D4.4.1 Ontology Mediation Management V1

Jos de Bruijn (UIBK)
Francisco Martín-Recuerda (UIBK)
Marc Ehrig (UKARL)
Axel Polleres (UIBK)
Livia Predoiu (UIBK)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D4.4.1 (WP4)
This deliverable provides a survey on differences and overlap that can occur between (concepts of) different ontologies. We formalize an ontology mapping language which can be used to declaratively specify the overlap between two different ontologies. We then demonstrate how the mapping language fits in an overall framework which enables ontology mediation on the Semantic Web. An important part of this framework is mapping discovery, a semi-automatic means to find overlap between different ontologies.
Keyword list: Ontology Mediation, Semantic Integration, Ontology Mapping, Mapping Discovery

Document Id.	SEKT/2004/D4.4.1/v1.0
Project	SEKT EU-IST-2003-506826
Date	February 8, 2005
Distribution	public

Copyright © 2005 Digital Enterprise Research Institute, University of Innsbruck

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid
Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Technikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EAD, Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

Different applications on the Semantic Web are expected to use different ontologies for the annotation and interpretation of data on the Web. Such differences hamper interoperation between applications and hamper reuse of data and ontologies across applications.

Reuse of data and interoperation between applications on the Semantic Web can be achieved in two principally different ways: by ontology mapping or ontology merging. In the case of ontology mappings, semantic overlap between ontologies needs to be detected and described using a formal language. Such a mapping can then be used for querying across ontologies, transforming data between representations, etc. Eventually, such mappings can be used to integrate autonomous heterogeneous applications over the Semantic Web. In the case of ontology merging, semantic overlap between ontologies needs to be detected and, based on this overlap, a new ontology is created which can be shared between applications which used the original ontology. This ontology can now be used to enable interoperation between applications on the Semantic Web.

This deliverable describes a framework for ontology mediation. We formalize an ontology mapping language which can be used to declaratively specify the overlap between two different ontologies. We then demonstrate how the mapping language fits in an overall framework which enables ontology mediation on the Semantic Web. An important part of this framework is mapping discovery, a semi-automatic means to find overlap between different ontologies.

Contents

Chapter 1

Introduction

It is expected that differences between ontologies will cause problems in interoperation between applications on the Semantic Web, as differences in data schemas cause problems in interoperation of current information systems.

Several approaches to creating ontology mappings have been described in the literature, e.g. MAFRA [43] and OntoMerge [22]. However, these approaches do not take into account the most recent Semantic Web languages and they do not describe how to use the mappings in order to achieve certain tasks on the Semantic Web.

We provide means to overcome problems of interoperability by describing an ontology mapping language for the Semantic Web and an ontology mediation framework which can be used to achieve specific tasks on the Semantic Web.

It is the aim of this deliverable to provide a framework for ontology mediation in the SEKT project. In order to achieve this, we first describe a number of generic ontology mediation use cases as well as a number of application scenarios relevant for SEKT. Based on these scenarios we develop a framework for ontology mediation with different stages, namely mapping creation, storage and run-time mediation. In this deliverable we particularly focus on the mapping creation phase where we describe a method for discovering ontology mappings, as well an on OWL-based ontology mapping language for the specification of such mappings. Storage and run-time mediation are subject for the deliverable D4.5.1: Ontology Mediation as Service Component.

We describe certain preliminaries and their coherence. We then focus on the task of discovering mappings between ontologies based on lexical and structural similarities and the application of certain heuristics. We describe how all these preliminaries and the mapping discovery relate in an overall mediation framework.

In this deliverable we describe a formal mapping language, as well as a framework for creating and using ontology mappings. The formal mapping language is based on the mapping language developed in deliverable D4.3.1 [9]. In fact, in this deliverable we formally ground the mapping language to OWL DL in order to enable mapping between

OWL ontologies. The DIP deliverable D1.5 [57] provides a formal grounding to WSML-Flight [8] in order to enable mapping between WSML-Flight ontologies.

In the remainder of this introduction we describe the terminology used throughout the deliverable.

1.1 Terminology

In order to make this deliverable self-contained we present here a slightly adapted version of the terminology clarification we have provided earlier in deliverables D4.2.1 [11] and D4.3.1 [9].

This section provides some clarification on the terminology used throughout this deliverable. We deem this necessary, because there exist many different understandings of the terminology in the literature.

Ontology An *ontology* O is a 4-tuple $\langle C, R, I, A \rangle$, where C is a set of concepts, R is a set of relations, I is a set of instances and A is a set of axioms. Note that these four sets are not necessarily disjoint (e.g. the same term can denote both a class and an instance), although the ontology language might require this. Each concept can have a number of attributes associated with it. An attribute is a special kind of relation, namely a binary relation associated with a concept.

All concepts, relations, instances and axioms are specified in some logical language. This notion of an ontology coincides with the notion of an ontology described in [59, Section 2] and is similar to the notion of an ontology in OKBC [5]. Concepts correspond with classes in OKBC, slots in OKBC are particular kinds of relations, facets in OKBC are a kind of axiom and individuals in OKBC are what we call *instances*[1].

In an ontology, concepts are usually organized in a subclass hierarchy, through the *is-a* (or subconcept-of) relationship. More general concepts reside higher in the hierarchy.

Instance Base Although instances are logically part of an ontology, it is often useful to separate between *an ontology* describing a collection of instances and *the collection of instances* described by the ontology. We refer to this collection of instances as the *Instance Base*. Instance bases are sometimes used to discover similarities between concepts in different ontologies (e.g. [63], [20]). An instance base can be any collection of data, such as a relational database or a collection of web pages. Note

[1] We use the terms instance and individual interchangeably throughout this document. Note that an instance is not necessarily related to a class.

that this does not rule out the situation where instances use several ontologies for their description.

Instances are an integral part of an ontology. However, we expect that most instance data will be stored in private data stores and will not be shared along with the ontology. The instances contained in the ontology itself are typically those instances that are shared.

Ontology Language The ontology language is the language which is used to represent the ontology. Semantic Web ontology languages can be split up into two parts: the logical and the extra-logical parts. The *logical* part amounts to a theory in some logical language, which can be used for reasoning. Class (concept) definitions, property (relation) definitions, and instance definitions correspond with axioms in the logical language. In fact, such definitions are merely a more convenient way to write down such axioms.

The *extra-logical* part of the language typically consists of non-functional properties (e.g. author name, creation date, natural language comments, multi-lingual labels; see also Dublin Core [68]) and other extra-logical statements, such as namespace declarations, ontology imports, versioning, etc.

Non-functional properties (also called *annotations*) are typically only for the human reader, whereas the other extra-logical statements are made for machine processing. For example, namespace declarations can be used to resolve Qualified Names to full URIs and the importing of ontologies can be achieved automatically by either (a) appending the logical part of the imported ontology to the logical part of the importing ontology to create one logical theory or (b) using a *mediator*, which resolves the heterogeneity between the two ontologies (see also the definition of **Ontology Mediation** below).

Ontology Mapping An *ontology mapping* M is a (declarative) specification of the semantic overlap between two ontologies O_S and O_T. This mapping can be one-way or two-way. In a one-way mapping we specify how to express terms in O_T using terms from O_S in a way that is not easily invertible. A two-way mapping works both ways, i.e. a term in O_T is expressed using terms of O_S and the other way around.

Ontology Mediation Ontology mediation is the process of reconciling differences between heterogeneous ontologies in order to achieve inter-operation between data sources annotated with and applications using these ontologies. This includes the discovery and specification of *ontology mappings*, as well as the use of these mappings for certain tasks, such as query rewriting and instance transformation. Furthermore, the *merging of ontologies* also falls under the term ontology mediation.

Matching We define *ontology matching* (sometime also called *mapping discovery*) as the process of discovering similarities between two source ontologies. The result of a matching operation is a specification of similarities between two ontologies.

Ontology matching is done through application of the *Match* operator (cf. [58]). Any schema matching or ontology matching algorithm can be used to implement the *Match* operator, e.g. [20, 28, 42, 47].

We adopt here the definition of *Match* given in [58]: "[*Match* is an operation], which takes two schemas [or ontologies] as input and produces a mapping between elements of the two schemas that correspond semantically to each other".

Mapping Language The mapping language is the language used to represent the *ontology mapping* M. Mapping languages often allow arbitrary transformation between ontologies, often using a rule-based formalism and typically allowing arbitrary value transformations.

Mapping Pattern Although not often used in current approaches to ontology mediation, patterns can play an important role in the specification of ontology mappings, because they have the potential to make mappings more concise, better understandable and reduce the number of errors (cf. [55]). A *mapping pattern* can be seen as a template for mappings which occur very often. Patterns can range from very simple (e.g. a mapping between a concept and a relation) to very complex, in which case the pattern captures comprehensive substructures of the ontologies, which are related in a certain way.

For the definitions of merging, aligning and relating ontologies, we adopt the definitions given in [16]:

Ontology Merging Creating one new ontology from two or more ontologies. In this case, the new ontology will unify and replace the original ontologies. This often requires considerable adaptation and extension.

Note that this definition does not say how the merged ontology relates to the original ontologies. The most prominent approaches are the *union* and the *intersection* approaches. In the union approach, the merged ontology is the union of all entities in both source ontologies, where differences in representation of similar concepts have been resolved. In the intersection approach, the merged ontology consists only of the parts of the source ontology which overlap (c.f. the *intersection* operator in ontology algebra [69]).

Ontology Aligning Bringing the ontologies into mutual agreement. The ontologies are kept separate, but at least one of the original ontologies is adapted such that the conceptualization and the vocabulary match in semantically overlapping parts of the ontologies. However, the ontologies might describe different parts of the domain in different levels of detail.

Relating Ontologies Specifying how the concepts in the different ontologies are related in a logical sense, i.e. creating an **Ontology Mapping**. This means that the original

ontologies have not changed, but that additional axioms describe the relationship between the concepts. Leaving the original ontologies unchanged often implies that only a part of the integration can be done, because major differences may require adaptation of the ontologies.

The term "Ontology Mapping" was defined above as a specification of the relationship between two ontologies. We can also interpret the word "Mapping" as a verb, i.e. the action of *creating* a mapping. In this case the term corresponds with the term "Relating Ontologies":

Mapping Ontologies Is the same as relating ontologies, as specified above.

Note that most disagreement in the literature is around the term *alignment*. We do not use the term alignment as such, but we do use the term *ontology aligning*. In most literature (e.g. [51]), alignment corresponds with what we call *relating ontologies* or *mapping ontologies*. Ontology aligning is also sometimes called *ontology reconciliation*.

The remainder of this deliverable is structured as follows.

Chapter 2 presents a number of use cases and scenarios for ontology mediation. These use cases and scenarios give the reader an idea about where and how to apply ontology mediation in Semantic Web applications. Chapter 3 describes the overall framework for ontology mediation. Chapter 4 describes a number of consideration with respect to ontology mediation. Chapter 5 describes APFEL, a method for the automatic discovery of ontology mappings. Chapter A describes the abstract mapping language which was developed in deliverable D4.3.1 [9], and described a formal grounding for the mapping language to mediate between OWL ontologies. We present related work in Chapter 7 and conclusions in Chapter 8.

Chapter 2

Use Cases and Scenarios for Ontology Mediation

This chapter describes the three generic use cases we see in the area of Ontology Mediation, as well as a number of typical scenarios which illustrate these use cases.

A use case, as in UML, corresponds to a generic task and thus abstracts from particular applications. An application scenario describes an actual application of ontology mediation in a particular application domain. One or more use cases might be applicable to this particular scenario.

The use cases and scenarios motivate the application of ontology mediation in the SEKT context. They will form the basis for the development of the mediation framework.

2.1 Generic Use Cases

This section describes the core technical use cases which need to be supported by the Ontology Mediation framework. We distinguish two use cases, which are detailed in the remainder of this section:

- Instance Mediation
- Ontology Merging

The first use case, Instance Mediation, addresses the tasks of instance transformation, unification and query rewriting. The second use case, Ontology Merging, addresses the way two source ontologies can be merged into one target ontology. The third use case, Creating Ontology Mappings, is about actually finding similarities between ontologies and creating mappings between the ontologies.

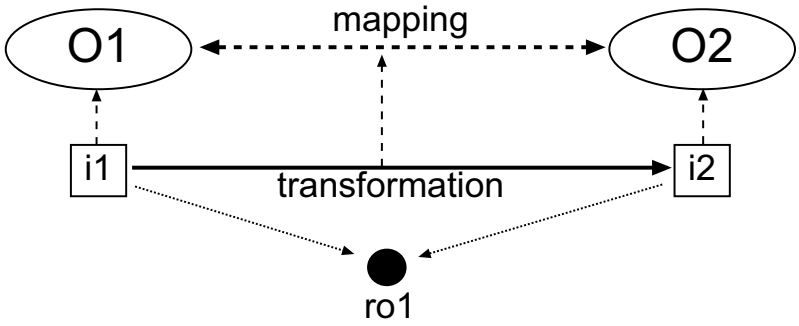

Figure 2.1: Instance Transformation

2.1.1 Use Cases for Instance Mediation

The following use cases are the typical use cases for instance mediation, where the emphasis is on instance transformation and unification.

Instance mediation as the process of overcoming differences between two instance bases, each described by an ontology. This includes the discovery and specification of ontology mappings, as well as the use of these mappings for certain tasks, such as query rewriting, instance transformation and instance unification.

As we can see from the above, instance mediation also requires the discovery and specification of ontology mappings. This makes apparent the inter-dependencies between the different use cases. We do not describe the discovery and specification of ontology mappings here; instead, these use cases are discussed later, because of the use in different other areas of ontology mediation.

Instance Transformation

For the instance transformation use case we assume two separate applications with separate instance stores both described by ontologies. The task to be performed is the transformation of an instance of a source ontology, say $\mathcal{O}_S$, to an instance of the target ontology $\mathcal{O}_T$. Figure 2.1 illustrates the process of instance transformation. An instance $i1$, which refers to ontology $O1$, is transformed into instance $i2$, which refers to ontology $O2$. What is important to note here is that the transformation itself is derived from the mapping between the two ontology and that both the original and the transformed instance provide information about the same real-world object.

This kind of transformation needs to be supported by the ontology mapping in the sense that the ontology mapping specifies the relationship between instances of the source ontology $\mathcal{O}_S$ and instances of the target ontology $\mathcal{O}_T$.

When an instance has been translated from $\mathcal{O}_S$ to $\mathcal{O}_T$, it is often necessary to detect whether the transformed instance corresponds to an existing instance in the instance store

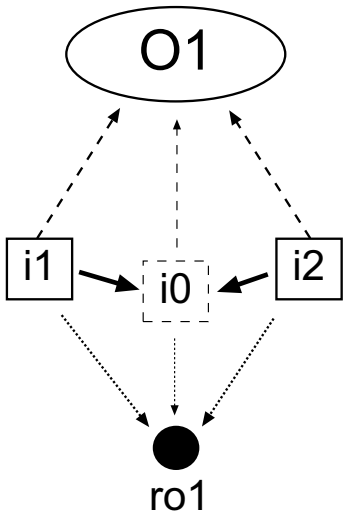

Figure 2.2: Instance Unification

of the target application in order to avoid duplication of information and in order to find out more about the instances in the knowledge base. We discuss this issue below.

Instance Unification

The instance unification problem can be summarized as follows:

Say, we have an ontology $\mathcal{O}$ and two instances I_1 and I_2 of that ontology. We want to check whether I_1 and I_2 refer to the same real-world object. In this case we need to unify I_1 and I_2 into a newly created instance I_0, which is the union of I_1 and I_2. Therefore, the instance unification task can be decomposed into (1) the identification of instances referring to the same real-world object and (2) taking the union of the two instances in order to obtain the unified instance.

If the instances I_1 and I_2 have been identified as referring to the same real-world object, but contain contradictory information, it is not possible to create a unified instance and the user should be informed of the inconsistency.

Figure 2.2 illustrates the process of instance unification. Two instances ($i1$ and $i2$) of the same ontology $O1$, which refer to the same real-world object $ro1$, are unified into one new instance, $i0$, which is the union of both instances, is also an instance of the ontology $O1$ and also describes to the same real-world object $ro1$.

The problem of instance unification is highly related to the problem of data and tuple matching. There has been much research in this area, e.g. [20, 41].

Instance transformation and instance unification are often required in a querying scenario where an application A queries another application B and the query results (consisting of instances) are transformed to the representation of A and unified with instances

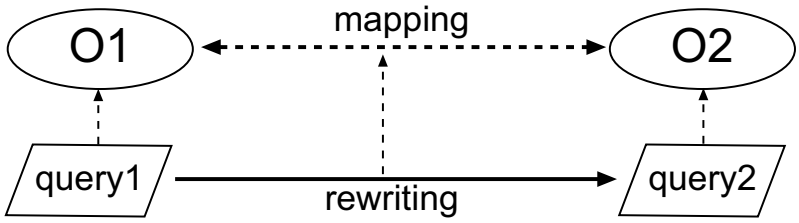

Figure 2.3: Query Rewriting

in the instance base of A.

In order to be able to query a data source which uses a different (unknown) ontology, the query originally formulated in terms of the application's ontology needs to be rewritten in terms of the other ontology. The next section describes the generic query rewriting use case.

Query Rewriting

An operation occurring very frequently in Knowledge Management application is querying of information sources. We want to allow an application to query different heterogeneous information sources without actually knowing about all the ontologies. In order to achieve this, a query written in terms of the application's ontology, needs to be rewritten using the terms in the target data source's ontology.

Say, we have an application A, which uses an ontology $\mathcal{O}_A$ for its information representation. Say now that this applications want to query a different data source, which uses ontology $\mathcal{O}_B$, but A does not know about the structure of this ontology. The application A now formulates a query $\mathcal{Q}_A$ in terms of ontology $\mathcal{O}_A$. In order to execute this query on the target data source, it needs to be rewritten onto query $\mathcal{Q}_B$, which is formulated in terms of ontology $\mathcal{O}_B$. This rewriting process is illustrated in figure 2.3.

After execution of the query, the results are transformed back to the $\mathcal{O}_A$ representation and unified with the local instances using the techniques for instance transformation and unification described above.

2.1.2 Ontology Merging

Besides the instance transformation and unification and query rewriting, we see another major use case for ontology mediation: Ontology Merging.

In the case of Ontology Merging [52], two source ontologies shall be merged into one target ontology[1] based on the source ontologies. In the general case, the source ontologies

[1]This merged ontology would be the union of the two source ontologies.

would disappear and only the target (merged) ontology remains. A special case is when the source ontologies remain, along with mappings to the merged ontology.

In the case where the source ontologies are discarded after the merge, the complete instance stores of the source ontologies have to be merged. In the latter case, the source ontologies can maintain their instance stores and during run-time of the application, processes of instance transformation and instance unification (cf. the previous subsection) are necessary.

Of course, when the source ontologies do not have instance stores associated with them, these problems do not occur. However, in the general case an ontology will have one or more instance stores associated with it. In special cases, such as the (distributed) development of ontologies, there will not be instance stores.

2.2 Scenarios for Ontology Mediation

This section describes a number of typical scenarios for ontology mediation.

2.2.1 Data Integration

Data Integration is concerned with the use of data from different sources in one application. The data from the different sources needs to be presented to the user in a unified way.

Using a Relational Database in a Semantic Web application

Relational databases are currently the most popular data storage paradigm in enterprises. As was shown in [66], a large amount of the information currently available over the Web is actually stored in relational databases. This clearly demonstrates the necessity of the ability to use a relational database source in a Semantic Web application.

Typically, a Semantic Web application would not want to deal with the peculiarities of a specific database schema. Especially since legacy database schemas are often specified using incomprehensible, organization of application-specific relation and attribute names, such as ‘TAJO003’.

In order for the application to use a Relational Database, the database schema has to be “lifted” to the ontology level[2], after which an ontology mapping can be created between the ontology used by the application and the ontology based on the database schema. Examples of relating relational database schemata to ontologies can be found in [7, 45, 67].

[2]This lifting can be done either directly by rewriting the database schema into an ontology (cf. [67]) or indirectly by relating the database schema to an existing ontology [7].

Once a relational data schema has been lifted to the ontology level and provided that the connector lifting the ontology[3] performs a two-way translation, i.e. it translates both instances from the relational representation to the ontology representation and queries from the ontology to the relational representation, the relational data source can be used in a Semantic Web application. In fact, the relational source can then be treated as an ontology with a corresponding instance store. The ontology corresponding to the relational database needs to be mapped to other ontologies on the Semantic Web in order to enable reuse of the database.

Using different heterogeneous Ontologies in a Semantic Web application

Larger applications typically make use multiple data sources to fulfill the information needs of its users. For instance, in an enterprise it could be the case that customer information and employee information are stored in separate sources. An application which wants to provide a search facility to search through all people known to the enterprise would have to integrate these separate sources.

We distinguish two cases for the use of different ontologies by one application. In the first case, the application uses a global ontology, where all specific local ontologies are mapped to the global ontology. In the second case, we assume a peer-to-peer like setting, where each application has it's own ontology and mappings exist between these different applications.

Using a Global Ontology Because one-to-one mappings between all involved ontologies do not scale in the general case (cf. [65, 64]), it is often preferred to have a central, global ontology to which all local ontologies are mapped.

We do not think of one global upper-level ontology for the Semantic Web, but rather different islands, consisting of a central ontology and local ontologies, along with the mappings between them, with mappings between the islands where appropriate. This layering could again define a hierarchy, where other central ontologies can combine several of these islands and so forth.

One could think of creating an ontology for a specific integration problem in an organisation or take the (more preferred) approach of relating the local ontologies to a domain ontology, which is a conceptual description of the domain and thus can be shared with others.

A clear advantage of this approach is that each application can use its own terminology, while still being able to communicate with other applications, which use different terminologies.

[3]This lifting involves mediation again which can be viewed as another ontology mediation problem as such using different "ontology languages", for example OWL and SQL. However, in this paper we restrict ourselves to mediation between ontology in the same language.

Using only Local Ontologies In the case of using only local ontologies, an ontology mapping needs to exist between every pair of ontologies, if one wants to mediate between. This approach is not very scalable in general, because it requires $O(n^2)$ ontology mappings, where n is the number of ontologies.

However, in some special cases, where there are very few ontologies or where using a global ontology is just not possible, this scenario can occur.

2.2.2 Ontology Evolution

One can expect ontologies to evolve over time. This holds also for ontologies that participate in mappings to other ontologies.

If either of the two ontologies involved in an ontology mapping changes (evolves), the mapping might become invalid. Therefore, the mapping between the ontologies needs to evolve together with the ontologies and versioning of both the ontologies and the mapping is required, as we explain below.

Evolving Ontology Mappings

Not only ontologies evolve, but also mappings evolve, especially in early stages of the mapping design.

For this scenario we assume static (non-evolving) ontologies and a changing (evolving) mappings between the ontologies. There are various reasons why a mapping may evolve. Examples are evolving insights into the source and target ontology and their similarities, new requirements on the mapping (e.g. a new subpart of the ontologies needs to be mapped, which was not considered before) and inadequate or faulty specification of the mapping.

This scenario does not only indicate the need for evolution support for ontology mappings, but also for versioning. Each change of the mapping requires a new uniquely identifiable and accessible version of the mapping, so that applications which use a certain version do not break because of changes in the mapping.

Mapping different versions of Evolving Ontologies

The evolution of ontology mappings is relatively simple compared to the mapping of evolving ontologies. In this case, both the source and the target ontology evolve over time and the mapping between the ontologies needs to evolve accordingly. The mapping between an evolving source ontology and a target ontology might even make apparent the need for the target ontology to evolve along with the source ontology in order to enable inter-operation.

The evolution of ontologies has the following implications for the mappings between ontologies:

- *Versioning* of the ontologies is required and the ontology mapping needs to be specified between *specific versions* of the source and target ontology. The mapping needs to refer to specific versions of the ontology.
- Evolution of ontologies indicates the need for *evolution of ontology mappings*. In many cases, when a new version of an ontology is created, a new version of each of the mappings in which the ontology is involved needs to be created. If the changes in the ontologies are formally and explicitly documented, these changes can be used as the basis for changes to be made to the mapping. We believe that in many cases, the evolution of the mapping can be done semi-automatically.
- Evolution of an ontology $\mathcal{O}_B$ which is mapped to an ontology $\mathcal{O}_A$ is mapped, might *indicate the need for subsequent evolution of* $\mathcal{O}_A$. An example of this is a case of database integration, using the local-as-view (cf. [39]) paradigm, where $\mathcal{O}_B$ is the global ontology and $\mathcal{O}_A$ is the local ontology. Evolution of ontologies indicates evolution of the domain and if two ontologies present a view on the same domain, both ontologies need to evolve.

Mappings between different Ontology Versions

When an ontology on the Semantic Web evolves, some mappings to the old ontology might become invalid. This problem can be solved by either evolving the mapping or by mapping to a specific version of the ontology.

When mapping to a specific version, problems may occur, for example because the instance base of the ontology evolves with the ontology. It is not feasible to maintain an instance base for each version of an ontology, because this can easily lead to inconsistencies and would actually be a maintenance nightmare. A way to deal with this problem is by providing a mapping between different versions of the ontology.

2.2.3 Browsing and Querying

One core use case for knowledge management is information access. Users have the need to not only create and process the information, but to access it in an easy and comprehensive way. This section focuses on the two access techniques browsing and querying and shows their intersection with mapping considerations.

Browsing

Browsing through the internet is a most common expression nowadays. The same sense can be used in the context of knowledge. People using a knowledge management system typically look for specific information which they expect to find in the system. As the users often don't exactly know what they are looking for, it is often easier to start with related information and then navigate to the goal object. This navigation is done most intuitive through graphical access as known from internet browsers. Extending this functionality for semantics is one core issue of the semantic web.

As an example the Spectacle Cluster Map[4] [27] is an application for the visualization of instantiated taxonomies such as class and concept hierarchies. A user can select the instances of which class to display from a list of all classes. The Cluster Map visualizes overlap between classes through shared instances. This way, the user can see clearly how the classes relate to each other, because of the instances that they share. This makes it very useful for analysis of hierarchically organized datasets. Figure 2.4 shows the Cluster Map Viewer displaying an example repository. All classes are represented by large green spheres. The smaller yellow spheres represent instances. All instances are connected to one or more classes through the balloon-shaped edges, which indicate the class membership. This is just a simple example, because it consists of a limited number of classes and instances, and there is not a lot of opportunity for overlap between classes (e.g., a resource is unlikely to be both an artist and an artefact).

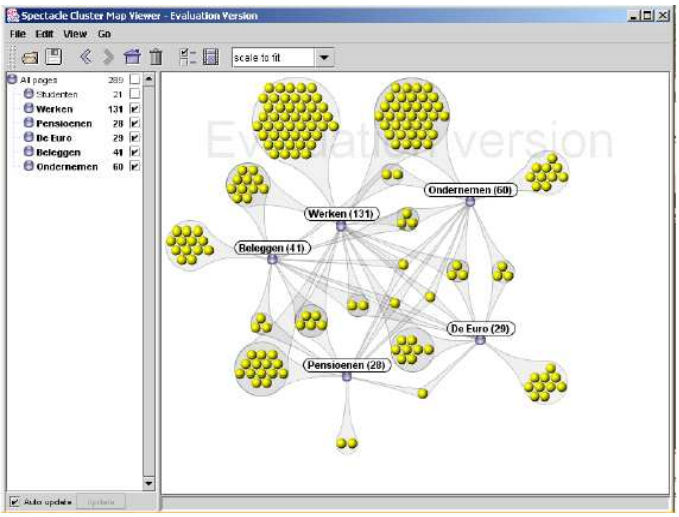

Figure 2.4: Screenshot of the Spectacle tool

Currently the systems are more and more distributed i.e. a big number of different ontologies arise, not only from one single PC but across peers. To allow efficient browsing the user only wants to be confronted with one coherent structure, without worrying

[4]Now called AutoFocus, see also http://www.aduna.biz/

about the original structure and source. Through integration of the various ontologies it is possible to achieve this singular view. And integration requires mediation.

Querying

Slightly related to the browsing task is the concept of querying information. Querying can be applied if the user exactly knows what she is looking for. Through complex queries she expects to directly receive the correct information. Such a query is shown in the following example:

```
SELECT
   Author, Paper
FROM
   {Paper} <rdf:type> {<foo:Paper>};
           <foo:keyword> {"RDF", "Querying"};
           <dc:author> {Author}
```

Unfortunately, again as for browsing as well, there may be different representations of the same thing. Context of the query and context of the result may differ. To allow a query system to find the correct answer it is necessary to translate the original query into a format processable by the result repository. This process is commonly known as query rewriting. Afterwards, when a result has been found the result again has to be translated back into the format of the original context. To create these rewriting rule ontology mediation comes back into play.

From the scenarios described in this chapter the need for ontology mediation becomes clear. On the basis of these use cases and scenarios we will describe a general ontology mediation framework in the following.

Note that the set of use cases and scenarios described in this chapter is by no means complete. The practice of ontology mediation will have to show which scenarios present themselves in practice.

Chapter 3

Ontology Mediation Framework

Taking the use cases and the scenarios of the previous chapter as a motivation for ontology mediation, we develop a general ontology mediation framework in this chapter. We describe the creation, storage and usage of mappings, as well as merging of ontologies.

3.1 Mapping Creation

Re-interpreting a bit [3]: *"the result of a mapping process of two (or more) ontologies produces a set of mappings across ontologies which allow ontology coordination*[1]*; said differently, mappings are tools that may enable the "flow" of information across heterogeneous ontologies by describing the relations existing between entities of different ontologies*."

Three dimensions of heterogeneity at the conceptual level can affect the definitions of mappings between ontologies [3]:

- **Coverage:** mapped ontologies can cover different domains (with possible overlap or not). If there is no overlap, mapping is not possible because they represent disjoint domains. On the other hand, overlap between different domains can also drive to inconsistencies that should be considered during the exploitation process.
- **Granularity**: although several mapped ontologies can describe the same domain, they can use different level of detail in their descriptions. This situation can difficult very much the mapping process, since the reasonable approach is to equalize the different level of detail.

[1] **Ontology Coordination:** broadest term that applies whenever knowledge from two or more ontologies must be used at the same time in a meaningful way (e.g. to achieve a single goal) [3].

- **Perspective**: ontologies represent the point of view of its designers, so mapped ontologies can represent different point of views of the same domain. In this case, the pre-processing process should provide a way of "rotating" the perspective of an ontology, or to shift its viewpoint. For some forms of heterogeneity, this can be done systematically and in a relatively simple way (e.g. for indexical descriptions); however, in general the change of perspective is a very hard task for any ontology alignment method.

[3] characterizes the mapping process as the generation of a mapping specification, A, between two ontologies o and o'. Several parameters can enrich the definition of mapping process by including a set of external resources (r), restricting or parameterize the mapping process. Given that the mapping process can be considered a cyclic process where a continuous refinement can be applied to improve the mapping results, previous mapping specification can be considered as a part of the input of a mapping process.

Based in the definition of Alignment process that [3] provides, the mapping process can be defined as a function f which, from a pair of ontologies o and o' to align, an input alignment A, a set of parameters p, a set oracles and resources r, returns a new alignment A' between these ontologies: A' = f(o; o'; A; p; r).

In order to finish the characterization of the mapping process, we will adapt the proposal of [43] to divide the mapping process in five fundamental phases:

- **Pre-processing (Lift & Normalization)**. Ontologies are first imported in a uniform representation formalism that facilitates later operations. Then the vocabulary of the ontology is normalized by eliminating lexical and semantic differences (like special characters, upper case letters and acronyms). As a refinement step, [23] proposes a set of heuristics to reduce the number of candidate mapping pairs in order to gain in efficiency.

- **Discovery (Similarity).** In this step, similarities between ontology entities are calculated. [43] proposes the use of a combination of different matchers that improve the result of the identification of equivalent terms. [61] include an extensive survey of different techniques.

- **Specification (Semantic Bridging)**. After having defined the similarities between entities in the different ontologies, a mapping needs to be specified between the similar entities of the ontologies. This specification is usually a manual process, but it can be aided by a tool. PROMPT [52], for example, comes up with concrete proposals for merge operations, so that for many operations the user only needs to say "execute", instead of having to specify the complete operation. In many cases (e.g. PROMPT), there is a feedback loop from this phase to the previous phase. Typically, the tool can offer more precise similarity measures when the user has already specified part of the mapping. Many matching algorithms do not include this feedback loop. However, these algorithms can often be readily applied

in an overall algorithm which executes the match algorithm in each iteration in the process.

- **Exploitation (Execution)**. When the mappings are specified, the next step is to use the ontologies and their mapping in a meaningful way. [43] focused this phase in instance transformation from the source ontology representation into the representation of the target ontology by evaluating the transformation functions associated with the mapping defined in the previous stage.

- **Post-processing**. Based in the exploitation results, the mapping specification is again analyzed to improve the quality of the exploitation tasks. For instance, the identification of inconsistencies as a result of the mapping process can be tackled by using techniques for repairing inconsistencies or the selection of reasoning techniques based in paraconsistent approaches that can provide meaningful answers from inconsistencies sources.

3.2 Merging Ontologies

Ontology Merging can be seen conceptually as taking the union of two ontologies, as in Wiederhold's ontology algebra [69], resolving the overlap between the ontologies. This can be written down as:

$$C = A \cup B \tag{3.1}$$

Where C is the resulting ontology from the merge of ontologies A and B.

Notice that C is not literally the union of A and B. We assume that in the merged ontology, all overlap in concepts, relations, instances and axioms has been resolved. Ideally, a new concept C would be created for each pair of concepts from A and B that overlap. Similarly for relations, instances and axiom.

Thus, for ontology merging we envision an approach as in PROMPT [52]. PROMPT takes the two source ontologies as input and supports the user in creating a new target ontology, based on the source ontologies. PROMPT detects overlap in classes in the source ontologies and suggests new classes to be created in the target ontology, based on this overlap. Classes in the source ontologies that do not overlap are typically copied directly from the source to the target ontology.

Ontology Mappings in Ontology Merging When performing ontology in an interactive process as in PROMPT [52], mapping do not really seem to fit in, because classes and relations in the target ontology are created directly, based on the source ontologies. However, during the merging process, overlap between classes and relations in the source ontologies is detected. In ontology mapping, this detected overlap is used to specify

ontology mappings. In ontology merging, this detected overlap is used to create new classes/relations in the target ontology.

In ontology merging as in PROMPT the detected overlap is not made explicit. Instead, it is implicit in the newly created entities in the target ontology. Therefore, it seems advantageous to create an ontology mapping first and do the ontology merging based on this mapping. If the ontology mapping has been specified, the relationships between the ontologies have been made explicit and this can then be used to aid in creating the target ontology. Another advantage of using an ontology mapping as a basis of the ontology merging, is that algorithms and tools that have been developed to aid in the development of ontology mappings can be reused for the ontology merging problem.

3.3 Storage

Mapping and merged ontologies need to be stored. Important aspect is the relationship with the overall ontology management.

Storage of merged ontologies is equivalent to the storage of any other ontology and thus we will not pay attention to this aspect here. We do pay additional attention to the problem of ontology storage. To our knowledge this problem of storage and retrieval of ontology mappings has not yet been addressed explicitly in the literature, although when viewing a mapping as a set of instances of a mapping ontology (e.g. MAFRA [43]), a regular ontology store can be used for storing mappings.

An ontology mapping repository should implement methods for the basic storage and retrieval of ontology mappings. It should be possible to store any ontology mapping and to retrieve an ontology mapping based on the identifier of the mapping. Furthermore, it should also be possible to query for ontology mappings based on the source and/or target ontology. In other words, if the user knows the ontologies he/she wants to map and there already exists a mapping, the user must be able to find the existing mapping.

Besides storing and retrieving, a mapping repository should also support basic versioning of ontology mappings. Users might want to use specific versions of an ontology mapping and also, a specific version of a mapping would map between specific versions of the source and target ontologies.

3.4 Run-time Mediation

Mappings between ontologies are established in order to solve a particular problem in interoperability between ontologies. The question now is: How to use the mappings? What good are they?

The most important use case for ontology mapping throughout this deliverable was

querying. A mapping between ontologies enables querying one ontology in terms of the other. This type of querying need to be supported by the mediation component.

Such querying can be achieved in two principled ways: (1) by loading the source and target ontologies, together with the mapping rules, in the reasoner and then posing queries and (2) by rewriting queries in terms of the target ontology to queries in terms of the source ontology and then querying the source knowledge base, after which the query answers must be transformed to the target ontology.

Both ways have advantages and disadvantages. In case all ontologies along with the mapping rules are loaded in the reasoner, one can pose simple queries and immediately retrieve the answers in terms of the target ontology. The additional steps of rewriting the query and transforming the answers are not required. Disadvantage is that the reasoner must have access to the instance store which corresponds with the source ontology. Such an instance store would typically be a relational database and thus the reasoner must be aware how to translate queries on the ontology concepts to queries in the relational database and have access to the database to execute the queries.

In the second case, the additional steps of query rewriting and transformation of the query results are required. Especially query rewriting is a very complicated and costly task. This scenario is appropriate in case the source knowledge base exposes only a simple query interface and there is no access to the instance store.

In this chapter we have described a general ontology mediation framework. In the remainder of this deliverable we will focus on the mapping creation phase, where we distinguish two distinct activities: automatic discovery of mappings and refinement and additional specification of mappings. We describe a method for the automatic discovery of mappings, called APFEL, and we describe a mapping language for the mapping of OWL ontologies.

Chapter 4

Ontology Mediation Preliminaries

In this chapter we describe a number of consideration relevant for ontology mediation. In particular, we consider here ontology languages, mapping languages, ontologies, mapping patterns, mapping, instance bases and an ontology mapping tool.

4.1 Ontology Language

An essential part of the ontology mediation framework is the language which is used to specify the ontologies. This ontology language restricts the way mappings between ontologies can be expressed. We identify the following requirements and critical components for a general mediation framework concerning the underlying ontology languages.

4.1.1 Expressivity and Language Mismatches

In the simpler case we assume that we only want to establish mappings between ontologies in the same ontology language.

In a more complicated scenario you might also want to map between languages of different expressivity. Here, when mapping between a language with richer language features to a language with less expressivity information loss might be inevitable. The other way around should be less problematic. Even worse there might be semantic differences with languages which only overlap semantically, i.e. either language has some features not expressible in the other and vice versa.

When mapping or merging ontologies in different ontology languages we might encounter two general scenarios. Either (i) only mappings between the fragments of the ontologies where the semantics overlaps are allowed or (ii) a merged ontology is defined in terms of an ontology language which provides a superset of the respective underlying ontology languages. In other words, we require in a mediation framework dealing with

different ontologies the definition of the (i) intersection of different ontology languages and (ii) a minimal unifying ontology language covering and unifying all expressive features of the underlying ontology languages.

4.1.2 Combinability with Rules

Mappings are typically expressed by some form of rules. Here, logic programming style rules, offering the expressivity of a powerful query language are often the choice. There is a whole bunch of literature on combining rule languages with ontology languages (cf. [40, 21, 29, 50, 25]. Note that a mapping language might need much more features than expressible in simple Horn rules. While Horn-rules are sufficient to express conjunctive queries on Relational Databases for several advanced mapping additional features such as an exhaustive set of pre-defined functions, aggregations, negation, etc. might be needed.

4.1.3 Scalability and Decidability

Expressivity of an ontology language usually comes at cost. This is also a reason why certain ontology languages restrict expressivity in order to keep computational cost of deciding certain reasoning tasks low. Therefore most ontology languages are layered. Depending on your application, the user might decide which expressivity is needed for the ontology at hand.

In most of the approaches to combine rules with ontology languages decidability issues are the major concern. For instance, inferencing in Ontologies expressed in Description Logics style might become undecidable when rules are simply added to the language, an example of such an undecidable combination of rules with an ontology languages is the rule language SWRL [31] as an extension of the Web ontology language OWL (more specifically its OWL DL fragment).

On the contrary, certain inferences in the ontology language itself might already be undecidable which is for instance the case for OWL Full. So, in order to have a decidable language, one has to find a reasonable tradeoff between expressivity of the mapping language and the ontology language.

4.1.4 Grounding Mappings to Different Ontology Language

We focus in this deliverable on mappings between Ontologies in OWL and the WSML ontology languages. This indicates that we require for our ontology mapping language a clear specification of the semantics of mapping between ontologies in either language. Since we want to keep the definition of mappings as general as possible, we need to specify for each pair of ontology languages A, B and elementary mapping pattern m:

1. Whether the respective mapping m might be used to map from A to B.
2. What the semantics of m in terms of source ontology language A and target ontology language B is.

4.1.5 Tool support

The requirements on the tool support for the ontology language depend on what kind of ontology languages will be supported (both syntaxes and semantics) and what kind of internal representation is used for the actual mediation. Depending on these requirements, several translators between different representation languages might be required. Given a source ontology O_A in a specific ontology language A and a target ontology language B the tools should allow for identification of the respective subset of O_A which can be mapped to ontology language B and/or which parts of O_A are incompatible with the expressivity of language B. The tools should only allow for the definition of compatible mappings with respect to the source ontology languages as identified in Section 4.1.4

4.2 Mapping Language

The mapping language itself is used to specify the actual mappings between the ontologies. This mapping language interacts with the patterns in order to enable the reuse of patterns and in order to make the mappings more easily understandable.

We use the language-independent mapping language specified in SEKT deliverable D4.3.1 [9] (see also Appendix A). As can be seen from D4.3.1, this language is highly related with the elementary mapping patterns described in that deliverable.

Tool support A reasoner is required that can deal with the mapping language. The kind of reasoner which is required depends on the actual grounding of the mapping language. In Chapter 6 we ground the mapping language to OWL DL. Such a grounding requires a Description Logic reasoner in order to work with the mappings. In the case of mapping between Logic Programming-based grounding, such as grounding to WSML-Flight [57], a Logic Programming or deductive database engine is required for reasoning.

In the case of grounding to SWRL (which is a straightforward extension of the grounding to OWL DL) full first-order theorem prover is required.

4.3 Ontologies

Ontologies are the actual representations we want to mediate between and therefore form an integral part of the mediation framework. In order to create a mapping or in order to

discover mappings, the source and target ontologies need to be imported in the respective tools. For many run-time tasks, reasoning with the source and target ontologies is also required.

Certain types of mappings can only be applied to certain (kinds of) ontologies. It is also to be expected that there will be many mappings between ontologies within a particular domain, whereas between domains, there will not be many mappings between ontologies. Also because there is not much overlap between the different domains.

Tool support Mechanisms need to be in place to access ontologies, both at the design-time, when the mapping is created, and at run-time when the actual mediation is performed. We assume here that there are already ontology management facilities in place and we do not see this to be within the scope of the ontology mediation effort.

4.4 Mapping Patterns

The mapping patterns capture recurring patterns found in ontology mappings. These patterns are defined in D4.3.1 and used in the mediation framework.

Tool support Mapping patterns need to be stored and retrieved from the patterns library. This patterns library is described in more detail in D4.3.1.

4.5 Mappings

The mappings themselves are used to enable mediation between different representations. Important aspects related to the mappings are the creation of mappings using specific tools, as well as the representation of mappings themselves through the mapping language.

One important aspect to keep in mind is that there might be several versions of an ontology mapping, along with several versions of the source and target ontologies. Often a change in either the source or target ontology would require a change in the ontology mapping.

Tool support Mappings need to be stored and retrieved. It might be possible to store them alongside the ontologies.

4.6 Instance Bases

Instance Stores are used to store large sets of instances (*instance bases*) related to ontologies. Connectors are used to import sets of instances into the mediation framework in order to do achieve certain tasks.

Essentially, these data source connectors are not specific for the mediation framework. In fact, a Semantic Web application, which uses a large data source, would use such a data source connector or *wrapper*.

4.7 Mapping Tool

The mapping tool is used to create the actual mappings and perform the actual merging of ontologies and heavily relies on user interaction. Therefore, the GUI presented by the tool is an essential part of the tool.

Another important part of the tool is the connection to a matching algorithm, which can be used to partially automate the detection of similarities between ontologies. These similarities are used to assist the user in creating mappings or performing the merging of ontologies.

Chapter 5

Mapping Discovery

The first phase in the ontology mediation framework described in the previous chapter is mapping creation. Manually creating mappings between ontologies is a very tedious, time-consuming and error-prone task. We therefore propose to create mappings in an automated fashion. This chapter of the deliverable provides new insights on full-automatic methods for mapping discovery.

In recent years different methods for automatic ontology mapping have been proposed to deal with this challenge (Figure 5.1). Thereby, the proposed methods were constricted to one of two different paradigms: Either, *(i)*, proposals would include a manually predefined automatic method for proposing mappings, which would be used in the actual mapping process (cf. [23, 26, 53]). They typically consist of a number of substrategies such as finding similar labels. Or, *(ii)*, proposals would learn an automatic mapping method based on instance representations, e.g. bag-of-word models of documents (cf. [1, 18]). Both paradigms suffer from drawbacks. The first paradigm suffers from the problem that

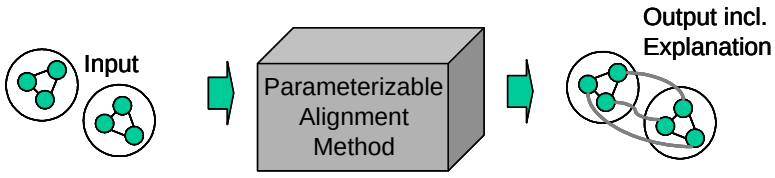

Figure 5.1: Ontology Mapping

it is impossible, even for an expert knowledge engineer, to predict what strategy of mapping entities is most successful for a given pair of ontologies. Furthermore, it is rather difficult to combine the multiple different substrategies to behave optimally. This is especially the case with increasing complexity of ontology languages or increasing amounts of domain specific conventions. The second paradigm is often hurt by the lack of instances or instance descriptions, because not in every case an ontology has many instances and in many cases instances exist only for some part of the ontology. Also, knowledge encoded

in the intensional descriptions of concepts and relations is only marginally exploited by this way.

Hence, there remains the need to automatically combine multiple diverse and complementary mapping strategies of *all* indicators, i.e. extensional *and* intensional descriptions, in order to produce comprehensive, effective and efficient semi-automatic mapping methods. Such methods need to be flexible to cope with different strategies for various application scenarios, e.g. by using parameters. We call them "PArameterizable Mapping Methods" (PAMM).

5.1 Bootstrapping with APFEL

In the course of SEKT, we have developed a bootstrapping approach for acquiring the parameters that drive such a PAMM. We call our approach APFEL for "Alignment Process Feature Estimation and Learning".

APFEL is based on four major considerations. First, at the level of *executing the mapping method*, APFEL is based on the general observation that mapping methods like QOM [23] or PROMPT [53] may be mapped onto a generic mapping process. Major steps of this generic process include:

1. Feature Engineering, i.e. select small excerpts of the overall ontology definition to describe a specific entity (e.g., the LABEL to describe the concept O1:DAIMLER).
2. Search Step Selection, i.e. choose two entities from the two ontologies to compare (e.g., O1:DAIMLER and O2:MERCEDES).
3. Similarity Assessment, i.e. indicate a similarity for a given description of two entities (e.g., $\text{simil}_{\text{label}}$(O1:DAIMLER,O2:MERCEDES)=0).
4. Similarity Aggregation, i.e. aggregate multiple similarity assessment for one pair of entities into a single measure (e.g., simil(O1:DAIMLER,O2:MERCEDES)=0.5).
5. Interpretation, i.e. use all aggregated numbers, some threshold and some interpretation strategy to propose the equality for the selected entity pairs (map(O1:DAIMLER)='$\perp$').
6. Iteration, i.e. as the similarity of one entity pair influences the similarity of neighboring entity pairs, the equality is propagated through the ontologies (e.g., it may lead to a new simil(O1:DAIMLER,O2:MERCEDES)=0.85, subsequently resulting in map(O1:DAIMLER)=O2:MERCEDES).

Second, at the meta level of *representing a mapping method*, APFEL parameterizes each of these steps by maintaining a declarative representation of features engineered Q_F,

similarity assessments Q_S for the features, a weighting scheme Q_W for such similarity assessments and a threshold Q_T to feed into the interpretation strategy (see Section 5.3.1).[1]

Third, such a declarative representation, e.g. of QOM or PROMPT, can be given to a *parameterizable mapping method*, PAMM, for execution such as indicated in Figure 5.1. In fact, we initialize PAMM with the representation of a QOM-like strategy, PAMM(QOM), before some initial mappings of two given ontologies are generated through it. The mappings are then handed over to the user for validation (cf. Section 5.3.2).

Fourth, APFEL generates hypotheses of useful features H_F for a domain-specific pair of ontologies and proposes similarity assessments H_S for these hypotheses (cf. Section 5.3.3). APFEL uses the validated initial mappings for machine learning the weighting scheme. The aggregation scheme recurs to all feature/similarity combinations under consideration, which are represented by $D_F := Q_F \cup H_F$ and $D_S := Q_S \cup H_S$. Finally, it outputs the weighting scheme D_W and the threshold it has learned D_T (cf. Section 5.3.4).

The APFEL process is summarized in Figure 5.3 and will be explained in detail in Section 5.3. The result of APFEL is a representation of an mapping scheme. The scheme then has been optimized by machine learning to consider the indicators initially used for bootstrapping as well as the newly generated domain/ontology-specific indicators. Thus, it may integrate indicators working at the level of intensional *and* extensional ontology descriptions to result in a comprehensive improved mapping method (cf. Section 5.4).

5.2 General Mapping Process

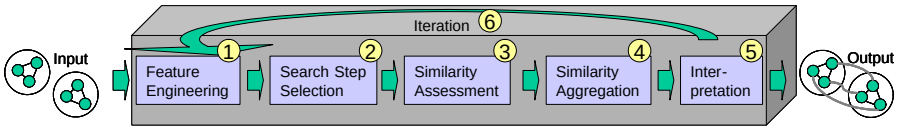

Figure 5.2: General Mapping Process in PAMM

We briefly introduce our definition of the generic mapping process that subsumes all the mapping approaches we are aware of. It has previously been presented in [23, 24]. Here, we only focus on its definition to the extent that is necessary to understand how APFEL operates on the steps of the generic process. Figure 5.2 illustrates the six main steps of the generic mapping process. As input, two ontologies are given which are to be mapped. The steps are illustrated through examples where necessary.

1. Feature engineering selects only parts of an ontology definition in order to describe

[1]Unlike done in QOM [23], we do not vary the search step selection, as QOM was about the trade-off between efficiency and effectiveness and in this paper we focus on effectiveness alone. Further, we do not vary the interpretation and iteration strategies to limit the exploration space.

a specific entity. Implicitly, [26] made a similar observation. For instance, mapping of entities may be based only on a subset of all RDFS primitives in the ontology. A feature may be as simple as the label of an entity, or it may include intensional structural descriptions such as super- or sub-concepts for concepts, or domain and range for relations. Instance features may be instantiated attributes. Further, we use extensional descriptions.

```
<rdf:Description rdf:about=‘‘o1:Daimler’’>
    <rdf:type rdf:resource=‘‘auto:automobile’’>
    <rdf:type rdf:resource=‘‘auto:luxury’’>
    <auto:speed rdf:resource=‘‘fast’’>
</rdf:Description>
```

Example 1: Fragment of First Example Ontology.

```
<rdf:Description rdf:about=‘‘o2:Mercedes’’>
    <rdf:type rdf:resource=‘‘auto:automobile’’>
    <auto:speed rdf:resource=‘‘fast’’>
</rdf:Description>
```

Example 2: Fragment of Second Example Ontology.

In our Examples 1 and 2 we have fragments of two different ontologies, one describing the instance DAIMLER and one describing MERCEDES. Both O1:DAIMLER and O2:MERCEDES have a generic feature called TYPE. The values of this feature are *(i)*, AUTOMOBILE and LUXURY, and, *(ii)*, AUTOMOBILE, respectively. As stated before also domain ontology-specific features are included through this step e.g. SPEED, the value of which is FAST for both the first and second entity.

The reader may note that another generic feature in our running example could be ISINPROJECTIONOFRELATION. For O1:DAIMLER and O1:BEETLE and O1:PICKUP the corresponding feature-value pair would be ISINPROJECTIONOFRELATION SPEED. As one may imagine in this context, the domain ontology-specific feature-value pair (SPEED FAST) will be more important to correctly and only map O1:DAIMLER and O2:MERCEDES than the generic (ISINPROJECTIONOFRELATION SPEED), which would not have the domain knowledge that both ontologies use FAST for characterization.

Nevertheless manually determining the correct features for ontology mapping is a difficult process. In the next section we will present an approach using machine learning to identify the most important features.

2. Selection of Next Search Steps. The derivation of ontology mappings takes place in a search space of candidate pairs. This step may choose to compute the similarity of a restricted subset of candidate concepts pairs $\{(e, f)|e \in \mathcal{E}_{O_1}, f \in \mathcal{E}_{O_2}\}$ and to ignore others. For the running example we simply select every possible entity pair as an mapping candidate. In our example this means we will continue the comparison of O1:DAIMLER and O2:MERCEDES.

3. Similarity Assessment determines similarity values of candidate pairs. We need inexact heuristic ways for comparing objects i.e. similarity functions such as on strings [38], object sets [6], checks for inclusion or inequality, rather than exact logical identity. The result lies within a range between 0 and 1. Returning back to our example we use a similarity function based on the instantiated results, i.e. we check whether the two concept sets, parent concepts of O1:DAIMLER (AUTOMOBILE and LUXURY) and parent concepts of O2:MERCEDES (only AUTOMOBILE), are the same. In the given case this is true to a certain degree, effectively returning a similarity value of 0.5. The corresponding feature/similarity assessment (FS1) is represented in Table 5.1. For APFEL we refer to them as Q_F/Q_S assessments.

***FS1**: if parent concepts are the same, the instances are also the same to a certain degree*

Comparing	No.	Feature Q_F	Similarity Q_S
Instances	FS1	(parent,X_1)	set equality(X_1, X_2)

Table 5.1: Informal and Formal Feature/Similarity Assessment

4. Similarity Aggregation. In general, there may be several similarity values for a candidate pair of entities (e, f) from two ontologies O_1, O_2, e.g. one for the similarity of their labels and one for the similarity of their relationship to other terms. These different similarity values for one candidate pair must be aggregated into a single aggregated similarity value. This may be achieved through a simple averaging step, but also through complex aggregation functions using weighting schemes Q_W. For the example this leads to: simil(O1:DAIMLER,O2:MERCEDES)=0.5.

5. Interpretation uses the aggregated similarity values to map entities from O_1 and O_2. Some mechanisms here are e.g. to use thresholds Q_T for similarity [53], to perform relaxation labelling [18], or to combine structural and similarity criteria. simil(O1:DAIMLER,O2:MERCEDES)$\geq$0.5 leads to map(O1:DAIMLER)=O2:MERCEDES.

6. Iteration. Several algorithms perform an iteration (see also similarity flooding [46]) over the whole process in order to bootstrap the amount of structural knowledge. Iteration may stop when no new mappings are proposed, or if a predefined number of iterations has been reached. Note that in a subsequent iteration one or several of steps 1 through 5 may be skipped, because all features might already be available in the appropriate format or because some similarity computation might only be required in the first round. We use the intermediate results of step 5 and feed them again into the process and stop after a predefined number of iterations.

5.3 Supervised Learning of an Ontology Mapping Process

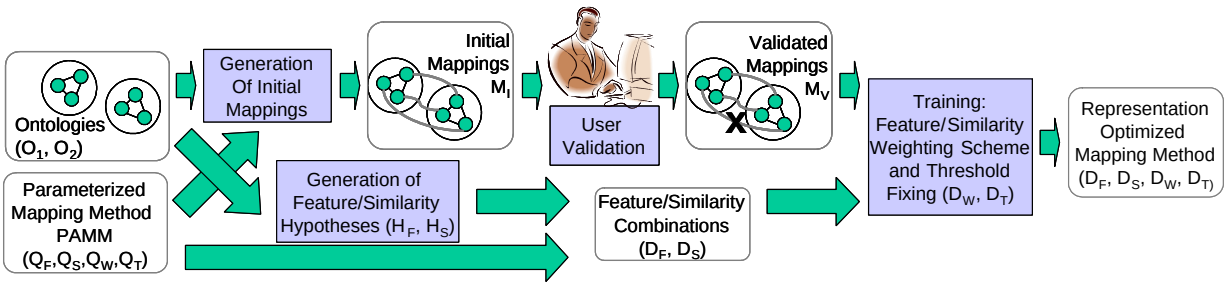

Figure 5.3: Detailed Process in APFEL

In this section the APFEL process is explained in detail following the Figure 5.3. Data structures are illustrated through white boxes and process steps through colored boxes. We will describe first the data structures, then the process steps. Finally, we describe how the result of APFEL is applied. Some steps are still work in progress and research will be continued in the scope of SEKT.

5.3.1 Data Structures

We here describe the data structures on which APFEL operates. APFEL requires two ontologies O_1 and O_2 as inputs to its processing. Either these are the ontologies for which the further mapping process will be optimized directly. Or, they exemplarily represent a type or domain which requires an optimized mapping method.

Core to APFEL is the representation of the generic mapping process. Relevant data structures for representation include:

(*i*) Q_F: features engineered (e.g. label, instances, domain), (*ii*) Q_S: similarity assessments corresponding to the features of Q_F (e.g. equality, subsumption), (*iii*) Q_W: weighting scheme for an aggregation of feature-similarity assessments (e.g. weighted averaging), and (*iv*) Q_T: interpretation strategy (e.g. mappings occur if similarity is above the fixed threshold).

Such a declarative representation can be given to a parameterizable mapping method, PAMM, for execution. In fact, we can initialize PAMM with a representation of different strategies. Thus, an initial mapping function, map_{init}, may be defined by map_{init}:=PAMM(PROMPT) or map_{init}:=PAMM(QOM).

Then, APFEL uses user validations M_V of the initial proposals of map_{init}.

In general, the described input does not explicitly require a knowledge engineer. The two ontologies, an arbitrary (predefined) mapping method, and the validation of the initial

mappings may be processed by a typical user as well.

The output of APFEL is an improved mapping method, $\text{map}_{\text{learn}}$, defined as $\text{map}_{\text{learn}}$:=PAMM(APFEL($O_1, O_2, Q_F, Q_S, Q_W, Q_T, M_V$)). The parameters that characterize APFEL($O_1, O_2, Q_F, Q_S, Q_W, Q_T, M_V$) constitute the tuple ($D_F$, D_S, D_W, D_T).

The reader may note that $\text{map}_{\text{learn}}$ and map_{init} both are functions, such that the result of map_{init}(o1:DAIMLER, O_1, O_2) might be '$\perp$' and differ from the result of $\text{map}_{\text{learn}}$(o1:DAIMLER, O_1, O_2) = o2:MERCEDES.

5.3.2 Generation and Validation of Initial Mappings

Machine learning as used in this paper requires training examples. The assistance in their creation is necessary as in a typical ontology mapping setting there are only a small number of really plausible mappings available compared to the large number of candidates, which might be possible a priori. Presenting every candidate for validation makes the process tiring and inefficient for the human user.

Therefore, we use an existing parametrization as input to the Parameterizable mapping Method, e.g. map_{init}=PAMM(QOM) to create the initial mappings A_I for the given ontologies O_1 and O_2. As these results are only preliminary, PAMM does not have to use very sophisticated processes: very basic features and similarities (e.g. label similarity) combined with a naïve simple averaging and fixed threshold are sufficient in most cases. Resulting pairs are stored starting with the highest probability mappings first as shown in Table 5.2.

Entity 1	Entity 2	User Mark
car	car	to be rated
auto	automobile	to be rated
wheel	tire	to be rated
speed	hasSpeed	to be rated
driver	gear	to be rated

Table 5.2: Initial Mappings Returned for Validation

This allows the user to easily validate the initial mappings and thus generate correct training data A_V. If the user further knows additional mappings he can add these mappings to the validated list. Entity pairs not marked by the user are by default treated as disjunct entities. Obviously the quality of the later machine learning step depends on the quality and quantity of the validated mappings done at this point.

5.3.3 Generation of Feature/Similarity Hypotheses

As mentioned in the introduction it becomes difficult for the human user to decide which features and similarity heuristics make sense in indicating an mapping of two entities. Our approach therefore generates these feature/similarity combinations automatically.

The basis of the feature/similarity combinations is given by an arbitrary mapping method such as PAMM(QOM) with which we have achieved good results (see [33]).

Further, from the two given ontologies APFEL extracts additional features H_F by examining the ontologies for overlapping features. Those being part of both ontologies are added. At this point domain-specific features are integrated into the mapping process. These features are combined in a combinatorial way with a generic set of predefined similarity assessments including similarity measures for, e.g., equality, string similarity, or set inclusion. Thus, APFEL derives similarity assessments H_S for features H_F.

$$\left\{ \begin{array}{l} extras \\ licensenumber \end{array} \right\} \times \left\{ \begin{array}{l} equality \\ inclusion \end{array} \right\} \Rightarrow$$

Comparing	No.	Feature H_F	Similarity H_S
Cars	FS1	(extras,X_1)	set equality(X_1, X_2)
Cars	FS2	(extras,X_1)	subset(X_1, X_2)
Cars	FS3	(license no.,X_1)	equality(X_1, X_2)
Cars	FS4	(license no.,X_1)	substring(X_1, X_2)

Figure 5.4: Generation of Additional Hypotheses

Figure 5.4 illustrates this process for generating hypotheses for feature/similarity combinations. In the given example two domain attributes EXTRAS and LICENSE NUMBER are compared using the EQUALITY and the INCLUSION heuristic. All feature/similarity combinations are added for now. Some feature/similarity combinations will not be very useful, e.g. FS4, comparing whether one license number is a substring of another license number. However, in the subsequent training step machine learning will be used to pick out those which improve mapping results.

From the feature/similarity combinations of (Q_F, Q_S) and of the extracted hypotheses (H_F, H_S) we derive an extended collection of feature/similarity combinations (D_F, D_S) with $D_F := Q_F \cup H_F$ and $D_S := Q_S \cup H_S$.

5.3.4 Training

After determining the classification of two entities of being mapped or not (A_V), all validated mapping pairs are processed with the previously automatically generated collection of features and similarities. From each set a numerical value is returned which is saved together with the entity pair as shown in Table 5.3.

Entity1	Entity2	Mark	FS1	FS2	FS3	FS4
car	car	1	1.0	1.0	0.8	0.0
auto	automobile	1	0.7	1.0	0.7	0.0
wheel	tire	0	0.0	1.0	0.8	0.0
speed	hasSpeed	1	0.7	0.0	0.0	1.0
driver	gear	0	0.2	0.0	0.0	0.0

Table 5.3: Training Data for Machine Learning (including user validation and value returned by each feature/similarity combination FS_i)

We can now apply machine learning algorithms to the automatically generated features D_F and similarities D_S using the example training mappings M_V. More specifically, the numerical values of all feature/similarity combinations are the input for the algorithm. The classification of being mapped or not marks the output. Different machine learning techniques for classification (e.g. decision tree learner, neural networks, or support vector machines) assign an optimal internal weighting D_W and threshold D_T scheme. Machine learning methods like C4.5 (J4.8 in Weka) capture relevance values for feature/similarity combinations. Feature/similarity combinations which do not have any (or only marginal) relevance values for the mapping are given a weight of zero and can thus be omitted.

To give the reader an intuition of the results of this training step we refer to Example 3, a decision tree. Depending on the output of each individual rule we traverse the tree and reach a leaf indicating either that two entities are mapped (`1`) or not (`0`).

```
rule0 <= 0.5
 |   rule4 <= 0
 |   |   rule3 <= 0.196429: 0 (149.0/3.0)
 |   |   rule3 > 0.196429
 |   |   |   rule5 > 0.964286
 |   |   |   |   rule2 <= 0.267857: 1 (22.0)
 |   |   |   |   rule2 > 0.267857: 0 (10.0)
 |   |   |   rule5 <= 0.967612: 1 (50.0/12.0)
 |   rule4 > 0: 1 (10.0/4.0)
 rule0 > 0.5: 1 (32.0/1.0)
```

Example 3: Decision Tree Output.

From this we finally receive the most important feature/similarity combinations (features D_F and similarity D_S) and the weighting D_W and threshold D_T thereof. With this we can set up the final ontology mapping method which we call map_{learn}:=PAMM(APFEL($O_1, O_2, Q_F, Q_S, Q_W, Q_T, M_V$)). Depending on the complexity of the mapping problem it might be necessary to repeat the step of test data generation (based on the improved mapping method) and training.

From first evaluation results one can derive that decision tree learning yields the best

results, although one has to be aware that the current evaluation is only preliminary and based on a restricted set of ontologies. Apart from that, a typical advantage of decision trees is that they may be understood by users. The features and similarities can be transformed into natural language and serve as an explanation for the automatically found mappings: "If labels are similar to a degree of 0.5 or more, the involved entities are mapped." Thus, the black box of ontology mapping becomes more transparent for users.

As with every machine learning approach the results are dependent on the number of training examples. Simple feature/similarity combinations such as labels/syntactic similarity can probably be learned with as little as ten examples. More complex combinations based on less frequent features, will only be included in the result with larger numbers of training examples. An upcoming thorough evaluation of our approach will provide the correct numbers soon.

5.4 Application

The final system is parameterized with D_F, D_S, D_W, and D_T. It allows for fully or semi-automatic mapping of the two ontologies — and further uses domain-specific optimization of the mapping system. Depending on the weighting and threshold scheme this may also include an explanation facility which provides evidence why two entities are mapped.

The presented approach is currently being implemented in Java using the capabilities of the KAON2-framework (based on [34]), which can handle OWL-DL ontologies. The optimized (learned) mapping method $\text{map}_{\text{learn}}$ will be applied in the mediation framework presented in this deliverable. We expect to provide a high quality automatic ontology mapping approach for SEKT.

5.5 Related Work

The tools PROMPT and AnchorPROMPT [53] use labels and to a certain extent the structure of ontologies. However, their focus lies on ontology merging, i.e. how to create one ontology out of two. Potential matches are presented to the user for confirmation, thus making it a semi-automatic tool. In their tool ONION [48] the authors use rules and inferencing to execute mappings, but the inferencing is based on manually assigned mappings or heuristics simpler than PROMPT. Besides equality first steps are taken in the direction of complex matches. These could also include concatenation of two fields such as "first name" and "last name" to "name"[17]. [4] further present an approach for semantic mapping based on SAT-solvers. In their approach an mapping can only be created if there are no inherent semantic rules restricting this. In [44] the authors present a practical approach and a manual tool to map ontologies.

[18] use machine learning in their approach GLUE. However, their learning is restricted on concept classifiers for instances based on instance descriptions, i.e. the content of web pages. From the learned classifiers they derive whether concepts in two schemas correspond to each other. Additional relaxation labelling is based solely on manually encoded predefined rules. Nevertheless, from all ontology mapping approaches their work is closest to APFEL. In [19] the same authors introduce the notion of the use of domain specific attributes, thus restricting their work on databases.

In this chapter we have described a way to automatically discover mappings between ontologies. However, such mappings are not guaranteed to be correct or complete. Thus, the domain expert needs to have the opportunity to refine the discovered mappings and to introduce new mappings which have not been discovered before. Such mappings, as well as the output of the discovery process, need to be specified using an ontology mapping language. In the next chapter we describe such a language for the mapping of OWL DL ontologies.

Chapter 6

Ontology Mapping Language

The output of the mapping creation phase in the ontology mediation framework is a specification of the ontology mappings. Furthermore, the outcome of the mapping discovery (in the mapping creation phase) is also a specification of ontology mappings. In order to adequately specify such mappings we describe an ontology mapping language in this chapter.

This chapter will analyze the requirements on the mapping language as well as the elementary mapping patterns in D4.3.1 [9] as a starting point and define the ontology mapping language. After that we describe the syntax, semantics and pragmatics of the language.

6.1 Requirements on the Ontology Mapping Language

This section shall present a number of requirements on the Ontology Mapping Specification Language, partly derived from the use cases and scenarios presented in Chapter 2 and the ontology mismatches which surveyed [35]. Furthermore, we present a number of additional requirements that follow from our setting.

Particularly, we identify the following requirements on an Ontology Mapping Specification Language:

Mapping on the Semantic Web Our goal is to develop an ontology mapping language for the Semantic Web. Therefore, we must be able to specify mappings between ontologies on the Web and the ontology mapping itself must also be available on the Web. The current standard for specifying ontologies on the web is the Web Ontology Language OWL [14]. We must therefore support mapping between ontologies written in OWL. An important species of OWL is OWL DL, which is a syntactical variant of the $\mathcal{SHOIN}(\mathbf{D})$ Description Logic language [30]. Therefore, mappings between OWL DL ontologies can be reduced to mappings between

Description Logic ontologies.

Specify Instance Transformations It follows from the generic use cases presented in the previous section that the ontology mapping language must support transformations between instances of the different ontologies. In fact, [58] defines the mapping process as the set of activities required to transform instances of the source ontology into instances of the target ontology. Also MAFRA [43] explicitly addresses the transformation of instances on the basis of a mapping between two ontologies.

In instance transformation, we identify two dimensions: structural transformation and value transformation:

- A *structural* transformation is a change in the structure of an instance. This means that an instance might be split into two instances, two instances might be merged into one, properties might be transformed to instances, etc. For example, an instance of the concept PhD-Student in one ontology might need to be split into two instances, one of Student and one of Researcher, in the target ontology. A different example is the use of aggregate functions. An ontology O_S might have a concept Parent with a property hasChild, whereas the ontology O_T might also have a class Parent, but in this case only with the property nrOfChildren. An aggregate function is required to count the number of children of a specific parent in O_S in order to come up with a suitable property filler for nrOfChildren.
- A *value* transformation is a simple transformation from one value into another. An example of such a value transformation is the transformation from kilograms into pounds

An example of a transformation, which requires both a structural and two value transformations is the transformation from a full name to separate first and last names. Splitting the full name instance into both the first and the last name requires structural transformation. After the structural transformation, two value transformations are required; on for the first and one for the last name.

Specify Instance Similarity Conditions One of the generic use cases presented in Chapter 2 is the instance unification use case. It turned out in this use case that when instances need to be unified, first the similarity between the instances must be established. In order to detect the similarity, one can compare the values of all properties and describe the similarity as a function over all the individual property similarities. The other extreme is to designate one property as the identifying property (cf. primary keys in relational databases) and designate instances that have equivalent values for these designated properties as equivalent and unify them[1].

[1]Note that, as with primary keys in relational databases, it is possible to designate several properties as the unique identification for instances

We shall take a hybrid approach, where it is possible to specify equality of instances as a logical condition over its property values. We call this the *exact* approach for instance unification. In the second case, the *probabilistic* approach, it is possible to specify a matching function over the property values, which yields a probability between 0 and 1, specifying the similarity between the instances. When combined with a threshold, this function also becomes a condition for similarity.

We are currently not aware of any existing approach which addresses the specification of instance similarity in the same sense we do here.

Query Rewriting and Ontology Merging The Query Rewriting and Ontology Merging use cases presented in the previous chapter indicate the need for the ontology mapping to not only map instances of the ontologies but to also map concepts and relations in the source and target ontologies. This is necessary for the case when a query written in terms of ontology $\mathcal{O}_S$ needs to be executed on an instance base, which is described by ontology $\mathcal{O}_T$. The mapping needs to specify exactly how concepts and relations in $\mathcal{O}_S$ relate to concepts in relations in $\mathcal{O}_T$ in order to enable the rewriting.

After execution of the query, the result instances need to be transformed back to $\mathcal{O}_S$ which involves all requirements for instance transformation described above. The querying use case scenario does, however, indicate the need for a mapping which supports query rewriting in one direction and instance transformation in the other direction.

One mapping for all tasks It is clearly advantageous to have one common declarative mapping language, which suffices for the different use cases of instance transformation, instance unification, query rewriting and ontology merging.

MAFRA [43] combines relating entities (such as concepts and relations) in ontologies with instance transformations. So-called *semantic bridges* specify the relationship between entities in different ontologies. Each instance of a semantic bridge has a transformation attached to it, which specify the instance transformations. The semantic bridges can be used for query rewriting and ontology merging, whereas the attached transformations can be used for instance transformation.

Use of Mapping patterns It is our expectation that many similar ontologies will appear on the Semantic Web. When many similar ontologies exist in a specific domain, the mappings between the ontologies will also be similar. In order to capture these similarities and to reuse existing mapping specification we aim to identify recurring mapping patterns. A mapping pattern can be seen as a template for mappings between classes of ontologies, which can be instantiated to create specific mappings between specific ontologies (cf. [55]).

Mapping patterns furthermore reduce the complexity of a mapping for the user and can be used as a way to modularize a mapping.

Versioning support The ontology mapping language must support constructs for the versioning of the mapping and for referring to specific version of the source and target ontologies.

The latter of course depends on the scheme chosen for ontology versioning. In the case of a new name for each new version of the ontology, no additional provisions have to be taken in the mapping language. This is currently the only way to do versioning in the Web Ontology Language OWL. Therefore, we will assume this situation.

Treating classes as instances Different ontologies might be modeled within slightly different domains with different granularity. What is seen as a class in one ontology might be seen as an instance of a different class in another ontology [60]. In order to support inter-operation between two ontologies with such differences, classes need to be mapped to instances and vice versa.

In fact this can be seen more general. The mapping language should support the mapping of any entity in the source ontology, whether it is a class, instance, relation, to any entity in the target ontology. For example, it should be possible to have a relation-instance mapping, a class-relation mapping, etc.

Mappings of different cardinalities It might be necessary to map a class in one ontology to a number of different classes in the other ontology. It might also be necessary to map a class in one ontology to a class and a relation in the other ontology. In other words, the language needs to support mappings of arbitrary cardinalities. One-to-one mappings are not enough.

6.2 Relation between the Ontology Language and the Mapping Language

We have introduced a language-independent ontology mapping language, based on a set of mapping patters, in deliverable D4.3.1 [9]. This mapping language is independent of the ontology language used for the source and target ontologies, because it does not have a formal semantics associated with it. A major limitation of this approach is that it is not possible to automate any tasks with this mapping. Nonetheless, the user can still do interesting things with the language, such as specifying the relations between elements in different ontologies on a conceptual level and easily discovering relations between different ontologies by inspecting the mapping.

In order to automate tasks on the Semantic Web which require multiple ontologies, a formal semantics is required for the mapping language. The conceptual mapping language has serious drawbacks in its expressiveness, because of lacking built-in functions and other limitations in expressiveness, such as writing down arbitrary formulae.

The mapping language accounts for lacking expressiveness by allowing arbitrary logical expressions. These logical expressions, however, depend on the actual formal language which is used for the mapping. The formal language to which the mapping language is grounded might pose some restrictions on the use of several constructs of the mapping language, because such constructs are beyond the expressive power of the formal language.

There are several considerations when choosing a formal language for the grounding of the mapping language:

Reasoning support When automating tasks on the Semantic Web, tool support is required. We assume the formal mapping language to be a declarative logical language and thus working with this language requires a reasoner. A reasoner could provide support in, e.g. query answering or subsumption reasoning. The availability of efficient reasoners is a prerequisite for ontology mediation on the Semantic Web.

Compatibility with the Ontology language An expression in the mapping language denotes a relation between elements in two (or more) ontologies. These ontologies (we assume) are expressed using a formal ontology language. Clearly, in order to reason with the mapping and the ontologies, some level of compatibility is required between the mapping language and the ontology language. This is because in most tasks to be automated, some form of reasoning with statements in the source and target ontologies is required.

In case the mapping language is equivalent to or strictly more expressiveness than the ontology language, no problems with reasoning will occur, because the reasoner can reason with both the ontologies and the mapping. If, however, the ontology language has different or more expressiveness than the mapping language, issues may occur when attempting to reason over the ontologies and the mapping. In this case, the reasoner should support the union of the mapping and the ontology languages. This is often hard to achieve in an efficient manner. If one wants to fully integrate, for example, the Logic Programming and the Description Logic paradigms, a full first-order theorem prover with extensions to handle nonmonotonic negation is required.

Map between ontologies in different languages There are currently several languages being proposed for the specification of ontologies on the Semantic Web. In this deliverable we focus on the Description Logic-based language. DIP deliverable D1.5 [57] provides a grounding to the Logic Programming-based WSML-Flight [10]. In order to allow interoperation between ontologies expressed on both languages, ideally, it should be possible to map between ontologies in different languages.

It is not trivial to map between ontologies in different language because of differences in expressiveness of the language. As pointed out above, the reasoner which reasons with the mapping needs to reason with the source and target ontologies as

well. As also pointed out above, we would require a first-order theorem prover with nonmonotonic extensions to handle the nonmonotonic negation coming from Logic Programming. Furthermore, we would need to specify a new logical language for this. Finally, in order to properly map between ontologies in different languages there need to be constructs in the mapping language which link different kinds of entities in the different languages. This would lead to an overly complex mapping language. Last but not least, the tool used for specifying the mapping would require to have additional mapping primitives in order to allow mapping between ontologies in different languages.

We argue that it is not feasible to create a mapping between ontologies expressed in different languages. In order to reason with a mapping, the mapping, as well as the source and target ontologies, need to be loaded into the reasoner. Thus, in order to reason with the mapping, the ontologies need to be translated to a common format anyway. In order to keep the mapping tool usable, only one type of constructs should be used and the user of the tool should not be bothered with different constructs coming from different ontology languages. Finally, the mapping language would become overly complex and there would be no efficient reasoners for dealing with the language.

We argue that in order to map between two ontologies, both ontologies need to be expressed in the same language. This way the mapping language is kept simple and we can use existing reasoner implementations to reason over the ontologies. However, this does not make it impossible to map between ontologies expressed in different languages. There is a clear layer of interoperation between WSML-Flight and OWL DL through WSML-Core [10]. Thus, in this way it is always possible to map a subset of an OWL DL ontology to a WSML-Flight ontology or to map a subset of a WSML-Flight ontology to an OWL DL ontology.

6.3 Built-ins in Ontology Mapping

As argued in Section 6.1, instance transformation can be split into two types of transformations:

- *Structural* transformations, which include composing several instances from one instance and splitting one instance into several instances. When the instance is a string data value, this corresponds with concatenating strings and taking substrings, respectively. Structural transformations often correspond with granularity mismatches.
- *Value* transformations, which are required to overcome mismatches in the *encoding* of values in the different ontologies [36].

Structural transformation can either be applied to abstract individuals or to concrete data values. Let's take composition as example. When composing a new abstract individual from a number of source individuals, a new individual is created based on the source individual. When composing a new concrete individual from source concrete individuals, built-in functions need to be applied to create the new data value. For example, if the source values and the target value are all strings, string concatenation would be common. When the source individuals are abstract individuals, the situation becomes more complex. However, the target would often be a count of source individuals. In this case, a built-in aggregate function can be used.

Value transformation can only be applied to concrete data values. Both the source and the target are data values. The transformation would often be a simple function which does, for example, a string permutation or a simple calculation, such as converting kilometers to miles.

As we can see from the above, built-ins are typically required as soon as we are dealing with concrete data values. Therefore, the implementation of the transformation engine should ideally include a large number of built-in functions, including aggregate functions, which are not all that common in current implementations. However, some practical implementations, such as OntoBroker [15] and DLV [37], do offer a large number built-in operators.

We do not require support for specific built-ins from the implementation of the language. Rather, built-in operators are referred to using URIs. If a specific built-in is not supported by the implementation, the user should be alerted of this fact and should be given a choice how to proceed.

6.4 Mediation between OWL ontologies

This section contains the formal grounding in OWL for the ontology mapping language.

In the mapping, we refer to productions in the OWL abstract syntax specification [56].

We translate all statements in the mapping language to OWL abstract syntax through the mapping function $t()$. The function $t()$ takes an expression in the mapping function as argument and returns an OWL DL statement in OWL abstract syntax.

6.4.1 Translating expressions and conditions

The logical expressions that may be written down in the mapping correspond with the axioms that may be written down in OWL DL:

logicalExpression $\doteq$ **axiom**

Class identifiers in the mapping language and in OWL are equivalent:

$t($**classID**$) \doteq$ **classID**

An attribute identifier in the mapping language is mapped to a property identifier in OWL. However, OWL makes a distinction between individual and data valued properties.

Of course the mapping language is agnostic as to whether the properties are individual valued or data valued. Because OWL makes a strict distinction between individual valued and data valued properties, it is only possible to map between two individual valued or two data valued properties[2].

$t($**attributeID**$) \mapsto$ **individualvaluedPropertyID** | **datavaluedPropertyID**

The same holds for the relation identifiers. OWL only supports binary relations in the form of properties. However, this should not be a problem because we only allow mapping between two OWL ontologies.

$t($**relationID**$) \mapsto$ **individualvaluedPropertyID** | **datavaluedPropertyID**

Different class expressions in the mapping language correspond to different types of descriptions in OWL:

$t($'and(' **classExpr**$_1$ ... **classExpr**$_n$ ')'$) \mapsto$ 'intersectionOf(' $t($**classExpr**$_1)$ ... $t($**classExpr**$_n)$ ')'

$t($'or(' **classExpr**$_1$ ... **classExpr**$_n$ ')'$) \mapsto$ 'unionOf(' $t($**classExpr**$_1)$ ... $t($**classExpr**$_n)$ ')'

$t($'not(' **classExpr** ')'$) \mapsto$ 'complementOf(' $t($**classExpr**$)$ ')'

Because OWL is based on the function-free two-variable fragment of First-Order logic, it is not possible to do a join mapping, because a join mapping requires a function symbol in order to construct a new term based on several existing terms. Thus, join mappings are not allowed in a mapping between OWL ontologies if the mapping language is grounded in OWL. Note that is a more expressive rule language would be used for mapping between OWL ontologies, the join mapping could be re-introduced.

The types of attribute expressions in OWL is also limited. OWL allows inverse, symmetric and transitivity of mappings.

In OWL it is possible to directly assert that a property is the inverse of another property. However, it is not possible to use the transitivity and symmetry of a particular property in an axiom. It is only possible to state symmetry and transitivity of a particular property.

[2]This restriction would not necessarily apply if we map between two OWL ontologies using a mapping language which is more expressive than OWL. However, we are using OWL as the mapping language here and thus this restriction applies. It would be possible, for example, to have a certain mapping table between data values and individuals values or to have a mapping function which produces data values based on individual values or the other way around, in order to achieve the mapping.

There are two different ways to deal with this limitation in the mapping language: we can either create a new property, state that it subsumes the original property and assert symmetry and/or transitivity for this new property or assert symmetry and/or transitivity for the original property. The problem with the latter is that it changes the definition of the original property and it also affects other mapping axioms which involve this property. Therefore, we choose the former solution:

Thus, we have a translation:

t('symmetric(' **attributeExpr** ')') $\mapsto R_{new}$

With R_{new} a newly introduced property:

'ObjectProperty(' t(**attributeExpr**) 'super(' R_{new} ')' ')'

'ObjectProperty(' R_{new} 'Symmetric' ')'

and

t('trans(' **attributeExpr** ')') $\mapsto R_{new}$

With R_{new} a newly introduced property:

'ObjectProperty(' t(**attributeExpr**) 'super(' R_{new} ')' ')'

'ObjectProperty(' R_{new} 'Transitive' ')'

For inverse we apply a similar trick: t('inverse(' t(**attributeExpr**) ')') $\mapsto R_{new}$

With R_{new} a newly introduced property:

'ObjectProperty(' t(**attributeExpr**) 'inverseOf(' R_{new} ')' ')'

'ObjectProperty(' R_{new} ')'

We map the attribute value condition in the following form:

t('attributeValueCondition(' **attributeID** (**individualID** | **dataLiteral**) ')') $\mapsto$ 'restriction(' t(**attributeID**) 'value(' t(**individualID** | **dataLiteral**) '))'

t('attributeTypeCondition(' **attributeID classExpr** ')') $\mapsto$ 'restriction(' t(**attributeID**) 'someValuesFrom(' t(**classExpr**) '))'

We map the attribute occurrence condition in the following way: v

t('attributeOccurrenceCondition(' **attributeID** ')') $\mapsto$ 'restriction(' t(**attributeID**) 'minCardinality(1))'

The mapping of the attribute conditions (i.e. conditions that are allowed inside a mapping) takes a parameter which is the identifier of the attribute (called $AttID$:

t('valueCondition('**individualID** | **dataLiteral** ')', $AttID$) $\mapsto$ 'restriction(' $t(AttID)$ 'value(' t(**individualID** | **dataLiteral**) ')' ')'

t('valueCondition(' **classExpr** ')', $AttID$) $\mapsto$ 'restriction(' $t(AttID)$ 'value(' t(**classExpr**) ')' ')'

Expression conditions cannot be mapped.

Translating mappings to OWL

For expressing mappings between ontologies in OWL we rely on the ontology import mechanism of OWL, which is expressed through annotation properties in the OWL abstract syntax (in case **mappingID** is missing, it is simply omitted from the resulting OWL ontology).

t('Mapping(' **mappingID** 'source(' **ontologyID**$_1$ ')' ... 'source(' **ontologyID**$_{n-1}$ ')' 'target(' **ontologyID**$_n$ ')' **directive**$_1$ ... **directive**$_n$ ')') $\mapsto$
'Ontology(' t(**mappingID**) 'Annotation(owl:imports' t(**ontologyID**$_1$) ')' ... 'Annotation(owl:imports' t(**ontologyID**$_n$) ')' t(**directive**$_1$) ... t(**directive**$_n$) ')'

A directive is either an annotation or a mapping expression. We translate annotations in the following way:

t('Annotation(' **propertyID** **propertyValue** ')') $\mapsto$ 'Ontology(' t(**propertyID**) t(**propertyValue**) ')'

A mapping expression can be a class mapping, an attribute mapping, a relation mapping, an instance mapping or a mapping between these.

Note that a class mapping which omits the 'two-way' is equivalent to a class mapping which has the 'one-way' integrated.

t('classMapping(' ['one-way'] **classExpr**$_1$ **classExpr**$_2$ **classAttributeMapping**$_1$ ... **classAttributeMapping**$_n$ **classCondition**$_1$ ... **classCondition**$_n$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'SubClassOf(' 'intersectionOf(' t(**classExpr**$_1$) t(**classAttributeMapping**$_1$) ... t(**classAttributeMapping**$_n$) t(**classCondition**$_1$) ... t(**classCondition**$_n$) ')' t(**classExpr**$_2$) ')' t(logicalExpression)

A two-way mapping simple translated to a class equivalence axiom:

t('classMapping(two-way' **classExpr**$_1$ **classExpr**$_2$ **classAttributeMapping**$_1$ ... **classAttributeMapping**$_n$ **classCondition**$_1$ ... **classCondition**$_n$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'EquivalentClasses(' 'intersectionOf(' t(**classExpr**$_1$) t(**classAttributeMapping**$_1$) ... t(**classAttributeMapping**$_n$) t(**classCondition**$_1$) ... t(**classCondition**$_n$) ')' t(**classExpr**$_2$) ')' t(logicalExpression)

There exist two kinds of attribute mappings, namely attribute mappings inside class mappings and attribute mappings separate of class mappings. Because of limitations in

OWL, attributes can only be mapped outside of the context of the class. Furthermore, attribute conditions are not allowed, because they cannot be applied in property axioms in OWL.

t('attributeMapping(' ['one-way'] **attributeExpr**$_1$ **attributeExpr**$_2$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'SubPropertyOf(' t(**attributeExpr**$_1$) t(**attributeExpr**$_2$) t(logicalExpression)

t('attributeMapping(two-way' **attributeExpr**$_1$ **attributeExpr**$_2$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'EquivalentProperties(' t(**attributeExpr**$_1$) t(**attributeExpr**$_2$) t(logicalExpression)

When mapping between OWL ontologies, attributes are equivalent to relations, because OWL only has the concept of binary relations:

t('relationMapping(' ['one-way'] **relationExpr**$_1$ **relationExpr**$_2$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'SubPropertyOf(' t(**relationExpr**$_1$) t(**relationExpr**$_2$) t(logicalExpression)

t('relationMapping(two-way' **relationExpr**$_1$ **relationExpr**$_2$ ['{' **logicalExpression** '}'] ')') $\mapsto$
'SubPropertyOf(' t(**relationExpr**$_1$) t(**relationExpr**$_2$) t(logicalExpression)

An instance mapping is simply equivalent to individual equality in OWL:

t('instanceMapping(' **individualID**$_1$ **individualID**$_2$ ')') $\mapsto$
'SameIndividual(' t(**individualID**$_1$) t(**individualID**$_2$)

Expressiveness of OWL vs. expressiveness of the Mapping language

It turns out that not all concepts in the mapping language can be mapped to OWL. This indicates that OWL is clearly lacking expressiveness for mapping between ontologies. The major problem is that OWL does not allow chaining over predicates. The expressiveness of OWL which is not captured in the mapping language are the enumeration of individuals and the universal value restrictions. Existential value restrictions are captured through the attributeValueCondition. The mapping language also does not allow to specify cardinality directly. However, it is possible to write down arbitrary OWL axioms and thus the language is in the end exactly as expressive as OWL.

6.4.2 Mappings not allowed in OWL

When compared with full first-order logic, OWL DL has severe limitations. Namely, function symbols are not allowed (although their expressiveness can be simulated using existential quantification and equality), only unary and binary predicates are allowed and

Mapping construct	Feature
join	Join mappings are not allowed because they require function symbols, which are not in OWL
classAttributeMapping	Vocabularies of classes and properties in OWL are disjoint
classRelationMapping	Vocabularies of classes and properties in OWL are disjoint
classInstanceMapping	Vocabularies of classes and individuals in OWL are disjoint
reflexive	It is not possible in OWL DL to use the same variable in both positions in a binary predicate
and for attribute and relation mappings	OWL DL does not allow conjunction of properties
or for attribute and relation mappings	OWL DL does not allow disjunction of properties
not for attribute and relation mappings	OWL DL does not allow negation of properties

Table 6.1: Mapping constructs not allowed in OWL DL mapping

it is not possible to chain variables over predicates, because OWL DL stays in the so-called two-variable fragment of first-order logic, which guarantees decidability of the satisfiability problem, but which does impose severe restrictions on expressiveness.

Table 6.1 lists all constructs of the mapping language which are not supported in the grounding to OWL along with the feature missing in OWL which would allow this construct.

OWL DL allows only limited logical expressions, namely OWL DL axioms. OWL DL does not support built-in predicates[3], which we identified as very important for ontology mapping (Section 6.3).

6.4.3 Extending the mapping language to SWRL

The Semantic Web Rule Language SWRL [32] is an extension of OWL DL to a full-fledged rule language (without function symbols). It might be worthwhile to discuss how SWRL could be used to exploit more of the mapping language for mapping between OWL ontologies. The advantage of using SWRL for this is that the inter-operation between OWL ontologies and SWRL rules is clear. SWRL is a direct extension of OWL DL and is thus a strict superset. A drawback of SWRL is that there are currently no efficient implementations which can handle a significant amount of instance data. In fact, the only reasoners of which we are aware that they can deal with SWRL are full first-order theorem provers. Theorem provers are not guaranteed to terminate[4] and perform poorly if many

[3]Note that there exist extensions of OWL DL which do allow the use of built-ins, e.g. OWL-E [54].

[4]Actually, any reasoner that can deal with SWRL cannot guarantee to terminate, because entailment in SWRL is undecidable.

entailment checks are required for a reasoning task (e.g. query answering). There exist subsets of SWRL which are known to be decidable, but such subsets either reduce the expressiveness of the Description Logic fragment (thus imposing limitations on the use of OWL constructs in the ontology) or the rules component [40]. A limitation of OWL DL which fits nicely in the rules world is OWL DL^- [13], which is based on Description Logic Programs [29].

SWRL defines a comprehensive set of built-in predicates for use with data values. As we have argued above (Section 6.3), we expect built-ins to play a major role in ontology mapping. However, we are not aware of any implementation for SWRL which can handle such built-ins, as theorem provers usually do not support the use of built-ins and typically require axiomatization of the entire datatype domain.

In this chapter we have described an ontology mapping language for OWL ontologies. This concludes our treatment of the mapping creation phase and the mapping framework. In the following we will describe related work and conclusions.

Chapter 7

Related Work

In this chapter we describe two major efforts in ontology mapping, namely OntoMerge [22] and MAFRA [43]. We have chosen these efforts because they represent the two major directions in ontology mapping: the use of bridging axioms to describe ontology mappings (OntoMerge) and the use of instance transformation procedures (MAFRA).

7.1 OntoMerge

[22] introduce an approach to ontology mediation "ontology translation be ontology merging and automated reasoning". In this approach, ontologies are merged by taking the union of both ontologies, where all terms are separated through the differences in the namespace. So-called *Bridging Axioms* are used to connect the overlapping part of the two ontologies.

In general, when merging ontologies, one would either create a new namespace for the merged ontology or import one ontology into the other, so that the merged ontology uses the namespace of the importing ontology. Having in the end an ontology which uses different namespaces in its definitions can be very confusing for the user, since an ontology is intended to be *shared* among multiple parties. Furthermore, the bridging axioms in the merged ontology might also be very confusing for the user, since they serve no other purpose than linking together related terms in the ontology. Thus, the merged ontology contains a lot of clutter, which makes the ontology hard to understand and hard to use. The clutter in the ontology consists of: (1) terms with different namespaces, (2) similar and equivalent terms exist in the ontology and (3) bridging axioms between the related terms. These three factors impede usability and especially sharing of the ontology.

On the other hand, [22] does not propose to use the merged ontologies as such, but to merely use them for three different tasks:

1. *Dataset translation* (cf. instance transformation in [12]). Dataset translation is the problem of translating a set of data (instances) from one representation to the other.

2. *Ontology extension generation*. The problem of ontology extension generation is the problem of generating an extension O_{2s}, given two related ontologies O_1 and O_2 and an extension (subontology) O_{1s} of ontology O_1. The example given by the authors is to generate a WSDL extension based on an OWL-S description of the corresponding Web Service.

3. *Querying different ontologies*. This relates very much to the query rewriting described in [12]. However, query rewriting is a technique for solving the problem of querying different ontologies, whereas the authors of [22] merely stipulate the problem.

As we have also suggested in [12], [22] uses mappings between ontologies in order to enable the translation. In fact, the ontology translation (except for the extension generation) can be seen as run-time mediation [12].

Dou et al. use an internal representation for the ontologies, called *Web-PDDL*, which is a typed first-order logic language. They support im- and export of DAML+OIL and OWL, but im- and exporters for other languages could be written as well, because Web-PDDL is able to capture many different ontology languages, because of its expressiveness.

Dataset translation Dou et al. perform dataset translation in two distinct steps. First, given the source dataset (a set of facts) and the merged ontology, all possible inferences are drawn from the source facts. Secondly, the results are projected on the target vocabulary, retaining only the results expressed in terms of the target ontology. These two steps guarantee that a maximal translation is performed, with respect to the merged ontology and the source dataset.

In their practical evaluation of the system, the authors only work with very small datasets consisting of several thousands of facts. The fact that they use a theorem prover leaves open questions about scalability for large numbers of facts.

Ontology extension generation Say you have two related ontologies O_1 and O_2 and a subontology O_1s of O_1. It is now possible, using the relationships between the two ontologies, to automatically generate a subontology O_2s of O_2 which corresponds with O_1s.

The subontology O_{2s} is identified by creating skolem constants during inference with the merged ontology and then creating predicates based on these skolem constants. The new predicates should be made sub-predicates of the existing predicates, in which the skolem constant is.

The major disadvantage to this approach for ontology extension generation, identified by the authors, is that the generated subontology only contains subproperty axioms, whereas many subontologies might be specified using general axioms.

Querying through Different Ontologies Querying is done in OntoMerge by selecting the merged ontology which merges the query ontology and the other ontology. Then, a query selection and reformulation module (not described in detail) is used to select subqueries and reformulate the subqueries. Each subquery is executed on respective knowledge bases and the results are combined.

In fact, what we call an *Ontology mapping* is very similar to a set of bridging axioms in [22]. However, we do not presume the source and target ontologies use the same language as the mapping, whereas [22] requires the merged ontology to consist of the source and target ontologies *and* the bridging axioms.

A major drawback of OntoMerge is that bridging axioms need to be written using a first-order language. Only very few people are familiar with the first-order logic.

7.2 MAFRA

MAFRA (MApping FRAmework for distributed ontologies) [43, 62] is a framework for mapping between ontologies. Figure 7.1 outlines the conceptual architecture of MAFRA where a set of main phases is identified and organized along two dimensions. Horizontal modules correspond to five fundamental phases in the ontology mapping process (Lift & Normalization, Similarity, Semantic Bridging, Execution and Post-processing). The vertical modules (Evolution, Domain Knowledge & Constraints, Cooperative Consensus building and GUI) interact with the horizontal phases during the entire ontology mapping process.

The horizontal dimension is subdivided into the following five modules:

- *Lift & Normalization* Defines a uniform representation (in RDF(S)) in order to normalize the ontologies we want to map. In this step, differences (like special characters, upper case letters and acronyms) are eliminated and the semantic differences are slightly reduced.
- *Similarity* It is a multi-strategy process that calculates similarities between ontology entities using different approaches. The combination of all of the matchers proposed by Maedche and colleagues allow the system to obtain better results in this phase.

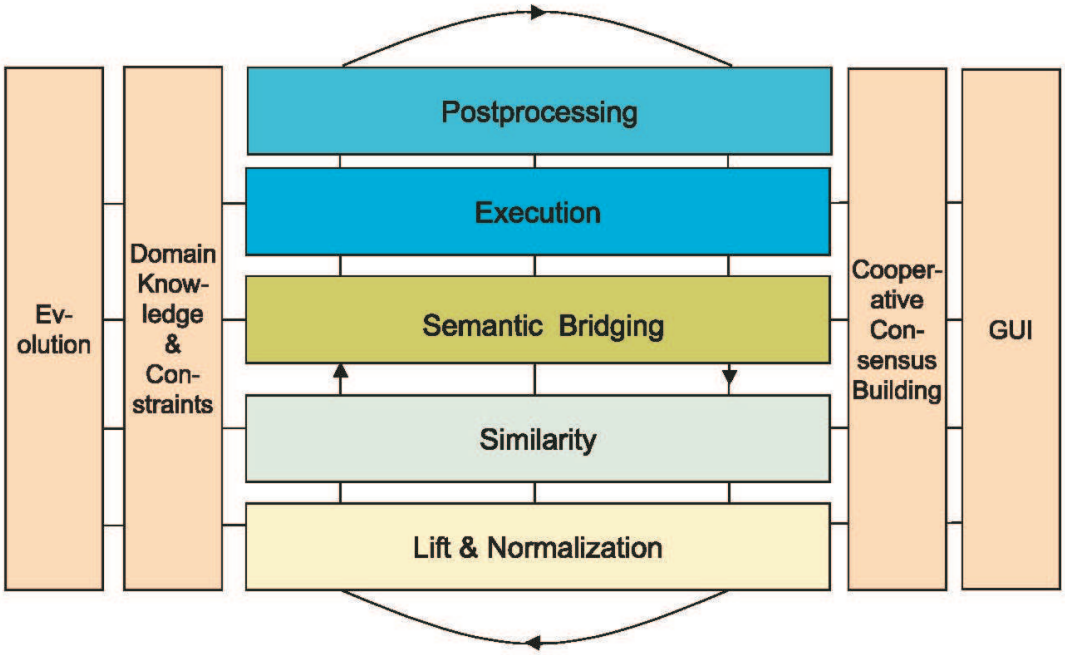

Figure 7.1: MAFRA Conceptual Architecture [43]

- *Semantic Bridging* Semantically relate entities (i.e. classes, relations, attributes) from the source and target ontologies, encapsulating all necessary information to transform instances of an entity in the source ontology into instances of one (or more) target ontology entity. The result is close to the notion of *articulation ontology* in ONION[49].

- *Execution* This module actually transforms instances from the source ontology representation into the representation of the target ontology by evaluating the semantic bridges which have been defined in the previous phase. There are two possible operational modes: offline (all the transformations are executed one time) and online (the transformations are continuously executed, and modifications in the source or target ontologies are immediately reflected).

- *Postprocessing* Take the results of the execution module to check and improve the quality of the transformation results (e.g. object identity: recognize that two instances represent the same real-world object).

The vertical dimension comprises the following modules:

- *Evolution* Synchronize the changes in the source and target ontologies with the semantic bridges defined by the Semantic Bridge module.

- *Cooperative Consensus Building* From multiple alternative possible mappings the tool helps to set up a consensus between the various proposals of people involved in the mapping task.
- *Domain constraints and Background Knowledge* The tool allows users to include extra information (e.g. lexical ontologies like Wordnet can help in the identification of synonyms) in order to improve the quality of the mapping.
- *GUI* Visualization of the elements of the source and target ontologies makes the mapping task a lot easier in the same way as do the semantic bridges established to represent the mapping between entities.

The main goal in MAFRA is to transform instances of the source ontology into instances of the target ontology. Semantic Bridges specify how to perform these transformations and categorize them between concept bridges and property bridges. Concept bridges define the transformations between source instances and target instances, whereas property bridges specify the transformations between source properties and target properties.

MAFRA includes a formal representation to specify the mappings. The formalism that is used to describe the Semantic Bridges is based in an ontology specified in DAML+OIL, called the Semantic Bridging Ontology (SBO), which includes the following concepts:

- *Classes Concepts, Relations and Attributes* Represent the main type of entities that can be found in the source and target ontologies.
- *Class Semantic Bridge* This is the most generic bridge and defines the relations between source and target entities. It allows for the definition of Abstract Semantic Bridges, which allow users to define common characteristics that can be used in the definition of other (concrete) semantic bridges. Abstract Semantic Bridges does not define concrete relations between source and target entities.
- *Class Service* These are reference resources that are responsible to connect to or to describe transformations.
- *Class Rule* Represent constraint specifications and relevant information for a transformation.
- *Class Transformation* This class specifies a transformation procedure for each semantic bridge, and it is obligatory (except in abstract semantic bridges).
- *Class Condition* Represent the conditions that should hold before a semantic bridge can be executed.
- *Composition modeling primitives* Allow each semantic bridge to aggregate several different bridges that will be processed one by one when the transformations of the

parent semantic bridge are executed. This modeling primitive belongs to the class Semantic Bridge.

- *Alternative modeling primitives* Supported by the class SemanticBridgeAlt; its function is to group several mutually exclusive semantic bridges.

We adopt the different phases of the MAFRA ontology mapping process in the mapping creation phase of out mediation framework. The major differences between our approach and MAFRA's approach are that MAFRA only considers mapping between RDFS ontologies, whereas we consider mapping between and merging of ontologies specified in different ontology languages, with a special emphasis on OWL DL. MAFRA only considers the use case of instance transformation, whereas we address the wider ontology mediation problem and also support querying and ontology merging.

Chapter 8

Conclusions

In this deliverable we have motivated ontology mediation through a number of use cases and application scenarios. Furthermore, we have described an ontology mediation framework inspired by these use cases and scenarios. We have focussed on the mapping creation step in the mediation process which can be divided into two parts: automated discovery of mappings and further refinement and specification of these mappings. We have addressed the mapping creation problem by describing a method for mapping discovery, called APFEL, and an ontology mapping language based on OWL DL.

In this deliverable we have described a formal grounding in OWL DL for the ontology mapping language which was originally developed in SEKT deliverable D4.3.1 [9]. A formal mapping enables performing certain tasks across ontologies, such as querying of remote knowledge bases in terms of a local ontology and the transformation of data between different representations.

Besides the formal grounding of the mapping language we have provided an overall framework for ontology mediation. An important part of this mediation framework is the discovery of ontology mappings. We have described an approach called APFEL (Alignment Process Feature Estimation and Learning) which can be used to estimate parameters for a so-called PArameterized Mapping Method (PAMM). Such an alignment method can be used to actually discover mappings between ontologies.

The mapping discovery and the mapping language are the major components in the overall mediation framework. Mapping discovery is used to discover mappings between ontologies which are then presented to the user. The user can then refine these mappings or choose to add or remove mapping rules.

Two other important components are storage and retrieval of mappings and the actual use of mappings in run-time mediation.

We have briefly touched on the topic of ontology merging. Our main focus, however, is on ontology mapping and the use of such mappings in querying and data transformation. Future versions of the mediation framework might develop more in the direction of

ontology merging, depending on the requirements of the SEKT case studies.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. On integrating catalogs. In *Proceedings of the tenth international conference on World Wide Web*, pages 603–612. ACM Press, 2001.

[2] T. Berners-Lee, R. Fielding, U. C. Irvine, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. RFC 2396, Internet Engineering Task Force, 1998.

[3] Paolo Bouquet, Jerome Euzenat, Enrico Franconi, Luciano Serafini, Giorgos Stamou, and Sergio Tessaris. Specification of a common framework for characterizing alignment. Deliverable D2.2.1, Knowledge Web, 2004.

[4] Paolo Bouquet, B. Magnini, L. Serafini, and S. Zanobini. A SAT-based algorithm for context matching. In *IV International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'2003)*, Stanford University (CA, USA), June 2003.

[5] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 600–607, Madison, Wisconsin, USA, 1998. MIT Press.

[6] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[7] Jos de Bruijn. Semantic information integration within and across organizational boundaries. Master Thesis, TU Delft, The Netherlands, 2003.

[8] Jos de Bruijn, Douglas Foxvog, Holger Lausen, Eyal Oren, Dumitru Roman, and Dieter Fensel. The WSML Family of Representation Languages. Deliverable D16v0.2, WSML, http://www.wsmo.org/wsml/, 2004. Available from http://www.wsmo.org/2004/d16/v0.2/.

[9] Jos de Bruijn, Douglas Foxvog, and Kerstin Zimmerman. Ontology mediation patterns library v1. Deliverable D4.3.1, SEKT, 2004.

[10] Jos de Bruijn, Holger Lausen, and Dieter Fensel. The WSML Family of Representation Languages. Deliverable D16.1v0.2, WSML, 2004. Available from http://www.wsmo.org/2004/d16/d16.1/v0.2/.

[11] Jos de Bruijn, Francisco Martín-Recuerda, Dimitar Manov, and Marc Ehrig. State-of-the-art survey on ontology merging and aligning v1. Deliverable D4.2.1, SEKT, 2004.

[12] Jos de Bruijn and Axel Polleres. Towards and ontology mapping language for the semantic web. Technical Report DERI-2004-06-30, DERI, 2004.

[13] Jos de Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL$^-$. Deliverable d20.1v0.2, WSML, 2004.
Available from http://www.wsmo.org/2004/d20/d20.1/v0.2/.

[14] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004.

[15] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. *Ontobroker: Ontology based Access to Distributed and Semi-Structured Information*. Kluwer Academic, 1999.

[16] Ying Ding, Dieter Fensel, Michel C. A. Klein, and Borys Omelayenko. The semantic web: yet another hip? *Data Knowledge Engineering*, 41(2-3):205–227, 2002.

[17] H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[18] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *VLDB Journal*, 50:279–301, 2003.

[19] A. Doan, Y., Lu, Y. Lee, and J. Han. Object matching for data integration: A profile-based approach. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, Acapulco, Mexico, August 2003.

[20] AnHai Doan, Jazant Madhaven, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag, 2004.

[21] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252, 1998.

[22] Dejing Dou, Drew McDermott, and Peishen Qi. Ontology translation by ontology merging and automated reasoning. In *Proc. EKAW2002 Workshop on Ontologies for Multi-Agent Systems*, pages 3–18, 2002.

[23] Marc Ehrig and Steffen Staab. QOM - quick ontology mapping. In Frank van Harmelen, Sheila McIlraith, and Dimitris Plexousakis, editors, *Proceedings of the Third International Semantic Web Conference (ISWC2004)*, LNCS, pages 683–696, Hiroshima, Japan, 2004. Springer.

[24] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In *Proceedings of the First European Semantic Web Symposium, ESWS 2004*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91, Heraklion, Greece, May 2004. Springer Verlag.

[25] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR04)*, 2004.

[26] Jerome Euzenat and Petko Valtchev. Similarity-based ontology alignment in owl-lite. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI2004)*, pages 333–337, Valencia, Spain, August 2004.

[27] Christiaan Fluit, Marta Sabou, and Frank van Harmelen. *Supporting User Tasks through Visualisation of Light-weight Ontologies*. Springer-Verlag, 2003.

[28] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS'04*, number 3053 in LNCS, pages 61–75, Heraklion, Greece, 2004. Springer-Verlag.

[29] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.

[30] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida, 2003.

[31] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. Daml draft v0.5, DAML, 2003. Available from http://www.daml.org/2003/11/swrl/.

[32] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. Available from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/, May 2004.

[33] Todd Hughes. Information interpretation and integration conference, August 2004. http://www.atl.external.lmco.com/projects/ontology/i3con.html.

[34] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In *Proceedings of Ninth International Conference on Knowledge Representation and Reasoning 2004*, pages 152–162, Whistler, Canada, June 2004.

[35] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA, August 4–5, 2001.

[36] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI01*, Seattle, USA, August 4-5 2001.

[37] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *CoRR*, cs.AI/0211004, 2002.

[38] I. V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 1966.

[39] Alon Y. Levy. *Logic-Based Techniques in Data Integration*, pages 575–595. Kluwer Publishers, 2000.

[40] Alon Y. Levy and Marie-Christine Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165 – 209, 1998.

[41] X. Li, P. Morie, and D. Roth. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine. Special Issue on Semantic Integration*, 2005.

[42] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proc. 27th Int. Conf. on Very Large Data Bases (VLDB)*, 2001.

[43] Alexander Maedche, Boris Motik, Nu no Silva, and Raphael Volz. Mafra a mapping framework for distributed ontologies. In *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW-2002*, Madrid, Spain, 2002.

[44] A. Maier, H.-P. Schnurr, and Y. Sure. Ontology-based information integration in the automotive industry. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference: The Semantic Web (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science (LNCS)*, pages 897–912, Sanibel Island, FL, USA, 2003. Springer.

[45] Andreas Maier, Hans-Peter Schnurr, and York Sure. Ontology-based information integration in the automotive industry. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, number LNCS 2870, pages 897 – 912. Springer-Verlag, 2003.

[46] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 117. IEEE Computer Society, 2002.

[47] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Developing metadata-intensive applications with rondo. *Journal of Web Semantics*, 1(1), December 2003.

[48] Prasenjit Mitra, Gio Wiederhold, and Martin Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of the Conference on Extending Database Technology 2000 (EDBT'200)*, volume 1777, pages 86+, Konstanz, Germany, 2000.

[49] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000)*, Konstanz, Germany, March 2000.

[50] Boris Motik, Ulrike Sattler, and Rudi Studer. Adding DL-safe rules to OWL DL. 2004. to be submitted.

[51] Natalya F. Noy and Mark A. Musen. Smart: Automated support for ontology merging and alignment. Technical Report SMI-1999-0813, Stanford Medical Informatics, 1999.

[52] Natalya F. Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proc. 17th Natl. Conf. On Artificial Intelligence (AAAI2000)*, Austin, Texas, USA, July/August 2000.

[53] Natalya F. Noy and Mark A. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

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

[55] John Y. Park, John H. Gennari, and Mark A. Musen. Mappings for reuse in knowledge-based systems. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modelling and Management (KAW 98)*, Banff, Canada, 1998.

[56] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C, 2004.

[57] Livia Predoiu, Francisco Martín-Recuerda, Axel Polleres, Fabio Porto, Adrian Mocan, Kerstin Zimmermann, Cristina Feier, and Jos de Bruijn. Framework for representing ontology networks with mappings that deal with conflicting and complementary concept definitions. Deliverable D1.5, DIP, 2004. Available from http://dip.semanticweb.org/.

[58] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[59] Dumitru Roman, Holger Lausen, and Uwe Keller. Web service modeling ontology standard (WSMO-standard). Working Draft D2v1.0, WSMO, 2004. Available from http://www.wsmo.org/2004/d2/v1.0/.

[60] Guus Schreiber. The web is not well-formed. *IEEE Intelligent Systems*, 17(2), 2002. Contribution to the section Trends and Controversies: Ontologies KISSES in Standardization.

[61] Pavel Shvaiko and Jerome Euzenat. A survey of schema-based matching approaches. Technical Report DIT-04-087, University of Trento, 2004.

[62] Nuno Silva and Jo ao Rocha. Service-oriented ontology mapping system. In *Proceedings of the Workshop on Semantic Integration of the International Semantic Web Conference (ISWC2003)*, Sanibel Island, USA, 2003.

[63] Gerd Stumme and Alexander Maedche. Fca-merge: Bottom-up merging of ontologies. In *7th Intl. Conf. on Artificial Intelligence (IJCAI '01)*, pages 225–230, Seattle, WA, USA, 2001.

[64] Mike Uschold. Creating, integrating, and maintaining local and global ontologies. In *Proceedings of the First Workshop on Ontology Learning (OL-2000) in conjunction with the 14th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, Germany, August 2000.

[65] Pepijn R. S. Visser and Zhan Cui. On accepting heterogeneous ontologies in distributed architectures. In *Proceedings of the ECAI98 workshop on applications of ontologies and problem-solving methods*, Brighton, UK, 1998.

[66] Raphael Volz, Siegfried Handschuh, Steffen Staab, Ljiljana Stojanovic, and Nenad Stojanovic. Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the semantic web. *Journal of Web Semantics*, 2004. to appear.

[67] Raphael Volz, Daniel Oberle, Steffen Staab, and Rudi Studer. Ontolift prototype. Deliverable 11, WonderWeb, 2003.

[68] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin core metadata for resource discovery. RFC 2413, IETF, 1998.

[69] Gio Wiederhold. An algebra for ontology composition. In *Proceedings of 1994 Monterey Workshop on formal Methods*, pages 56–61, U.S. Naval Postgraduate School, Monterey CA, 1994.

Appendix A

Ontology Mapping Language Abstract Syntax

This Appendix contains the Abstract syntax, which was developed in Deliverable D4.3.1: "Ontology Mediation Patterns Library V1" [9]. This appendix is a direct copy of the corresponding appendix of D4.3.1 and is only included in order to make this deliverable self-contained.

The abstract syntax is written in the form of EBNF, similar to the OWL Abstract Syntax [56]. Any element between square brackets '[' and ']' is optional. Any element between curly brackets '{' and '}' can have multiple occurrences.

Each element of an ontology on the Semantic Web, whether it is a class, attribute, instance, or relation, is identified using a URI [2]. In the abstract syntax, a URI is denoted with the name **URIReference**. We define the following identifiers:

mappingID ::= **URIReference**
ontologyID ::= **URIReference**
classID ::= **URIReference**
propertyID ::= **URIReference**
attributeID ::= **URIReference**
relationID ::= **URIReference**
individualID ::= **URIReference**

We allow concrete data values. The abstract syntax for data values is taken from the OWL abstract syntax:

dataLiteral ::= **typedLiteral**|**plainLiteral**

typedLiteral ::= **lexicalForm**'^^'**URIReference**
plainLiteral ::= **lexicalFrom**['@'**languageTag**]

The lexical form is a sequence of unicode characters in normal form C, as in RDF. The language tag is an XML language tag, as in RDF.

First of all, the mapping itself is declared, along with the ontologies participating in the mapping.

mapping ::= 'Mapping(' [**mappingID**]
{ 'source(' **ontologyID** ')' }
'target(' **ontologyID** ')'
{ **directive** } ')'

A mapping consists of a number of annotations, corresponding to non-functional properties in WSMO [59], and a number of mapping expressions. The creator of the mapping is advised to include a version identifier in the non-functional properties.

directive ::= **annotation**
|**expression**

annotation ::= 'Annotation(' **propertyID URIReference** ')'
'Annotation(' **propertyID dataLiteral** ')'

Expressions are either class mappings, relation mappings, instance mappings or arbitrary logical expressions. The syntax for thees logical expressions is not specified; it depends on the actual logical language to which the language is grounded.

A special kind of relation mappings are attribute mappings. Attributes are binary relations with a defined domain and are thus associated with a particular class. In the mapping itself the attribute can be either associated with the domain defined in the (source or target) ontology or with a subclass of this domain.

A mapping can be either uni- or bidirectional. In the case of a class mapping, this corresponds with class equivalence and class subsumption, respectively. In order to distinguish these kinds of mappings, we introduce two different keywords for class, relation and attribute mappings, namely ‘unidirectional’ and ‘bidirectional’. Individual mappings are always bidirectional. Unidirectional and bidirectional mappings are differentiated with the use of a switch. The use of this switch is required.

It is possible, although not required, to nest attribute mappings inside class mappings. Furthermore, it is possible to write an axiom, in the form of a class condition, which

defines general conditions over the mapping, possibly involving terms of both source and target ontologies. Notice that this class condition is a general precondition for the mapping and thus is applied in both directions if the class mapping is a bidirectional mapping. Notice that we allow arbitrary axioms in the form of a logical expression. The form of such a logical expression depends on the logical language being used for the mappings and is thus not further specified here.

expression ::= 'classMapping(' 'unidirectional'—'bidirectional' { **annotation** }
classExpr classExpr { **classAttributeMapping** }
{ **classCondition** } ['{' **logicalExpression** '}'] ')'

There is a distinction between attributes mapping in the context of a class and attributes mapped outside the context of a particular class. Because attributes are defined locally for a specific class, we expect the attribute mappings to occur mostly inside class mappings. The keywords for the mappings are the same. However, attribute mappings outside of the context of a class mappings need to be preceded with the class identifier, followed by a dot '.'.

classAttributeMapping ::= 'attributeMapping(' 'unidirectional'—'bidirectional' **attributeExpr**
attributeExpr { **attributeCondition** } ')'

expression ::= 'attributeMapping(' 'unidirectional'—'bidirectional' **attributeExpr**
attributeExpr { **attributeCondition** }
['{' **logicalExpression** '}'] ')'

expression ::= 'relationMapping(' 'unidirectional'—'bidirectional' **relationExpr**
relationExpr { **relationCondition** }
['{' **logicalExpression** '}'] ')'

expression ::= 'instanceMapping(' **individualID individualID** ')'
expression ::= 'classAttributeMapping(' 'unidirectional'—'bidirectional' **classExpr**
attributeExpr ['{' **logicalExpression** '}'] ')'

expression ::= 'classRelationMapping(' 'unidirectional'—'bidirectional' **classExpr**
relationExpr ['{' **logicalExpression** '}'] ')'

expression ::= 'classInstanceMapping(' 'unidirectional'—'bidirectional' **classExpr**
individualID ['{' **logicalExpression** '}'] ')'

expression ::= '{' **logicalExpression** '}'

For class expressions we allow basic boolean algebra. This corresponds loosely with Wiederhold's ontology algebra [69]. Wiederhold included the basic intersection and union, which correspond with our and and or operators. Wiederhold's difference operator corresponds with a conjunction of two class expressions, where one is negated, i.e. for two class expressions C and D, the different $C - D$ corresponds with and(C,not(D)).

The join expression is a specific kind of disjunction, namely a disjunction with an additional logical expression which contains the precondition for instances to be included in the join.

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

Attribute expressions are defined as such, allowing for inverse, transitive close, symmetric closure and reflexive closure, where inverse(A) stands for the inverse of A, symmetric(A) stands for the symmetric closure of A[1], reflexive(A) stands for the reflexive closure of A[2] and trans(A) stands for the transitive closure of A:

attributeExpr ::= **attributeID**
|'and(' **attributeExpr attributeExpr** { **attributeExpr** } ')'
|'or(' **attributeExpr attributeExpr** { **attributeExpr** } ')'
|'not(' **attributeExpr** ')'
|'inverse(' **attributeExpr** ')'
|'symmetric(' **attributeExpr** ')'
|'reflexive(' **attributeExpr** ')'
|'trans(' **attributeExpr** ')'

Relation expressions are defined similar to class expressions:

relationExpr ::= **relationID**
|'and(' **relationExpr relationExpr** { **relationExpr** } ')'
|'or(' **relationExpr relationExpr** { **relationExpr** } ')'
|'not(' **relationExpr** ')'

[1]Notice that the symmetric closure of an attribute is equivalent to the union of the attribute and its inverse: or(A inverse(A)).

[2]The reflexive closure of an attribute A includes for each value v in the domain a tuple with equivalent domain and range v: $\langle v, v \rangle$.

classCondition ::= ’attributeValueCondition(’ **attributeID** (**individualID** | **dataLiteral**) ’)’

classCondition ::= ’attributeTypeCondition(’ **attributeID classExpr** ’)’

classCondition ::= ’attributeOccurrenceCondition(’ **attributeID** ’)’

attributeCondition ::= ’valueCondition(’ (**individualID** | **dataLiteral**) ’)’

attributeCondition ::= ’typeCondition(’ **classExpression**) ’)’

Especially when mapping several source ontologies into one target ontology, different classes and relations need to be joined. Although apparently similar, a join mapping is fundamentally different from an intersection mapping.

Appendix B

Mapping Example

In this appendix, we show an example mapping, including the complete mapping, both in the form of patterns and in the form of concrete mapping rules.

B.1 Source ontology

Herewith the source ontology in OWL abstract syntax:

```
Class(Person partial restriction(name allValuesFrom(xsd:string))
                     restriction(age allValuesFrom(xsd:integer))
                     restriction(hasGender allValuesFrom(Gender))
                     restriction(marriedTo allValuesFrom(Person))
                     restriction(hasChild allValuesFrom(Person))
                     restriction(name maxCardinality(1))
                     restriction(age maxCardinality(1))
                     restriction(gender maxCardinality(1)))
EnumeratedClass(Gender male female)
Class(Parent partial Person
                    restriction(hasChild minCardinality(1)))
DatatypeProperty(name)
DatatypeProperty(age)
ObjectProperty(hasGender)
ObjectProperty(hasChild)
```

B.2 Target ontology

Herewith the complete target ontology of the running example:

Class(Human partial restriction(name allValuesFrom(xsd:string))
restriction(age allValuesFrom(xsd:integer))
restriction(noOfChildren allValuesFrom(xsd:integer))
restriction(name maxCardinality(1))
restriction(age maxCardinality(1))
restriction(noOfChildren maxCardinality(1)))

Class(Marriage partial restriction(hasParticipant allValuesFrom(Human))
restriction(date allValuesFrom(xsd:date))
restriction(location allValuesFrom(xsd:string))
restriction(hasParticipant cardinality(2))
restriction(date maxCardinality(1))
restriction(location maxCardinality(1)))

Class(Man partial Human)

Class(Woman partial Human)

DisjointClasses(Man Woman)

B.3 Mapping

Herewith the complete mapping between the source and target ontology, both in mapping-patterns format and in rules format, where we again use the WSML syntax because of its readability. For the source ontology we use the namespace prefix `o1`; for the target ontology we use the namespace prefix `o2`.

B.3.1 Mapping format

```
classMapping (o1:Person o2:Human
        attributeMapping(o1:name o2:name)
        attributeMapping(o1:age o2:age)
        )
```

```
classMapping (o1:Person o2:Man
        attributeValueCondition(hasGender male)
        )

classMapping (o1:Person o2:Woman
        attributeValueCondition(hasGender female)
        )
```

As we can see, the attribute `marriedTo` is not mapped to the class `Marriage`, because an attribute-class mapping is not possible to express in OWL. Furthermore, there is no mapping from the attribute `hasChild` to the attribute `noOfChildren`, because aggregate functions over datatypes are not supported in OWL.

B.3.2 OWL format

In this section we provide the translation to OWL of the previously mentioned mapping:

```
SubClassOf(o1:Person o2:Human)
SubPropertyOf(o1:name, o2:name)
SubPropertyOf(o1:age, o2:age)
SubClassOf(IntersectionOf(o1:Person restriction(hasGender value(male)))
            o2:Man)
SubClassOf(IntersectionOf(o1:Person restriction(hasGender value(female)))
            o2:Woman)
```