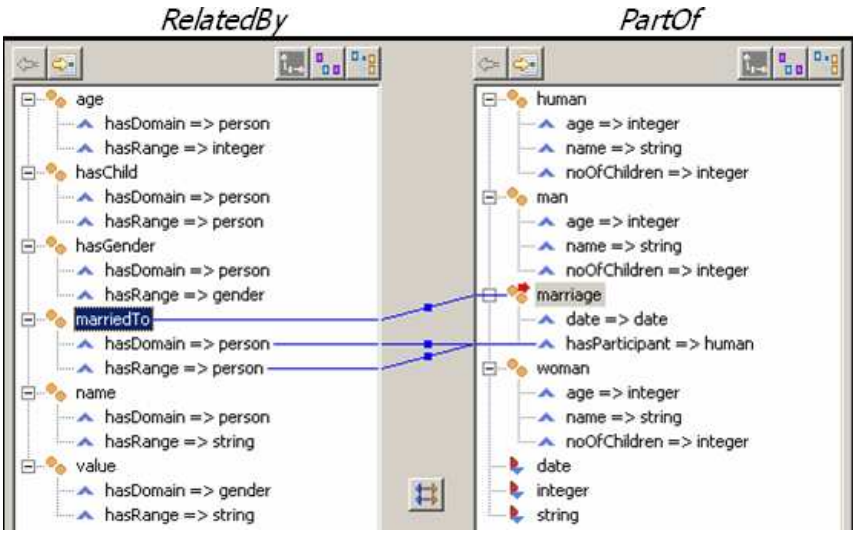

Figure 3.2: *RelatedBy* Perspective for O_1 and *PartOf* Perspective for O_2

It is interesting to note that in the above examples only one nesting level is expanded. As it is discuss in sequel, a decomposition mechanism provides access to subsequent nesting levels while graphically only one nesting level is displayed at all time. By this the design-time phase can benefit of several advantages, like higher efficiency of the suggestion algorithms and more effective guidance through the mapping process.

3.2 Perspectives and their Elements - Formal Definitions

In this approach the ontologies are presented to the user using *perspectives*. A perspective can be seen as a vertical projection of the ontology and it will be used by the domain expert to visualize and browse the ontologies and to define mappings. It is possible to define several perspectives on an ontology as introduced in Section 3.1 and formalized in Section 4, all of them characterized by a set of common elements.

Table 3.1: Types of items for the perspectives in Listing 3.1

	PartOf	*InstanceOf*	*RelatedBy*
ci	person, gender	person	name, hasGender, marriedTo, value
pi	string, integer	string, integer, gender	person, gender, integer, string
di	name, age, hasGender, hasChild, marriedTo	hasGender	hasDomain, hasRange
si	string, integer, gender, person	male, female	person, gender, integer, string

There have been four types of such elements (items) identified: *compound*, *primitive*, *description* and *successor*. The following unary relations are used to denote each of them:

$ci(x)$ - where x is a compound item,

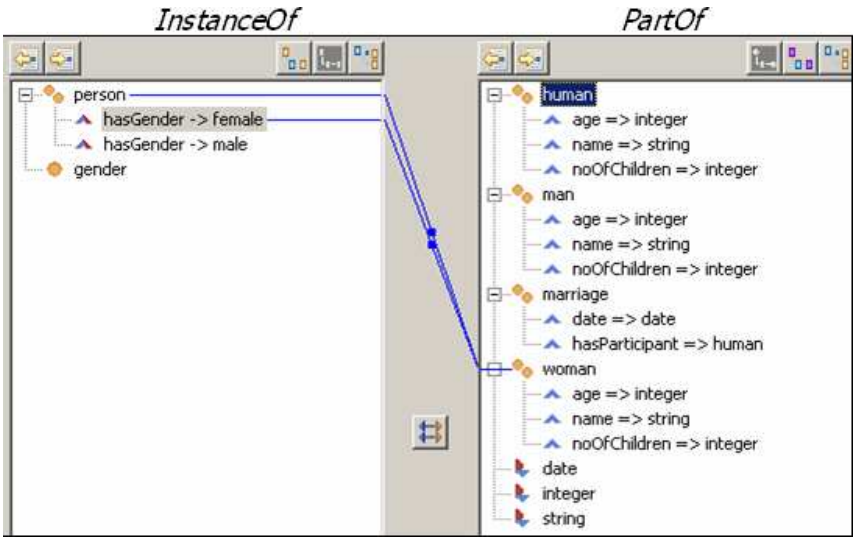

Figure 3.3: *InstanceOf* Perspective for O_1, and *PartOf* Perspective for O_2

$pi(x)$ - where x is a primitive item,

$di(x)$ - where x is a description item and

$si(x)$ - where x is a successor item.

Both primitive and compound items represent first-class citizens of a perspective while description and successor items link the compound and the primitive items in a graph-based structure. In addition we define a set of general relationships between these items that hold for all perspectives:

- Each compound item is described by at least one description item:

$$\forall x.(ci(x) <=> \exists y.(di(y) \wedge describes(y, x))) \tag{3.1}$$

  where *describes* is a binary relation that holds between a compound item and one of its description items. The participants in this relation are always a compound item and a description item:

$$\forall x.\forall y.(describes(x, y) => ci(y) \wedge di(x)) \tag{3.2}$$

- Each description item points to at least one successor item:

$$\forall x.(di(x) <=> \exists y.(si(y) \wedge successor(y, x))) \tag{3.3}$$

  where *successor* is a binary relation that holds between a description item and one of its successor items. The participants in this relation are always a description item and a successor item:

$$\forall x.\forall y.(successor(x, y) => di(y) \wedge si(x)) \tag{3.4}$$

- The successor items are either primitive or compound items:

$$\forall x.(si(x) => pi(x) \vee ci(x))) \tag{3.5}$$

- The compound, primitive and description items are mutually exclusive for the same perspective:

$$\forall x.(\neg((ci(x) \wedge pi(x)) \vee (ci(x) \wedge di(x)) \vee (di(x) \wedge pi(x)))) \tag{3.6}$$

To summarize, a description item can be seen as a link that points to another item which from a certain point of view describes the original item. As such, for the concept *person* a description item is *hasGender* which is a link to the definition of the concept *gender* (where *gender* is a successor item). When defining a perspective, one can choose as description items a class of ontology entities (e.g. the attributes), a general ontology relation (e.g. the *is-a* relationship) or even a custom defined relation (e.g. *hasChild*). The only requirement is to be able to define for the chosen description item the *describes* and *successor* binary relationships.

This is a set of minimal descriptions for our model, but by inference other useful consequences can be inferred. For example, note that Formula 3.1 and Formula 3.6 imply that primitive items have no description items. Table 3.2 shows examples of relationships for the perspectives defined on the ontology in Listing 3.1.

Table 3.2: Relations between items in the perspectives depicted in Figures 3.1,3.2 and 3.3

	successor(x, y)	*describes*(x, y)
PartOf	*successor*(string, name) *successor*(integer, age) *successor*(person, hasChild) *successor*(gender, hasGender) *successor*(person, marriedTo)	*describes*(name, person) *describes*(age, person) *describes*(hasChild, person) *describes*(hasGender, person) *describes*(marriedTo, person)
InstanceOf	*successor*(gender, (hasGender, male)) *successor*(gender, (hasGender, female)	*describes*((hasGender, male), person) *describes*((hasGender, female), person)
RelatedBy	*successor*(person, hasDomain) *successor*(string, hasRange) *successor*(gender, hasRange) *successor*(person, hasRange) *successor*(gender, hasDomain)	*describes*(hasDomain, name) *describes*(hasRange, name) *describes*(hasRange, hasGender) *describes*(hasDomain, hasGender) *describes*(hasDomain, marriedTo) *describes*(hasRange, marriedTo) *describes*(hasDomain, value) *describes*(hasRange, value)

Please note that conforming to the model, in *InstanceOf* a description item is a tuple formed from the attribute and the value this attribute takes. Additionally, the successor is the type of the attribute's value. As described in Section 4.3, this allow generally defined techniques (e.g. decomposition) to be applied to this perspective as well. The graphical representation of *InstanceOf* perspective in Figure 3.3 is just a way of displaying its elements. However, behind it, all the formulas defined in this model hold.

As a consequence we can define a perspective as being a set $\phi = \{x_1, x_2, ..., x_n\}$ for which we have:

$$\forall x.(member^3(x, \phi) => pi(x) \vee ci(x) \vee di(x)) \tag{3.7}$$

In addition, for any perspective ϕ the following formulas hold:

$$\forall x.\forall y.(describes(y, x) => (member(x, \phi) <=> member(y, \phi))) \tag{3.8a}$$
$$\forall x.\forall y.(successor(y, x) => (member(x, \phi) <=> member(y, \phi))) \tag{3.8b}$$

Formulas 3.8a and 3.2 state that the description of a compound item appears in the perspective *iff* the compound item appears in the perspective as well. Similarly (based on Formula 3.8b and Formula 3.4), a successor of a description item appears in a perspective *iff* the description item appears in the perspective too.

3.3 Contexts

Not all of the information modeled in the ontology is useful in all stages of the mapping process. The previous section shows that a perspective represents only a subset of an ontology, but we can go further and define the notion of *context*. A context is a subset of a perspective that contains only those ontological entities and their descriptions from that perspective, which are relevant to a concrete operation. We can say that γ_ϕ is a context of the perspective ϕ if:

$$\forall x.(member(x, \gamma_\phi) => member(x, \phi)) \tag{3.9}$$

For a context from Formulas 3.8a and 3.8b only 3.8a holds, such that:

$$\forall x.\forall y.\forall \gamma_\phi.(describes(y, x) => (member(x, \gamma_\phi) <=> member(y, \gamma_\phi))) \tag{3.10}$$

This means that if compound item is part of a context all its description items are included as well, On the other hand if a description item is included in a context its successors are not necessarily part of the same context (but they can be reached by decomposition). As a consequence, we can say that all perspectives are contexts but not all contexts are perspectives.

A notion tightly related with contexts is the process of *decomposition*. Decomposition is the process of exploring the descriptions and the successors' descriptions of a given item and organizing this information in a new context. A context can be created from

[3] *member* is a relationship expressing the membership of an element to a list. It could be defined like this:
$\forall x.\forall l.(member(x, x.l))$
$\forall x.\forall y.\forall l.(member(x, l) => member(x, y.l))$

another context (this operation is called *context update*) by applying decomposition on an item from a perspective or a context. Decomposition allows navigating between contexts and links consecutive nested levels; the way the contexts are navigated when creating mappings influences the creation types of mappings that are created.

Let $decomposition(x, \gamma_\phi)$ be a binary function which has as value a new context obtained by decomposing x in respect with the context γ_ϕ. If x is a primitive item the decomposition does not change the current context. That is, any item part of the current context before the decomposition over a primitive item is part of the current context after the decomposition as well (and other way around):

$$\forall x.\forall y.\forall \gamma_\phi.(member(x, \gamma_\phi) \wedge pi(x) => \\ (member(y, decomposition(x, \gamma_\phi)) <=> member(y, \gamma_\phi))) \quad (3.11)$$

If decomposition is applied on a compound item, the new context contains that particular compound item and all its description items:

$$\forall x.\forall y.\forall \gamma_\phi.(member(x, \gamma_\phi) \wedge ci(x) => \\ (member(y, decomposition(x, \gamma_\phi)) <=> y = x \vee describes(y, x))) \quad (3.12)$$

Decomposition can be applied on description items as well, case in which it is transposed to the corresponding successor items. The context remains unchanged if either the successor item is a primitive item or it is already contained in the current context:

$$\forall x.\forall y.\forall z.\forall \gamma_\phi.(member(x, \gamma_\phi) \wedge di(x) \wedge successor(z, x) \wedge \\ (pi(z) \vee member(z, \gamma_\phi)) => \\ (member(y, decomposition(x, \gamma_\phi)) <=> member(y, \gamma_\phi))) \quad (3.13)$$

If the successor item is a compound item and it is not already included in the current context, the decomposition is applied as for the compound items in the perspective:

$$\forall x.\forall y.\forall z.\forall \gamma_\phi.(member(x, \gamma_\phi) \wedge di(x) \wedge successor(z, x) \wedge \\ ci(z) \wedge \neg(member(z, \gamma_\phi)) => \\ (member(y, decomposition(x, \gamma_\phi)) <=> member(y, decomposition(z, \phi)))) \quad (3.14)$$

It is important to note that when the successor z is a compound item, the new context is given by the decomposition of ϕ over z since z is not part of the current context γ_ϕ (as mentioned above Formula 3.8b does not apply to contexts).

To summarize, there are several cases when decomposition cannot be applied (i.e. the context remains unchanged): formula 3.11 specifies that the decomposition of a primitive concept does not update the current context. Additionally, decomposition applied on a description item that has a primitive successor (formula 3.13) leaves the current context unchanged as well. The same formula also does not allow the decomposition of those description items that have as successor a compound item already contained by the current context (recursive structures).

Table 3.3 presents some examples of decompositions and context updates: each column shows how the context changes by decomposing any of the marked items in the top row. The decomposition can be applied simultaneously on multiple items, and the result of decomposing each item is contributing to the new context. Note as described over for column 1 no change occurs as all of the marked items cannot trigger decomposition conforming to formulae 3.11 and 3.13.

Table 3.3: Decomposition and context updates

Original Context		
• string • integer • person ⊢ name → string ⊢ age → integer ⊢ hasGender → gender ⊢ hasChild → person └ marriedTo → person • gender └ value → string	• string • integer • person ⊢ name → string ⊢ age → integer ⊢ hasGender → gender ⊢ hasChild → person └ marriedTo → person • gender └ value → string	• string • integer • person ⊢ name → string ⊢ age → integer ⊢ hasGender → gender ⊢ hasChild → person └ marriedTo → person • gender └ value → string
New Context		
• string • integer • person ⊢ name → string ⊢ age → integer ⊢ hasGender → gender ⊢ hasChild → person └ marriedTo → person • gender └ value → string	• person ⊢ name → string ⊢ age → integer ⊢ hasGender → gender ⊢ hasChild → person └ marriedTo → person	• gender └ value → string

3.4 Mappings

To create mappings between ontologies, a source and target perspective is used to represent the source and target ontologies. We refer to this approach as interactive mapping creation. The mapping creation process relies upon a domain expert, who has the role of choosing an item from the source perspective and one from the target perspective (or contexts) and explicitly marking them as mapped items. We call this action *map* and using this the domain expert states that there is a semantic relationship between the mapped items. Choosing the right pair of items to be mapped is not necessarily a manual task: a semi-automatic solution can offer suggestions that are eventually validated by a human user [MC05]. We refer to this approach as interactive mapping creation.

We define a *mapping context* as a quadruple $Mc =< \phi_S, \gamma_{\phi_S}, \phi_T, \gamma_{\phi_T} >$ where ϕ_S and ϕ_T are the source and target perspectives associated to the source and target ontologies. γ_{ϕ_S} and γ_{ϕ_T} are the current contexts derived out of the two perspectives ϕ_S and ϕ_T.

Initially, $\gamma_{\phi_S} \equiv \phi_S$ and $\gamma_{\phi_T} \equiv \phi_T$.

We also define $map_{Mc}(x, y)$ the action of marking the item x from ontology S and item y from ontology T as being semantically related with respect to the mapping context Mc. Thus, we have the following axiom:

$$\forall x.\forall y.\forall \phi_S.\forall \phi_T.\forall \gamma_{\phi_S}.\forall \gamma_{\phi_T} map_{Mc}(x,y) \wedge (Mc =< \phi_S, \gamma_{\phi_S}, \phi_T, \gamma_{\phi_T} >) \wedge ((ci(x) \vee pi(x)) \wedge (ci(y) \vee pi(y))) \vee (di(x) \wedge di(y)) \; => member(x, \gamma_{\phi_S}) \wedge member(y, \gamma_{\phi_T}) \quad (3.15)$$

Formula 3.15 defines the allowed types of mapping. Thus we can have mappings between primitive and/or compound items and between description items. As described in [MC05] the set of the allowed mappings can be extended or restricted by a particular, concrete perspective.

Each time a *map* action occurs the mapping context is updated; we denote the updates using: $Mc \twoheadrightarrow Mc^{'}$ meaning that at least one element of the quadruple defining Mc might have changed and the new mapping context is $Mc^{'}$. The mapping context updates occur as defined in axiom 3.16:

$$\forall x.\forall y.map_{Mc}(x,y) \wedge (Mc =< \phi_S, \gamma_{\phi_S}, \phi_T, \gamma_{\phi_T} >) \; => \quad (3.16)$$
$$(Mc' =< \phi_S, decomposition(x, \gamma_{\phi_S}), \phi_T, decomposition(y, \gamma_{\phi_T}) >) \wedge Mc \twoheadrightarrow Mc'$$

There are cases when Mc and Mc' are identical; such situations occur when the source and target context remain unchanged, e.g. when creating mappings between primitive items.

Chapter 4

Grounding the Model to Ontologies

This section explores the way in which the model presented above can be applied to a real ontological model and how we can use it to define concrete perspectives that could be used to create meaningful mappings between ontologies. WSMO ontologies are used for this purpose since the tools implementing these conceptual ideas are part of Web Service Execution Environment (WSMX) which is a references implementation of WSMO; however this model can be potentially grounded to any ontology representation language. We first introduce the main aspects of WSMO ontologies and a mechanism to link these ontologies with our model and then we will present the three types of concrete perspectives we identified as being useful in the mapping process. It is important to note that the concrete perspectives are not defined inside of the model. This is because a perspective's definition depends on the ontology language used, while the model itself is an abstract model, ontology-language neutral.

The first sub-section gives a short overview of WSML, followed by the definition of the three perspectives we have identified as useful for our scenario.

4.1 Web Service Modeling Language (WSML)

The Web Service Modeling Ontology (WSMO) defines the main aspects related to Semantic Web Services: Ontologies, Web Services, Goals and Mediators [FPD+05]. From these four, only Ontologies are of interest for this work. We will focus on concepts, attributes and instances in this deliverable; however we intend to address other ontological elements in the future. WSMO ontologies are expressed using the Web Service Modeling Language WSML [dBLK+05, HLF05] which is a language for the specification of different aspects of SWS; it takes into account all aspects identified by WSMO.

WSML comprises different formalisms, most notably Description Logics and Logic Programming, in order to investigate their applicability in the context of Ontologies and Web services. Three main areas can benefit from the use of formal methods in service

descriptions: Ontology description, Declarative functional description of Goals and Web services, and Description of dynamics. WSML defines a syntax and semantics for ontology descriptions. The underlying logic formalisms a re used to give a formal meaning to ontology descriptions in WSML, resulting in different variants of the language, which differ in logical expressiveness and in the underlying language paradigms, and allow users to make the trade off between provided expressiveness and the implied complexity for ontology modeling on a per-application basis. We briefly differentiate these variants:

WSML-Core is based on by the intersection of the Description Logic $\mathcal{SHIQ}$(**D**) [HST99] and Horn Logic, based on Description Logic programs. It has the least expressive power of all the WSML variants. The main features of the language are concepts, attributes, binary relations and instances, as well as concept and relation hierarchies and support for datatypes.

WSML-DL captures the Description Logic $\mathcal{SHIQ}$(**D**), which is a major part of the (DL species of) Web Ontology Language OWL [DE04].

WSML-Flight is an extension of WSML-Core which provides a powerful rule language. It adds features such as meta-modeling, constraints and nonmonotonic negation. WSML-Flight is based on a logic programming variant of F-Logic [KLW95] and is semantically equivalent to Datalog with inequality and (locally) stratified negation. WSML-Flight is a direct syntactic extension of WSML-Core and it is a semantic extension in the sense that the WSML-Core subset of WSML-Flight agrees with WSML-Core on ground entailments).

WSML-Rule extends WSML-Flight with further features from Logic Programming, namely the use of function symbols, unsafe rules, and unstratified negation under the Well-Founded semantics.

WSML-Full unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule.

Several features make WSML unique from other language proposals for the SW and SWS, amongst them the most important are:

- *One syntactic framework for a set of layered languages.* No single language paradigm will be sufficient for all SWS use cases, thus different language variants of different expressiveness are needed.
- *A normative, human readable syntax* which allows for easier adoption of the language by the users.
- *Separation of conceptual and logical modeling.* The conceptual syntax allows for easy modeling of Ontologies, Web Services, Goals, and Mediators, while the logical expression syntax allows expert users to refine definitions on the conceptual syntax.

- *Semantics based on well known formalisms.* WSML captures well known logical formalisms in a unifying syntactical framework, while maintaining the established computational properties of the original formalisms.
- *A frame-based syntax.* It allows the user to work directly on the level of concepts, attributes, instances and attribute values, instead of at the level of predicates.

4.2 PartOf Perspective

The *PartOf* perspective is the most common perspective that can be used to display an ontology, focusing on hierarchies of concepts, with their attributes and attributes' types. To link this perspective with our model we define the unary relations $ci_{PartOf}(x)$, $pi_{PartOf}(x)$ and $di_{PartOf}(x)$ such that:

$$ci(x) \textbf{ iff } ci_{PartOf}(x) \tag{4.1}$$

$$pi(x) \textbf{ iff } pi_{PartOf}(x) \tag{4.2}$$

$$di(x) \textbf{ iff } di_{PartOf}(x) \tag{4.3}$$

Formula 4.1 basically means that an ontology entity x has the role of a compound item in the formal model if and only if in the WSML ontology the $ci_{PartOf}(x)$ holds. Formulas 4.2 and 4.3 are similar for primitive items and descriptions items relative to $pi_{PartOf(x)}$ and $di_{PartOf}(x)$.

$ci_{PartOf}(x)$, $pi_{PartOf}(x)$ and $di_{PartOf}(x)$ have to be defined in the logical language used to represent the ontologies to be aligned. WSML[1] definitions for these relations are presented in Formulas 4.13, 4.5 and 4.6. In the *PartOf* perspective the role of compound items is taken by those concepts that have at least one attribute - we call them *compound concepts*. Naturally, the description items are in this case attributes, as stated in Axiom 4.5. Primitive items are data types or those concepts that have no attributes, as expressed by Axiom 4.6 where x **subconceptOf** $true$ holds iff x is a concept and *naf* stands for negation as failure. The *describes* and *successor* relations for WSML ontologies are defined by Axioms 4.7 and 4.8 and linked in Axiom 4.9. The ontology fragment presented in Table 3.1 can be visualized using the *PartOf* perspective as in Figure 3.1.

axiom ci_{PartOf} **definedBy** $ci_{PartOf}(?x)$ **equivalent**
exists $?y, ?z(?x[?y$ **ofType** $?z])$ (4.4)

axiom di_{PartOf} **definedBy** $di_{PartOf}(?y)$ **equivalent**
exists $?x, ?z(?x[?y$ **ofType** $?z])$ (4.5)

[1] In WSML $\alpha[\beta$ **ofType** $\gamma]$ is an atomic formulas called *molecule*; in here both α and γ identifies concepts while β identifies an attribute and terms beginning with '?' are used to denote variables. An example of a molecule for the ontology fragment in Table 3.1 is $person[name$ **ofType** $string]$

axiom pi_{PartOf} **definedBy** $pi_{PartOf}(?x)$ **:-**
$?x$ **subconceptOf** $true$ and **naf** $ci_{PartOf}(x)$ (4.6)

$$describes(y, x) \textbf{ iff } \textbf{exists } ?z(?x[?y \textbf{ ofType } ?z]) \quad (4.7)$$

$$successor(z, y) \textbf{ iff } \textbf{exists } ?x(?x[?y \textbf{ ofType } ?z]) \quad (4.8)$$

$$describes(y, x) \wedge successor(z, y) \textbf{ iff } ?x[?y \textbf{ ofType } ?z] \quad (4.9)$$

4.3 InstanceOf Perspective

The *InstanceOf* perspective can be used to create conditional mappings based on predefined values and instances. To link this perspective with our model we define $ci_{InstanceOf}(x)$, $pi_{InstanceOf}(x)$ and $di_{InstanceOf}(y, w)$ such that:

$$ci(x) \textbf{ iff } ci_{InstanceOf}(x) \quad (4.10)$$
$$pi(x) \textbf{ iff } pi_{InstanceOf}(x) \quad (4.11)$$
$$di(< y, w >) \textbf{ iff } di_{InstanceOf}(y, w) \quad (4.12)$$

In the same way as above, $ci_{InstanceOf}(x)$, $pi_{InstanceOf}(x)$ and $di_{InstanceOf}(x)$ are defined using WSML:

axiom $ci_{InstanceOf}$ **definedBy** $ci_{InstanceOf}(?x)$ **equivalent**
exists $?y, ?z, ?w(?x[?y$ **ofType** $?z]$ **and** (4.13)
$(?w$ **memberOf** $?z$ **or** $(ci_{InstanceOf}(?z)$ **and** $?w = \_\#)))$

In the *InstanceOf* perspective a compound item is a concept that has at least one attribute and that attribute has as type either another compound item or a concept for which there is at least one instance defined in the ontology. The description items are tuples $< y, w >$ where **y** is an attribute matching the above conditions and **w** is an instance member of *y*'s type explicitly defined in the ontology or an anonymous id representing a potential instance of the *y*'s type as defined in Axiom 4.14.

axiom $di_{InstanceOf}$ **definedBy** $di_{InstanceOf}(?y, ?w)$ **equivalent**
exists $?x, ?z(?x[?y$ **ofType** $?z]$ **and** (4.14)
$(?w$ **memberOf** $?z$ **or** $(ci_{InstanceOf}(?z)$ **and** $?w = \_\#)))$

The anonymous instances (denoted by _# in the previous axioms) can be created by a recursive algorithm that traverses the ontology graph; pseudo-code for such an algorithm is given in Axiom 4.1. The *createAnonymousInstances* takes as parameters the root of the concept tree for which the anonymous instances are to be created, the set of instances already known, and the set of concepts already processed. Initially, the instances' set would contain the instances explicitly defined in the ontology while the processed concepts would be the empty set. Anonymous instances are created for the root of the concept tree and every successor concept for which an instance exists.

Listing 4.1: An algorithm for creating the anonymous instances

```
void createAnonymousInstances(Concept C,
                              Collection instances, Collection processedConcepts){
    if not (processedConcepts.contains(C)) then
        instances.addAll(getInstancesOf(C));
        processedConcepts.add(C);
        Collection newConceptsSet;
        for each A in C.getAttributes() do
            for each R in A.getRanges() do
                if containsInstancesOf(instances, R) then
                    instances.add(new AnonymousInstance(C));
                endIf
                if not (processedConcepts.contains(R)) then
                    newConceptsSet.add(R);
                endIf
            endFor
        endFor
        for each C in newConceptsSet do
            createAnonymousInstances(C, instances, precessedConcepts);
        endFor
    endIf
}
```

An interesting aspect is that the initial instances set can be extended to contain also values of data types. For data types with discrete domains it is straight forward, e.g. for *boolean* the *false* and *true* values could be considered. For the rest of the data types (with continue domain) a default value ca be initially included and allowed later, in the tool, to be edited by the domain expert. In such cases the method `containsInstancesOf(`$instances^+$`, R)` will return *true* as well if `R` is a data type.

The primitive items for the *InstanceOf* perspective are defined in WSML as follows:

$$\textbf{axiom } pi_{InstanceOf} \textbf{ definedBy } pi_{InstanceOf}(?x) \textbf{ :-} \\ ?x \textbf{ subconceptOf } true \text{ and } \textbf{naf } ci_{InstanceOf}(?x) \tag{4.15}$$

Also the *describes* and *successor* relations can be linked with the WSML ontologies in a similar manner:

$$describes(< y, w >, x) \textbf{ iff} \\ \textbf{exists } ?z(?x[?y \textbf{ ofType } ?z] \textbf{ and } ?w \textbf{ memberOf } ?z) \tag{4.16}$$

$$successor(z, < y, w >) \textbf{ iff} \\ \textbf{exists } ?x(?x[?y \textbf{ ofType } ?z] \textbf{ and } ?w \textbf{ memberOf } ?z) \tag{4.17}$$

$$describes(< y, w >, x) \wedge successor(z, < y, w >) \textbf{ iff} \quad (?x[?y \textbf{ ofType } ?z] \textbf{ and } ?w \textbf{ memberOf } ?z) \tag{4.18}$$

The reader can find the fragment of ontology presented in Table 3.1 visualized using the *InstanceOf* perspective as in Figure 3.3.

4.4 RelatedBy Perspective

The *RelatedBy* perspective focus on the attributes of the ontology, and describes them in respect with their domain and type.

$$ci(x) \textbf{ iff } ci_{RelatedBy}(x) \tag{4.19a}$$
$$pi(x) \textbf{ iff } pi_{RelatedBy}(x) \tag{4.19b}$$
$$di(x) \textbf{ iff } di_{RelatedBy}(x) \tag{4.19c}$$

The WSML definitions for $ci_{RelatedBy}(x)$, $pi_{RelatedBy}(x)$ and $di_{RelatedBy}(x)$ are described in Axioms 4.20, 4.21 and 4.22.

$$\textbf{axiom } ci_{RelatedBy} \textbf{ definedBy } ci_{RelatedBy}(?y) \textbf{ equivalent exists } ?x, ?z(?x[?y \textbf{ ofType } ?z]) \tag{4.20}$$

In the *RelatedBy* perspective the attributes are considered compound items and all of them have only two description items: *hasDomain* and *hasRange*.

$$\textbf{axiom } di_{RelatedBy} \textbf{ definedBy } di_{RelatedBy}(?y) \textbf{ equivalent } ?y = hasDomain \textbf{ or } ?y = hasRange) \tag{4.21}$$

As primitive items we consider in this perspectives the concepts and the data type (i.e. *hasDomain* and *hasRange* have no mining for them so they have no descriptions):

$$\textbf{axiom } pi_{RelatedBy} \textbf{ definedBy } pi_{RelatedBy}(?x) \textbf{ :-} \quad ?x \textbf{ subconceptOf } true \textbf{ or } ?x \textbf{ memberOf } \_datatype \tag{4.22}$$

The construct "$?x$ **memberOf** $\_datatype$" checks if x is a data type: in WSML each data type is a member of the meta-concept *_datatype*.

Finally we link the *describe* and *successor* relations with the WSML ontologies:

$$describe(hasDomain, y) \wedge \quad succesor(x, hasDomain) \textbf{ iff } \textbf{exists } ?z(?x[?y \textbf{ ofType } ?z]) \tag{4.23a}$$

$$describe(hasRange, y) \wedge \quad succesor(z, hasRange) \textbf{ iff}$$
$$\textbf{exists } ?x(?x[?y \textbf{ ofType } ?z] \tag{4.23b}$$

It is important to note that for these perspective no decomposition can be applied on the description items since they always point to primitive items.

The fragment of ontology presented in Table 3.2 can be visualized using the *RelatedBy* perspective as in Figure 3.2.

Chapter 5

Linking the Model to a Mapping Language

In this section we specify the allowed mappings for each of the perspectives described in Chapter 4. We start from the following premise $map_{Mc}(x_S, y_T) \wedge Mc =< \phi_S, \gamma_{\phi_S}, \phi_T, \delta_{\phi_T} >$ which means that the elements x_S and y_T from the source and target ontology, respectively, are to be mapped in the mapping context Mc. Sections 5.2 to 5.4 we will discuss the situations that can occur for a pair of perspectives (for brevity we address only those cases when the source and target perspectives are of the same type). The types of mapping that can be created will be analyzed with respect to the Alignment Format proposed in [ESS06], briefly described in Section 5.1.

5.1 The Alignment Format

We chose to express the mappings in the alignment format proposed in [ESS06] because it does not commit to any specific ontology representation language. Mappings expressed in this way, just state that a semantic relationship exists between the mapped entities; the actual semantics and interpretation is associated in a second step, called grounding. Mappings in the alignment format have the following form:

⟨map⟩ ::= `<map>`⟨cell⟩`*</map>`
⟨cell⟩ ::= `<Cell id>`⟨entity1⟩⟨entity2⟩⟨measure⟩⟨relation⟩`</Cell>`
⟨id⟩ ::= `id=`⟨uriref⟩
⟨entity1⟩ ::= `<entity1>`⟨uriref⟩`</entity1>`
| `<entity1>`⟨expression⟩`</entity1>`
⟨entity2⟩ ::= `<entity2>`⟨uriref⟩`</entity2>`
| `<entity2>`⟨expression⟩`</entity2>`

An alignment is composed of a header (not described here) and a set of a set of cells,

each representing a mapping between schema entities.

The ⟨id⟩ is a unique identifier of the mapping. Each cell has two entities which represent objects of the two ontologies to be related. Four types of entities are distinguished: Classes, Attributes, Relations and Instances. The URI representing an entity can be directly used in the case it refers to an atomic element, such as a particular class or attribute. An entity can also be a complex expression containing multiple elements as well as conditions.

The ⟨measure⟩ field represents the confidence given to the correspondence. A measure is a real number taking values in [0..1]. This field is particularly useful when the alignment is generated as the output of an algorithm, but in our approach since the mappings are validated by the domain expert its value will be always set to 1[1]. The type of entities being part of the correspondence and the relation standing between them (equivalence or subsumption) are represented by the ⟨relation⟩.

The following grammar stands for Class expressions.

```
⟨classexpr⟩   ::= <omwg:Class rdf:about="⟨uriref⟩">⟨classcond⟩*</omwg:Class>
                | <omwg:Class>⟨classconst⟩⟨classcond⟩*</omwg:Class>
⟨classconst⟩  ::= <and rdf:parseType="Collection">⟨classexpr⟩⟨classexpr⟩+</and>
                | <or rdf:parseType="Collection">⟨classexpr⟩⟨classexpr⟩+</or>
                | <not>⟨classexpr⟩</not>
⟨classcond⟩   ::= <attributeValueCondition>⟨restriction⟩</attributeValueCondition>
                | <attributeTypeCondition>⟨restriction⟩</attributeTypeCondition>
                | <attributeOccurenceCondition>
                  ⟨restriction⟩</attributeOccurenceCondition>
```

Class expression can be built using operators ⟨and⟩ for the intersection, ⟨or⟩ for the union and ⟨not⟩ for negation. Conditions specifying in which cases the mapping is effective are specified using restrictions on the constituted expression.

```
⟨restriction⟩ ::= <Restriction>
                    <onProperty>⟨path⟩</onProperty>
                    <comparator rdf:resource="⟨uriref⟩"/>
                    <value>⟨pov⟩</value>
                  </Restriction>
      ⟨pov⟩   ::= ⟨path⟩ | ⟨value⟩ | ⟨uriref⟩
    ⟨value⟩   ::= ⟨simplevalue⟩
                | <Apply rdf:resource="⟨uriref⟩"
                      rdf:parseType="Collection">⟨pov⟩*</Apply>
```

[1]The measure can be seen in here as a measure of confidence in respect with a reference alignment. Being validated by a domain expert this is considered to be the reference alignment, 100% correct from the domain expert point of view.

In different words, the measure does not necessarily show the confidence of the domain expert in the quality of the mapping but its confidence in the fact that the mapping formally expresses its intention.

A restriction targets a specific set of instances answering given certain constraints on their properties.

```
⟨attexpr⟩    ::=  <omwg:Attribute rdf:about="⟨uriref⟩">⟨attcond⟩*</omwg:Attribute>
              |   <omwg:Attribute>⟨attconst⟩ ⟨attcond⟩*⟨atttranf⟩</omwg:Attribute>
⟨attconst⟩   ::=  <and rdf:parseType="Collection">⟨attexpr⟩⟨attexpr⟩+</and>
              |   <or rdf:parseType="Collection">⟨attexpr⟩⟨attexpr⟩+</or>
              |   <not>⟨attexpr⟩</not>
⟨attcond⟩    ::=  <valueCondition><value>⟨pov⟩</value>
                  <comparator rdf:resource="⟨uriref⟩"/></valueCondition>
              |   <domainRestriction>⟨classExpr⟩</domainRestriction>
              |   <typeCondition>⟨pov⟩</typeCondition>
⟨atttransf⟩  ::=  <transf rdf:resource="⟨uriref⟩">⟨pov⟩*</transf>
              |   <service rdf:resource="⟨uriref⟩" id="⟨uriref⟩">
                  ⟨pov⟩*</service>
```

In the same way as class, complex attribute expressions are constructed using operators and restrictions. Transformations can also be specified in order to transform instance values in the case, for example, of a currency exchange. Attribute expressions can be restricted on their value, domain, or range.

Instance mappings always stand between atomic entities.

```
⟨instexpr⟩   ::=  <omwg:Instance rdf:about"⟨uriref⟩"/>
```

The alignment format has not only the role of linking the efforts in Knowledge Web Workpackage 2.2 (Heterogeneity) and Workpackage 2.4 (Semantic Web Services), but it also aims to become a common mapping representation language for mediation efforts in various other European projects (e.g. DIP[2], SemanticGov[3] and SEEMP[4]).

In the next sections we illustrate how such mapping language statements are generated at design time through the use of different pairs of perspectives. Although generated mappings are frequently bidirectional (i.e. *equivalent*), for brevity in this work all the mappings are considered to be unidirectional (i.e. *subsumption* in the alignment format). There are four categories of (unidirectional) mappings that can be generated with the combination of perspectives proposed below:

• $ClassMapping$: The two entities being mapped are both classes.

• $AttributeMapping$: The two entities being mapped are both attributes.

[2]"Data, Information, and Process Integration with Semantic Web Services";more details available at http://dip.semanticweb.org/

[3]"Providing Integrated Public Services to Citizens at the National and Pan-European level with the use of Emerging Semantic Web Technologies"; more details available at http://www.semantic-gov.org/

[4]"Single European Employment Market Place"; more details available at http://www.seemp.org/

- $ClassAttributeMapping$: The first entity is a class while the second entity participating in the mapping is an attribute.

- $AttributeClassMapping$: The first entity is an attribute while the second entity participating in the mapping is a class.

5.2 *PartOf* to *PartOf* Mappings

When using the *PartOf* perspective to create mappings for both the source and target ontologies we have the following allowed cases (derived from Formula 3.15):

- $pi_{PartOf}(x_S) \wedge pi_{PartOf}(x_T)$ In this case, the mapping will generate a *ClassMapping* statement in the mapping language and leaves the mapping context unchanged (axioms 3.11 and 3.16).

- $ci_{PartOf}(x_S) \wedge ci_{PartOf}(x_T)$ Generates a *ClassMapping* statement and updates the context for the source and target perspectives (axioms 3.12 and 3.16).

- $di_{PartOf}(x_S) \wedge di_{PartOf}(x_T)$ In this case $successor(y_S, x_S) \wedge successor(y_T, x_T)$ holds and there can be distinguished the following situations:

  - $pi_{PartOf}(y_S) \wedge pi_{PartOf}(y_T)$ An *AttributeMapping* is generated between x_S and x_T followed by a *ClassMapping* between y_S and y_T. Conforming to the axioms 3.13 and 3.16, the mapping context remains unchanged.

  - $ci_{PartOf}(y_S) \wedge ci_{PartOf}(y_T)$ An *AttributeMapping* is generated having as participants x_S and x_T. The mapping context is updated conform to the axioms 3.13, 3.14 and 3.16.

  - $pi_{PartOf}(y_S) \wedge ci_{PartOf}(y_T)$ Generates a *ClassAttributeMapping* between z_S and x_T, where $describes(x_S, z_S)$. The new mapping context keeps the source context unchanged while decomposing the target context over y_T.

  - $ci_{PartOf}(y_S) \wedge pi_{PartOf}(y_T)$ This case is symmetric with the one presented above and it generates a *AttributeClassMapping* between x_S and the z_T, where we have $describes(x_T, z_T)$.

- $ci_{PartOf}(x_S) \wedge pi_{PartOf}(x_T)$ It is not allowed for this combination of perspectives. To take an example, such a case would involve a mapping between $ci_{PartOf}(person)$ and $pi_{PartOf}(string)$ where $describes(hasName, person) \wedge successor(string, hasName)$, which does not have any semantic meaning. A correct solution would be a mapping between $ci_{PartOf}(person)$ and $ci_{PartOf}(u_T)$ such as $\exists v_T.(describes(v_T, u_T) \wedge successor(string, v_T)$.

- $pi_{PartOf}(x_S) \wedge ci_{PartOf}(x_T)$ The same explanation applies as above.

5.3 *InstanceOf* to *InstanceOf* Mappings

When using the *InstanceOf* perspectives we can create similar mappings to those created with the *PartOf* perspectives, the difference being that conditions are added to the mappings, and by this, the mappings hold only if the conditions are fulfilled. The mappings between two primitive items or between two compound items in the *InstanceOf* perspective are identical with the ones from the *PartOf* perspective. For the remaining cases we have:

- $di_{InstanceOf}(x_S, w_S) \wedge di_{InstanceOf}(x_T, w_T)$ In this case, we have $successor(y_S, < x_S, w_S >) \wedge successor(y_T, < x_T, w_T >)$ and we can distinguish the following situations:
  - $pi_{InstanceOf}(y_S) \wedge pi_{InstanceOf}(y_T)$ An *AttributeMapping* is generated between x_S and x_T conditioned by two *valueCondition*s imposing the presence of w_S and w_T in the mediated data. Conforming to the axioms 3.13 and 3.16 the mapping context remains unchanged.
  - $ci_{InstanceOf}(y_S) \wedge ci_{InstanceOf}(y_T)$ An *AttributeMapping* is generated having as participants x_S and x_T conditioned by two *typeConditions*. The mapping context is updated conforming to the axioms 3.13, 3.14 and 3.16.
  - $pi_{InstanceOf}(y_S) \wedge ci_{InstanceOf}(y_T)$ This case generates a *ClassAttributeMapping* between z_S and the x_T, where $describes(x_S, z_S)$. A *typeCondition* is added for x_T attribute. The new mapping context keeps the source context unchanged while decomposing the target context over y_T.
  - $ci_{InstanceOf}(y_S) \wedge pi_{InstanceOf}(y_T)$ This case is symmetric with the one presented above and it generates a *AttributeClassMapping* between x_S and the z_T where $describes(x_T, z_T)$. A *typeCondition* is added for x_S.
- $di_{InstanceOf}(x_S, w_S) \wedge pi_{InstanceOf}(x_T)$ *InstanceOf* extends the set of allowed mappings as defined in Formula 3.15. For z_S such that $describes(< x_S, w_S >, z_S)$, a *ClassMapping* between z_S and x_T is generated, conditioned by an *valueCondition* on the attribute x_S and value w_S.
- $pi_{InstanceOf}(x_S) \wedge di_{InstanceOf}(x_T, w_T)$ Similar with the above case.
- $ci_{InstanceOf}(x_S) \wedge pi_{InstanceOf}(x_T)$ It is not directly allowed for this combination of perspectives, but the intended mapping can be created as described by previous case.
- $pi_{InstanceOf}(x_S) \wedge ci_{InstanceOf}(x_T)$ The same explanation applies as above.

5.4 *RelatedBy* to *RelatedBy* Mappings

In the *RelatedBy* perspective attributes are seen as root elements, having only two descriptions: their domain and their type. We identify the following cases:

- $pi_{RelatedBy}(x_S) \wedge pi_{RelatedBy}(x_T)$ The *RelatedBy* perspective it is not suited to create mappings between concepts, such as the mappings involving primitive items trigger a change of the perspective from which the primitive item is part of to the *PartOf* perspective.

  In this case, both source and target perspectives are changed into the *PartOf* perspective and the current contexts will be obtained by decomposing the perspectives over x_S and x_T.

- $ci_{RelatedBy}(x_S) \wedge ci_{RelatedBy}(x_T)$ The mapping will generate an *attributeMapping* statement in the mapping language having as participants x_S and x_T.

- $di_{RelatedBy}(x_S) \wedge di_{RelatedBy}(x_T)$ When trying to create such mappings the source and the target perspectives will be changed from *RelatedBy* to *PartOf* perspectives and the current contexts will be obtained by decomposing the perspectives over z_S and z_T, where $successor(z_S, x_S) \wedge successor(z_T, x_T)$

5.5 Mapping Examples

Table 5.1 shows examples of mappings in the Alignment Format (using the human readable syntax) and how these mappings look like when grounded to WSML when mapping the person concept in the the source ontology with human (and man) in the target ontology. When evaluated, the WSML mapping rules will generate instances of *man* if the *gender* condition is met, or of *human* otherwise. The construct $mediated(X, C)$ represents the identifier of the newly created target instance, where X is the source instance that is transformed, and C is the target concept we map to.

Table 5.2 depicts the same mappings represented in the RDF syntax of the alignment format.

5.6 Grounding the Alignment Format

As presented in Chapter 4, in order to use our model (i.e. an abstract model), it had to be ”grounded” to concrete ontologies. In the same fashion, in order to use the alignment format in real mediation scenarios, it has to be grounded to a concrete ontology representation language: the same language the ontologies to be mapped are expressed in.

Table 5.1: Mappings example

Alignment Format (human readable format)	**Mapping Rules in WSML**

Mapping(o1#persono2#man
  ClassMapping(*one-way*
  person man)
  attributeValueCondition(
  [(person)hasGender => gender] male))
 	
Mapping(o1#ageo2#age
  AttributeMapping(one-way
  [(person)age=>integer]
  [(human)age=>integer]))
 	
Mapping(o1#nameo2#name
  AttributeMapping(one-way
  [(person)name => string]
  [(human)name => string]))
 	
Mapping(o1#hasGendero2#man
  AttributeClassMapping(one-way
  [(person)hasGender => gender] man))
  valueCondition(
  [(person)hasGender => gender] male)
	axiom mapping001 **definedBy**
	  mediated(X_1, o2#man) **memberOf** o2#man:-
	  X_1 **memberOf** o1#person **and**
	  X_1[o1#hasGender **hasValue** o1#male].
	axiom mapping001 **definedBy**
	  mediated(X_2, o2#human) **memberOf** o2#human:-
	  X_2 **memberOf** o1#person.
	axiom mapping005 **definedBy**
	  mediated(X_5, o2#human)[o2#age **hasValue** Y_6]:-
	  X_5[o1#age **hasValue** Y_6]:o1#person.
	axiom mapping006 **definedBy**
	  mediated(X_7, o2#human)[o2#name **hasValue** Y_8]:-
	  X_7[o1#name **hasValue** Y_8]:o1#person.
	axiom mapping007 **definedBy** [a]
	  mediated(Y_11, o2#man)[A_9 **hasValue** AR_10]:-
	  mediated(Y_11, o2#human)[A_9 **hasValue** AR_10]
	  and Y_11[o1#hasGender **hasValue** o1#male].
	
	[a] By omitting this rule the mediator delegates to the target system the responsibility to populate the attributes of *man* that have been mapped in due to its superclass, *human*. Otherwise, if the rule is present, the mediator directly sends a complete instance of *man* (i.e. including its inherited attributes values) to the target party.

When creating a grounding for the abstract mappings, a meaning (i.e. a formal semantics) is associated with these mappings. We can say that the grounding prescribes how the mappings are interpreted and how are they going to be used in the context of that particular ontology language. As a consequence, it is very likely to need different grounding for different mediation scenarios (e.g. instance transformation vs query rewriting) while using the same set of abstract mappings.

The grounding mechanism we used is a grounding to WSML-Rule [HLF05][5] designed for instance transformation. The output of this mechanism is a set of WSML axioms representing rules; when evaluated by a WSML reasoner that already contains he source instances in its knowledge base t, these rules will generate a set of target instances, i.e. the mediation results. The new instances can be retrieved by simple queries and used in further computations. For example such a query would look like "`?x memberOf O2#man`" which translates into "what are the instances of concept *man* in the target ontology?". The WSML mapping rules fire and new instances of the concept *man*, are created having the source instances as inputs.

It is important to note that we assume that the mappings represented in the alignment format are correct and consistent. By correctness in this context it is understood that the given mappings conform to the domain expert's view on each particular heterogeneity issue. Unfortunately the correctness as defined above cannot be checked or insured by tools since this is the very role of the domain expert: to validate and to approve the tool's

[5]The interested reader can find grounding to OWL and RDF in [dBFZ04, Sch05].

suggestions.

Part of this validation also involves the consistency checking. A set of mappings would be inconsistent if the mediation results (in our case the mediated instances) do not conform to the target ontology constraints and its entities' formal semantics. Since the alignment format does not have an associated formal semantics associated, the consistency of mappings cannot be checked at this level. However, after the grounding is applied and the mapping rules are loaded into the reasoner together with the required source and target schema information and constraints, such checking is possible. If the mappings are grounded to a rule-based ontology representation language (e.g. WSML-Rule) it might be necessary to include samples of instances to be mediated in order to detect the potential inconsistencies. If the mappings are grounded to a DL-based ontology language several mappings' properties (including consistency) can be checked as in [SSW06] without using sample instances.

Table 5.2: Mappings example in the Alignment Format

```
    <map>
      <Cell id="o1#persono2#man">
        <entity1><omwg:Class rdf:resource="&o1;person"/>
            <attributeValueCondition>
            <Restriction>
              <property>
                <Path>
                    <first><Attribute rdf:about="&o1;hasGender"/></first>
                </Path>
              </property>
              <comparator rdf:resource="equal"/>
              <value>&o1;male</value>
            </Restriction>
        </attributeValueCondition>
        </entity1>
        <entity2><omwg:Class rdf:resource="&o2;man"/></entity2>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>subsumption</relation>
      </Cell>
      <Cell id="o1#ageo2#age">
        <entity1><omwg:Attribute rdf:resource="&o1;age"/></entity1>
        <entity2><omwg:Attribute rdf:resource="&o2;age"/></entity2>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>subsumption</relation>
      </Cell>
      <Cell id="o1#nameo2#name">
        <entity1><omwg:Attribute rdf:resource="&o1;name"/></entity1>
        <entity2><omwg:Attribute rdf:resource="&o2;name"/></entity2>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>subsumption</relation>
      </Cell>
      <Cell id="o1#hasGendero2#man">
        <entity1>
            <omwg:Attribute rdf:resource="&o1;hasGender"/>
            <valueCondition>
                <Restriction>
                    <property>
                        <Path>
                            <first><Attribute rdf:about="&o1;hasGender"/></first>
                        </Path>
                    </property>
                    <comparator rdf:resource="equal"/>
                    <value>&o1;male</value>
                </Restriction>
            </valueCondition>
        </entity1>
        <entity2><omwg:Class rdf:resource="&o2;man"/></entity2>
        <measure rdf:datatype='&xsd:float'>1.</measure>
        <relation>subsumption</relation>
      </Cell>
</map>
```

Chapter 6

Implementation

The ideas and methods presented in this deliverable are used in the mediation component of the WSMX architecture [HCM+05, MMCZ06]. The WSMX Data Mediation component is designed to support data transformation, which is intended to transform source ontology instances entering the system into instances expressed in terms of the target ontology. As described above, in order to make this possible the data mediation process consists of a design-time and a run-time phase. Each of these two phases has its own implementations: the *Ontology Mapping Tool*, described in Section 6.1 and the *Run-time Data Mediator*, described in Section 6.2. Figure 6.1 gives an overview of the two tools and of the relationship between them. The Ontology Mapping Tool offers a graphical way of creating the mappings and supports the storage of the mappings for further usage or refinement. The Run-time Data Mediator is able to load the mappings created at design-time and use them to transform the incoming data from terms of one ontology (the source ontology) to terms of another ontology (the target ontology) during run-time.

The creation of mappings (at design-time) is a semi-automatic, computer-assisted process that keeps the domain expert in control of the process. The run-time phase, on the other hand, is a completely automatic process that runs from end-to-end with no human intervention as long as mappings between the source and target ontology exist in the mapping repository.

In the following subsections the main features of these tools are presented, emphasizing their relationship with the formal model presented above.

6.1 Ontology Mapping Tool

The Ontology Mapping Tool is implemented as an Eclipse plug-in, part of the Web Service Modeling Toolkit (WSMT)[1] [Ker06], an integrated environment for ontology creation, visualization and mapping. The Ontology Mapping Tool is currently compatible with WSMO ontologies (but by providing the appropriate wrappers different ontology languages could be supported). It offers different ways of browsing the ontologies using perspectives and allows the domain expert to create mappings between two ontologies (source and target) and to store them in a persistent mapping storage.

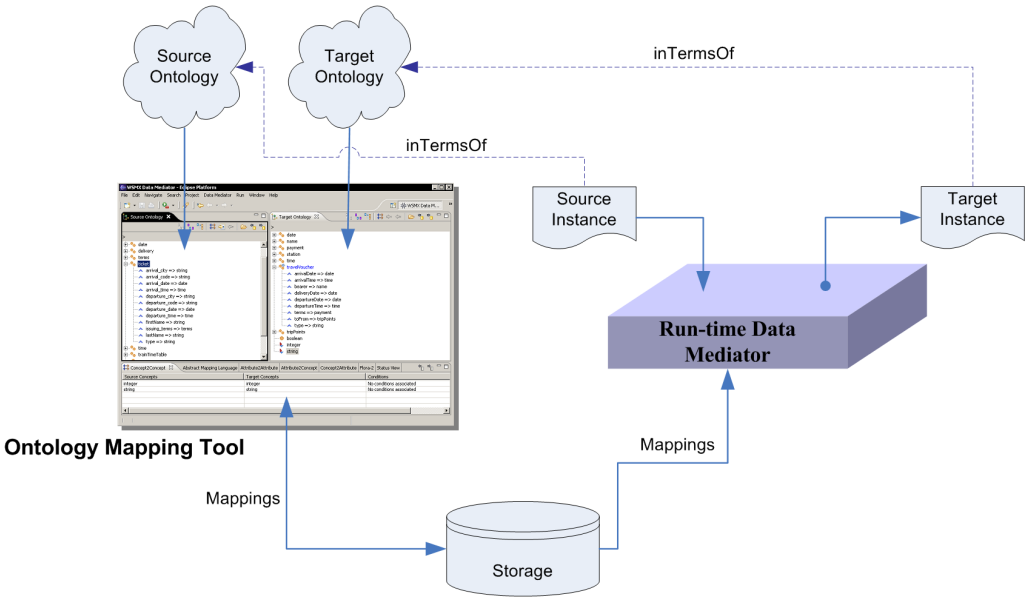

Figure 6.1: Overview on the Ontology Mapping Tool and Run-time Data Mediator

6.1.1 Perspectives

Currently only the *PartOf* and *InstanceOf* perspectives are fully implemented. They follow the principles of the formal model and support the features described in the next subsections.

Figure 6.2 shows how the perspectives are represented in the Ontology Mapping Tool.

6.1.2 Decomposition algorithm

The decomposition algorithm is one of the most important algorithms in our approach and it is used to offer guidance to the domain expert in the mapping process and to com-

[1] Open source project available at http://sourceforge.net/projects/wsmt

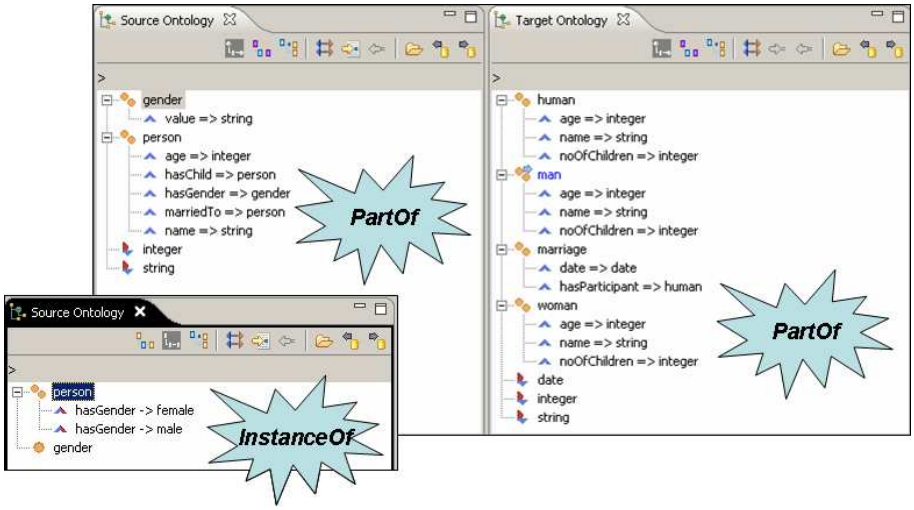

Figure 6.2: Perspectives in the Ontology Mapping Tool

pute the structural factor as part of the suggestions algorithms (described later in this chapter). By decomposition the descriptions of a compound item are exposed and made available to the mapping process. The decomposition algorithm can be applied to description items and returns the description items (if any) for the successors of those particular description items. An overview of this algorithm is presented in Listing 6.1

Listing 6.1: The Decomposition algorithm

```
void decompose(Collection collectionOfItems){
    Collection result;
    for each item in collectionOfItems do{
        result = result + [item];
        if isCompound(item){
            result = result + getDescriptions(item);
        }
        if isDescriptionItem(item){
                Item successorItem = getSuccessor(item);
                if (not createsLoop(succesorItem)){
                    result = result + [successorItem];
                    result = result + getDescriptions(item);
                }
        }
    }
    return result;
}
```

The implementation of *isCompound*, *isDescriptionItem*, *getDescriptions*, *getSuccessor*, and *createsLoop* differ from one view to another - for example, the cases when loops are encountered (i.e. the algorithm will not terminate) have to be addressed for each view in particular.

6.1.3 Suggestion Algorithms

Suggestion algorithms do not represent the focus of the work presented here. However, in order to deliver a truly semi-automatic mapping tool, suggestion algorithms are a necessity.

The suggestion algorithms are used to help the domain expert making decisions during the mapping process, regarding the possible semantic relationships between source and target items in the current mapping context. A combination of two types of such algorithms is used: the first being a lexical algorithm and the second being the structural algorithms that consider the description items in their computations. Brief descriptions of these algorithms are provided below.

For each pair of items the suggestion algorithms compute an *eligibility factor* (EF), indicating the degree of similarity between the two items: the smallest value (0) means that the two items are completely different, while the greatest value (1) indicates that the two items maybe similar. For dealing with the values between 0 and 1 a threshold value is used: the values lower than this value indicate different items and values greater than this value indicate similar items. Setting a lower threshold assures a greater number of suggestions, while a higher value for the threshold restricts the number of suggestion to a smaller subset.

The EF is computed as a weighted average between a *structural factor* (SF), referring to the structural properties and a *lexical factor* (LF), referring to the lexical relationships determined for a given pair of items. The weights can be chosen based on the characteristics of the ontologies to be mapped. For example when mapping between ontologies developed in dissimilar spoken languages the weight of LF should be close to 0 in contrast with the case when mapping between ontology developed in the same working groups or institutions (the usage of similar names for related terms is more likely to happen).

Even if the structural factor is computed using the decomposition algorithm, the actual heuristics used are dependent on the specific views where it is applied. In a similar manner the current views determine the weight for the structural and lexical factors as well as the exact features of the items to be used in computations.

Lexical Factor

The lexical factor is computed based on the syntactic similarities between the names of a given pair of items. There are two main aspects used in these computations: first, the lexical relationships between the terms as given by WordNet[2] and second, the results returned by string analysis algorithms. WordNet is an on-line lexical reference, inspired by current psycholinguistic theories of human lexical memory [MBF+90]. English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one

[2]More details available at http://www.cogsci.princeton.edu/ wn/

underlying lexical concept. Different relations such as hyponymy and hypernymy link the synonym sets.

The first step in computing the lexical factor involves the splitting of both source and target term in tokens. Tokens are separated in a term by one or more symbols like '_', '-', or by the usage of capital letters inside the term's name. For example the term *PurchaseOrder_Request* is formed of the following tokens: *purchase*, *order* and *request*. For each of the source tokens a set of lexically related terms are retrieved by using WordNet (generically referred further as synonyms for brevity even if hyponims and hypernims are considered as well). The Monge and Elkan's recursive matching algorithm [ME96] is used to compute the matching score between each of the source tokens' synonyms and the target term[3]. As such, the lexical factor is computed by considering the number of tokens in a term and the sum of the maximum matching score between each source token's collection of synonyms and the target term. The following formula is used for determining the LF:

$$LF = M\frac{1}{S}\sum_{i=1}^{S}\max_{k=1}^{SS_i}(ME(A_{i,k}, B))$$

where:

- S represents the number of tokens in the source term. T is used in later formulas to denote the number of tokens in the target term.
- A represents the initial source term. A_i represents the set of synonyms for the i-th token. $A_{i,k}$ is the k-th synonym term from the i-th set of synonyms. SS_i gives the number of synonyms for the i-th token.
- B represents the target term.
- ME returns the matching score conforming to Monge and Elkan's algorithm.
- M is a multiplicity factor (*multiplier*) used to compensate for the difference in the number of tokens from the source and the target:

$$M = \begin{cases} \frac{\alpha}{S} & if \;\; S > \alpha \\ \\ \frac{S}{\alpha} & if \;\; S \leq \alpha \end{cases}$$

  where α is the average of number of tokens in source and target: $\alpha = \frac{S+T}{2}$. As such M can be calculated as:

[3]Since it would be too time expensive two compute to sets of synonyms and to compere each of them with eachother, we trade some accuracy for a better efficiency. We also do not tokenize the target since the Monge and Elkan's algorithm covers the cases when the target is formed by multiple tokens.

$$M = \begin{cases} \dfrac{S+T}{2S} & if \;\; S > T \\ \dfrac{2S}{S+T} & if \;\; S \leq T \end{cases}$$

A graphical representation of the values M takes when S and T vary from 1 to 10 is presented in Figure 6.3.

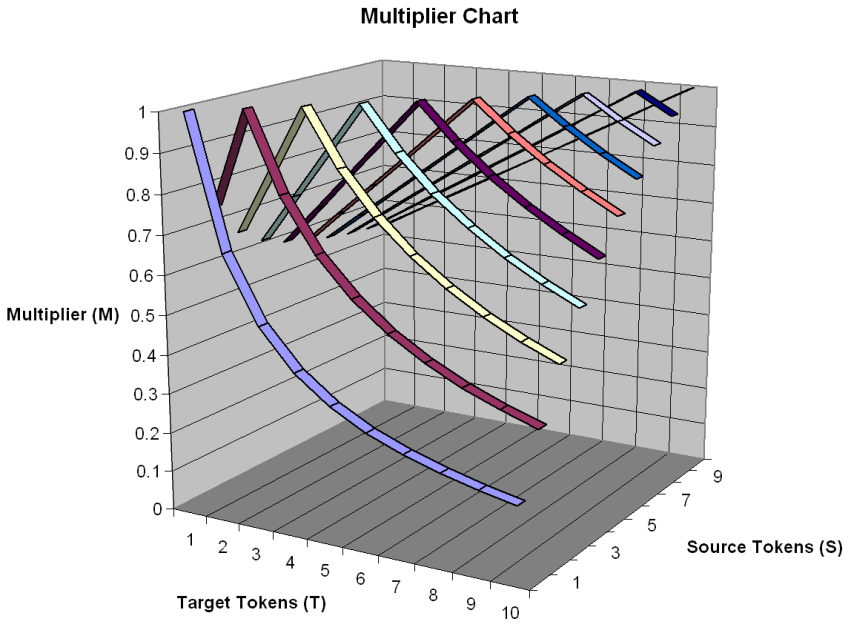

Figure 6.3: Values taken by the multiplicity factor when the number of tokens in the source and target terms varies from 1 to 10

It is important to note that M does not affect the score given by the string matching algorithms when the numbers of tokens in the source and the target are the same (i.e. $M = 1$). In the rest of the cases, M decreases as the difference between the numbers of tokens increases.

Structural Factor

The structural factor is computed based on the structural similarities between the two terms. The following formula is used:

$$SF = \begin{cases} \frac{2D}{S+T}\frac{S}{T} & if \;\; S < T \\ \\ \frac{2D}{S+T}\frac{T}{S} & if \;\; S \geq T \end{cases}$$

where:

- D is determined by the number of mappings between *primitive description items* that can be found by decomposition (see Listing 6.1) starting from the given source and target items. A *primitive (compound) description item* is a description item that has as successor a primitive (compound) item. For example in the *PartOf* perspective, a typical primitive description item would be an attribute that has as range a data type.

  As D is computed by a recursive algorithm, if two mapped *compound description items* are encountered during decomposition, their contribution to D is equal with $(SS + TS)/2$, where SS and TS represent the number of primitive description items that can be found by decomposition starting from the successor of the source, respectively of the target description item.

- S represents the number of primitive description items that can be found by decomposition starting from the initial source item.

- T represents the number of primitive description items that can be found by decomposition starting from the initial target item.

The eligibility factor might seem very expensive to compute between a selected source item and all the items from the target, especially when mapping large ontologies. The performance is drastically improved by the use of contexts since the set of item pairs for which the eligibility factor has to be computed is significantly cut down.

The algorithm provided in this section assists the user in finding correspondence between ontological entities. It provides suggestions based on simple measures. Research on schema matching is, however, a dynamic area. Various algorithms are continuously developed. More meaningful and complex mappings are discovered using these algorithms. To achieve improved results they exploit additional information like the structure of the ontology, external resources like the web or a corpus, and the available instance data.

The alignment format succinctly described Section 5.1 provides an output format to represent mappings for such algorithms[4]. Used as an exchange format, it allows our tool to use suggestions made by these algorithms.

[4]The Ontology Alignment Evaluation Initiative (http://oaei.ontologymatching.org/) evaluates the performance of matching algorithms. It uses this alignment format as the exchange format to compare generated mappings.

6.1.4 Bottom-Up vs Top-Down Approach

Considering the algorithms and methods described above, two possible approaches regarding ontology mapping are supported by the Ontology Mapping Tool: bottom-up and top-down approaches.

The *bottom-up approach* means that the mapping process starts with the mappings of the primitive items (if possible) and then continues with items having more and more complex descriptions. By this the pairs of primitive items act like a minimal, agreed upon set of mappings between the two ontologies, and starting from this minimal set more complex relationships could be gradually discovered. This approach is useful when a complete alignment of the two ontologies is desired.

The *top-down approach* means that the mapping process starts with mappings of compound items and it is usually adopted when a concrete heterogeneity problem has to be resolved. This is done when the domain expert is interested only in resolving a particular item's mismatches and not in fully aligning the input ontologies. The decomposition algorithm and the mapping contexts it updates, will help the expert to identify all the relationships that can be captured by using a specific type of view and which are relevant to the problems to be solved.

In the same way as for the other algorithms, the applicability and advantages/disadvantages of each of these approaches depends on the type of view used.

6.2 Run-time Data Mediator

The *Run-time Data Mediator* plays the role of the data mediation component in WSMX (available together with the WSMX system[5]). It uses the abstract mappings created at design time, grounds them to WSML, and uses a reasoner to evaluate them against the incoming source instances. The mapping rules, the source instances and if necessary, source and target schema information are loaded into the reasoning space in what could be seen as a "pseudo-merged" ontology (i.e. the entities from the source and the target and the rules are strictly separated by namespaces). By querying the reasoning space for instances of target concepts, if semantically related source instances exist, the rules fire and produce as results the target instances.

The storage used is a relational data base but the mappings could be stored using the mapping documents in the file system as well. Figure 6.4 gives a more detailed overview of the mediation tools and their relationships, together with the several possible usage scenarios identified for the Run-time Data Mediator.

It is important to note that the mappings grounding module is integrated as part of the run-time component. In this way, the same set of mappings (i.e. abstract, ontology

[5]Open Source Project available at http://sourceforge.net/projects/wsmx

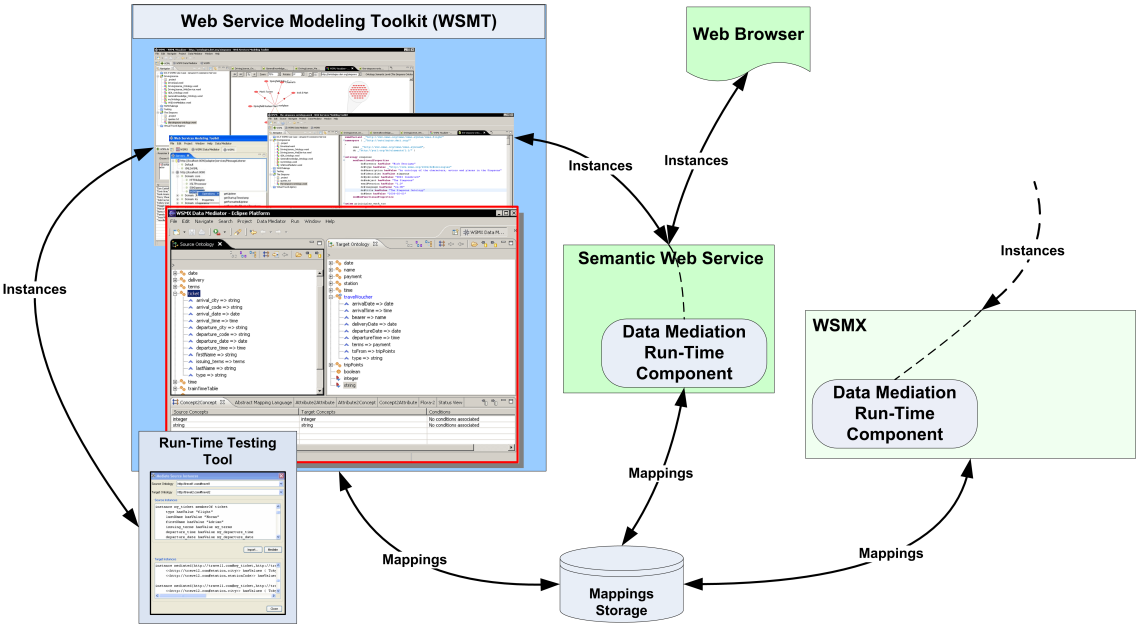

Figure 6.4: Run-time Data Mediator Usage Scenarios

language-independent mappings) can be grounded to different languages depending on the scenario in which the run-time mediator is used. By grounding, a formal semantics is associated with the mappings, such that only at this stage is it stated what exactly it means to have two items mapped to each other. These formal semantics differ from one mediation scenario to another, i.e. different grounding has to be applied when using the abstract mappings in instance transformation than when using them in query rewriting. An additional advantage is an easier management of mappings that form the ontology alignment. If ontologies evolve, the mappings have to be updated accordingly and it becomes more efficient to perform these updates free of any language-related peculiarities.

The main scope of the Run-time Data Mediator is to be deployed as a component in the WSMX environment. Additionally it can be made available as a Web service (i.e. a Semantic Web service) that can be invoked directly via SOAP from specialized tools (one of such invokers is deployed as a plug-in in WSMT) or through a Web interface using a Web browser. And finally, the Run-time mediation can be offered as a stand-alone application that helps in testing and evaluating the mappings during the mapping process at design-time. For more convenience, the stand-alone version can be integrated and delivered together with WSMT as a helper tool for the ontology mapping.

Chapter 7

Related Work

The resource decoupling and isolation principles implemented by Web services could become a disadvantage, considering the heterogeneity problems introduced by this approach. For this reason, this area have been intensely investigated in the last few years, in order to offer robust solutions towards solving such problems that may appear in operations involving Web services.

[WS03] proposes a mechanism to solve the differences between a requester's needs and a provider's offer in the discovery process. This approach applies syntactic and structural algorithms to asses the similarity between the desired WSDL[1] specifications and the set of WSDL specifications advertised in UDDI[2] . The syntactic algorithm uses a *WordNet-Powered Vector-space model* to retrieve published WSDL services that are most similar to the input description. The structural algorithm involves the comparison of the operation set offered by the services, which is based on the comparison of the structures of the operations input and output messages, which, in turn, is based on the comparison of the data types of the objects communicated by these messages. The main difference with our approach is that in our case the computations of the lexical and structural factors are part of an interactive mapping process. This means that the more mappings are created the better results the structural algorithm returns.

Even if such a method might present clear advantages over the ”traditional” manual browsing of UDDI repository, we think that it is not enough for a truly dynamic and (semi-) automatic web service discovery. We consider that comprehensive semantic descriptions based on ontologies have to be associated with Web services and used in solving the heterogeneity problems. Ontology mapping-based mediation is the key to dynamic and (semi-)automatic solutions due to its generality and its great potential to reuse. Our approach proposes methodologies and solutions that could be applied in most of the Web services' operations (including discovery, selection, invocation, etc.) where heterogeneity problems arise. However, some of the lexical algorithms' main principles

[1]Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl

[2]UDDI technical paper, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

presented in [WS03] (including the use of WordNet) can be also found in our approach in the suggestion algorithms (see Section 6.1.3 for more details). On the other hand, while in our approach synonyms, hypernyms and hyponyms are treated the same when applying the Monge and Elkan's algorithm, Wang and Stroulia use three separate weights values to compute the overall lexical similarity factor in order to "bypass the problem of lacking effective automated word sense disambiguation techniques", as they claim.

Another important reference point for our work is represented by the work done in the area of database integration in general and database schema matching in particular. A survey of approaches towards schema matching is presented in [RB01] and a classification of the Match operator is provided (Figure 7.1).

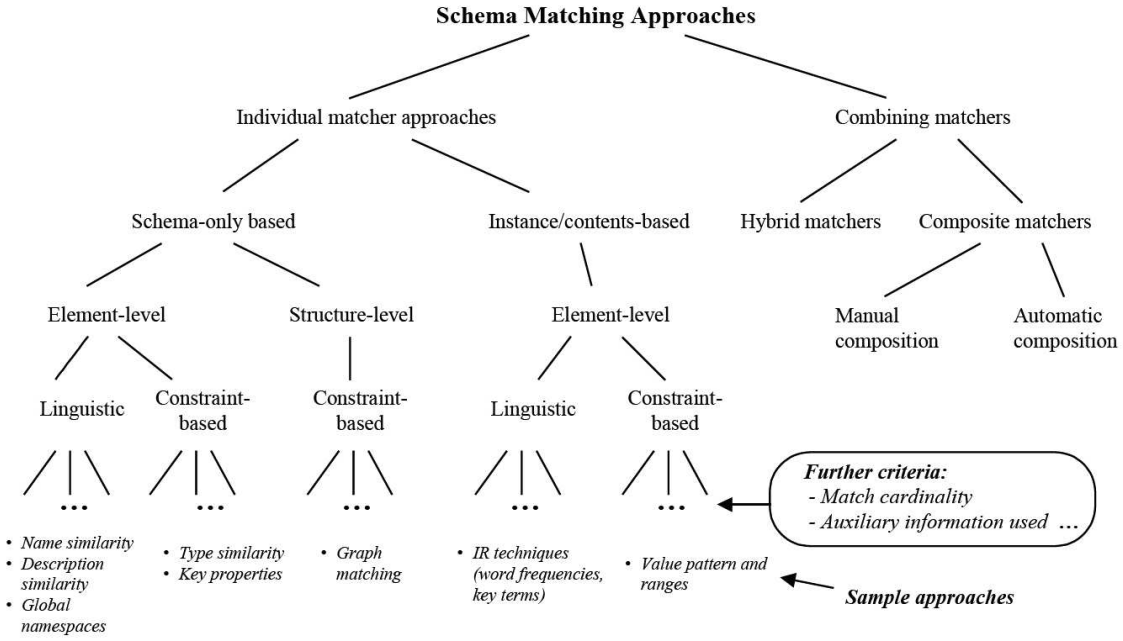

Figure 7.1: Classification of schema matching approaches (from [RB01])

In our approach, we use *schema-only*-based [3] mechanisms to semi-automatically determine the matches between elements of the source and target schema. The mappings created by using the perspectives presented above are 1:1 element level mappings but by layering support for transformation functions on top of perspectives 1:n and n:1 *element level* can be created. These mappings represented in the alignment format are grounded to a set of rules in an ontology language to be evaluated into a reasoner. By using reasoning and appropriate queries, the existing rules behave like virtual *structure-level* mappings

[3]The *InstanceOf* perspective does make use of instances but they are not application-specific instances that are part of an instance store, but instances that are normally shared together with the schema (the distinction made in here is similar with the one made in [Gua98] between the ontology and the "core" knowledge base). They are later used as "constants" when creating an instance base, based on that particular schema.

allowing the creation of data structures expressed in terms of the target ontology based on data expressed in terms of the source ontology. Furthermore, not explicitly creating mappings between whole structures enables the reuse of the mapping information from one particular heterogeneity case to another. Regarding the suggestion mechanisms, we use a set of lexical algorithms for *name matching* as described in Section 6.1.3. Additionally, the usage of the already existing mappings in the structural algorithms can be categorized as a *constraint-based* approach, as defined in [RB01].

Kalfoglou and Schorlemmer [KS03] provide a survey of ontology mapping approaches and classify them in several categories based on the type of work behind them. We've selected three of these approaches which we consider relevant for our work: Frameworks, Methods and Tools, and Translators. For each of these categories one approach has been identified and analyzed below in comparison to our work: MAFRA [SR03] for Frameworks, PROMPT [NM03] for Methods and Tools and Abiteboul and colleagues' approach [ACT97] for Translators.

MAFRA and PROMPT have been chosen because of the similarities they share with our work: they both take a semi-automatic approach towards ontology mapping having the human user involved in an interactive mapping process where he or she has to choose between a set of available actions to solve the existing mismatches between the ontologies. The work regarding "correspondence and translation for heterogeneous data" presented in [ACT97], even though it tackles integration of heterogeneous data in the database world, has been chosen due to the similarity with our approach to instance transformation based on ontology mappings created before hand.

MAFRA proposes a *Semantic Bridge Ontology* to represent the mappings. This ontology has as central concept, the so called "Semantic bridge", which is the equivalent of our mapping language statements. The main difference with our approach is that MAFRA does not define any explicit relation between the graphical representation of the ontologies in their tool and the generation of these semantic bridges or between the user's actions and the particular bridges to be used. The formal abstract model we propose links the graphical elements of the user interface with the mapping representation language, ensuring a clear correspondence between the user actions and the generated mappings.

PROMPT is an interactive and semi-automatic algorithm for ontology merging. The user is asked to apply a set of given operations to a set of possible matches, based on which the algorithm recomputes the set of suggestions and signals the potential inconsistencies. The fundamental difference in our approach is that instead of defining several operations we have only one operation (*map*) which will take two ontology elements as arguments, and multiple *perspectives* to graphically represent the ontologies in the user interface. Based on the particular types of perspectives used and on the roles of the *map* action arguments in that perspective the tool is able to determine the type of mapping to be created. This allows different ontology mismatches to be addressed using a single map action by switching between perspectives. PROMPT defines the term *local context* which perfectly matches our *context* definition: the set of descriptions attached to an item

together with the items these descriptions point to. While PROMPT uses the local context in decision making when computing the suggestions, we also use the context when displaying the ontology.

Instead of allowing browsing on multiple hierarchical layers, as PROMPT and MAFRA do, we adopt context-based browsing that allows the identification of the domain expert's intentions and generates mappings.

Abiteboul and colleagues [ACT97] propose a middleware data model, as basis for the integration task, and declarative rules to specify the integration. The model is a minimal one and the data structure consists of ordered label trees. The authors claim that "even though a mapping from a richer data model to this model may loose some of the original semantics, the data itself is preserved and the integration with other models is facilitated". Since the main scenario they target is data translation while in our approach we apply our framework to instance transformation, the relation is obvious. The tree model they use, resembles our items model for the perspectives with the difference that our model can be grounded to several "view-points" on the two conceptualizations to be mapped. The authors also consider two types of rules: *correspondence rules* used to express relationships between tree nodes (similar with our mappings in the alignment format but expressed in Datalog) and *translation rules*, a decidable subcase for the actual data translation (resembling our mapping rules in a concrete ontology language, such as WSML). Additionally, in our approach, we extend our model to the design-time phase as well in order to directly capture the relationships between the ontologies in terms of the "middleware" model.

Chapter 8

Conclusion and Further Work

In this deliverable we define a formal model for creation of mappings. This model sits between the graphical elements used to represent the ontologies and the result of the mapping process, i.e. the ontology alignment. By defining both the graphical instruments and the mapping strategies in terms of this model we assure a direct and complete correspondence between the human user actions and their effect on the generated ontology alignment. The model formally defines the kind of effect a user action has in terms of the generated alignment (for example, a mapping between two compound items in the *PartOf* view always generates only a *classMapping* in the alignment). Additionally the model formally defines the allowed user actions in the graphical interface and what each of them means in terms of the model and in terms of the alignment that is generated.

We also propose a set of different graphical perspectives that can be linked with the same model, each of them offering a different viewpoint on the displayed ontology. By combining these types of perspectives different types of mismatches can be addressed in an identical way from one pair of perspectives to the other. On top of these perspectives a lexical and a structural algorithm offer suggestions to the domain expert. A mechanism of context updates and decomposition offers guidance through the whole mapping process.

As future work, we plan to focus on identifying more relevant perspectives and to investigate the possible combination of these perspectives from the point of view of the types of mappings that can be generated. These would lead in the end to defining a set of mapping patterns in terms of our model, which will significantly improve the mechanism for finding mappings. Another point to be investigated is mapping using multiple participants from the source and/or from the target ontology. In this deliverable we investigated only the cases when exactly one element from the source and exactly one element from the target can be selected at a time to be mapped. We also plan to address transformation functions from the perspective of our model. Such transformation functions (e.g. string concatenation) would allow the creation of new target data, based on a combination of given source data.

Bibliography

[ACT97] S. Abiteboul, S. Cluet, and T.Milo. Correspondence and translation for heterogeneous data. In *Proceeding of International Conference on Database Theory*, pages 351–363, 1997.

[BM04] P.V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, October 2004.

[dBFZ04] J. de Bruijn, D. Foxvog, and K. Zimmerman. Ontology mediation patterns library. SEKT Project Deliverable D4.3.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.

[dBLK+05] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. The Web Service Modeling Language WSML. Technical report, WSML Working Draft, `http://www.wsmo.org/TR/d16/d16.1/v0.2/`, October 2005.

[DE04] M. Dean and G. Schreiber (Eds.). OWL Web Ontology Language Reference. W3C Recommendation, Available from `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`, February 2004.

[ESS05] M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Fourth International Semantic Web Conference (ISWC-2005)*, November 2005.

[ESS06] J. Euzenat, F. Scharffe, and L. Serafini. Specification of the alignment format. Technical report, Knowledge Web Deliverable, 2006.

[FPD+05] C. Feier, A. Polleres, R. Dumitru, J. Domingue, M. Stollberg, and D. Fensel. Towards intelligent web services: The web service modeling ontology (WSMO). In *International Conference on Intelligent Computing (ICIC)*, 2005.

[GN88] M. R. Genesereth and N. J. Nilson. *Logical Foundations of Artificial Inteligence*. Morgan-Kaufmann, 1988.

[Gru93] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–229, 1993.

[Gua98] N. Guarino. Formal ontology and information systems. In *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems FOIS'98*, pages 3–15, Trento, Italy, June 1998. Amsterdam, IOS Press.

[HCM+05] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *International Conference on Web Services (ICWS 2005)*, July 2005.

[HLF05] A. Polleres H. Lausen, J. de Bruijn and D. Fensel. WSML - A Language Framework for Semantic Web Services. In *W3C Workshop on Rule Languages for Interoperability*, April 2005.

[HST99] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *In Proceeding of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[Ker06] M. Kerrigan. WSMOViz: An Ontology Visualization Approach for WSMO. In *10th International Conference on Information Visualization*, 2006.

[KLW95] 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.

[KS03] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review Journal (KER)*, 18(1):1–31, 2003.

[MBF+90] G.A. Miller, R. Beckwith, C. Felbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[MC05] A. Mocan and E. Cimpian. Mapping creation using a view based approach. In *1st International Workshop on Mediation in Semantic Web Services (Mediate 2005)*, December 2005.

[ME96] A. Monge and C. Elkan. The field-matching problem: algorithm and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[MMCZ06] A. Mocan, M Moran, E. Cimpian, and M. Zaremba. Filling the Gap Extending Service Oriented Architectures with Semantics. In *Second IEEE International Symposium on Service-Oriented Applications, Integration and Collaboration (SOAIC-2006)*, October 2006.

[NM03] N.F. Noy and M. A. Munsen. The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 6(59):983–1024, 2003.

[RB01] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, (10):334–350, 2001.

[Sch05] F. Scharffe. Mapping and merging tools design. Ontology management working group (omwg) working draft, Digital Enterprise Research Institute, University of Innsbruck, 2005.

[SR03] N. Silva and J. Rocha. Semantic web complex ontology mapping. In *Proceedings of the IEEE Web Intelligence (WI2003)*, page 82, 2003.

[SSW06] H. Stuckenschmidt, L. Serafini, and H. Wache. Reasoning about Ontology Mappings. In *ECAI 2006 Workshop on Context Representation and Reasoning*, Riva del Garda, Italy, 2006.

[WS03] Y. Wang and E. Stroulia. Semantic structure matching for assessing web-service similarity. In *First International Conference on Service Oriented Computing*, Trento, Italy, December 2003.