



D4.2.2 State-of-the-art survey on Ontology Merging and Aligning V2

Livia Predoiu (DERI)
Cristina Feier (DERI)
Francois Scharffe (DERI)
Jos de Bruijn (DERI)
Francisco Martín-Recuerda (DERI)
Dimitar Manov (SIRMA)
Marc Ehrig (UKARL)

Abstract.

EU-IST Integrated Project (IP) IST-2003-506826 SEKT

Deliverable D4.2.2 (WP4)

This deliverable contains a comprehensive state-of-the-art survey on Ontology Merging and Aligning methods, tools and specification languages.

Keyword list: Ontology Mediation, state-of-the-art survey, ontology merging, ontology aligning, ontology mapping

WP4: Ontology Mediation

Nature of the Deliverable:	Report	Dissemination level:	PU
Contractual date of delivery:	2005-12-31	Actual date of delivery:	2005-12-31

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
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH

Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck

Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contactperson: 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
Contactperson: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EOOD (Ltd.)

135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768, Fax: +359 2 9768 311
Contactperson: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autònoma 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
Contactperson: Pompeu Casanovas Romeu
E-mail: pompeu.casanovasquab.es

Changes

Version	Date	Author	Changes
0.1	2005-12-01	Cristina	creation and fill in a lot of the content of D4.2.1
0.2	2005-12-05	Livia	added more content
0.3	2005-12-10	Cristina	added more content
0.4	2005-12-19	Livia	added more content
0.5	2005-12-20	Cristina	Version for QA
0.6	2006-01-31	Livia	final Version for QA

Executive Summary

This report provides a state-of-the-art survey of Ontology Merging and Aligning methods, tools and techniques. In this deliverable, we provide a framework for the evaluation of these approaches, as well as a categorization of approaches for ontology merging and aligning.

We categorize the approaches in methods and tools, integration systems and specific techniques. We compare the approaches according to the evaluation framework and to a set of generic use cases for ontology mediation in order to evaluate the applicability of the approach to the ontology mediation problem on the Semantic Web.

This is the second version of this deliverable in the course of the SEKT project. We kept much of the content that was delivered in the first version because the systems did not change and parts of the state-of-the-art thus did not change. In this version we have refined the definitions for the common-encountered terms in the deliverable and we have updated the classification of the integration approaches. We have also added the description of five new systems that are remarkable and make a big contribution to the field of Ontology Mediation and updated the descriptions of the previous systems, where necessarily.

Contents

1	Introduction	3
1.1	Terminology	5
1.2	The Ontology Mapping Process	7
1.3	Ontology Mismatches	9
1.3.1	Ontology-level Mismatches	9
1.3.2	Language-level mismatches	11
1.4	Integration approaches	12
1.5	Wrappers and Mediators	18
2	Motivational Use Cases	21
2.1	Use Cases for Instance Mediation	21
2.1.1	Instance Transformation	22
2.1.2	Instance Unification	23
2.1.3	Query Rewriting	24
2.2	Ontology Merging	25
2.3	Creating Ontology Mappings	26
2.3.1	Finding Similarities	26
2.3.2	Specifying Mappings	26
3	The Evaluation Framework	27
4	The Survey	30
4.1	Methods and Tools	31
4.1.1	MAFRA	31
4.1.2	RDFT	40
4.1.3	PROMPT	43
4.1.4	GLUE	50
4.1.5	Semantic Matching	52
4.1.6	OntoMap	56
4.1.7	RDFDiff	60
4.1.8	OntoMerge	63
4.1.9	OMEN	65
4.1.10	WSMT Data mediation module	67

4.1.11	DOME mapping module	72
4.2	Integrated Systems	74
4.2.1	InfoSleuth	74
4.2.2	ONION	78
4.2.3	OBSERVER	81
4.2.4	MOMIS	87
4.2.5	INFOMIX	90
4.2.6	AutoMed	95
4.3	Specific Techniques	98
4.3.1	FCA-Merge	98
4.3.2	OntoMorph	99
4.3.3	QOM Quick Ontology Mapping	99
5	Comparison of the Methods	101
5.1	Ontology Languages	101
5.2	Mapping Language	103
5.3	Mapping Patterns	105
5.4	Automation Support	105
5.5	Applicability to Use Cases	105
5.6	Implementation	107
5.7	Experiences	107
6	Conclusions	110
	Bibliography	113

Chapter 1

Introduction

This report presents a state-of-the-art survey of ontology mapping and merging methods and tools with an emphasis on ontology mapping and inter-operability over the Semantic Web.

This is the second version of this deliverable in the course of the SEKT project. We kept a lot of the content that was delivered in the first version unchanged because the systems did not change and parts of the state-of-the-art thus did not change as well. In this version we have refined the definitions for the common-encountered terms in the deliverable and we have updated the classification of the integration approaches. We have also added the description of five new systems that are remarkable and make a big contribution to the field of Ontology Mediation and updated the descriptions of the previous systems, where necessarily. The five new systems which were added are OMEN, WSMT Data Mediation module, Dome mapping module, INFOMIX and AutoMed.

The issue of inter-operability between information systems has already existed for many years. With the recent advent of the Semantic Web [BLHL01], the issues have only increased, because of the abundance, heterogeneity and independence of the various data sources.

Traditional data integration systems focus on inter-operability between data sources and applications within enterprises. Within enterprises a certain coherence between data sources can be expected, although data integration within enterprises still faces many challenges which remain to be resolved.

We focus on information integration on the Semantic Web. This means that we not only take into account data integration within organizations¹, but also explicitly address integration across organizational boundaries. Between organizations, even more heterogeneity between the data sources can be expected.

Fortunately, ontologies [Fen03], the backbone of the Semantic Web, can help us with a

¹Arguably, the intranet of an organization is an isolated part of the Web and could be part of the Semantic Web in the near future

part of the integration problem. Because ontologies are *explicit* and *formal* specifications of knowledge, they help in disambiguating data and can help in finding correspondences between data sources because of the explicit specification of the knowledge in an ontology.

On the Semantic Web, data is annotated using ontologies. Concepts (also called *classes*)² in ontologies give meaning to data on the Web. Because ontologies are *shared* specifications, the same ontologies can be used for the annotation of multiple data sources, not only limited to Web pages, but also collections of xml documents, relational databases, etc. This enables a certain degree of inter-operation between data sources using the same ontologies, because of their shared terminology³. However, it cannot be expected that all individuals and organizations on the Semantic Web will ever agree on using one common terminology or ontology. It can be expected that many different ontologies will appear and, in order to enable inter-operation, mediation is required between these ontologies.

As was argued in [VC98, Usc00], it is very hard to create standard ontologies. In fact, even inside organizations the standardization of a terminology is not feasible, because of the lengthy process of standardization and because the use of a big standard impedes changes in the organization (any change would require consensus among a large group of people, which is hard to achieve). Across organizations this problem becomes more severe, because the group of people which need to reach consensus is much bigger and conflicts of interest are more likely to occur. Therefore, it is likely that there will be many different heterogeneous ontologies on the Semantic Web and in order to enable interoperability between applications on the Semantic Web, *mediation* is required between different representations (ontologies) of knowledge in the same domain.

This report presents a survey on the state of the art of ontology mapping, merging and mediation. It includes both well-known approaches in database integration and recent approaches specifically addressing ontology mapping on the Semantic Web. The Semantic Web has a number of distinguishing features relative to older data integration systems:

- The Semantic Web relies heavily on *standardization* of both the protocols for the transport of data (HTTP) and the syntax for the specification of data and knowledge (RDF [LS99] and OWL [DS04]).
- In contrast with the database schemas in many data integration systems, the semantics of the data are made explicit through a logic-based language.
- Ontologies capture knowledge in a way understandable to humans and processable by machines. Furthermore, ontologies ideally represent a *consensual* view of a particular domain, which is shared among a group of people.

²We use the words *concept* and *class* interchangeably in this document.

³Note that this is not the end of the story. For many applications it is necessary to detect whether pieces of data, coming from different data sources annotated with the same ontology, actually refer to the same thing. This is a challenge we also address in the course of the SEKT project; it is described in more detail in Section 2.1.2 of this report.

These features help in the task of ontology mediation. For example, because of the standardization of the languages on the Semantic Web, syntax does not play a big role, so the mediation can focus on the semantics of the data.

Note that Semantic Web technologies can not only be put to use on the World Wide Web. They can also be employed within company intranets in order to achieve interoperability between applications within an organization.

This chapter is further structured as follows. We first clarify the terminology used in this survey in Section 1.1. In Section 1.2 we explain the *ontology mapping process* as we see it, which is used during the survey to identify the use of certain methods in this process. Then, we present a list of mismatches, which can occur between ontologies, in Section 1.3. In Section 1.4 we summarize different approaches to achieve integration of multiple heterogeneous data sources. We conclude with a few remarks about the wrapper/mediator architecture, which is used in several of the approaches in this survey, in Section 1.5.

1.1 Terminology

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

Ontology A widespread definition for the notion of ontology is "an explicit specification of a conceptualization"[Gru94]. In the following we consider an ontology to be 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. A concept in an ontology is a category of real or virtual objects of interest, which we call instances. Relations specify how different concepts and instances are interconnected, and axioms are formulae that allow to further specify interdependencies between all mentioned ontology entities. All concepts, relations, instances, and axioms are specified in some logical language. Note that the sets C and I are not necessarily disjoint (e.g. the same term can denote both a class and an instance), although the ontology language often requires this.

In an ontology, concepts are usually organized in a subclass hierarchy, through the *subclass-of* (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. [SM01], [DMDH04]). Note that this does not

rule out the situation where instances use several ontologies for their description. However, most approaches in this survey which make use of instances assume a collection of instances described by *one* ontology.

Ontology Language The ontology language is the language which is used to represent the ontology. Popular ontology languages for the Semantic Web are RDFS [BG04] and OWL [DS04]. Semantic Web ontology languages can be split up into two parts: the logical and the extra-logical parts. The *logical* part usually amounts to a theory in some logical language, which can be used for reasoning. The logical part basically consists of a number of logical axioms, which form the class (concept) definitions, property (relation) definitions, instance definitions, etc.

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) and other extra-logical statements, such as namespace declarations, ontology imports, versioning, etc.

Non-functional properties are typically only for the human reader, whereas many of the other extra-logical statements are machine-processable. For example, namespaces can be resolved by the machine 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 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.

Ontology Mapping An *ontology mapping* M is a (declarative) specification of the semantic overlap between two ontologies O_S and O_T . The correspondences between different entities of the two ontologies are typically expressed using some axioms formulated in a specific mapping language. Mappings can be unidirectional (that specify how terms in one ontology can be expressed using terms from the other ontology) or bidirectional (that work in both ways, i.e terms from one ontology are expressed using terms from the other and the other way around).

Mapping Language The mapping language is the language used to represent the *ontology mapping* M . It is important here to distinguish between a specification of the similarities of entities between ontologies and an actual ontology mapping. The specification of similarities between ontologies is usually a level of confidence (usually between 0 and 1) of the similarity of entities, whereas an ontology mapping

actually specifies the relationship between the entities in the ontologies. This is typically an exact specification and typically far more powerful than simple similarity measures. 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. [PGM98]). 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.

Matching We define *ontology matching* 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. [RB01]). Any schema matching or ontology matching algorithm can be used to implement the *Match* operator, e.g. [DMDH04, GSY04, MBR01, MRB03].

The specification of similarities typically serves as an input to the ontology mapping (see also Section 1.2).

Merging Ontologies Merging refers to the creation of one ontology from two or more source ontologies. The new ontology will unify and in general replace the original source ontologies (see also Section 1.4).

Aligning Ontologies Specifying how the concepts in the different ontologies are related in a logical sense. 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. This concept is very similar with Ontology Mapping, however it is a more general concept: two ontologies can be aligned by creating an ontology mapping.

1.2 The Ontology Mapping Process

In order to clarify the role of many of the methods, tools and techniques in this survey, we will explain in this section what we see as the *ontology mapping process*. Many of the tools and techniques in this survey form a part of the overall mapping process and the integration systems typically form a superset of the mapping process, i.e. they typically

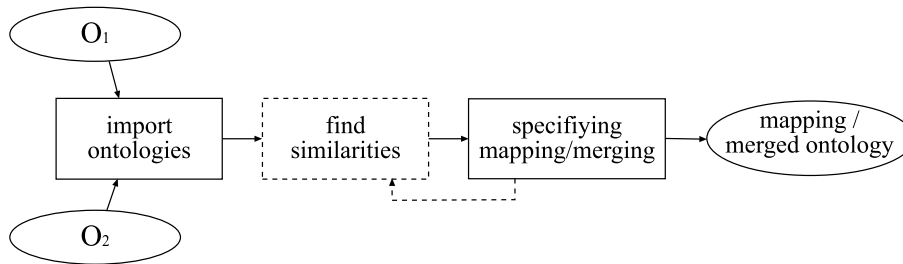


Figure 1.1: The Ontology Mapping Process

incorporate the complete mapping process, but also offer additional functionality, such as the use of the mapping to perform the actual querying and data integration.

First, we have to note that for simplicity we assume only two⁴ different ontologies O_1 and O_2 , which describe the same or similar domains, as input to the mapping process. The outcome of the mapping process is either a mapping M , which describes how O_1 and O_2 are related, or a new ontology O_M , which is the merger of O_1 and O_2 .

Figure 1.1 depicts the different phases in the generic mapping process as we see it. Not all phases are necessarily incorporated in every mapping tool and several phases in the process are optional. We distinguish the following phases (in temporal order) in the mapping process:

1. *Import of ontologies* Ontologies can be specified in different languages, which indicates a need to convert them to a common format in order to be able to specify the mapping. Furthermore, the ontologies need to be imported in the tool, which is used to specify the mapping.
2. *Finding Similarities* Many systems use the *Match* operator to (semi-)automatically find similarities between schemas or ontologies. For any two source ontologies, the Match operator returns the similarities between the ontologies. We distinguish this phase in the mapping process only when the similarities are determined in an automatic fashion. If the mapping process is completely manual, this phase is skipped.
3. *Specifying Mapping/Merging* After (potential) similarities between ontologies have been found, the mapping between the ontologies needs to be specified. This specification is usually a manual process, but it can be aided by a tool. PROMPT [NM00b], 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

⁴It is straightforward to scale up this approach to more than two ontologies.

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.

The three phases of the mapping process are specified at a very high level. Many of the approaches in this survey provide a more detailed description of (part of) this mapping process (e.g. PROMPT, Section 4.1.3, MOMIS, Section 4.2.4).

1.3 Ontology Mismatches

Different types of mismatches can occur between different ontologies. It is important to identify which kind of mismatches can and do occur between ontologies, in order to resolve these mismatches in the mapping or the merge of ontologies. The classification of ontology mismatches is also important to denote which kind of mismatches can be resolved with a particular mapping formalism (language) and which kind of mismatches can be detected with a particular matching algorithm.

Klein [Kle01] identifies two levels of mismatches between ontologies. The first level is the ontology language or meta-model level. These mismatches include syntactic mismatches, differences in the meaning of primitives in the different languages, and differences in the expressivity of the languages. We will describe these mismatches in more detail in Section 1.3.2. The second level of mismatches is the ontology or model level, which is described below.

1.3.1 Ontology-level Mismatches

Where mismatches at the language level include differences in encoding and meaning of language constructs, mismatches at the ontology level include mismatches in the meaning or encoding of concepts in different ontologies. Klein follows the basic types of ontology mismatches identified in [VJBCS97]:

- *Conceptualization mismatches* are mismatches between different conceptualizations of the same domain.
- *Explication mismatches* are mismatches in the way a conceptualization is specified.

[Kle01] distinguishes two different conceptualization mismatches:

Scope mismatch Two classes have some overlap in the extension (the set of instances), but the extensions are not exactly the same. [VJBCS97] call this a *class mismatch* and work it out further for classes and relations.

Model coverage and granularity This mismatch is a difference in the part of the domain that is covered by both ontologies or the level of detail with which the model is covered.

Klein furthermore distinguishes different types of explication mismatches. First, there are two mismatches in the style of modeling:

Paradigm These mismatches occur when different paradigms are used for the explication of the same concept. For example, one ontology might represent time using intervals, while another ontology might use points to represent time.

Concept description Mismatches in the way a concept is described. For example, differences in the way the subclass-of hierarchy is built or when in one ontology several subclasses are defined for groups of instances, while in the other ontology subclasses are created for these different groups.

Then there are the terminological mismatches:

Synonym terms Two terms are equivalent when they are semantically equivalent, but are represented by different names. It is possible to use dictionaries or thesauri to resolve this problem, but one should be aware of possible scope differences (see the first conceptualization mismatch above).

Homonym terms This problem occurs when semantically different concepts have the same name.

Finally, the last type of difference:

Encoding Values in different ontologies might be encoded in a different way. For example, one ontology might define distance in kilometers, while another uses miles.

Inter-ontology relationships [MIKS00] takes a slightly different approach. This paper identifies different types of inter-ontology relationships (based on relationships identified in [HM93]) that should be taken into account by ontology mapping systems:

Synonym Two terms in different ontologies have the same semantics. This corresponds to the synonym terms mismatch mentioned above.

Hyponym A term is less general than another one in a different ontology. This is a special kind of scope mismatch and can also be seen as a concept description mismatch.

Hypernym A term is more general than another one in a different ontology. This is a special kind of scope mismatch and can also be seen as a concept description mismatch.

Overlap There is an intersection in the abstraction represented by two terms. This corresponds to the scope mismatch.

Disjoint There is no intersection in the abstraction represented by two terms.

Covering The abstraction represented by a term in one ontology is the same as the abstraction represented by the union of other given abstractions which are subsumed individually by the term. This corresponds to the granularity mismatch identified by Klein.

1.3.2 Language-level mismatches

Typically, ontology mappings require the source and target ontologies to be represented in the same language. This translation may already resolve most of the language issues, which can occur. Typical language level mismatches are *syntax*, *logical representation*, *semantics of primitives* and *language expressivity* [Kle01]. Most systems presented in this survey do such a translation but do not say if and how the issues surrounding language level mismatches are resolved in the translation to the internal representation.

Also, many methods and tools for matching and mapping require the source ontologies to be expressed in a certain representational format. Although there typically exists a translation from any ontology language into this particular representation, the preservation of semantics can still not be guaranteed.

We will now go over the above mentioned language level mismatches and describe what the (potential) issues are with the current systems and techniques:

- Certainly, differences in *syntax* would be resolved by any such translation to an internal representation, since both ontologies then use the same syntax.
- Differences in *logical representation* occur when syntactically different, but logically equivalent statements are used to represent the same thing. An example of this is the way disjointness is expressed in the OWL Lite species of the Web Ontology Language OWL [MvH04], compared to the way disjointness is usually expressed in the OWL DL species⁵.

Arguably, this is not really an issue with the language itself, but rather an issue with the use of the language. However, when a language allows the user to model the same thing in different ways, it is easy for a user to mistakenly model certain things in an inconvenient way and it is harder for a user to understand the model created by a different user or indeed created by him/herself in the past. When the ontology language used by the technique/tool/system allows such different logical representations of equivalent statements, this mismatch still needs to be taken into account

⁵The OWL Lite statement `Class(owl:Nothing complete A B)`, although also valid in OWL DL, is usually modeled as `DisjointClasses(A B)` in OWL DL

in the ontology mapping process. In order to overcome these issues, one could think of a normalization step before the start of the mapping process or reasoning during the mapping process in order to detect equivalence in logical expressions.

- When the *semantics of primitives* is different in different ontology languages, i.e. a syntactically equivalent construct has a different meaning in the different languages, the translation to the common representation needs to take this into account. Fortunately, this problem can be resolved in the translation to the common representation. If both ontologies already use the common representation and this common representation does not allow ambiguous statements, this mismatch does not occur.
- Differences in *expressivity of the languages* are resolved in the translation to the common representation language. However, if the expressivity of the common representation language is not a superset of the language of the source ontology, some semantics might get lost in the translation, as was pointed out in [MWK00].

As we can see, the issues with language mismatches are less severe if there is a translation to a common representation. However, all ontology mismatches mentioned previously need to be taken into account when creating any mapping between ontologies.

1.4 Integration approaches

We identify two major paradigms in information integration: (1) merging data models into a central model and (2) aligning and mapping models. In the ontology engineering community these approaches are known as Ontology Merging and Ontology Aligning.

We distinguish two distinct approaches in ontology merging. In the first approach the input of the merging process is a collection of ontologies and the outcome is one new, merged, ontology which captures the original ontologies (see Figure 1.2). A prominent example of this approach is PROMPT [NM00b], which is an algorithm and a tool for interactively merging ontologies. In the second approach (see Figure 1.3) the original ontologies are not replaced, but rather a ‘view’, called *bridge ontology*, is created which imports the original ontologies and specifies the correspondences using *bridge axioms*. OntoMerge [DMQ02] is a prominent example of this approach.

[NM99] clarifies the difference between ontology merging and ontology aligning. When merging two ontologies, a single coherent ontology is created that is a merged version of the two original ontologies. When aligning two ontologies, the two original ontologies persist, with a number of links established between them, allowing the aligned ontologies to reuse information from one another. The alignment of ontologies is usually part of the ontology merging process.

Solutions can be further classified along two dimensions: a *run-time* and a *design-time* dimension. The run-time dimension concerns with the way the user views the data in the

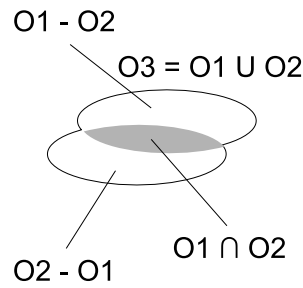


Figure 1.2: Output of the merging process - case 1

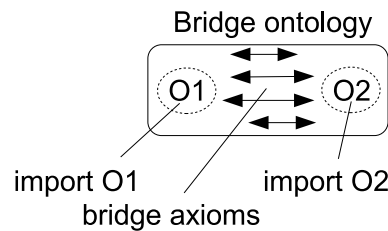


Figure 1.3: Output of the merging process - case 2

system during operation. The design-time dimension concerns with the way the models of the disparate data sources are integrated.

In the run-time, or user-centered dimension we distinguish two approaches: (1) *the local model* and (2) *the global model* approach. The difference between these two approaches is whether, in interactions with the system, the user can use his/her own local data model, or whether the user needs to conform to a global model when interacting with the system:

The Local Model/Ontology Approach. In this case the user is represented by an agent in the system and this agent represents the user with its own local data model. The agent performs the translation between the user's local model and either the global model or other local models in order to allow interaction with multiple data sources in the system.

The Global Model/Ontology Approach. The user views the system through the global data model using a mediator, which is "a system that supports an integrated view over multiple information sources"[Hul97]. Note that in the local model approach, a user agent will in most cases also contact a mediator in order to allow inter-operation with the system, which contains multiple information sources. An example is the approach taken in the COG project[db04].

In the design-time dimension we distinguish:

One-to-one mapping of ontologies. Mappings are created between pairs of ontologies. Problems with this approach arise when many such mappings need to be created,

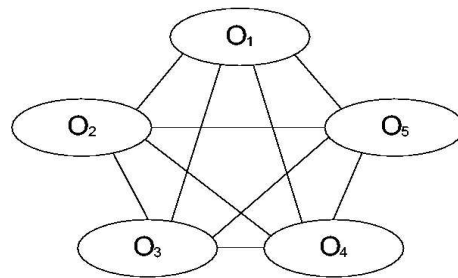


Figure 1.4: The one-to-one approach for ontology integration

which is often the case in organizations where many different applications are in use. The complexity of the ontology mapping for the one-to-one approach is $O(n^2)$, where n is the number of ontologies. An example of the one-to-one approach is OBSERVER [MIKS00] (see also Section 4.2.3). Figure 1.4 illustrates one-to-one mapping of ontologies. There exists a mapping between every pair of ontologies. In the worst case, these mappings are only one-way. This means that a single mapping can only translate from one model to another, not the other way around.

Mapping to a global ontology. Each ontology is mapped to a central shared ontology (Figure 1.5). Drawbacks of using a global ontology to which all other ontologies are mapped are similar to those of using any standard [VC98]. For example, it is hard to reach a consensus on a standard shared by many people (it is always a lengthy process), who use different terminologies for the same domain and a standard impedes changes in an organization (because evolution of standards suffers

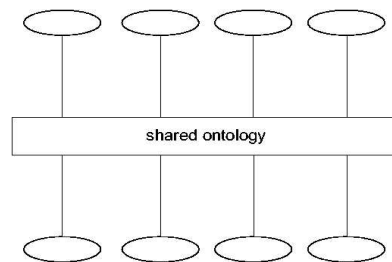


Figure 1.5: The global approach for ontology integration

from the same problems as the development of standards). An example of the global ontology approach is MOMIS [BCVB01] (see also Section 4.2.4).

Hybrid approach. On the Web, not much control over the use of ontologies can be enforced and the global integration scenario does not scale, because different organizations use different ontologies and do not want to commit to a single new ontology. However, the one-to-one integration approach also does not scale, because it requires the maintenance of too many mappings between ontologies. Therefore, a hybrid approach is suitable, where there are several "islands" around influential domain ontologies, where within the island there is a form of global integration; one ontology would be the global ontology of the islands and a number of local ontologies are mapped to this global ontology. Mappings between the islands are established, as illustrated in Figure 1.6.

Note that many methods, tools and techniques in the survey do not have a bias for one

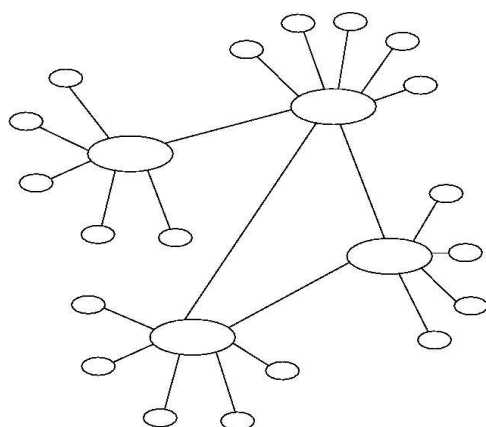


Figure 1.6: The hybrid approach for ontology integration

of the above approaches. They can often be used in all scenarios, although an ontology merging tool such as PROMPT [NM00b] does seem to have a bias towards using a shared ontology.

The more comprehensive integration systems typically prescribe which paradigm should be used. MOMIS [BCVB01], for example, prescribes the use of a global merged ontology for the integration of data sources, whereas OBSERVER [MIKS00] prescribes loosely coupled component ontologies with mappings between the ontologies.

The approaches that fall in the global-approach category can be further classified by the directionality of the mappings between the local ontologies and the shared ontology. This distinction is inherited from the database setting, where the data integration task is seen as a process by which several databases with **associated local schemas** are integrated to form **a virtual database** (thus it conforms to the global approach), and views connect local schemas with the global virtual schema. The terms LAV, GAV, and GLAV were introduced for describing different approaches in database integration [Len]:

GAV (Global as view). The constructs of a global schema are described as views over the local schemas (in database terminology) or the mappings describe entities from the shared ontology in terms of entities from the local ontologies (in ontology mediation terminology).

In the case where there are no integrity constraints in the global schema, query processing is rather simple, consisting in unfolding the original query. Unfolding means expanding the terms of the query, which are constructs of the global schema, according to the mappings, in order to obtain a query which contains only constructs from local schemas. Unfortunately, in the presence of integrity constraints (which is the case when the global schema is an ontology), additional steps must be performed for reasoning [CDGL01].

One disadvantage of this method is that whenever a source changes or a new one is added, the global schema needs to be reconsidered. Another disadvantage mentioned in [EA] is that it cannot fully capture the data integration semantics where a source schema construct can be defined by a non-reversible function over global schema constructs.

LAV (Local as view). The constructs of local schemas are described as views over the global schema (in database terminology) or the mappings define entities from the local ontologies in terms of entities from the global ontology (in ontology mediation terminology). This approach is characterized by a high modularity and extensibility (the local schemas are expected to be the subject of change more frequently than the global schema and changing a local schema does not require changes in the global one). A disadvantage of this method is the complexity of reasoning. Like GAV, there are situations where the approach is not expressive enough for fully capturing the data integration semantics [EA].

GLAV (Global-local as view). This approach can be seen as an extension of both LAV and GAV. In this case a view on the local schema is defined with respect to a view on the global schema (a mapping is an inclusion/equivalence relation between a query over the global ontology and a query over a shared ontology). Query processing is complex in this case as well, since there is no direct information about which data satisfy the global schema. One reasoning technique is to transform a GLAV rule into a GAV rule and an integrity constraint over the global schema[AC01]. An advantage of this method is that it is more expressive than both LAV and GAV⁶

Recently, as part of the AutoMed framework (see Section 4.2.6), a new approach for connecting schemas called BAV(Both as view) was introduced [MP03],[EA] that is more expressive than LAV/GAV/GLAV. Below we give an overview of this approach:

BAV (Both as view). This approach is based on the use of reversible sequences of primitive schema transformations, called **transformations pathways**.

Schemas are incrementally transformed by applying to them a sequence of primitive transformation steps t_1, \dots, t_n , where each such step adds, extends, deletes, contracts or renames a single schema concept. A query expressed in the **intermediate query language (IQL)** is attached to each `add` or `delete` step and specifies the extent of the new or the deleted construct in terms of the rest of the constructs in the schema.

[MP03] shows how the LAV and GAV definitions of views can be fully derived from BAV schema transformation sequences, and how BAV transformation sequences may be partially derived from LAV or GAV definitions. A network of pathways may join different schema together. Thus, this approach for specifying mappings could also be applied for the case of one-to-one mapping and for the hybrid approach.

1.5 Wrappers and Mediators

In the wrapper/mediator architecture, the main components are *wrappers*; there typically exists one wrapper for each data source, and one (or more) *mediator(s)*, which mediate between the differences in the individual data sources. In the global integration paradigm, there is typically one mediator which is accessed by the user for querying and information retrieval. In this case the mediator typically has one global schema along with mappings to all the local schemas, where each data source has one local schema. Each data source has a wrapper associated with it, which provides the translation between the representation of the data source and the system representation (this is typically between a database representation and the ontology representation) and translates queries from the system representation to the data source representation (typically, ontology queries would

⁶see <http://www.science.unitn.it/coopis/talks/Wednesday/Lenzerini.pdf> for an example

be translated to SQL queries to be executed on the individual database). An example of this approach is MOMIS [BCVB01] (see also Section 4.2.4). This wrapper/mediator approach with one global mediator is illustrated in Figure 1.7.

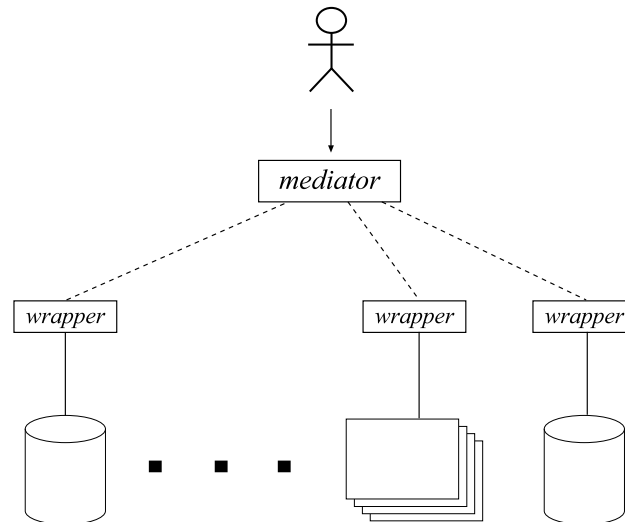


Figure 1.7: The wrapper/mediator architecture in the case of global integration. Note that the drums represent as usual databases and the sheets represent other data sources.

In a one-to-one integration scheme, there are typically several mediators, which mediate between the representations in the individual sources. The setting here is similar to a peer-to-peer setting, where each peer could have a number of data sources and an ontology, which describes the data of the source. The user would be one peer and would use that peer's ontology. If the user wants to query a different peer, the mediator has to mediate the differences between the ontologies. One example of this case is OBSERVER [MIKS00] (see also Section 4.2.3), where each peer has its own mediator, which does the query rewriting and querying of other peers, although one central mediator (called the Inter-ontology Relationship Manager IRM) still keeps track of the relationships between the ontologies. This central mediator is queried by other mediators to find out about related peers and the differences in representation.

Figure 1.8 illustrates the use of wrappers and mediators in the case of one-to-one integration. Note that in this case, all mediators need to be aware of all other mediators in order to perform query rewriting and to achieve effective query answering.

This report is further structured as follows. Chapter 2 presents the use cases which we have identified as crucial for ontology mediation on the Semantic Web. These use cases are later used to identify if and how the approaches in the survey would fit into a Semantic Web context. The framework we use for evaluating the approaches in this survey is presented in Chapter 3. The survey itself is presented in Chapter 4. Chapter 5 compares the approaches in the survey and Chapter 6 presents some conclusions.

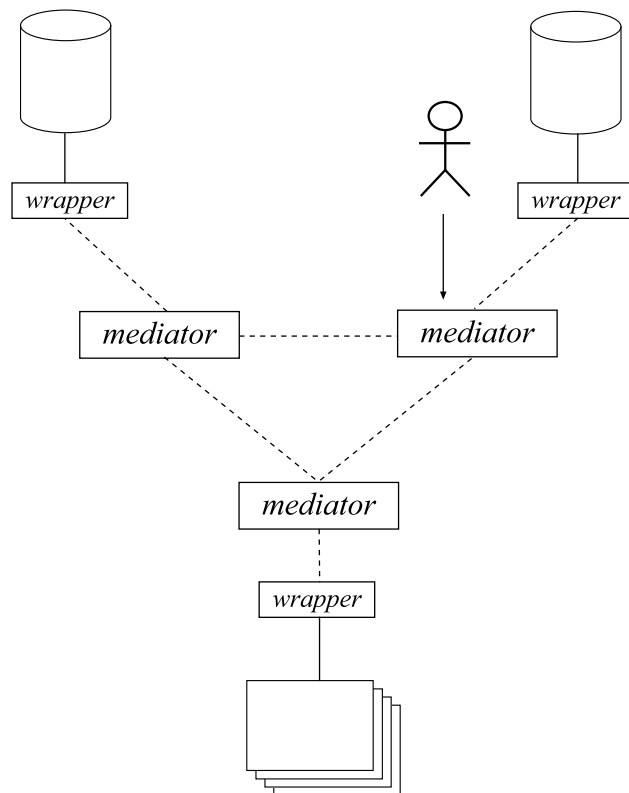


Figure 1.8: The wrapper/mediator architecture in the case of one-to-one integration

Chapter 2

Motivational Use Cases

In this chapter we present a number of generic use cases which capture the functionality required for ontology mediation on the Semantic Web. Any application of ontology mediation is expected to use all these use cases to some extent. Therefore it is interesting to see to what extent each of the approaches in this survey supports these use cases in order to evaluate their applicability to the ontology mediation problem on the Semantic Web. The Ontology Mediation framework developed in the SEKT project should support as many features as possible from the ones described here.

We distinguish three use cases, which are detailed in the remainder of this chapter:

- Instance Mediation
- Ontology Merging
- Creating Ontology Mappings

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.

The generic use cases correspond to three orthogonal dimensions in ontology mediation. Each application scenario can make use of all three use cases to some extent.

2.1 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.

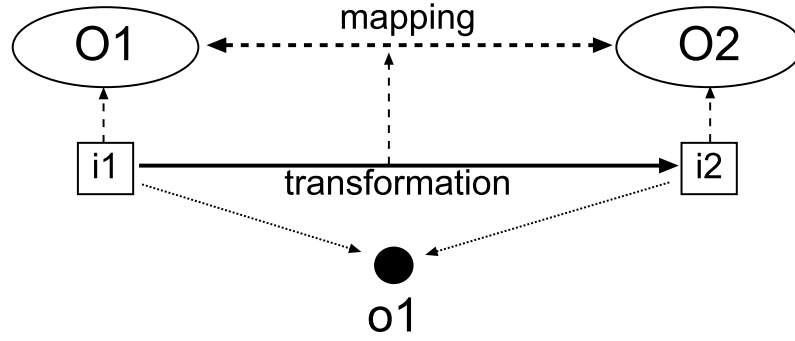


Figure 2.1: Instance Transformation

Definition 1 We define instance mediation as the process of reconciling 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 and instance transformation.

As we can see in the definition, 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 their use in different other areas of ontology mediation.

2.1.1 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 $\mathcal{O}1$, is transformed into instance $i2$, which refers to ontology $\mathcal{O}2$. What is important to note here is that the transformation itself is derived from the mapping between the two ontologies, and that both the original and the transformed instance provide information about the same real-world referant. Note that a real-world referant is not necessarily a physical object, but can also be, for example, a brand-name product, a date, an event or a message.

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 .

Different application scenarios have different requirements on the precision and coverage of the transformation. With *precision* in this context we mean the degree to which the intended meaning of the instance is preserved in the transformation. With *coverage*

we mean the fraction of instances that are intended to be transformed, which are actually transformed. The requirements of the application determine what these measures look like.

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 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.

2.1.2 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 referant. 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 ¹. Therefore, the instance unification task can be decomposed into (1) the identification of instances referring to the same real-world referant 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 referant, 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 $i2$) in the same ontology $O1$, which refer to the same real-world referant $r1$, are unified into one new instance, $i0$, which is also an instance in the ontology $O1$ and also describes the same real-world referant $r1$.²

We identify two general means of detecting whether two instances refer to the same real-world referant:

- In the ‘exact’ case, the ontology mapping specifies precise, exact conditions which unambiguously specify in which cases two instances refer to the same real-world referant and in which cases they refer to different real-world referant. In other words, in which cases the instances are unifiable.
- In the ‘probabilistic’ case, a similarity measure is created on the basis of the ontology mapping. The similarity measure expresses the probability that both instances

¹Note that I_0 could coincide with either I_1 or I_2 , which would be a less general case of the one described here.

²Note that contradictory information may result from unstated assumptions in an ontology or instance base. Unification of contradictory statements could be performed by adding in the appropriate assumption as a condition.

For example, “Canada [a real-world object] requires inbound travellers to carry passports” is true if the assumption is that travellers are EU citizens, but not if they are US citizens entering from the US.

Such techniques are not explored in this deliverable.

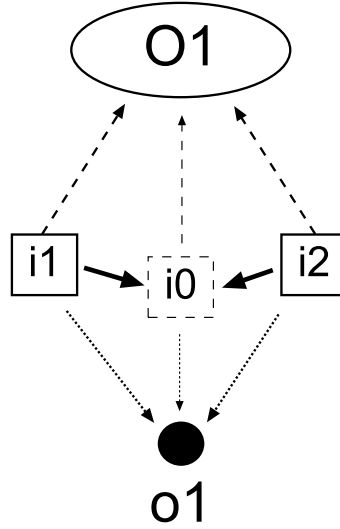


Figure 2.2: Instance Unification

refer to the same object. A threshold could be used to decide whether to unify the instances. Another possibility is to have the user decide about the unification, which is clearly undesirable in the general case, but could be useful when the accuracy of the resulting ontology is crucial or when dealing with very few instances.

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 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.

2.1.3 Query Rewriting

An operation occurring very frequently in Knowledge Management applications 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 application wants 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

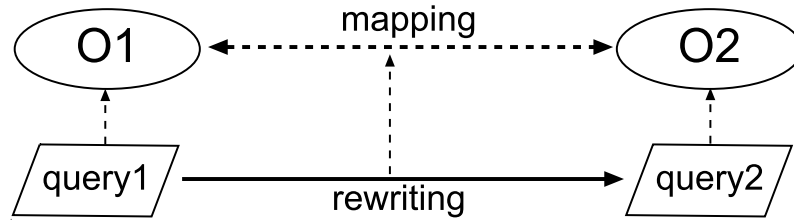


Figure 2.3: Query Rewriting

A now formulates a query 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 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.

2.2 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 [NM00b], two source ontologies shall be merged into one target ontology based on the source ontologies. In the general case, the source ontologies would disappear or become unavailable and only the target (merged) ontology remains. A special case is when the source ontologies remain, along with mappings to the merged ontology. Note that the target (merged) ontology could coincide with one of the source ontologies.

In the general case where the source ontologies become unavailable after the merge, the complete instance stores of the source ontologies have to be merged. In the special 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) might be necessary³. We can compare these two distinct cases with notions developed in the field of database integration, namely, the notions of *materialised* and *virtual* views [Hul97] respectively.

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.

³Note that mappings only can be derived when there is a semantical overlap. If there is no semantical overlap, the set of mappings is empty and thus instance transformations and instance unifications are not performed.

2.3 Creating Ontology Mappings

In order to be able to support the previously mentioned use cases, generally, a mapping needs to be created between the source and the target ontology. This does not apply to the case of ontology merging where the source ontologies do not remain. Because the source ontologies disappear, there needs to be no ontology mapping between these sources and the new merged ontology. However, the techniques for finding concepts to be merged in different ontologies and finding mappings between concepts in different ontologies are the same, since they are both based on the similarity of concepts. In fact, a mapping between two ontologies can be used as a basis for the merged ontology. In the case of Ontology Merging where the source ontologies remain, a mapping needs to be created between each source ontology and the merged ontology.

We split the "Creating Ontology Mappings" use case into two distinct use cases: finding similarities between ontologies and specifying mappings between ontologies.

2.3.1 Finding Similarities

In order to determine which mappings need to be created, the similarity between ontologies needs to be established. This can either be established manually or automatically using the so-called *Match* operator (cf. [RB01]). The *Match* operator takes as input two ontologies and returns as output a list of similarities between entities in the two source ontologies. These similarities can now be used as a starting point to semi-automatically create a mapping between the ontologies or to merge the two ontologies (cf. [NM00b]).

2.3.2 Specifying Mappings

After having defined the similarities between entities in the different ontologies, a mapping needs to be specified between the similar entities of the ontologies. The requirements of this mapping depend on the application scenario (cf. the various scenarios described in the next section) and in general the requirements of ontology mediation, as mentioned in the introduction.

Chapter 3

The Evaluation Framework

This chapter presents the framework used for evaluating and comparing different approaches in ontology merging and aligning, as well as data integration using ontologies. This framework is set up in such a way that it enables us to evaluate the applicability of the approaches to an ontology mediation setting in the Semantic Web context. Each of the approaches in the survey is described according to these criteria. If one of the criteria is not applicable to the approach, it will be omitted.

Summary of the approach We first summarize the approach to give the reader a feeling for what the approach is all about.

Ontology Languages For each tool or method we describe which ontology languages are supported as sources and targets of the mapping between ontologies. Furthermore, we describe how the ontology languages relate to the mapping language employed by the approach. In many cases the same language is used for both the ontologies and the mappings. In some cases, this can have drawbacks if the ontology language is not expressive enough to capture all the required ontology mappings.

Mapping language An important aspect in ontology mediation is the mapping language which is used to actually specify the mapping. The mapping language determines to some extent the complexity of creating mappings and also the possibilities of automation in creating the mappings and in transforming and unifying instances. The most important aspects of an ontology mapping language are its expressivity (i.e. what kind of relations between the ontologies can be expressed) and its usability.

An important aspect of a mapping language is the types of mappings that are supported, in other words the expressivity. We can distinguish several types of mappings here. The following is an (incomplete) list of types of mappings:

- Class mappings
- Property (i.e. relation) mappings

- Instance mappings
- Axioms / rules / constraints
- Value transformations (for properties)
- Conditional mapping

As was pointed out in Section 1.3, there are several mismatches between ontologies, both on the language and the ontology level. A mapping language needs to take these mismatches into account. These mismatches mostly concern the ontology level, although there are still some issues remaining on the language level, as was pointed out in Section 1.3.

Two notes about the mapping language with respect to the approach in the survey are in order here. Firstly, an ontology merging tool (e.g. PROMPT [NM00b]) does not produce a mapping and therefore does not need a mapping language. Secondly, we describe several methods and tools for ontology matching in this survey. These approaches typically do not produce a mapping, but rather a specification of similarities between entities in the ontologies.

Mapping Patterns One of the major goals of Work Package 4 in the SEKT project is to investigate the use of patterns for the creation of ontology mappings. One of the tasks is to find such patterns. Therefore, it would be interesting to see if and how existing approaches cope with this and how mapping patterns could be integrated. This issue is very closely related to the mapping language.

Automation support We describe the type of automation that is supported and the degree to which it is supported during creation of the ontology mapping. Ontology mapping can not be fully automated; the mapping process will always be an interactive one.

One important aspect in the automation support is the use of external information sources, such as domain-specific lexicons or existing ontologies or data schemas.

Applicability to use cases In order to see if and how an approach can be applied to our setting of ontology mediation in the Semantic Web we analyze the applicability of each approach to the use cases presented in Chapter 2. More specifically, we relate each approach to the following use cases:

- Instance Transformation
- Instance Unification
- Query Rewriting
- Ontology Merging

We will not treat the applicability of each of the approaches to each of the use cases in detail, but rather give an indication about the (in)applicability to each of the use cases.

Implementation For each approach we describe the tool support developed for the particular method. We distinguish the following two categories of tools:

- Tools that support the user in creating the mappings (and merging the ontologies). These tools fall in two categories: (1) components that implement the *Match* operator to find similarities between ontologies and (2) GUI tools that aid the user in specifying the mappings between the ontologies.
- Tools that do the run-time mediation. These tools take care of query-rewriting, data transformation, etc. . .

An important aspect of the implementation is the maturity of the tool(set). An academic prototype that has just been built to support a PhD thesis would be less stable and less usable than a product that has undergone much development over the years and is exploited by a commercial organization.

Experiences with the approach We summarize the experiences that have been reported in the literature for each approach. These experiences are very valuable, because they show the applicability of the methods to real ontology mapping and information integration problems. They also show the usability and limitations of the tools that have been developed for the method.

We structure the description of each of the approaches in the survey in Chapter 4 according to this evaluation framework. Furthermore, we provide a comparison of the approaches in the survey based on the presented evaluation framework in Chapter 5.

Chapter 4

The Survey

This chapter presents the actual survey on ontology merging and aligning approaches. We evaluate the approaches according to the criteria identified in the evaluation framework in Chapter 3.

In order to structure the survey, we have grouped the approaches into three categories:

- *Methods and Tools.* We describe several special-purpose methods and tools. The purpose of the approaches in this section ranges from ontology matching (GLUE, Semantic Matching) to ontology merging (PROMPT) and ontology mapping (MAFRA, RDFT, WSMT). Sometimes the lines between the purpose of the approaches becomes blurred, because, for example, the authors of MAFRA [MMSV02] also describe a way to do ontology matching. Also, in the case of PROMPT we not only describe the ontology merging tool, but also related tools in the area of matching (even PROMPT itself has a matching algorithm) and ontology versioning (PROMPTDiff).
- *Data Integration Systems.* We describe four approaches to data integration using ontologies, namely InfoSleuth, ONION, INFOMIX, MOMIS and OBSERVER. These integration systems are all comprehensive in the sense that they typically have different types of functionality. For example, both ONION and MOMIS have matching tools, which aid in creating mappings between ontologies. All data integration systems described in this survey support querying of the underlying data sources based on querying posed against an ontology; they typically implement the wrapper/mediator architecture, which was described in Section 1.5.
- *Specific Techniques.* We briefly describe a few specific techniques, which we do not evaluate according to the criteria in the evaluation framework. FCA-Merge is a method for ontology merging, based on formal concept analysis. OntoMorph is a system for syntactic and semantic rewriting of ontologies. QOM (Quick Ontology Mapping) is a method and tool for the discovery of ontology mapping, based on a

combined similarity measure. OMEN is a system that describes mappings with a probability and a by means of a Bayesian Network.

4.1 Methods and Tools

4.1.1 MAFRA

Summary MAFRA (MApping FRAmework for distributed ontologies) [MMSV02, SaR03b] is a framework defined for mapping distributed ontologies on the Semantic Web based on the idea that complex mappings and reasoning about those mappings is the best approach in a decentralized environment like the Web. MAFRA has been implemented as a plug-in of KAON¹ and introduces two interesting new concepts: Semantic Bridges and service-centric approaches. A Semantic Bridge is defined as "a declarative representation of a semantic relation between source and target ontologies entities" [SaR03b]. A Semantic Bridge provides the necessary mechanisms to transform instances and property fillers of a particular source ontology into instances and property fillers of a particular target ontology. Semantic Bridges are similar to the notion of articulation structures ("the points of linkage between two aligned ontologies") in [Kle01] and articulation ontologies in ONION [MWK00, MW01] (also Section 4.2.2).

The other novelty is the service-centric approach that the MAFRA Toolkit introduces [SaR03b]:

Each semantic bridge has an associated transformation service that determines the transformation procedure and the information the user must provide to the transformation engine. Each service is characterized by a set of arguments, which in turn are characterized by name, type, optionality and location (whether it is a source, target or condition argument). Services are not only responsible for the transformation capabilities but also for the validation of argument values and semi-automatic mapping.

The service oriented approach complements the Semantic Bridges mechanism providing the transformation services necessary to perform the mapping transformations. Silva and colleagues proposed a decentralized solution where independent transformation modules are attached to the system. An overview of the architecture of the MAFRA toolkit can be seen in Figure 4.1, where some transformation modules are included (copy instance, copy relation, concatenate, split, etc.).

Figure 4.2 outlines the conceptual architecture of the MAFRA System. In the conceptual architecture a set of modules is identified and organized along two dimensions. Horizontal modules correspond to five fundamental phases in the ontology mapping process

¹KAON is an Ontology Management tool developed by the University of Karlsruhe, <http://kaon.semanticweb.org/>

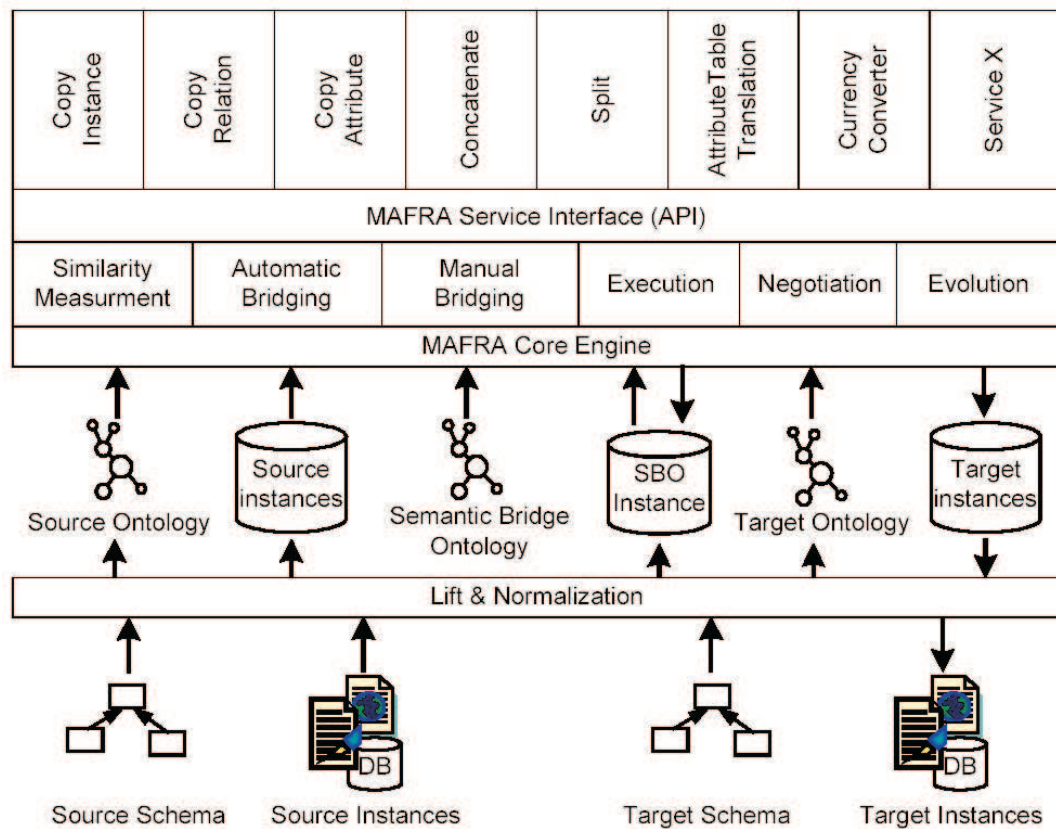


Figure 4.1: MAFRA Toolkit System Architecture [SaR03b]

(Lift & Normalization, Similarity, Semantic Bridging, Execution and Postprocessing). The vertical modules (Evolution, Domain Knowledge & Constraints, Cooperative Consensus building and GUI) interact with the horizontal phases during the entire ontology mapping process.

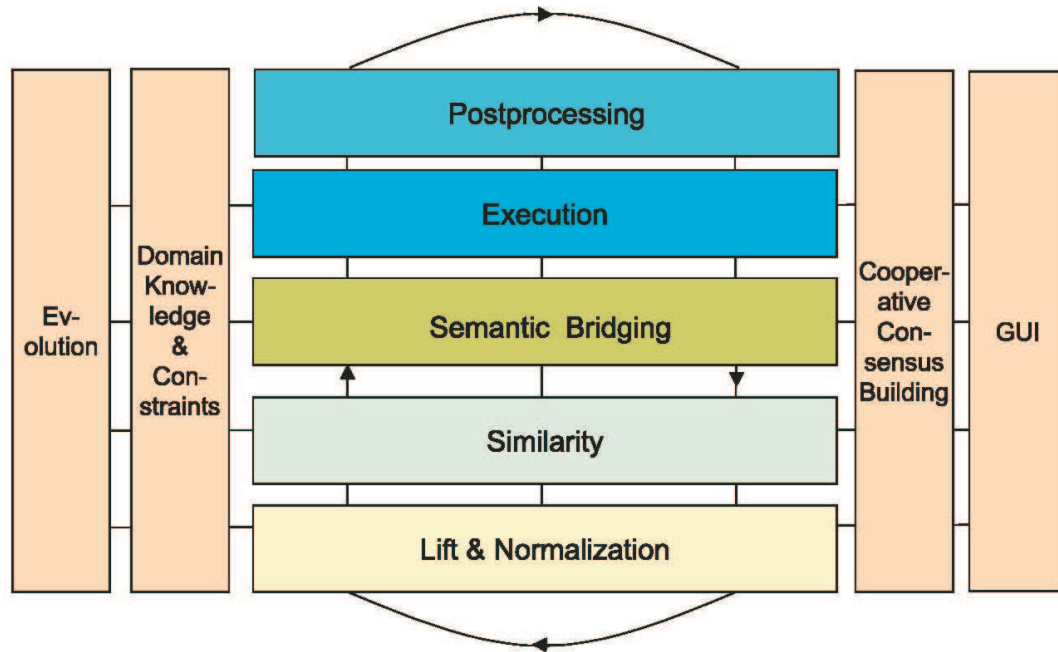


Figure 4.2: MAFRA Conceptual Architecture [MMSV02]

The main phases of the mapping process described along the horizontal dimension are:

- *Lift & Normalization.* Lifting refers to the process of importing the ontologies in a uniform representation formalism (**RDF**-Schema in this case) for facilitating later operations. The next step after lifting is the normalization of the vocabularies of the ontologies by eliminating lexical and semantic differences (like special characters, upper case letters and acronyms).
- *Similarity* In this step, similarities between ontology entities are calculated as a support for mapping discovery. A combination of different similarity algorithms is employed in MAFRA[MMSV02][Sil02] in order to improve the result of the identification of equivalent terms.
- *Semantic Bridging* After identifying the similarities between entities from different ontologies, the similar entities are semantically bridged, i.e. correspondences are established between them for enabling the transformation of instances of one source entity to instances of one target entity. Several sub-steps can be identified in this

phase. We describe these sub-steps in more detail after the enumeration of the mapping phases and of the vertical modules.

MAFRA includes a formal representation to specify the mappings. The formalism that is used to describe the Semantic Bridges is based on a DAML+OIL ontology, called the Semantic Bridging Ontology (SBO). The result is close to the notion of *articulation ontology* in ONION[MWK00] (see also Section 4.2.2). A mapping is a set of instances of the semantic bridges described by this ontology. We describe this ontology in more detail later.

- *Execution.* When the mappings/bridges are specified, the next step is to exploit them in a meaningful way. As already mentioned, MAFRA addresses only the task of instance transformation [MMSV02]. This module actually transforms instances from the source ontology representation into the representation of the target ontology by evaluating the transformation functions associated with the bridges defined in the previous stage. 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* Based on the execution results, the mapping specification is again analyzed, e.g. for discovering object identities, in order to improve the quality of the instance transformation task.

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 Semantic Bridging Ontology (SBO) (see Figure 4.3) is a taxonomy of generic bridges that are going to be instantiated for defining concrete bridges/mappings. We give an overview of the dimensions along which a bridge can be described in MAFRA, followed by a shallow description of the classes from SBO that allows one to express such bridges.

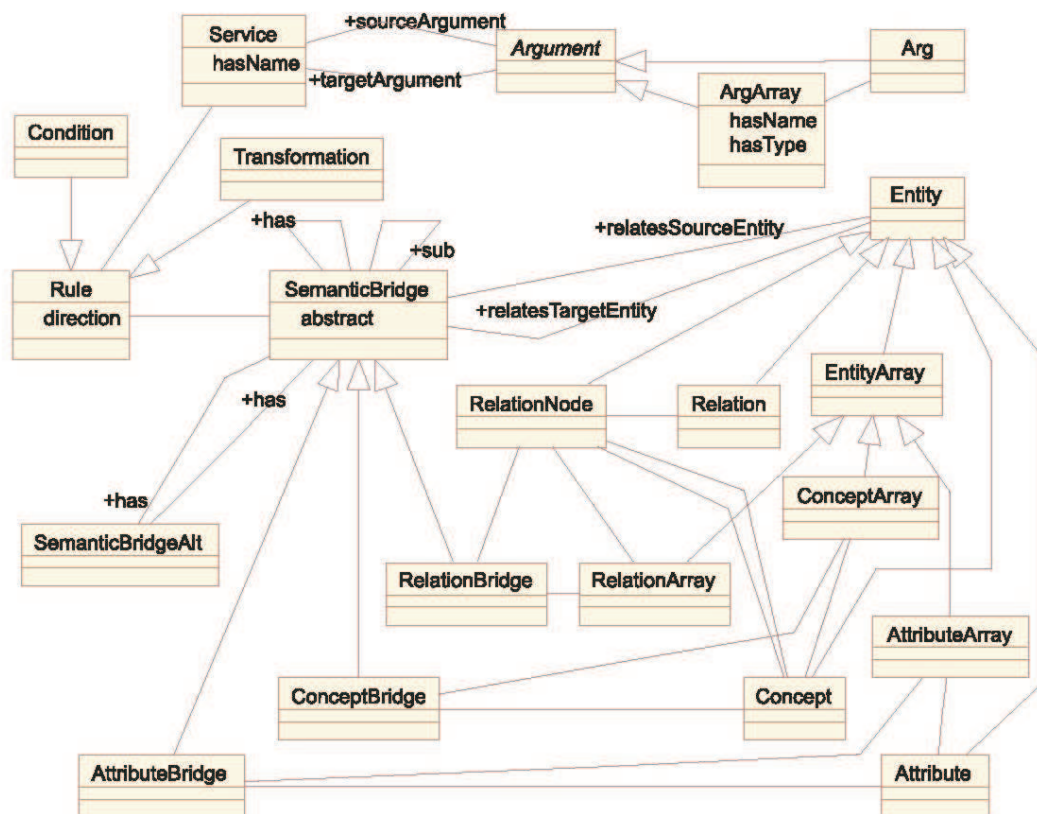


Figure 4.3: Semantic Bridging Ontology (SBO) in UML [MMSV02]

A bridge can be described along five dimensions:

- *entity dimension*: pertains to the entities related by a bridge which may be concepts (modeling classes of objects in the real world), relations, and attributes.
- *cardinality dimension*: pertains to the number of ontology entities at both sides of the semantic bridge (usually 1:n or m:1, m:n is seldom required and it can be usually decomposed into m:1:n)
- *structural dimension*: pertains to the way elementary bridges may be combined into a more complex bridge (relations that may hold between bridges: *specialization*, *alternatives*, *composition*, *abstraction*)
- *transformation dimension*: describes how instances are transformed by means of an associated transformation function.
- *constraint dimension*: allows one to express conditions upon whose fulfilment the bridge evaluation depends. The transformation rule associated with the bridge is not executed unless these conditions hold.

The abstract class `SemanticBridge` describes a generic bridge, upon which there are no restrictions regarding the entity types that the bridge connects or the cardinality. For supporting the *composition* modelling primitive, this class has defined a relation `hasBridge`. The class `SemanticBridgeAlt` supports the *alternative* modelling primitive by grouping several mutual exclusive semantic bridges. `SemanticBridge` is further specialized in the SBO according to the entity type, the ontology defining as subclasses of this class: `RelationBridge`, `ConceptBridge`, and `AttributeBridge`. There is no specialization of the `SemanticBridge` class to a class which bridges between individuals.

`Rule` is a class for describing generic rules. `Condition` and `Transformation` are its subclasses which are responsible for describing the condition necessary for the execution of a bridge, and the transformation function of a bridge, respectively. The `Service` class maps the bridge parameters with the transformation procedure arguments and call to procedures.

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. The Semantic Bridge phase defines in the following steps the necessary structures to describe the mapping between two ontologies:

1. Based on the analysis of similarities that were discovered in the Similarity phase, the first step is to select the pairs of entities, which could be concepts, relations

and attributes, to be bridged that correspond with concept bridges. MAFRA allows relations of different cardinality between source and target entities. Thus, a source or target entity can belong to one or more Semantic Bridges.

2. The property bridging step specifies matching properties for each concept bridge. The authors of MAFRA distinguish two types of properties: attributes and relations. In the case that the type of source and target properties is different the transformation specification information is required, and the domain expert is asked to supply this information. According to MAFRA's definition, an attribute defines a relation between a concept and a data type value and a relation defines a relation between two concepts.
3. This step (together with the next one) is part of a refinement process to improve the matching results, and focuses on looking for mapping alternatives where there is no target entities. If it is not possible to find a target entity for a source entity, the algorithm analyzes the hierarchy of the source ontology and proposes an equivalent mapping of some of the parents of the unmapped source entity. So the source entity is mapped to the same target entities as some of its parents.
4. As a part of the refinement process mentioned previously, in this step the system tries to improve the quality of bridges between source concepts and target concepts. It can be viewed as a complementary routine to the similarity phase.
5. Associate transformation procedures with the mapping relations identified in previous phases. Although one of the main goals of the authors of MAFRA is to provide an elevated level of automation in the mapping procedure, they recognize that in this step the intervention of an expert is highly recommended.

To finish this brief description of MAFRA, we present an example from [MMSV02] (see Figure 4.4). The goal of this exercise is to map two ontologies: the source ontology (*o1*) describes the structure of a family and its events are categorized in family events (marriage and divorce) and individual events (birth date, death date); and the target ontology (*o2*) characterizes individuals as Man and Woman. A Concept Bridge is defined to map *o1* : *Individual* with *o2* : *Individual*. All the attribute bridges are mapped using property/attribute bridges except for *o1* : *Individual* - *sex*. This attribute is mapped using an alternative semantic bridge with two concept bridges that map *o1* : *Individual* - *sex* with *o2* : *Man* and *o2* : *Woman*.

Ontology Languages MAFRA needs the Lift & Normalization module to translate the ontologies that participate in the mapping process into RDF (S). Precisely, the terminology specification is transformed to RDF Schema and the instances to RDF.

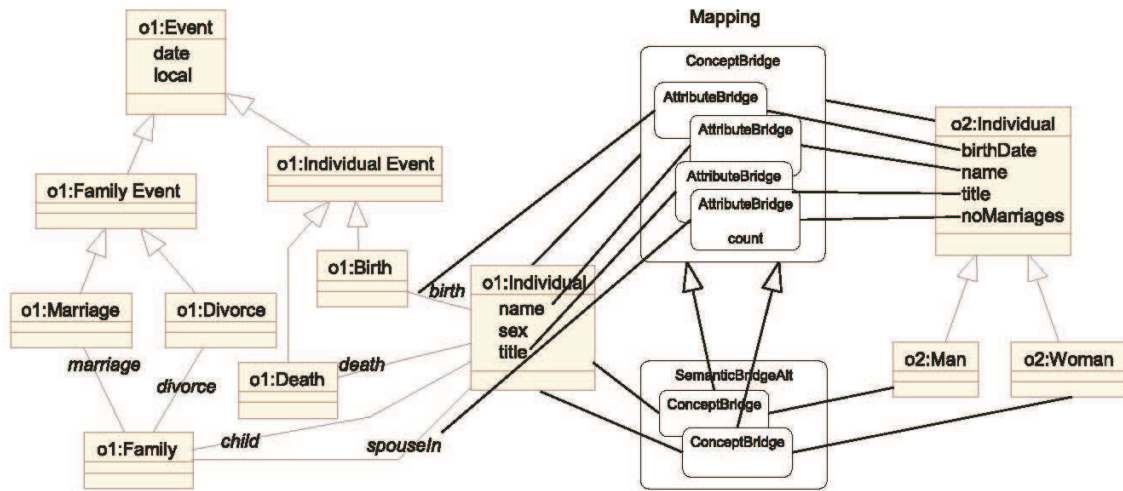


Figure 4.4: UML representation of the semantic bridge defined to map the ontologies of the example [MMSV02]

Mapping Language Semantic Bridges (SBs) in conjunction with transformations modules services provide all the functionality that a mapping language requires. The semantics of this mapping formalism is unambiguously specified through the SBO (Semantic Bridging Ontology).

Another important characteristic of MAFRA is that it supports several types of mapping like Class mappings, Property (i.e. relation) mappings and Instance mappings.

Mapping Patterns MAFRA does not support the use of mapping patterns in ontology mappings. However, one could see a Semantic Bridge as an elementary mapping pattern and a specific combination of a number of Semantic Bridges can be seen as a mapping pattern. Therefore, it should be possible to incorporate the use of mapping patterns into MAFRA.

Automation Support One of the main goals of the designers of MAFRA is to get a high level of automation support. Unfortunately, the papers that describe the tool do not indicate precisely which steps are automatic and which are not. Also MAFRA gives the user the opportunity to define Semantic Bridges manually. The modules that are directly involved in the mapping process (horizontal dimension) present the following level of automation:

- Lift & Normalization is probably a module that can work independently from users to provide a uniform representation of the ontologies that will be mapped.

- The calculation of similarities inside a multi-strategy process looks like it is fully automatic.
- The generation of Semantic Bridges is partially automated. The specification of mappings between properties (property bridging step) and the association of transformation procedures with mapping relations require the participation of a domain expert.
- The execution engine, implemented in Java, is fully automated, and achieves the transformations defined in the Semantic Bridges.
- The postprocessing module is not further elaborated in the papers that described MAFRA, and the level of automation is not specified.

The module *Domain and Background Knowledge* provides mechanisms to include background knowledge and domain constraints by using for example glossaries or lexical ontologies. This features can considerably improved the quality of the results of the similarity module and the semantic bridge module.

Applicability to Use Cases One of the goals of MAFRA is to support instance transformation through transformation procedures that are associated to Semantic Bridges. The postprocessing module tries to provide support for instance unification (recognizing that two instances represent the same real word object), but the authors recognized that it is a very challenging task and do not guarantee that it is fully implemented. On the other hand, Semantic Bridges define explicitly mappings between entities of two ontologies, and MAFRA provides a semantic specification for these mechanisms.

Finally, in the papers [SaR03a] and [SaR03b], the authors outline a mechanism is close to the idea of query rewriting to retrieve all the instances of a query that are stored in several ontologies which have mapping specifications between each other.

Implementation As we mentioned in the summary description of this tool, the MAFRA toolkit was implemented as plug-in of KAON. Silva and colleagues continuous with the development of this mapping system, and the latest versions can be founded at <http://mafra-toolkit.sourceforge.net>. Also some examples and documentation are available on this site. MAFRA's current approach is being used and tested under the Harmonise project². Harmonise intends to overcome the interoperability problems occurring between tourism operators due to the use of different information representation standards. The MAFRA Toolkit was adopted as the representation and transformation engine core technology for the Harmonise project. Harmonise uses an "Interoperability Minimum Harmonisation Ontology" (IMHO) as lingua franca between agents. The MAFRA Toolkit is responsible for the acquisition, representation and execution of the ontology mapping between each agent specific ontology and IMHO [SaR03a].

²<http://www.harmonise.org>

Experiences Silva and colleagues provided an informal evaluation in their papers ([SaR03a, SaR03b]) of the performance of MAFRA and they compare their results with OntoMerge [DMQ02], a tool for mapping and merge.

4.1.2 RDFT

Summary Omelayenko and Fensel [OF01] present an approach to the integration of product information over the web by exploiting the data model of RDF [LS99], which is based on directed labeled graphs. In their approach, Omelayenko and Fensel assume product catalogs from different organizations specified in XML documents. The problem they sketch is different organizations using different representations for their product catalogs. They intend to mediate between these different representations with the use of RDF triples³.

The approach to the integration of product catalogs is called two-layered because the product information itself is represented in XML, whereas the transformation between different representations is done in RDF. The general idea is that an XML document, whose structure is described by a DTD (Document Type Definition) or XML Schema, is (1) *abstracted* to an RDF graph, which in turn is described by an ontology, which could be specified using the RDF Schema [BG04] ontology language. The RDF document is then (2) *transformed* into a target representation, which is also described by an ontology. Then, the target RDF is (3) *refined* to the target XML representation, which can be used by applications at the target vendor. All three transformation steps are performed with the XML transformation language XSLT [Cla99]. The process of abstraction, transformation and refinement is illustrated in Figure 4.5.

[Ome02b] proposes a mapping meta-ontology for describing the transformation between RDF documents. This mapping meta-ontology, called RDFT (RDF Transformation) is specified using RDF Schema [BG04] and is used to describe the mapping between two RDFS ontologies. We describe this ontology and its use in more detail below.

[Ome02a] describes a technique for discovering semantic correspondence between different product classification schemes based on a Naive-Bayes classifier. The mappings between the different product classifications are represented using the bridges from the RDFT meta-ontology.

Ontology Languages Omelayenko [Ome02b] not only describes a way to map between different RDF Schema ontologies, but also describes the way to transform XML documents to RDF using RDFT, thereby effectively specifying the way to perform the abstraction step.

³An RDF triple consists of a *subject*, a *predicate* and an *object*. Subjects and objects form the nodes of the graph, whereas predicates form the edges. An object in a triple can also occur as a subject or an object of a different triple.

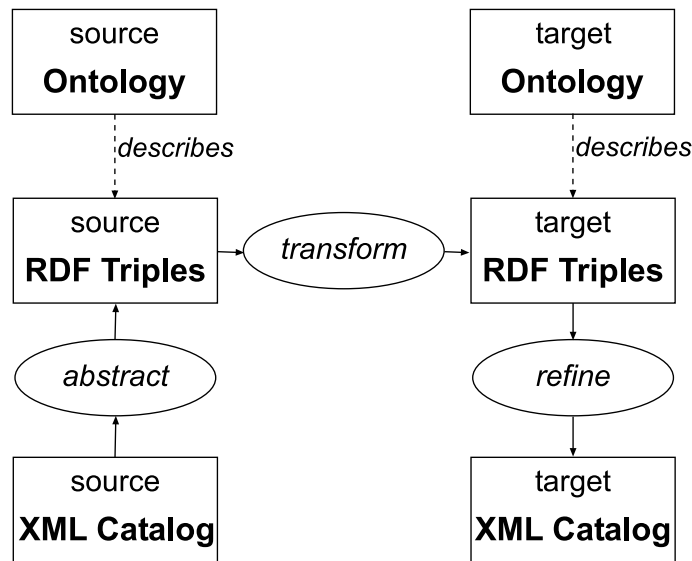


Figure 4.5: Two-layered integration of XML catalogs using RDF

RDFT can be used to express mappings between arbitrary ontologies specified in the RDF Schema ontology language. Furthermore, it can be used to specify the transformation between XML documents and the RDF representation.

Mapping language We will now give a short overview of the RDFT mapping meta-ontology.

The RDFT meta-ontology is used to describe three types of mappings denoted by classes in RDFT:

- An *EventMap* is used to specify the relationship between different events. Events in this context correspond to activity occurrences, such as sending or receiving a message. These events can be used, for example, to connect descriptions of two web services, described using the Web Service Definition Language WSDL⁴.
- A *DocumentMap* specifies the relationship between an XML and an RDF representation of a catalog.
- A *VocabularyMap* specifies the actual relationships between two ontologies.

For our purposes, the most interesting type of mapping is the *vocabulary mapping* (*VocabularyMap*).

A mapping between two ontologies (vocabularies) is expressed using a number of bridges. Bridges in RDFT are subclasses of the *RDFBridge* class. RDFT distinguishes two types of RDF bridges, namely *Class2Class* and *Property2Property*

⁴<http://www.w3.org/TR/wsdl>

bridges. `Class2Class` bridges are used to describe the mapping between two classes and the transformation of instances of these classes. The instance transformation is specified using XPath [CD99] expressions. `Property2Property` bridges are used to describe the mapping between two properties in the ontologies. Again, XPath can be used to specify instance transformations.

The types of mappings in RDFT (class-to-class and property-to-property) are probably sufficient in the domain of e-Marketplaces, which was the original target application domain [OF01], because ontologies can be expected to have a similar level of granularity and the goals of the different ontologies are similar. However, if ontologies are more diverse, different types of mappings, e.g. classes-to-instances, classes-to-properties, etc. will be necessary.

In the approach taken by Omelayenko (cf. [OF01, Ome02b]), the steps of *abstraction*, *transformation*, and *refinement* all use the XML Transformation language XSLT [Cla99] for specifying the transformations between XML and RDF documents, as well as transformations between different RDF representations. While certainly XSLT is expressive enough to express arbitrary transformations between XML documents, and can therefore also transform RDF documents represented in the RDF/XML [Bec03] serialization into a different representation, it is not well-suited for the specification of RDF transformations, because it does not take the data model of RDF, which is *graph based* into account, whereas the data model of XML is *tree based*. Therefore, the RDF data model needs to be in a sense encoded in the tree based XML model in each single XSLT transformation.

Automation Support [Ome02a] describes a way to discover similarity between classes based on the instance information for this class, using a machine-learning approach. In the use case, the class was a product classifier and the instance data consisted of the product descriptions.

The RDFT meta-ontology was presented as the preferred way to specify mappings between ontologies, based on the similarities discovered by a matching tool, but no explicit support is provided for this.

Applicability to Use Cases RDFT tackles the use case of *instance transformation* through the XPath specifications attached to the RDF Bridges. RDFT does not offer a solution for instance unification, nor for query rewriting, although the declarative mapping between classes and properties could be used for this purpose. The scope of RDFT is limited to the transformation of XML documents between different representations.

The use case of ontology merging is not addressed, although a specification of relationships between ontologies in terms of the RDFT meta-ontology could help in merging different ontologies, because it specifies the relationship between classes.

Implementation A prototype tool was created to create mappings based on the RDFT meta-ontology.

Experiences RDFT as well as the classification method proposed in [Ome02a] have been used for the discovery and specification of mappings between product classification schemes in the course of the GoldenBullet [DKO⁺02] project.

4.1.3 PROMPT

Summary The PROMPT suite consists of a set of tools that have had an important impact in the area of merging, alignment and versioning of ontologies. A relevant result of this development is the definition of a global strategy that looks to take advantage of the synergies that have been generated by the combination of tools that in the past were considered independent. The PROMPT suite [NM03b] includes an ontology merging tool (iPROMPT, formerly known as PROMPT [NM00b]), an ontology tool for finding additional points of similarity between ontologies for other tools like iPROMPT (AnchorPROMPT, [NM00a]), an ontology versioning tool (PROMPTDiff, [NM03a]), and a tool for factoring out semantically complete subontologies (PROMPTFactor, [NM03b]). The work of Natasha Noy and colleagues proves that different tasks in multiple ontology management⁵, like looking for differences between versions of an ontology or looking for similarities between two ontologies in a merging process, are closely interrelated and share several components and heuristics (see Figure 4.6). Thus tools for supporting some of the tasks in the context of multiple ontology management can benefit greatly from their integration with others [NM03a].

The key components of the PROMPT suite have been developed as extensions (plugins) of the Protégé 2000 ontology development environment⁶. We can distinguish the following components:

- iPROMPT is an interactive ontology merging tool, which helps users in the ontology merging task by providing suggestions about which elements can be merged, by identifying inconsistencies and potential problems and suggesting possible strategies to resolve these problems and inconsistencies.
- AnchorPROMPT extends the performances of tools like iPROMPT determining additional points of similarities between ontologies that are not identified by iPROMPT.

⁵Noy and colleagues [NM03b] define multiple ontology management as a set of concrete tasks for dealing with multiple ontologies such as maintaining libraries of ontologies, import and reuse of ontologies, translating ontologies to other formalisms, ontology versioning support, ontology merging-mapping-alignment support, inference across multiple ontologies and query across multiple ontologies

⁶<http://protege.semanticweb.org/>

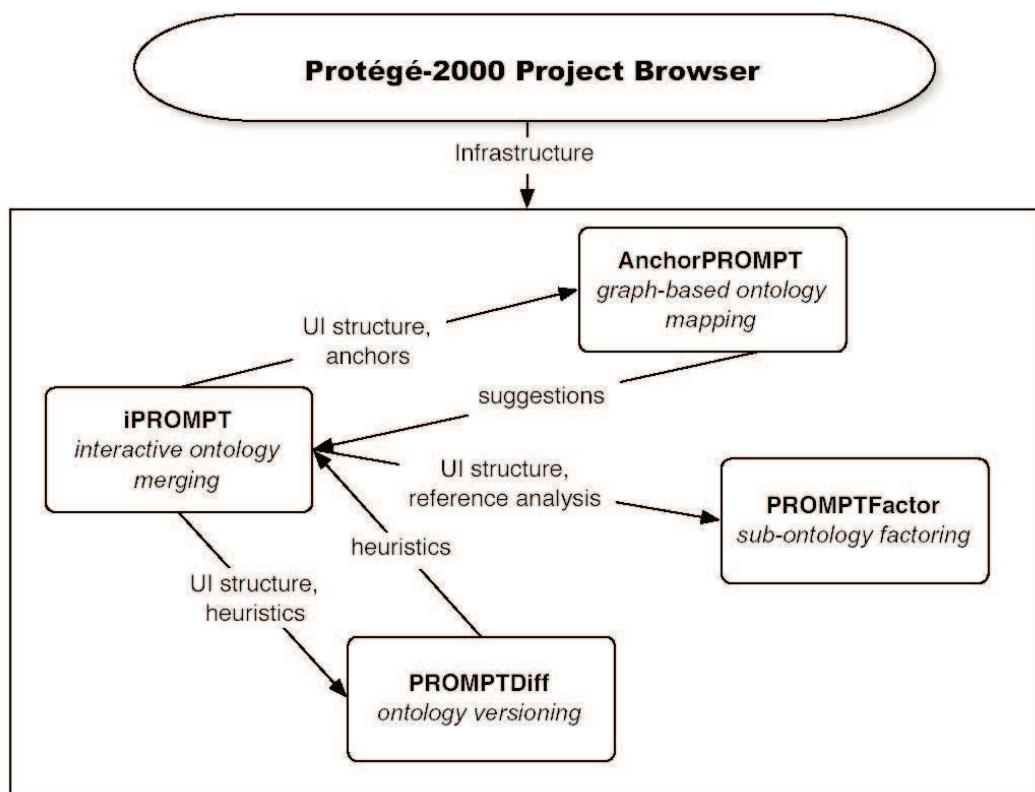


Figure 4.6: The PROMPT suite infrastructure and interactions between tools [NM03b]

- **PROMPTDiff** compares two version of an ontology and identifies structural differences between different versions of the same ontology.
- **PROMPTFactor** is a tool that enables users to create a new ontology factoring out part of an existing ontology. In this process, the tool guarantees that the terms of the resulting subontology are well-defined (for instance, every concept of the subontology includes as appropriate the superconcepts/subconcepts required for its specification).

One of the major contributions to the development of **PROMPT** suite was the identification of an important overlap in the functionality of its tools and the implementation of an integrated approach where all these tools benefit from each other. For instance, some of the components that were originally created for the interface of **iPROMPT** were reused in the implementation of the interfaces of the other tools of the suite. In addition, the initial sets of related terms between two ontologies that **AnchorPROMPT** requires as a starting point for a deeper analysis of similarities can be provided by **iPROMPT**. In return, **AnchorPROMPT** can supply an additional set of related terms that can be used by **iPROMPT** to improve the results of the merging process. A final example of this integrated approach can be found in the design of **PROMPTDiff** and **iPROMPT**. **PROMPTDiff** uses some of the heuristics that were initially developed in **iPROMPT** for comparison of concept names, slots attached to concepts, domains and range of slots and so on.

As mentioned above, **iPROMPT** [NM00b] is an interactive tool implemented as an extension of Protégé 2000. **iPROMPT** guides users in the process of merging two ontologies (see an example of the user interface in Figure 4.7). The tool was originally developed to handle ontologies specified in OKBC [CFF⁺98], but there are significant efforts to adapt the tool in order to support the OWL ontology language [DS04]. The central element of **iPROMPT** is the algorithm that defines a set of steps for the interactive merging process, see also Figure 4.8. The first step is to identify potential merge candidates based on class-name similarities. The result is presented to the user as a list of potential merge operations. The second step is initiated by the user who chooses one of the suggested operations from the list or specifies the operation directly. The system performs the requested action and automatically executes additional changes derived from the action. It then makes a new list of suggested actions for the user based on the new structure of the ontology, determines conflicts introduced by the last action, finds possible solutions to these conflicts and displays these to the user.

Initially, **PROMPT** identified a set of ontology merging operations (merge classes, merge slots, merge bindings between a slot and a class, etc) and a set of possible conflicts for these operations (name conflicts, dangling references, redundancy in the class hierarchy and slot-value restrictions that violate class inheritance). These lists of ontology merging operations and possible conflict operations have been extended by the authors of the tool as a part of an evolution process in the design of the system.

The goal of **AnchorPROMPT** [NM00a] is to augment the results of methods that

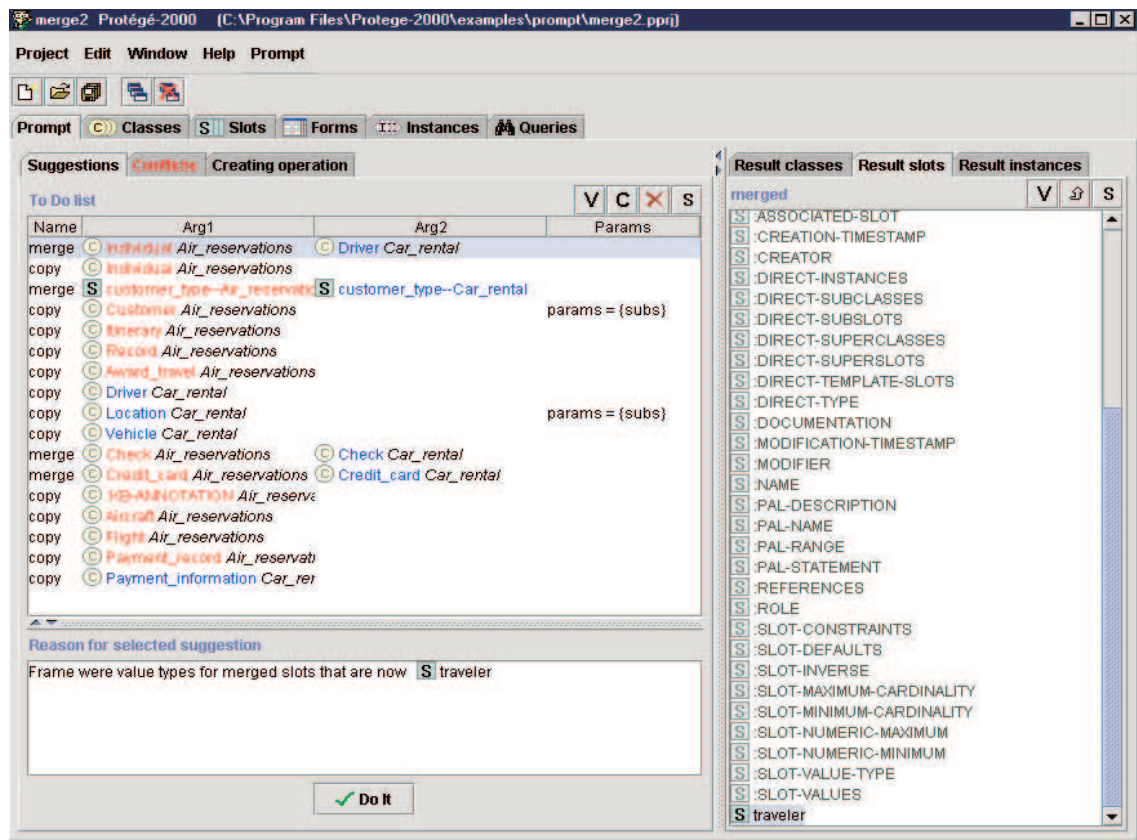


Figure 4.7: An example of ontology merging in iPROMPT

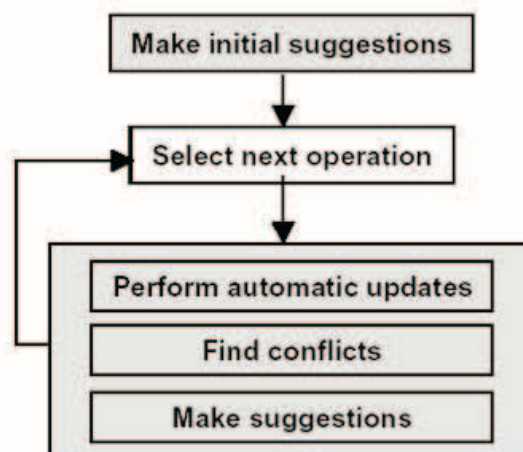


Figure 4.8: The flow of the iPROMPT algorithm [NM00b]

analyze only local context in ontology structures, such as Chimaera [MFRW00] and iPROMPT [NM00b], by finding additional possible points of similarity between ontologies. To do this AnchorPROMPT requires that the other tool or the user provides an initial set of related terms. Following a graph perspective, the tool establishes a set of paths that connects the terms of an ontology that are related with the terms of the other one. The algorithm takes two pairs of related terms as input and analyzes the elements that are included in the path that connect the elements of the same ontology with the elements of the equivalence path of the other ontology. So, we have two paths (one for each ontology) and the terms that compound these paths. The analysis looks for terms along the paths that might be similar to the terms of the other path, which belongs to the other ontology, assuming that the elements of those paths are often similar as well. These new potentially related terms the algorithm discovers are marked with a similarity score that can be modified during the evaluation of other paths in which these terms are also involved. Terms with high similar scores will be presented to the user to improve the set of possible suggestions in, for example, a merging process in iPROMPT.

If the two ontologies that we compare present important differences in the number of levels of their hierarchy or in the number of relations between classes, the algorithm does not work well.

The third element of the suite is **PROMPTDiff** [NM03a], which is used to compare the structure of two versions of a particular ontology and which identifies the frames (i.e. classes, slots or instances) that have no changes, frames with only changes in their properties, and frames that have also changed in other parts of their definitions. The name of the tool, PROMPTDiff, is influenced by tools like CVS, which is a version control system that is used to maintain the history of program source code files. This tool includes facilities to discover changes between versions of a document (finding a diff).

The last element of the PROMPT suite we will describe here is the tool **PROMPT-Factor** [NM03b] which allows users to extract from a larger ontology the elements that the user is interested in, in a way that also copies all the terms required for preserving the semantics of the descriptions. The authors of the tool call this process “factoring subontologies”.

During the analysis of the PROMPT suite, we concluded that the tool has some limitations in the area of ontology versioning and evolution. We present a summary of some of the most relevant conclusions of our study (some of them where confirmed by Natasha Noy):

- PROMPTDiff only detects differences between two versions using a structural diff. In [Kle04], we can find several complementary alternatives (change logs, conceptual relations and transformation set) that can give us a richer description of the changes that the original ontology has undergone.

- The description of the differences between two versions of an ontology that PROMPTDiff offers is limited. For this reason, Klein extended PROMPTDiff to support richer semantic descriptions of changes. He introduced a more complex classification of type of changes (implicitly-changed, directly-changed, changed, isomorphic and unchanged, see [Kle04]) and provides a high level description of the changes based on the idea of minimal transformation set and on an ontology of changes (again see, [Kle04]).

PROMPTDiff can find difference between ontologies but it does not mean that there is explicit support for versioning. PROMPTDiff does not allow the user to identify versions or to indicate that there is a versioning relationship between ontologies. Therefore, the user has to find a way to manage different versions of an ontology and to identify that a particular ontology is a version of another ontology.

Ontology Languages The knowledge model underlying PROMPT is the Open Knowledge Base Connectivity (OKBC) protocol [CFF⁺98]. OKBC is frame-based: frames are the main elements in this knowledge model for building ontologies, and three types of frames can be distinguished: classes, slots and instances. A class is a set of entities and the elements of such a set are called instances, which may be classes or slots as well as individuals. Slots define binary relations between pairs of entities, whether they be classes, instances, or primitive objects (such as a string or a number).

There has been a considerable effort to provide PROMPT with RDF and OWL support[NM03b]. In the tutorial about the Protégé OWL plug-in given at ISWC03⁷ it was mentioned that PROMPT can be used in conjunction with this plug-in⁸[KFNM]. Thus the OWL support for PROMPT is achieved through this plug-in.

Mapping Language iPROMPT and AnchorPROMPT do not include a language that specifies the mapping. We understand that there should exist an internal representation of the mapping results because there is a strong interaction between the tools of the suite, and they need to share these results, but the related bibliography does not describe this possible formalism. Michel Klein (see [Kle04]) implemented an extension of PROMPTDiff that provides a language for change specification that characterizes differences between two ontologies. This language was originally defined using OKBC and then translated and extended in OWL.

iPROMPT in combination with AnchorPROMPT can map classes, properties and instances using linguistic and structural similarity techniques.

Mapping Patterns Currently there is no support for mapping patterns in the PROMPT suite.

⁷<http://iswc2003.semanticweb.org/pdf/Protege-OWL-Tutorial-ISWC03.pdf>

⁸<http://protege.stanford.edu/plugins/owl/>

Automation Support iPROMPT is an interactive merging tool that guides users in the process of merging two ontologies. iPROMPT proposes to the user a set of ontology merging operations and a set of possible conflicts for these operations. Then, the user has two choices: select one of the suggestions generated by the tool, or specify the desired operation directly. After that, iPROMPT performs the operation and automatically executes additional changes that the operation requires. Finally the previous list of suggestions is modified as a result of the changes that the executed operation produced. This cycle is repeated until the merging process finishes, or the user decides to abort it.

Applicability to Use Cases The PROMPT suite is a set of tools that provides several solutions for ontology mediation, versioning and factoring. iPROMPT covers the complete merging process, and can also generate a list of initial similarities that AnchorPROMPT improves in generating a new list of related terms on which the mapping could be based.

Implementation All the tools of the PROMPT suite are plug-ins or extensions to the Protégé-2000 ontology development environment. Protégé-2000 provides an intuitive graphical user interface for ontology development, a rich knowledge model based on OKBC, and an extensible architecture that provides API access both to the Protégé-2000 knowledge bases and to its user interface components [NM03b]. One of the disadvantages of having Protégé-2000 as a base for PROMPT stems from the difficulties of supporting OWL on top of Protégé-2000. These are mentioned in [KFNM] and are due to the differences between OKBC, Protégé-2000's internal model, and OWL.

The PROMPT suite is clearly user oriented where the main goal is to support the user in creating the mappings (and merging the ontologies). The suite of tools provides a common user interface that follows the schema of Protégé-2000 GUI, and components that implement the Match operator to find similarities between ontologies.

The PROMPT suite was developed and improved in the context of several projects during the last 5 years, with the collaboration of many users who continuously evaluate and exploit the tools providing valuable feedback for the developers.

Experiences The papers that describe iPROMPT [NM00b], AnchorPROMPT [NM00a] and PROMPTDiff [NM03a] include evaluation tests to show the accuracy of these tools.

In the case of iPROMPT [NM00b], the authors tested the tool using two ontologies with 134 class and slot frames in total. The first ontology was developed for the unified problem solving method development language (UPML) [FMvH⁺03] and the second ontology for the method description language (MDL) [GGM98]. The evaluation showed that human experts followed 90% of iPROMPT's suggestions and 75% of the conflict resolution strategies. The users performed 74% of the operations suggested by iPROMPT during the merging process.

AnchorPROMPT was also tested in [NM00a]. The results show that the accuracy of AnchorPROMPT is directly proportional to the length of the paths considered. For example with path length 2 the accuracy is 100% and with path length 4 the accuracy decreases to 67%. Noy and colleagues also tested AnchorPROMPT with the same ontologies as iPROMPT. They discovered an important limitation of the tool: the algorithm does not provide good results when the structures of the ontologies differ considerably. The UPML ontology has a large number of classes distributed on many different levels. On the other hand, the MDL ontology has a simpler structure with fewer classes and with only two levels in the hierarchy.

Finally, the accuracy of PROMPTDiff [NM03a] was evaluated using several versions of two ontologies of two different projects: EON project and PharmGKB project. The tool identified 96% of the possible matches (recall) and 93% of the identified matches were correct (precision).

4.1.4 GLUE

Summary GLUE [DMDH04] is a system which employs machine learning technologies to semi-automatically create mappings between heterogeneous ontologies, where an ontology is seen as a taxonomy of concepts. With GLUE, the authors port their previous work on matching database schemas (called LSD) [DMDH02] to the Semantic Web domain. GLUE focuses on finding 1-to-1 mappings between concepts in taxonomies, although the authors say that extending matching to relations and attributes and involving more complex mappings (such as 1-to-n and n-to-1 mappings) is the subject of ongoing research.

The similarity of two concepts A and B in the two taxonomies O_1 and O_2 is based on the sets of instances that overlap between the two concepts. In order to determine whether an instance of concept B is also an instance of concept A , first a classifier is built using the instances of concept A as the training set. This classifier is now used to classify the instances of concept B . The classifier then decides for each instance of B , whether it is also an instance of A or not.

Based on these classifications, four probabilities are computed, namely $P(A, B)$, $P(\overline{A}, B)$, $P(A, \overline{B})$ and $P(\overline{A}, \overline{B})$, where, for example, $P(A, \overline{B})$ is the probability that an instance in the domain belongs to A , but not to B . These four probabilities can now be used to compute the *joint probability distribution* for the concepts A and B , which is a user supplied function, using these four probabilities as parameters. [DMDH04] describes two possible functions for the joint probability distribution. The first example is the *Jaccard* coefficient, where the similarity measure is computed by dividing the probability that an instance is in the intersection of two concepts by the probability that an instance is in the union of the concepts ($P(A \cap B)/P(A \cup B)$), which intuitively corresponds to the function of relevant instances, which are both in A and B . The second example is the “most-specific-parent”, where the similarity measure is positive (i.e. the measure is not

0) for any parent B of A and it is the highest for the most specific parent, i.e. the concept B_{MSP} , which represents the smallest superset of A .

The general architecture of the GLUE system is as follows:

- The *Distribution Estimator* takes as input the two taxonomies O_1 and O_2 , together with their instances and applies machine learning to compute the four aforementioned probabilities $P(A, B)$, $P(\overline{A}, B)$, $P(A, \overline{B})$ and $P(\overline{A}, \overline{B})$. Currently, the distribution estimator uses a content learner, which learns a classifier based on the textual context of the instances, and a name learner, which learns a classifier based on the name of the instance. It is possible to plug in different learners for different aspects using a meta-learner which uses a certain function to incorporate the predictions from all learners into an overall prediction.
- The *Similarity Estimator* applies a user supplied function, such as the mentioned Jaccard coefficient or the most-specific-parent, and computes a similarity value for each pair of concepts $\langle A \in O_1, B \in O_2 \rangle$.
- The *Relaxation Labeler* takes as input the similarity values for the concepts from the taxonomies and searches for the best mapping configuration, exploiting user supplied domain specific constraints and heuristics.

All in all, GLUE can be seen as an implementation of the *Match* operator and can be fit into the overall mapping process as illustrated in Section 1.2.

Ontology Languages The GLUE matcher uses two taxonomies, in which the nodes correspond to concepts, and edges correspond to *subclass-of* relationships in the ontologies. Clearly, such a taxonomy can be easily extracted from an ontology represented in any ontology language, although a lot of the relationships in the ontology are not taken into account. This, though, is not such a big problem for the approach, since the matching is mostly based on instance information.

Mapping Language The result of the matching done in GLUE is not a mapping between the two ontologies, but rather a set of similarity measures, stating which concepts in one ontology O_1 are similar to concepts in the other ontology O_2 .

Mapping Patterns Mapping patterns are not an issue in GLUE, since it is only concerned with discovering similarities between concepts based on their instances. GLUE could also not be used for matching patterns with an ontology, since a pattern does not have instances.

Automation Support GLUE has a semi-automatic algorithm for specifying the mapping between two ontologies. Ontologies are seen as taxonomies and the problem of matching is reduced to: “for each concept node in one taxonomy, find the *most similar* node in the other taxonomy”.

The input from the user in the matching process consists of the function to be used for computing the overall similarity value, based on the joint distribution of the concepts, and the domain specific constraints and heuristics, which are used for the relaxation labeling process.

GLUE takes a one-shot approach at determining the similarities between taxonomies, which means that there is no user interaction during the matching process. The user has to use the outcome of the matching process as-is and use it as a basis for creating a mapping between the ontologies. In other words, GLUE implements the “find similarities” step in the mapping process (Section 1.2), but does not provide support for the iteration step.

Applicability to Use Cases GLUE aids in creating mappings between ontologies in the sense that it makes the work of the human user easier by finding similarities between concepts in two ontologies based on their instances.

Implementation A prototypical implementation of GLUE was created and the performance of each of the components in the architecture was evaluated. The main components to be evaluated were the different types of learners used for the classification and the relaxation labeler, which applied domain constraints and heuristics in order to come up with better matches. It turned out that the combination of several combined classifiers together with domain heuristics can achieve significant performance enhancement in terms of accuracy, which can go up to 97% in some domains.

Experiences [DMDH04] reports only on small evaluations of the performance of their system for taxonomies in the domain of (university) course catalogs and company profiles. The matching accuracy for their chosen examples was typically between 70 and 90 percent. However, experiments on a broader scale need to be done to see if GLUE works in other domains and to evaluate the scalability of the approach.

4.1.5 Semantic Matching

Summary Semantic Matching [GS04] is an approach to matching classification hierarchies. The problem addressed by Semantic Matching is the following: say you have two different classification hierarchies, where each hierarchy is used to describe a set of documents, i.e. each term in the classification hierarchy describes a set of documents. How do the terms in one hierarchy relate to the terms in the other hierarchy?

Semantic Matching can also be seen as an implementation of the *Match* operator for purely tree-structures ontologies. The authors define *Match* as follows: “*Match* is an operator that takes two graph-like structures (e.g. database schemas or ontologies) and produces a mapping between elements of the two graphs that correspond semantically to each other”. This definition is similar to the definition provided in Section 1.1. However, in Semantic Matching the definition is limited to the graph representation format for ontologies. This distinction is fundamental to the Semantic Matching approach, since it performs matching based on the nodes and the edges between the nodes in a graph.

Until now Semantic Matching has been mostly developed and tested for the task of matching classification hierarchies. Classification hierarchies are tree-structured graphs in which each node has only one parent. A property of classification hierarchies is that there is only one type of relationship, which is a *more-specific-term* relation which subsumes the *subclass-of* relationship. It is currently not clear if and exactly how Semantic Matching can be applied to the problem of ontology matching, because most ontologies typically have different types of relationships between concepts and the *subclass-of* relationship in ontologies is a formal relationship, interpreted often as a strict logical implication or a subset relationship (as is the case for the semantics of Description Logics).

Of course, an ontology can usually be rewritten as a graph with labelled edges, although some information (e.g. axioms and higher-order relations) might be lost in the rewriting. Concepts could be the nodes and relationships between concepts could be the (labelled) edges; the label of the edge would denote the type of the relationship. This is similar to the labelled graphs used in ONION (see Section 4.2.2). There is currently work underway to incorporate the semantics of the *relationships* in the Semantic Matching algorithm, but this work is still in the early stages.

The authors of [GS04] have argued that almost all earlier approaches to schema and ontology matching have been *syntactic* matching approaches, as opposed to *semantic* matching. In syntactic matching, the labels and sometimes the syntactical structure of the graph is matched and typically some similarity coefficient $[0, 1]$ is obtained, which indicates the similarity between the two nodes. Semantic Matching computes a set-based relation between the nodes, taking into account the meaning of each node. The possible relations returned by the Semantic Matching algorithm are *equality* ($=$), *overlap* (\cap), *mismatch* (\perp), *more general* (\subseteq) or *more specific* (\supseteq). The correspondence of the symbols with set theory is not a coincidence, since each concept in the classification hierarchies represents a set of documents.

We will now briefly sketch the Semantic Matching (also *S-Match*) algorithm for graph matching.

Two levels of granularity for matching are distinguished in S-Match, namely *element-level* matching and *structure-level* matching. At the element level, which is concerned with individual nodes, the authors distinguish techniques with *weak semantics* and techniques with *strong semantics*. Techniques with weak semantics correspond to the syntactic matching which has been proposed in most previous literature (for an overview

see [RB01]). Element-level matching with strong semantics is done using thesauri (e.g. WordNet [Fel99]), which typically contain synonym and hypernym relations between terms. These relations can be used to find semantic relations between nodes in the graphs.

In the next phase, the *structure-level* matching, the matching problem, i.e. the two graphs together with the *mapping query* are translated into a propositional formula and then checked for validity (i.e. *satisfiability*). A mapping query is a pair of nodes and a semantic relationship between the pair of nodes. If the propositional sentence is valid, we know that the semantic relationship between the two nodes in the query holds and thus can be added to the mapping result.

A potential problem with this algorithm is that the propositional satisfiability check (which is known to have nondeterministic polynomial complexity) has to be performed for every pair of nodes from the two graphs. Clearly, this does not scale for large graphs.

Ontology Languages Currently, the semantic matching can work with classification hierarchies, but also directed acyclic graphs (DAGs) in general. Classification hierarchies can often be extracted from ontologies by treating classes in the ontology as nodes and the *subclass-of* relationships as edges, but all other relationships are lost in the translation. Note that multiple classification hierarchies can be extracted from an ontology in which concepts have more than one direct superclass. This of course does not rule out the use of the result of the algorithm as the input to a mapping process for the complete ontologies. Also, there is work underway to extend the semantic matching to work with labeled graphs, taking the semantics of the different relationships into account.

This does not mean that the algorithm in its current form is useless, on the contrary. There are currently many classification schemes around, such as dmoz⁹, Yahoo¹⁰, and many other (specialized) classification hierarchies are in use. However, for arbitrary ontology matching on the Semantic Web, it has not been shown that the algorithm performs well. That the algorithm works well has been shown only for the case of classification hierarchies with *more-specific-term* relationships. Ontologies typically implement both the formal *subclass-of* relation and many other types of relationships.

Mapping Language S-Match is a matching algorithm and as such does not have a language for the actual specification of the mappings, only for the specification of the similarities, although in this case the specification of similarities comes close to a real mapping specification.

As we have mentioned earlier, the specification of the similarity of concepts is done using set-based primitives, denoting the relationships of equality, disjointness, overlap and sub/superset. In later work (e.g. [GSY04]), the authors use the symbols commonly found in description logics, i.e. $\langle A, B, \sqcap \rangle$ for overlap, $\langle A, B, \sqsubseteq \rangle$ for subset, $\langle A, B, \sqsupseteq \rangle$

⁹<http://www.dmoz.org/>

¹⁰<http://www.yahoo.com/>

for superset and $\langle A, B, \perp \rangle$ for disjointness of the concepts. These relations could be translated to Description Logic [BCM⁺03] axioms, i.e. $A \sqcap B \not\equiv \perp$, $A \sqsubseteq B$, $B \sqsubseteq A$, and $A \sqcap B \equiv \perp$.

Mapping Patterns Currently, there is no use of mapping patterns in Semantic Matching. It might be worthwhile to see if mapping patterns can help to find similarities, although this is not a straightforward task. Perhaps it is possible to match ontologies against mapping patterns in order to find out if a certain mapping pattern might be applicable, but the authors do not give any hints as to if and how we can incorporate mapping patterns into the matching algorithm.

Automation Support Clearly, the proposed algorithm is an automatic one-pass (i.e. no user interaction) algorithm, which returns all similarities it can find between the two graphs. There is no user interaction during the execution of the matching.

It cannot be assumed that the mapping returned by the algorithm is either correct or complete. Therefore, the result of the S-Match algorithm can serve as a first step in the overall ontology mapping process. It can serve as the input for the next phase in the mapping process, in which the user validates the result of the matching and corrects any mistakes and does the necessary additions in order to make the mapping correct and complete¹¹.

Applicability to Use Cases For the purpose of ontology mediation on the Semantic Web, the role which can be played by S-Match could be in the discovery phase of the mappings between ontologies. Since S-Match provides an implementation of the *Match* operator, it fits into the “find similarities” step in the mapping process.

Implementation [GSY04] presents S-Match, an algorithm and implementation of Semantic Matching. It also compares the performance of the S-Match implementation in terms of speed, precision and recall with available implementations of existing approaches in syntactic matchings COMA [DR02], Cupid [MBR01] and Similarity Flooding [MGMR02], which was implemented in the RONDO system [MRB03].

It turned out that for most applications, S-Match outperformed the other systems in terms of precision and recall. However, the other systems typically outperformed S-Match in terms of time required to perform the actual matching. One possible explanation is that the S-Match implementation has not really been optimized. However, the S-Match implementation uses a propositional SAT solver, which can not be efficiently implemented, because the problem is known to be NP-Hard. Currently, there are no known algorithms that require less than exponential time for satisfiability checking.

¹¹Of course, it can also never be guaranteed that the outcome of the human mapping will be either correct or complete.

Experiences Semantic matching has so far only been tested with some toy examples. However, the results presented in [GSY04] do look promising with respect to the precision and recall achieved by the system compared to other existing matchers. Furthermore, S-Match is currently in the early stages of its development; there are plans to apply S-Match in other settings, which will show whether S-Match works for real-world problems on the Semantic Web.

4.1.6 OntoMap

OntoMap ([KSD01a]) is a knowledge representation formalism, reasoner, and web portal¹² for upper-level ontologies and lexical semantics. The project was developed by Ontotext Lab. in cooperation with the Bulgarian Academy of Sciences. Note that the Ontology Mapping system used in SEKT is also named OntoMap but is mainly developed by the company Ontoprise and is a completely different system. The portal provides access to the most popular upper-level ontologies and lexical resources, together with hand-crafted mappings between them. It facilitates the evaluation and comparison of upper-level ontologies and lexical knowledge bases. The portal is based on a unified representation of the resources, a proprietary inference engine, and a mapping methodology. It includes a number of alternative viewers: HTML, DHTML, a stand-alone GUI application.

In order to provide a uniform representation of the ontologies and the mappings between them, OntoMap introduces a relatively simple meta-ontology called OntoMapO. The knowledge representation language is more complex than RDF(S) and similar to OWL Lite⁻ [dBPF04], but it also includes specific primitives for ontology-mapping.

The following upper-level ontologies are hosted:

- Upper Cyc Ontology
- EuroWordnet Top Ontology
- EuroWordnet Meta-Ontology
- WordNet Meta-Ontology
- WordNet Tops (the top 41 classes)
- MikroKosmos Top (the top 13 classes)
- OntoMap Meta-Ontology
- Protege Meta-Ontology
- Simple Ontology of Business Entities

¹²<http://www.ontotext.com/projects/OntoMap.html>

- SENSUS Top (the top 257 classes)

Mappings between EuroWordnet Top and the other ontologies were created. There are almost no direct mappings between the other ontologies, but the equivalence and subsumption relations are automatically propagated through the mapping to EuroWordnet Top.

Mapping Language The full description of the OntoMapO could be found in [KSD01b]. Here we present just its mapping primitives, as follows:

- *MuchMoreSpecific* - the 1st concept is much more specific than the second one; transitive relation. Inverse of *MuchMoreGeneral* and a specialization of *ChildOf*;
- *MuchMoreGeneral* - the 1st concept is much more general than the second one; transitive relation. Inverse of *MuchMoreSpecific* and a specialization of *ParentOf*;
- *TopInstance* - the 1st concept is the most general instance of the second one, which is a meta-concept. Inverse of *ExactClass* and a specialization of *InstanceOf*;
- *ExactClass* - the 1st concept is a meta-concept, the second concept is the most general instance of the first one. Inverse of *TopInstance* and a specialization of *ClassOf*;
- *ParentAsInstance* - the 1st concept is more general than all the instances of the second one which is a meta-concept. Inverse of *ChildAsClass*;
- *ChildAsClass* - the 1st concept is a meta-concept (class), all its instances are more specific than the second concept. Inverse of *ParentAsInstance*.

Automation support OntoMap does not automatically create mappings. It assumes that either a mapping exists or it may be created manually. Although it may seem that automatic mapping may reduce the efforts, in the case of upper-level ontologies the typical heuristics involved for domain ontologies can play a very limited role. This is explained in detail in [KSD01b]. Once a mapping to one of the ontologies it supports is created, OntoMap could automatically create a mapping to any of the other ontologies.

Applicability to use cases OntoMap could be used for (semi-)automatic creation of Ontology Mappings between other domain ontologies and existing ones, but it requires that a mapping exists to one of the supported ontologies. The different upper-level ontologies are suited for different purposes, thus, a domain ontology may naturally map to one of these, and then OntoMap will automatically provide a mapping to the rest. Although OntoMap does not directly address the use-case of instance transformation, the mappings it creates could be used for such tasks. It is important to mention that OntoMap handles

classes and instances in an uniform fashion and thus could transform instances to classes and vice-versa (via *ParentAsInstance* and *ChildAsClass* mapping primitives).

Tool support

- The OntoMap web portal (<http://ontomap.ontotext.com>) requires the users to register (it is free) and then it allows the browsing of ontologies via a handy DHTML Tree View. The search for concepts throughout one or more ontologies is also supported. The portal allows the export of the ontologies to DAML+OIL.
- *CYC to EWN-Top mapping*. An online service, hosted at <http://demo.ontotext.com>, allows the browsing of the EuroWordnet Top ontology and its mapping into Upper Cyc Ontology. The corresponding Cyc concepts are represented with: their glosses, direct and indirect super-classes (*#\$genls*), direct and indirect classes (*#\$isa*). The mapping itself is expressed in terms of a CycL microtheory encoding of the EuroWordNet Top Ontology on top of the publicly available part of the Cyc knowledge base. This approach was chosen because such a mapping is impossible by means of equivalence and subsumption relations only. However, a simplified relational view that is sufficient for many purposes, is also provided. More theoretical details can be found in [KS00].
- The *OntoMap Viewer* is a standalone java application, which represents the main functionality of the OntoMap web portal. OntoMap Viewer is distributed for all popular platforms: Windows, Linux, Solaris, MacOS (<http://www.ontotext.com/projects/OntoMapView/install.htm>). All of the previously mentioned ontologies are encoded into OntoMapO language. The viewer allows the browsing and searching by concepts from any ontology. An example, shown in Figure 4.9, illustrates the supported mappings between the upper-level ontologies. The user chooses an ontology and then selects a concept from it, e.g. *Person* from SENSUS Top ontology. Then the viewer shows any equivalence, super- and sub-concepts from all ontologies including the current one, but also the others, if there are equivalence and/or subsumption relations (in this case the concept is equivalent to *Human* from EuroWordnet Top and *Person* from UpperCyc).

Summary OntoMap provides a mapping model for upper-level ontologies, and a few of the most popular ones are encoded in it. Using the mapping to the EuroWordnet Top ontology and a reasoner to support the knowledge representation language, a mapping between all of the ontologies is available. Thus, a new mapping from a domain ontology to one of the supported upper-level ontologies could be automatically mapped to each of the other ones. However, OntoMap is focused on the evaluation and the comparison of the ontologies, which are encoded into OntoMapO, rather than on Ontology mapping or instance transformation services.

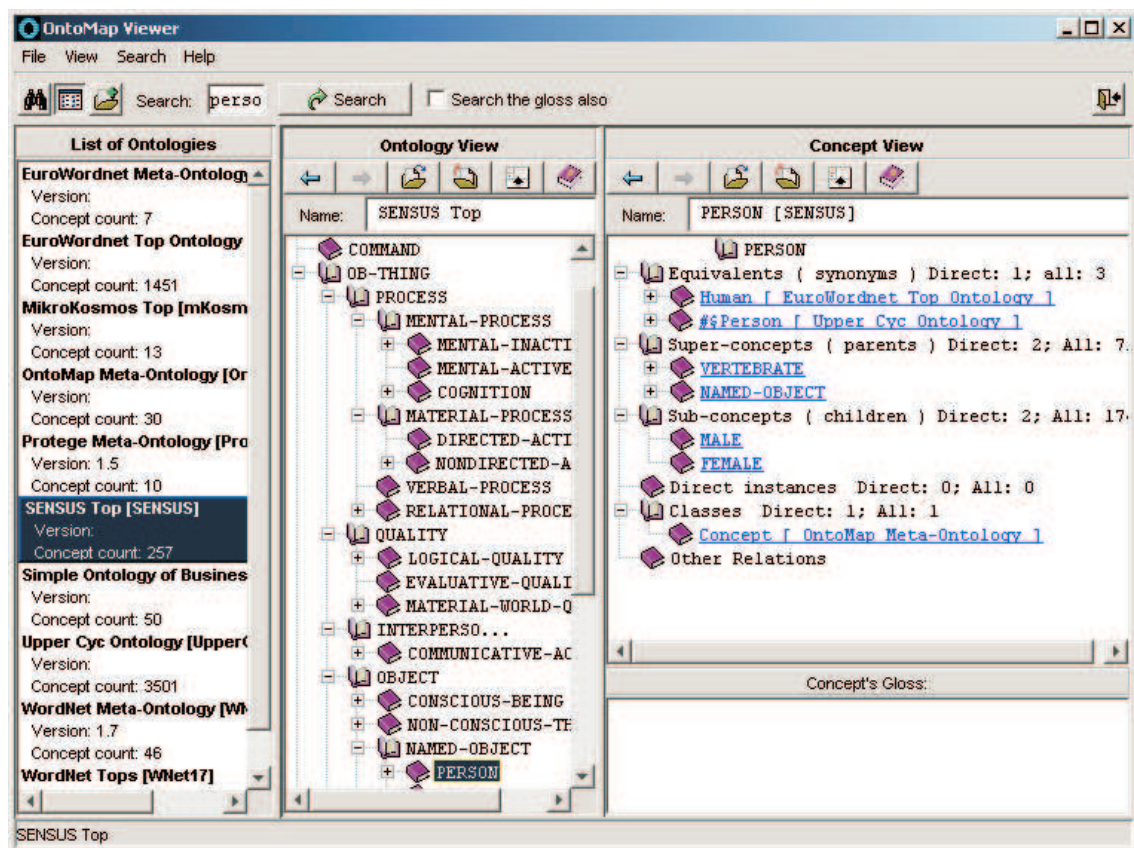


Figure 4.9: OntoMap Viewer - concept “Person” in different upper-level ontologies

4.1.7 RDFDiff

RDFDiff (formerly known as OntoView [KKOF02]) is an algorithm and a tool, developed jointly by VU Amsterdam and Ontotext Lab. It aims to detect and depict changes in RDF-encoded knowledge (such as ontologies encoded in RDF(S), DAML+OIL, and OWL). Basically, it is very much inspired by the CVS diff and the UNIX diff. It has been tuned for RDF(S) and compares the ontologies on a structural level, i.e. rather the data model of different RDF(S) ontologies than the syntactical encoding. RDFDiff utilizes change-classification rules, which are intended to serve as a basis for further semantic and structural analysis of the differences. Such rules can be classified as:

- robust categorization of the changes
- evaluation of the compatibility between the versions
- data-transformation or mapping

The change detection in RDFDiff includes:

- Matching anonymous resources and their descriptions
- Detection of renamed resources, based on the definition of the resource
- Detection of renamed resources, based on the usage of the resource (detection of sub-graphs, which do not change, given that the old ID/URI has been substituted with the new one)

The tool resembles PROMPTDiff [NM03a] in the following characteristics:

- change detection based on graphs
- an extensible set of rules to classify a change (called “heuristics” in both PROMPTDiff and RDFDiff)

It differs from PROMPTDiff in being able to compare XML-serializations of the RDF(S) ontology as well. RDFDiff detects changes mainly using the RDF-graph model, but it presents the results in a diff-like way (textual), while trying to preserve the order of the statements in the XML serialized files. It is focused on RDF(S) and thus it handles its specifics well (e.g. anonymous resource matching). However, the rules model the semantics of the logical language (i.e. Description Logics for DAML+OIL and OWL).

RDFDiff overview Essentially, the algorithm takes as input two RDF-encoded files, compares them, and produces a list of changes, organized into added, changed and deleted items.

It is important to mention that RDFDiff does not implement much semantics. The rules may implement the semantics of the data model of the RDF-encoded ontology but they need not to be used. RDFDiff does not need to be run on two ontologies (it could compare any arbitrary RDF-graphs) but it was developed to aid ontology comparison. In its first version RDFDiff was perceived as a diff for an old and a new version of an ontology, and thus the description of the algorithm, as well as the user interface of the tool implementing it, refer to the two ontologies being compared as "old" and "new". However, except for the visualization of the results, RDFDiff does not depend on the fact that the RDF graphs are two versions of the same graph and therefore it could well be used for any two ontologies.

The algorithm treats an RDF-encoded ontology as a sequence of resource definitions, where a definition of resource R is assumed to be the list of all statements where R is a subject. This grouping of the statements into resource definitions is optional, because the user might require the original grouping from the source file. In the case where the statements are not grouped into "resource definitions", they will be handled as grouped in first level XML elements. The resource definitions are ordered according to the new file, considering the position of the first statement from the definition.

Statements are considered to be changed *if and only if* the subject and the predicate in each versions match unambiguously, but the values might be different, i.e. when there is a single statement with such a subject and such a predicate in both versions. Because the resource definitions of the first file are reordered according to the second file, definitions of same terms that occur in different places in the two files can be found.

Mapping language The change-classification rules are defined via a simple RDFS schema, which contains two classes and a few properties. The classes are *Rule* and *Triplet*. Each rule is a set of triplets defining relationships between some triples via common variables used in place of 'subject', 'predicate' or 'object'. The main task is to find all the possible solutions (read possible bindings of the variables used in the triplets) where there exist triples that match the patterns of the whole set of triplets. Then all solutions found with the older version of the resource definition are compared against those found in its newer version, and the equal pairs are removed.

Each instance of a Rule class can be connected with several instances of the Triplet by the property *use*, which is defined as:

```
<rdfs:Property rdf:about="&rule;use" />
```

These are the properties that indicate which variable is to be observed for possible changes and what label should be added, removed or changed:

```
<rdfs:Property rdf:about="&rule;checkVar" /> <rdfs:Property
rdf:about="&rule;onChanged" /> <rdfs:Property
rdf:about="&rule;onAdded" /> <rdfs:Property
rdf:about="&rule;onRemoved" />
```

The instances of the Triplet class can be connected with predicates specifying a subject, a predicate and an object of a triple for matching. Each value of these predicates, starting with "\$", can be treated as a name of a variable, and the others as resource URIs or literals.

```
<rdfs:Property rdf:about="&rule;theSubj" /> <rdfs:Property
rdf:about="&rule;thePred" /> <rdfs:Property
rdf:about="&rule;theObj" />
```

The solutions are compared by the values bound to the used variables: if a value is a resource, by its URI, if it is a literal, lexically, and in the case of an anonymous resource, by the object value of the triple with a predicate "daml:onProperty". If there is at least one such solution in any of the versions, the rule is applicable and its 'label' is added.

If there are solutions which only differ by the value of the 'checkVar' variable: the generated 'label' is the one prepared by the literal, connected with a 'rule:onChanged' property to that rule instance. If there are no solutions in the older version, then for each 'extra' solution that is left unmatched in the newer version, a 'label' is generated from the literal, connected with a 'rule:onAdded' property to the rule. And in the case of unmatched solutions left from the older version, the rule label is generated with the literal connected to the rule with a rule:onRemoved property.

An example rule to detect the change of the parent class follows:

```
<rule:Rule rdf:about="&rule;ruleSubClassOf">
  <rule:use>
    <rule:Triplet>
      <rule:theSubj>$X</rule:theSubj>
      <rule:thePred rdf:resource="&rdfs;subClassOf" />
      <rule:theObj>$Z</rule:theObj>
    </rule:Triplet>
  </rule:use>
  <rule:checkVar>$Z</rule:checkVar>
  <rule:onChanged>subClassOf.changed to $Z</rule:onChanged>
  <rule:onAdded>subClassOf.added to $Z</rule:onAdded>
  <rule:onRemoved>subClassOf.removed to $Z</rule:onRemoved>
</rule:Rule>
```

Application to use-cases When applied to two ontologies, RDFDiff will find structural similarities. It does not utilize semantic (S-Match) or linguistic (PROMPT) similarities.

The change-classification rules could be a powerful declarative syntax for the automatic detection of changes or for the automatic creation of a mapping between two ontologies. Instance transformation is also possible, but all of the applications require extensions of the allowed actions in the rules.

Tool support The RDFDiff tool is web-based. It implements the RDFDiff algorithm, and in addition it allows one to specify some handy compare options:

- pairs of namespaces treated as equal;
- an ignore-list of properties (e.g. ignoring *rdf:Comment*).

Line 182 : BusinessObject	Line 184 : BusinessObject
subClassOf changed to IntangibleObject	
<rdfs:Class rdf:about="skimo_rdfs:BusinessObject" rdfs:label="BusinessObject"> <rdfs:comment>An almost abstract entity being used in business context. This includes markets, industry sectors, brands, etc. Many products can also be seen as a business abstraction, but most of the products bear other important aspects, such as engineering and design.</rdfs:comment> <rdfs:subClassOf rdf:resource="skimo_rdfs:IntangibleObject"/> </rdfs:Class>	<rdfs:Class rdf:about="skimo_rdfs:BusinessObject" rdfs:label="BusinessObject"> <rdfs:comment>An almost abstract entity being us context. This includes markets, industry sectors, b products can also be seen as a business abstraction, bear other important aspects, such as engineering an <rdfs:subClassOf rdf:resource="skimo_rdfs:Object </rdfs:Class>
Line 188 : CDG	Line 190 : CDG
subClassOf changed to LexicalResource	
<rdfs:subClassOf rdf:resource="skimo_rdfs:LexicalResource"/> </rdfs:Class>	<rdfs:subClassOf rdf:resource="skimo_rdfs:NERLex </rdfs:Class>
Line 201 : CalendarMonth	Line 199 : CalendarMonth
subClassOf changed to CalendarEntity	
<rdfs:subClassOf rdf:resource="skimo_rdfs:CalendarEntity"/> </rdfs:Class>	<rdfs:subClassOf rdf:resource="skimo_rdfs:Tempor </rdfs:Class>
Line 305 : ContactInformation	Line 303 : ContactInformation
comment added	
<rdfs:Class rdf:about="skimo_rdfs:ContactInformation" rdfs:label="ContactInformation"> <rdfs:subClassOf rdf:resource="skimo_rdfs:Abstract"/> </rdfs:Class>	<rdfs:Class rdf:about="skimo_rdfs:ContactInformation" rdfs:label="ContactInformation"> <rdfs:comment>Any instance of a particular notat allow contacting an individual or organisation.</rdf <rdfs:subClassOf rdf:resource="skimo_rdfs:Abstre </rdfs:Class>

Figure 4.10: RDFDiff - an example

Summary RDFDiff is a diff-like tool oriented to the comparison of XML-serialized RDF(S) graphs. It could be used for change detection between two versions of an ontology, or for a comparison of two arbitrary ontologies. Although not directly suited for mediation use-cases, its change classification rules allow for applications for automatic creation of mappings or instance transformation.

4.1.8 OntoMerge

Dou et al.[DMQ02] introduce an approach to ontology mediation “ontology translation by 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, Dou et al.[DMQ02] 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 [dBP04]). 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 [dBP04]. However, query rewriting is a technique for solving the problem of querying different ontologies, whereas Dou et al. [DMQ02] merely mention the problem.

As we have also suggested in [dBP04], OntoMerge 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 [dBP04].

[DMQ02] presents an internal representation for the ontologies, called *Web-PDDL*, which is a typed first-order logic language. The import and export of DAML+OIL and OWL is already supported. Due to the fact that Web-PDDL can capture many different ontology languages, importers and exporters for other languages could be written as well.

Dataset translation OntoMerge [DMQ02] performs 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 thousand 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} will contain new predicates with identifiers generated using skolem constants. The new predicates are made sub-predicates of the existing predicates.

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 [DMQ02]. However, we do not presume the source and target ontologies use the same language as the mapping, whereas OntoMerge 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.

4.1.9 OMEN

OMEN (Ontology Mapping Enhancer) [MNJ04] is a tool for describing mappings with probabilities and infer new mappings by means of Bayesian Network inference mechanisms. The motivation for using Bayesian Networks for this purpose is that especially

when the mappings are discovered automatically by heuristics or machine learning techniques, they are imprecise and attached with a certain uncertainty. Also experts are sometimes unsure about the exact match between ontological elements of different ontologies. This allows them to assign some certainty rating to a match.

The inference of new mappings happens by means of meta-rules based on the semantics of the ontology relations that expresses how each mapping affects other related mappings. These meta-rules are used by the Bayesian Network.

Within this tool, a simple ontology model is assumed. Similar to RDF Schema, only the following components are used to express ontologies:

- **Classes** are concepts in the domain that are organized in a hierarchy of subclass superclass relationships with multiple inheritance.
- **Properties** describe attributes of classes and relationships between classes. Properties are considered to have one or more domains and one or more ranges. Domains are classes to which the property can be applied, i.e. the union of the domains that are possible is taken as final domain. Ranges restrict the classes for the values of the property, i.e. the intersection of the ranges that are possible is taken as final range.

Construction of the Bayesian Network The nodes in the Bayesian Network correspond to individual pairs of matches, i.e. matches between individual pairs of classes or properties from different ontologies. Arrows in the Bayesian Network represent the influences between the nodes in the Bayesian Network. Each node is associated with a conditional probability table (CPT) that represents the influences of the parents of a node that affect the node itself.

Note that not all possible pairs of classes or properties can be represented as nodes in the Bayesian Network because there are too many pairs and the inference would not be efficient. Therefore, the Bayesian Network is pruned. For details on the pruning see [MNJ04]. Also the number of parents that each nodes has is adjusted because too many parents significantly increase the computations that are necessary and make the inference intractable. Furthermore, all cycles that may occur with the arrows are broken by rejecting the edges from the parents whose matching information is minimum.

For inferencing, the nodes that have an initial probability that is above a given threshold are considered to be part of the evidence. The CPTs are computed by means of a set of generic meta-rules that enables the automatic generation of CPTs for each particular pair of ontologies. The implementation of the OMEN system is parametrized with respect to the meta-rules. Meta-rules can be added and removed in order to enable a better evaluation of which one work best for a particular knowledge model.

Afterwards the inference on the Bayesian Network can be started and the output is a new set of matches.

The set of meta-rules used in the implementation are for example the following meta-rules:

- Say there are two concepts C and C' that match and there is a relationship between C and another concept C_1 in the ontology O and a relationship between C' and C'_1 in the ontology O' . Furthermore these two relationships match. Then, the probability of the match between C_1 and C'_1 is increased.
- Say there are two concepts C and C' that match and there is a relationship between C and another concept C_1 in the ontology O and a relationship between C' and C'_1 in the ontology O' . Furthermore these two relationships do not match. Then, the probability of the match between C_1 and C'_1 is decreased.
- Say there are two properties that match and each of them has a single range. Then, the probability of a match between the classes that represent the ranges is increased.
- Say there are two properties that match and the first property has a range that is a union of the classes C_1 and C_2 . The other property has a single range corresponding to C' . Then, the probability that C_1 is a specialization of C' can be increased. Analogously, the probability that C_2 is a specialization of C' can be increased.
- Say there are mappings between superconcepts of two certain concepts, each belonging to a different ontology, and all the siblings. Then, the probability of a match between the remaining concepts is increased.

Note that in the OMEN system probabilistic influences are combined as follows: If a node in a Bayesian Network has two parents, the conditional probability tables are combined for the child using the assumption that the two parents are independent. I.e. $P(N|P_1, P_2) = P(N|P_1)P(N|P_2)$

When the match of two pairs of parents influences each other, this assumption is not true. However, the system obtained encouraging results even with this simplifying assumption.

Experiments were conducted in order to see whether the inference of new mappings works. For this purpose, two ontologies expressed in RDF using RDF Schema have been used. The experiments showed that new mappings can be inferred by OMEN. E.g. by giving only 3 out of 11 matches, up to 7 missing matches were able to be generated.

4.1.10 WSMT Data mediation module

The Web Services Modelling Toolkit (WSMT) [Ker05] has different modules to model Semantic Web Services based on the WSMO framework [DR05]. The data mediation module [Moc05] presented in this section contains three sub-modules or components: a graphical user interface for defining the mappings, a component that reads the mappings

from the storage and generates the appropriate mapping rules and an environment that provides the means for executing the mapping rules. The first module represents the design-time component while the second two modules represent the run-time component. Figure 4.11 shows how these components interact.

Overview of the Data Mediation Module

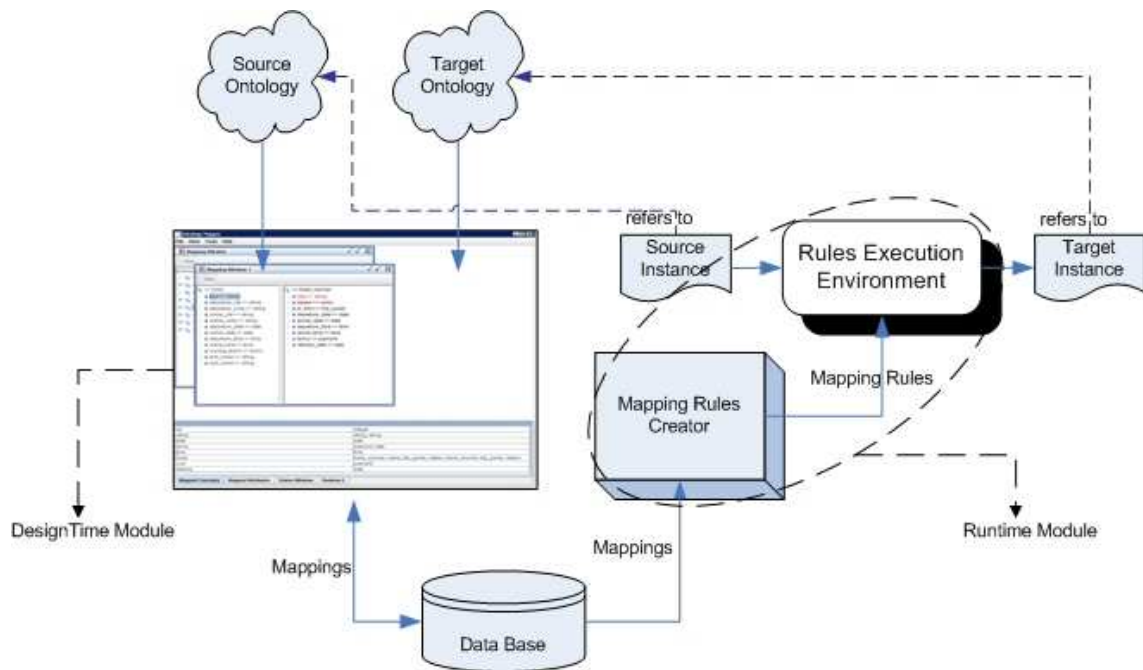


Figure 4.11: Overview of the WSMT Data Mediation Module

Design Time Component

The mapping tools offer graphical interfaces in order to assist the user in the mapping creation and to reduce his or her effort to simple choices and validations.

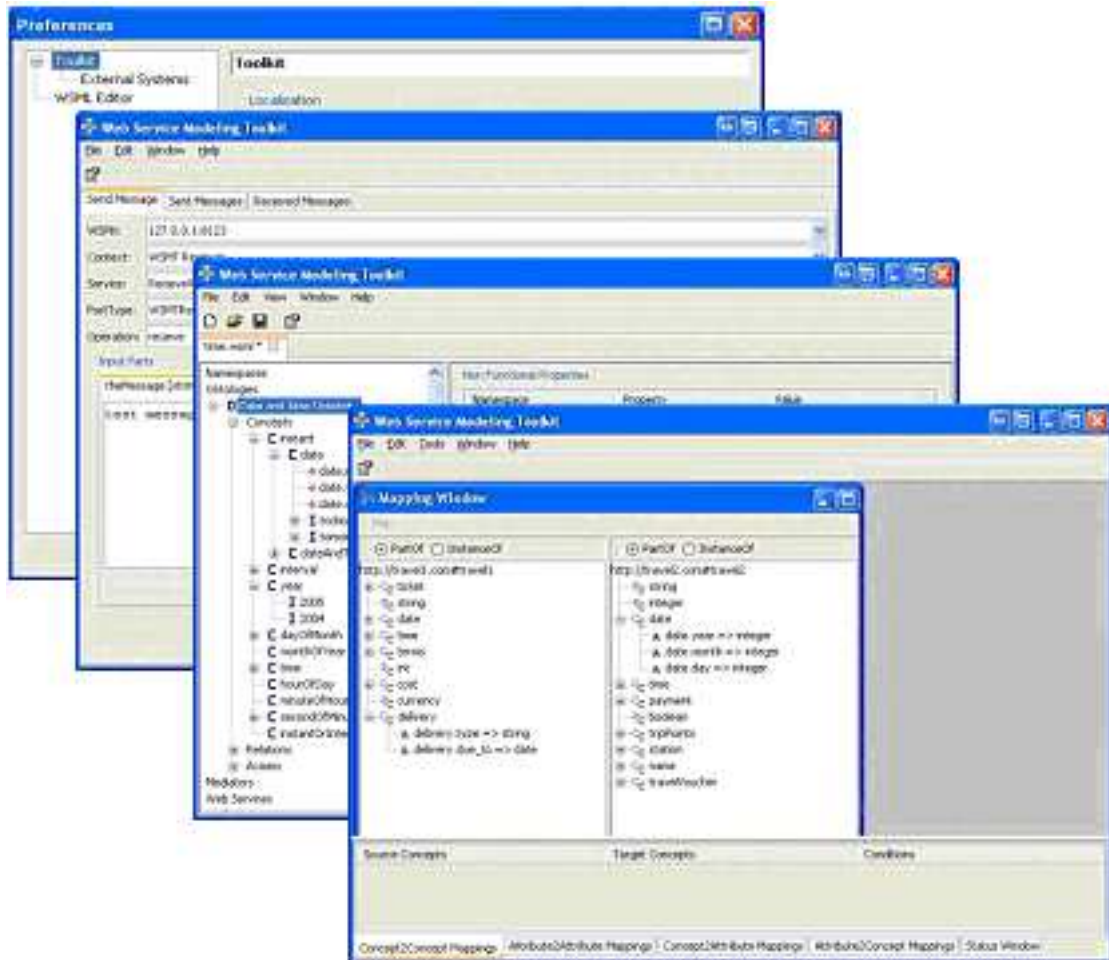
The graphical interface is built using Java 1.5¹³ and the default ontology language supported is WSMML. Future versions are expected to propose various wrappers that could transliterate ontologies expressed in other languages into WSMML (a simplified one for Flora-2 already exists). The object model behind the mapping tool is fully compatible with the WSMO API and WSMO4J (and this applies to the runtime component as well). The WSMO API provides a set of interface for manipulating WSMO entities, including ontologies. WSMO4J provides an implementation of these interfaces and a parser for WSMML¹⁴. The mapping tool was integrated as a plug-in in the Web Service Modelling

¹³see <http://java.sun.com> for more information

¹⁴for more details see <https://sourceforge.net/projects/wsmo4j>

Toolkit, which also offers as plug-ins a WSMML editor and a WSMX invoker (see Figure 4.12). The editor can be used by the domain expert to operate on the ontologies they are mapping and the framework offers basic synchronization functions such as reloading the ontologies into the mapping tool or invalidating mapping rules affected by changes in the ontologies. The WSMML invoker can be used to test mappings with run-time components deployed as Web Services, directly from the mapping tool (more details about this are provided in section 4.1.10).

Figure 4.12: Screenshots of WSMT Plug-ins (from back to front: the Preferences Panel, WSMX Invoker, WSMML Editor and Mapping Tool)



As the mapping tool and WSMT are two of the surrounding efforts around WSMO and WSMX, they are part of the WSMX open source project¹⁵ - Windows and Linux installers for WSMT can be found on SourceForge¹⁶.

¹⁵see <https://sourceforge.net/projects/wsmx>

¹⁶downloads available at: http://sourceforge.net/project/showfiles.php?group_id=113321

Map<Identifiable, List<Identifiable>>	mediate(Ontology sourceOntology, Ontology targetOntology, Set<Identifiable> data) Transforms a set of source ontology instances into instances of the target ontology.
List<Identifiable>	mediate(Ontology sourceOntology, Ontology targetOntology, Identifiable data) Transforms a given source ontology instance into instances of the target ontology.
StringBuffer	mediate(Ontology sourceOntology, Ontology targetOntology, StringBuffer payload) Transforms source ontology instances into instances of the target ontology. The payload represents a WSMML document containing the instances to be mediated. It will be parsed and after the mediation takes places a new WSMML document is created containing the target instances.

Table 4.1: Interfaces of the Run-time Mediator Method Summary

The mappings are stored in an external storage, in this case a relational database, from where they can be loaded by the mapping rules generator module. Also, by means of this graphical interface, the user can load existing mappings from the external storage for further refinements or as support in computing the suggestions.

Run-time component

The Mapping Rules Creator implements the grounding mechanism described in Section 4.2 while the Rule Execution Environment has the role of executing the mapping rules against the incoming source ontology instances. The Run-time component is designed to be part of the WSMX architecture [MZ05], and as a consequence it offers well-defined interfaces explaining how it can be invoked. These interfaces are presented in Table 4.1.

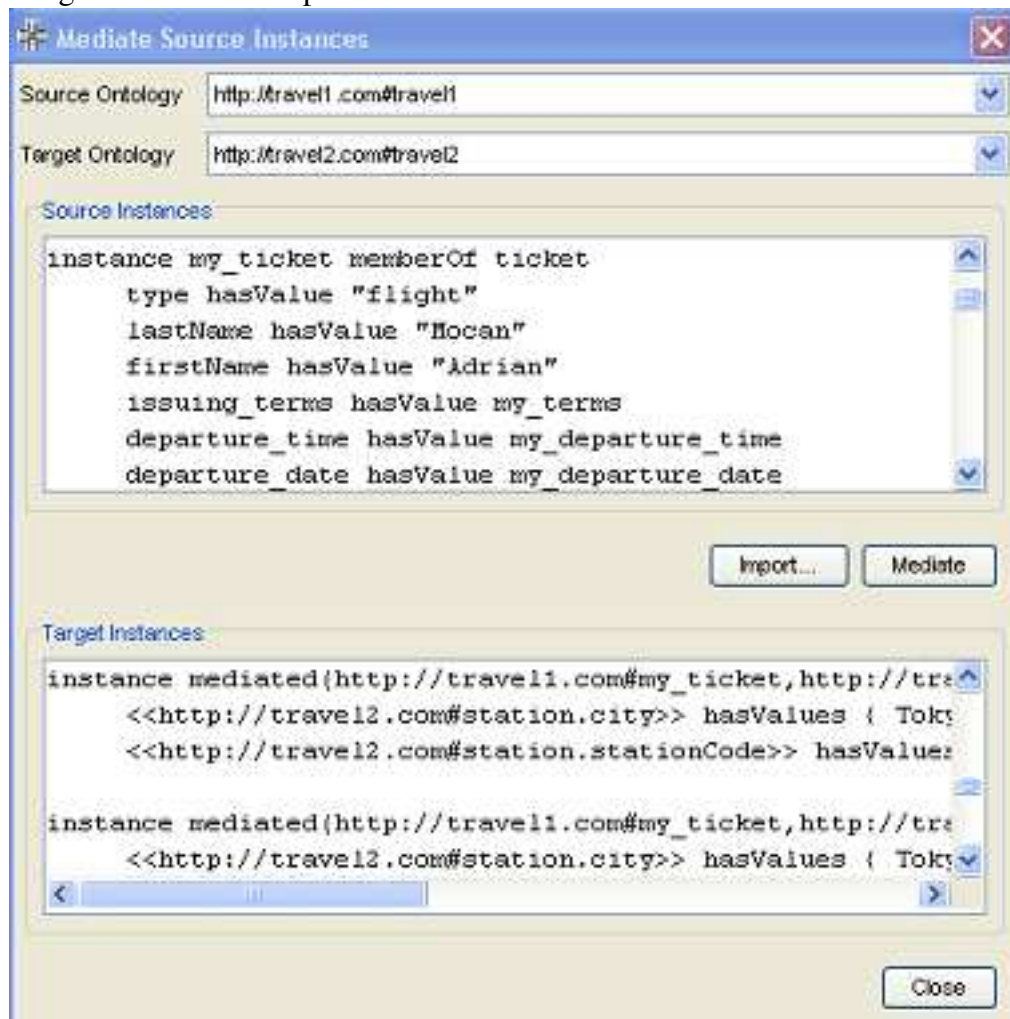
The first two methods refer to the ontological entities in terms of the WSMO API objects, while the last method manipulates WSMML documents embedded in StringBuffer objects.

Outside the WSMX architecture the Run-time component can be used in the following

ways:

- as a standalone application able to connect to the provided mapping storage and to perform mediation of instances provided as WSMML documents. It offers a small graphical interface where the user can set the source and the target ontology, provide the data to be mediated and retrieve the mediated data.

Figure 4.13: The Graphical Interface of the Stand-alone Run-time Mediator



- as a deployed Web Service that can be invoked with the source and target ontology IDs and data to be mediated and that returns the mediated data. The mapping tool can directly invoke such Web Services by using the WSMT Invoker.

4.1.11 DOME mapping module

The Distributed Ontology Management Environment(DOME [Hen05])¹⁷ is an ongoing project aiming at building a suite of tools to manage distributed ontologies. Proposed as a stand-alone application or as an Eclipse plug-in, it includes three modules: an ontology editor, a mapping editor, and a versioning editor. DOME is developed using concepts from Human Interface Design and ontology engineering to facilitate the ontology management in being easily usable and intuitive. Figure 4.14 present a screen-shot of the software.

Figure 4.14: Screenshot of the Distributed Ontology Management Environment(DOME)

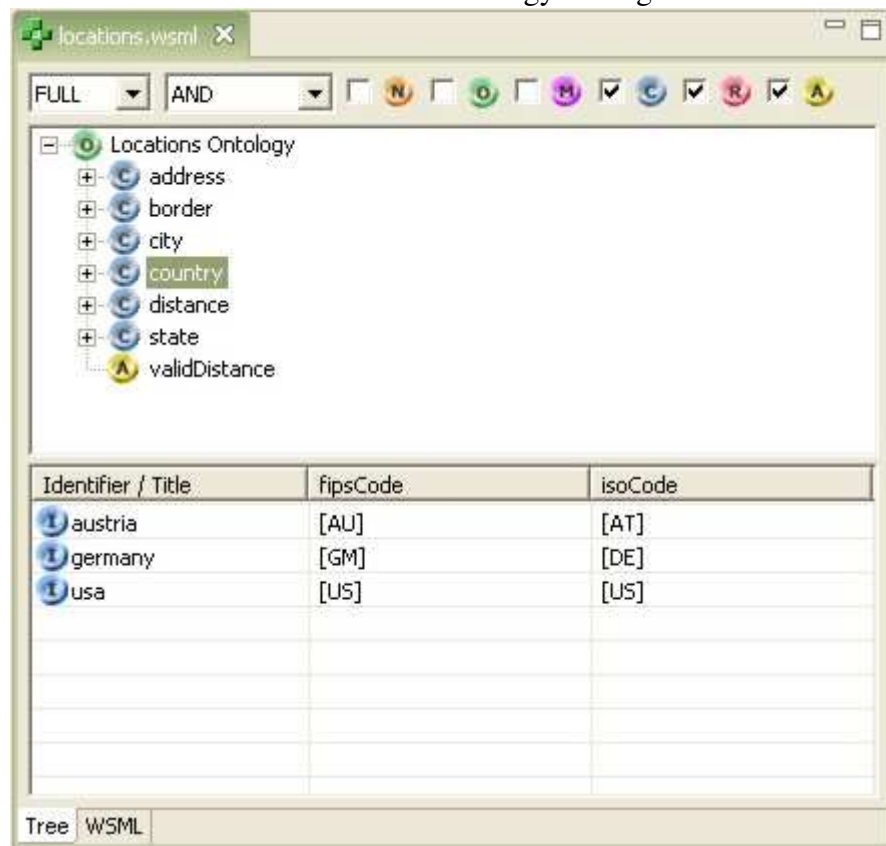


Figure 4.15: DOME screenshot

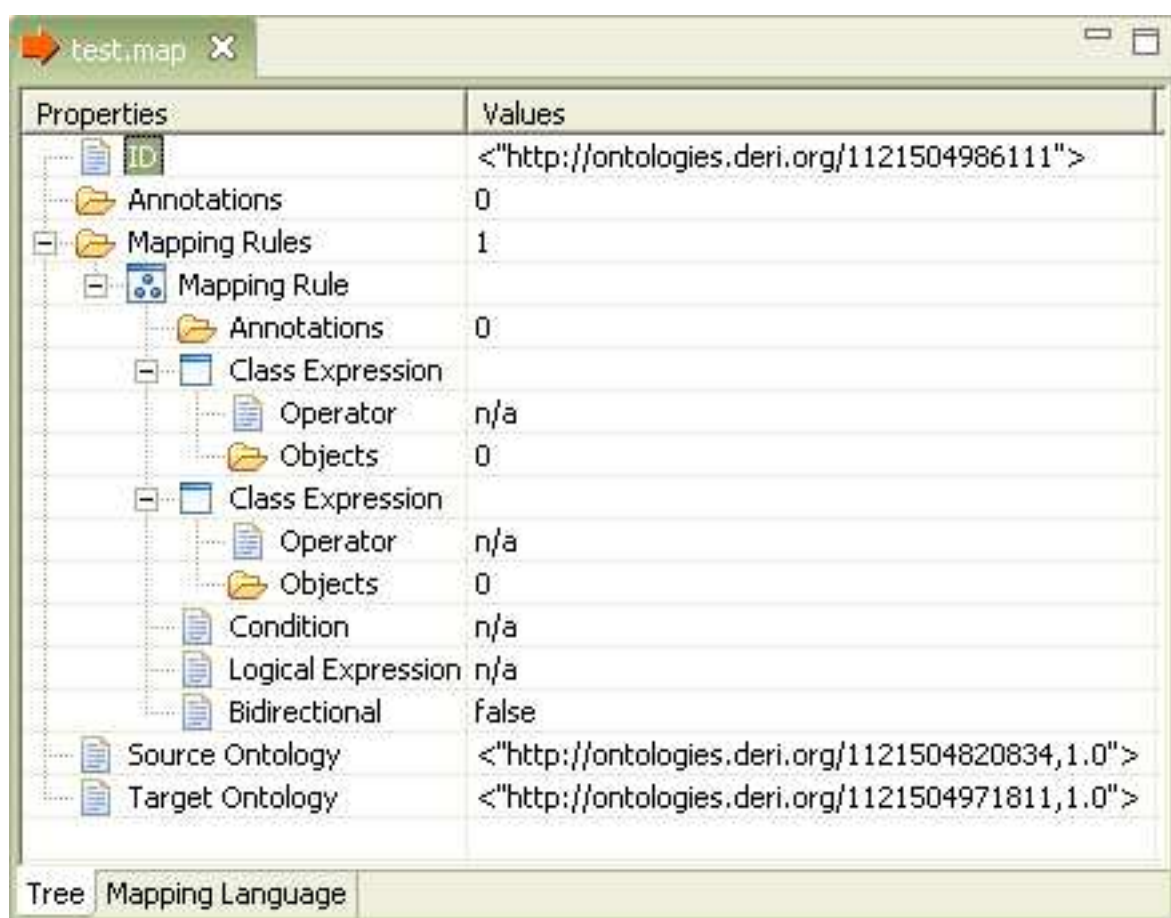
Dome is based on WSMO4j to editing WSML ontologies.

The mapping editor is based on an abstract ontology mapping language[SdB05] and presents an original mapping document tree view. It allows mapping between classes, attributes, relations and instance, and exploits the capability of the abstract mapping language to relate many to many entities by operator selection. It is to our knowledge the only mapping tool providing this capability.

¹⁷The software is available at <http://dome.sourceforge.net>

In order to realize a mapping, the user select one or many entities in the source ontology, and drags'n'drops them to the entities in the target ontologies. If many entities were selected, they can be combined via a selected operator. It is in then possible to map, for example, the conjunction of *male* and *female* classes to the *person* class using a unique mapping rule. The mapping document tree editor gives the possibility of adding conditions to the rules, for example mapping *person* to *female* require the condition that the *gender* attribute of *person* is equal to "*female*". Figure 4.16 presents the mapping document tree editor.

Figure 4.16: Mapping document tree editor



It is interesting to note that the mapping module is fully integrated in the whole ontology management environment, to realize the mapping the user only has to open another ontology and can directly relate the entities.

4.2 Integrated Systems

4.2.1 InfoSleuth

Summary InfoSleuth [FNPB99, NFK⁺00] is an agent-based system, which supports construction of complex ontologies from smaller component ontologies so that tools tailored for one component ontology can be used in many application domains. The purpose of the system is to provide an interface to very dynamic data sources which can appear and disappear from the system at any given time. Examples of reused ontologies include units of measure, chemistry knowledge, geographic metadata, and so on. Mapping is explicitly specified among these ontologies as relationships between concepts in one ontology and related concepts in other ontologies.

All mappings between ontologies are maintained by a special class of agents known as resource agents. A resource agent encapsulates a set of information about the ontology mapping rules, and presents that information to the other agents in concepts of one or more ontologies (called *domain ontologies*). All mapping is encapsulated within the resource agents. Ontologies are represented in OKBC (Open Knowledge Base Connectivity) [CFF⁺98] format and stored in an OKBC server by a special class of agents called ontology agents, which provide ontology specifications to users (for request formulation) and to resource agents (for mapping).

The InfoSleuth architecture [NFK⁺00] (Figure 4.17) consists of a number of different types of agents. User agents and resource agents are the main agents in the system. User agents request information to fulfil the user's information needs and resource agents provide that information. The remaining agents in the system provide the “glue” (or *mediation*) between the two.

- The *user agents* act on behalf of the user and maintain the user's state. They provide a system interface that enables users to communicate with the system.
- The *resource agents* wrap and activate databases and other repositories of information. They translate queries and data stored in external repositories between their local forms and their InfoSleuth forms. There are resource agents for different types of data sources, including relational databases, flat files, and images.
- *Service agents* provide internal information to the operation of the agent system. Service agents include *Broker agents*, which collectively maintain the information the agents advertise about themselves, *Ontology agents*, which maintain a knowledge base of the different ontologies used for specifying requests, and *Monitor agents*, which monitor the operation of the system.
- *Query and analysis agents* fuse and/or analyze information from one or more resources into single (one-time) results. Query and analysis agents include *Multi-resource query agents*, which process queries that span multiple data sources, *De-*

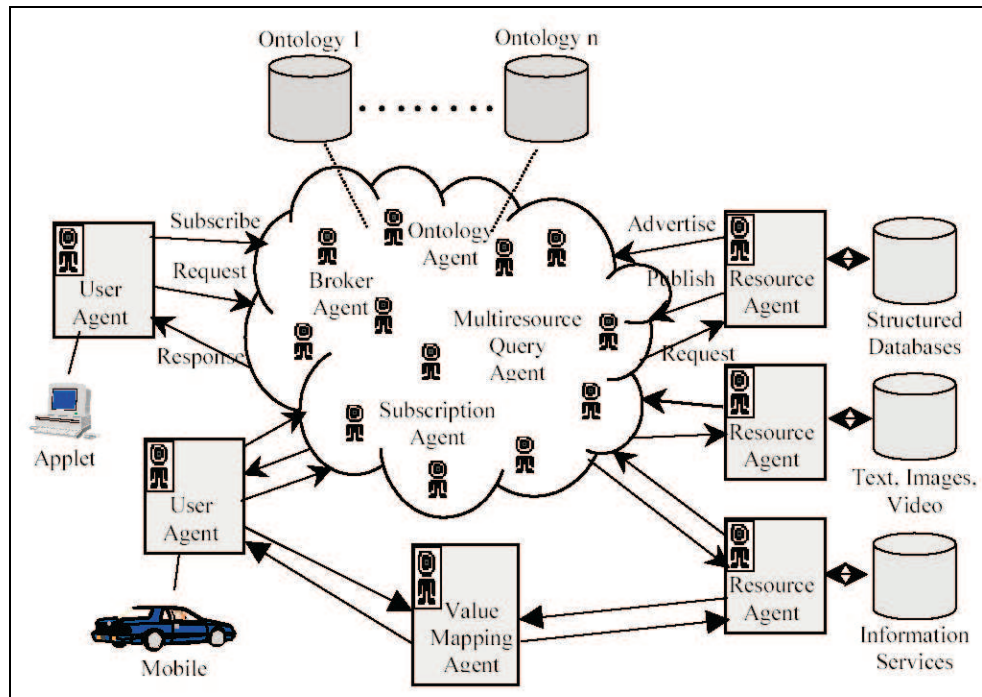


Figure 4.17: The InfoSleuth architecture

violation detection agents, which monitor streams of data to detect deviations, and other data mining agents.

Multi-resource query agents query multiple heterogeneous resources. The queries posed to the agent are specified in terms of some domain ontology. In InfoSleuth, applications can use several domain ontologies. However, a query is always posed over one domain-specific ontology.

- *Planning and temporal agents* guide the request through some processing which may take place over a period of time, such as a long-term plan, a workflow, or the detection of complex events. Planning and temporal agents include *Subscription agents*, which monitor how a set of information (in a data source) changes over time, *Task planning and execution agents* plan the processing of user requests in the system, and *Sentinel agents* monitor the information and event stream for complex events.
- *Value mapping agents* provide value mappings, i.e. mappings between values, among equivalent representations of the same information.

InfoSleuth uses a variant of SQL in which a query consists of a *select*, *from* and *where* clause as its query language. Functions are allowed in the *select* and *where* clauses and the syntax is consistent with that used in popular relational database management systems. For the user queries, a layer on top of this query language has been developed,

called Template-based Query Markup Language (TQML), which uses templates and materialized views to aid the user in creating queries.

When agents come online they advertise their capabilities to a specific broker agent in terms of the InfoSleuth ontology. This ontology is a special ontology used for advertisement and querying of agents. When a query is posed to a broker agent, the brokering process is initiated. First, syntactic matching is done for example to, determine which resource agents speak the desired language. The semantic matching is done in order to find out which resources contain information about the desired concepts. Finally, pragmatic matching is done to restrict the set of resources to those that, for example, have the correct access permissions.

Resource agents in InfoSleuth function as a wrapper of the underlying data source. A resource agent advertises the part of the overall domain ontology that it supports, advertises its query capabilities and does the query rewriting and transforms the retrieved data to facts of the domain ontology.

Mappings between different value domains are encapsulated in *value mapping agents*, which perform simple and complex mappings between domains. Examples of complex mappings are sophisticated functions (e.g. differences in time intervals) and incorporating values from (multiple) external ontologies.

The execution of queries is done by the *query agent*. This query agent decomposes a query into a number of subqueries, one for each resource agent involved in the query. Furthermore, it creates a number of global queries for fusing the results of the subqueries in order not to have redundancy in the overall query result.

Ontology Languages Ontology agents which provide an OKBC interface to the knowledge base can all be connected to the InfoSleuth agent system. All ontologies within InfoSleuth are expressed using the OKBC knowledge model. Each resource agent must wrap an external information source and provide a mapping with the domain ontologies currently in use in the InfoSleuth system.

Mapping Language [NFK⁺00] reports no mapping between ontologies in InfoSleuth. In fact, this was seen as future work. However, because InfoSleuth is mostly a data integration system, it is more relevant that a mapping between data sources and the domain ontologies is possible. InfoSleuth does not provide a mapping language, but does provide a number of Java templates, which can be used for the development of wrappers, which contain a procedural mapping between the data schema and the domain ontologies in the agent system. An important point here is that it is possible to map to multiple domain ontologies and it would be very interesting to combine this with actual mappings between ontologies, as is done in ONION [MWK00], for example.

Mapping Patterns Although some aid in the creation of mappings through the use of Java templates is offered to the user, there is no concept of mapping patterns in InfoSleuth. Extensions of InfoSleuth, which would enable mapping between ontologies, would benefit from the use of mapping patterns. However, we are not aware of any continuation of the work on InfoSleuth after the work reported in [NFK⁺00].

Automation Support There is no automation support in creating mappings between data schemas and ontologies. However, the query rewriting and data fusion is completely automated, based on the mappings between the data schemas and the ontology. A query written in terms of a domain ontology is automatically decomposed in terms of the resources, and after execution the results are automatically fused by a different decomposition of the original query.

Applicability to Use Cases As stated above, the resource agents take care of transforming data from the underlying sources to the ontology representation of the system and also of rewriting the query in terms of the data schema.

The querying agent fuses query results from different sources in order to remove redundancies. The fusion of query results is based on a different decomposition of the user query, which defines a union of the query results and eliminates any redundancy in the results.

Implementation The InfoSleuth agent system has been implemented in two prototype projects.

There are Java templates available to make the development of new agents easier. To create a resource agent using such a template, it is generally sufficient to just supply a configuration and a mapping file to complete the agent [NFK⁺00]. It is possible to use different ontologies in an InfoSleuth system. Each OKBC-compliant Knowledge Base can be used in InfoSleuth by wrapping it using an *ontology agent*.

Experiences [NFK⁺00] reports the use of InfoSleuth in two prototype projects. The first is the EDEN (Environmental Data Exchange Network) project. The aim of the EDEN project was to provide integrated access to environmental information resources over the Web. EDEN posed many challenges in the area of the integration of legacy databases and mappings of values of different representations of similar information.

Another prototype project in which InfoSleuth was applied is MCC's Competitive Intelligence System [NFK⁺00].

4.2.2 ONION

Summary ONION (ONtology compositiON) [MWK00, MW01] is an architecture based on a sound formalism to support a scalable framework for ontology integration that uses a graph-oriented model for the representation of the ontologies. The special feature of this system is that it separates the logical inference engine from the representation model (the graph representation) of the ontologies as much as possible. This allows for the accommodation of different inference engines in the architecture.

In ONION there are two types of ontologies, individual ontologies, referred to as *source ontologies* and *articulation ontologies*, which contain the concepts and relationships expressed as articulation rules (rules that provide links across domains). Articulation rules are established to enable knowledge inter-operability, and to bridge the semantic gap between heterogeneous sources. They indicate which concepts individually or in conjunction, are related in the source ontologies [MWK00]. SKAT (the Semantic Knowledge Articulation Tool) [MWJ99] uses the structure of these graphs together with term-matching and other rules to propose articulation rules for the articulation ontologies. The source ontologies are reflected in the system by the use of wrappers.

The mapping between ontologies is executed by ontology algebra [Wie94, MW01]. Such algebra consists of three operations, namely, intersection, union and difference. The objective of ontology algebra is to provide the capability for interrogating many largely semantically disjoint knowledge resources, given the ontology algebra as input. The description of the algebra operators is as follows:

- The *intersection* produces an ontology graph, which is the intersection of the two source ontologies with respect to a set of articulation rules, generated by an articulation generator function. The nodes in the intersection ontology are those that appear in the articulation rules. The edges are those edges between nodes in the intersection ontology that appear in the source ontologies or have been established as an articulation rule. The intersection determines the portions of knowledge bases that deal with similar concepts.
- The *union* operator generates a unified ontology graph comprising the two original ontology graphs connected by the articulation. The union presents a coherent, connected and semantically sound unified ontology if the original ontologies are consistent.
- The *difference* operator, to distinguish the difference between two ontologies ($O_1 - O_2$) is defined as the concepts and relationships of the first ontology that have not been determined to exist in the second. This operation allows a local ontology maintainer to determine the extent of one's ontology that remains independent of the articulation with other domain ontologies so that it can be independently manipulated without having to update the articulation.

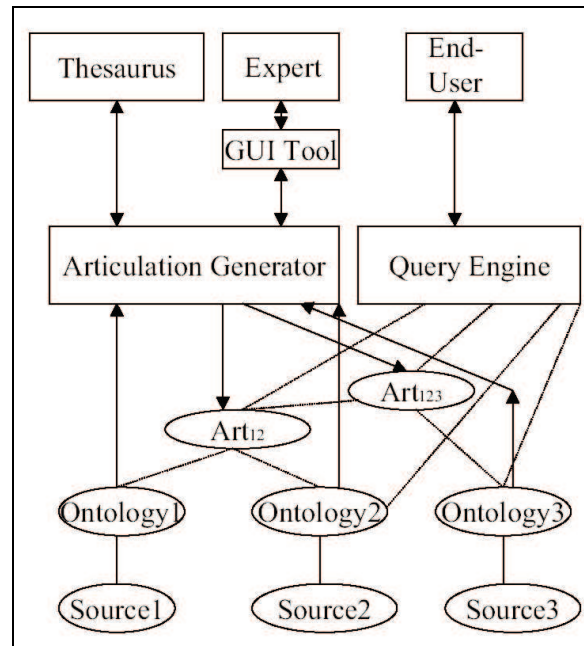


Figure 4.18: The components of the ONION system

ONION tries to separate as much as possible the logical inference engine from the representation model of the ontologies, allowing the accommodation of different inference engines. It also uses articulations of ontologies to inter-operate among ontologies. These articulation ontologies can be organized in a hierarchical fashion. For example, an articulation ontology can be created for two other articulation ontologies that unify different source ontologies. The ontology mapping is based on the graph mapping, and at the same time, domain experts can define a variety of fuzzy matching.

Ontology Language Before ontologies are integrated in the ONION system, they are translated to the ONION graph-based conceptual model. An ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R . The graph $G = (V, E)$ consists of a finite set of nodes V and a finite set of edges E . Nodes in the graph correspond to *concepts* in the ontology. Edges correspond to *semantic relationships* between the concepts.

In the ONION conceptual model, there are several semantic relationships with a built-in meaning, namely $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$. Furthermore, the user can create user-defined semantic relationships. The user then has to axiomatize the meaning of the relationship. The better the meaning of the relationship is axiomatized, the more accurate the articulation will be. A more detailed description of the meaning of the built-in semantic relationships can be found in [MW01].

An ontology graph can be represented in the Semantic Web language RDF [LS99], because RDF has a graph-based data model. The set of logical rules R are expressed as

Horn clauses.

An ontology in any source language can be translated to the graph-based model using a custom wrapper. It could happen that during the translation to the ONION conceptual model, some semantic information is lost. This information can no longer be used for the articulation of relationships with other ontologies, however, the user can still access this information by querying the underlying ontology directly.

Mapping Language Inter-operation in ONION is achieved through the use of *articulation ontologies*. An articulation ontology denotes the semantic intersection of two source ontologies. The intersection is an operation in the so-called *ontology algebra* [Wie94].

The articulation ontology is constructed based on so-called *articulation rules*. An articulation rule specifies the relationship between concepts in the source ontologies. An articulation rule is a rule of the form $P \Rightarrow Q$, which can be intuitively read as “P semantically implies Q”. In other words, P is a specialization of Q , or “P is *subsumed by* Q”.

ONION distinguishes between simple and compound rules. A simple articulation rule, which specifies the relationship between nodes in two ontology graphs, is of the form $O_1.A \Rightarrow O_2.B$, where A depicts a node in ontology O_1 and B depicts a node in ontology O_2 . The rule specifies the fact that A is a specialization of B . This rule translates to the simplest semantic bridge, the *semantic implication bridge*, which is an edge (A , “SIBridge”, B), connecting the two nodes. Compound rules incorporate conjunction and/or disjunction in the rule. Such rules are modeled by adding one or more nodes to the articulation ontology and creating the appropriate semantic implication bridges between the nodes in the source ontologies and the new node in the articulation ontology. For more information, see [MWK00].

In order to allow for value transformations, ONION offers the possibility of associating a function with an edge in the articulation ontology. Examples of such functions are currency conversion and conversion between different distance measures.

Automation Support The articulation rules are created in a semi-automatic process with SKAT [MWJ99] (Semantic Knowledge Annotation Tool), which proposes articulation rules to the expert and the expert can either accept or decline these proposals and also specify rules which are not proposed by the tool.

SKAT does matching of the two source ontologies using both term matching and structural matching.

Applicability to use cases ONION is a system for the unification of heterogeneous ontologies through the use of articulation ontologies with the purpose of query processing. The resulting articulation ontology is presented to the user and is used (together with the

source ontologies) by the user for querying. The ONION query system translates a query on the articulation ontology to the actual source ontologies and executes the query on the underlying ontologies. The results are then translated back to the representation of the articulation ontology.

ONION does not propose a strategy for unifying instances. The ontology obtained from applying the union operator can be seen as the result of a merge operation.

The complete mapping process is included in ONION. In fact, the mapping is just one aspect of ONION, because ONION also provides the run-time environment for data integration.

Implementation The ONION architecture [MWK00, MW01] (Figure 4.18, taken from [MW01]) consists of four components:

- *The ONION data layer.* This layer contains the wrappers for the external sources and the articulation ontologies that form the Semantic Bridges between the sources.
- *The ONION viewer.* This is the user interface component of the system. The viewer visualizes both the source and the articulation ontologies.
- *The ONION query system.* The query system translates queries formulated in term of an articulation ontology into a query execution plan and executes the query.
- *The Articulation Engine.* The articulation generator takes articulation rules proposed by SKAT [MWJ99], the Semantic Knowledge Articulation Tool, and generates sets of articulation rules, which are forwarded to the expert for confirmation.

The different components in the architecture have been implemented as a research prototype to support a PhD thesis.

Experiences [MWK00, MW01] do not show any real experiences with the application of ONION besides toy examples described in the papers.

4.2.3 OBSERVER

Summary OBSERVER (Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity Resolution) [MIKS00] is a system which is intended to overcome problems with heterogeneity between distributed data repositories by using component ontologies and the explicit relationships between these components. OBSERVER presents an architecture consisting of component nodes, each of which has its own ontology, and the Inter-ontology Relationship Manager (IRM), which maintains mappings between the ontologies at the different component nodes. Besides the ontology, each component node contains a number of data repositories along with mappings to the ontology,

to enable semantic querying of data residing in these repositories. When other components need to be queried, the IRM provides mappings to ontologies of other component nodes in order to enable querying. The user views the data in the system through its own local ontology, located at the user's component node.

OBSERVER uses a component-based approach to ontology mapping. It provides brokering capabilities across domain ontologies to enhance distributed ontology querying, thus avoiding the need to have a global schema or collection of concepts.

OBSERVER uses multiple pre-existing ontologies to access heterogeneous, distributed and independently developed data repositories. Each repository is described by means of one or more ontologies expressed using the Description Logic (DL) system CLASSIC. The information requested from OBSERVER is expressed according to the user's domain ontology, also expressed using DL. DL allows matching the query with the available relevant data repositories, as well as translating it to the ontologies used in the local repositories.

The system contains a number of component nodes, one of which is the user node. Each node has an ontology server that provides definitions for the terms in the ontology and retrieves data underlying the ontology in the component node. If the user wants to expand his query over different ontology servers, the original query needs to be translated from the vocabulary of the user's ontology into the vocabulary of another's component ontology. This is done in the following way: the source and target ontologies are integrated, after which all terms in the query for which a synonym exists in the target ontology, are replaced by this synonym and all other terms are replaced by *semantically similar* expression that could lead to a full translation of the user query. Each such term for which there is no synonym in the component ontology is replaced by the intersection of its immediate parents or the union of its immediate children, recursively, until the rewritten query contains only terms from the component ontology. Several alternative translations might be generated in this process, which are not exact in the case where terms were replaced with semantically similar expression and not synonyms.

A method for evaluating the *information loss* for the case of inexact translations was developed, with the purpose of enabling the system to choose "the best" among alternative translations. The user can opt for translations for which the information loss is smaller than a certain value by setting a certain threshold. By setting this parameter to 0, the user specifies no loss at all. The loss of information threshold is used by the query processor, which discards queries exceeding the threshold.

The information loss is computed based on the commonly encountered metrics in Information Retrieval, *precision* and *recall*. The idea is to take into account both these metrics, that is the loss in precision (percentage of retrieved objects which are not relevant) and the loss in recall (percentage of relevant objects which are not retrieved), because it is not known a priori the preference of the user towards one of these measures. The formula for computing information loss based on precision loss and recall loss contains a weight which expresses the bias towards one of the two metrics, which can be set by the user.

However these metrics, i.e. precision and recall, had to be adapted in order to give higher priority to semantic relationships than to those suggested by the underlying extensions. This problem arises due to the fact that extensions are coming from different ontologies and the extensional relationships may not reflect the semantic relationships. For example, a term in a user ontology which semantically subsumes a term in the target ontology may have a smaller extension than the child term.

An Inter-ontology Relationship Manager (IRM) provides the translations between the terms among the different component ontologies. The IRM effectively contains a one-to-one mapping between any two component ontologies. This module is able to deal with (intentional) *Synonym*, *Hyponym*, *Hypernym*, *Overlap*, *Disjoint* and *Covering* inter-ontology relationships. Furthermore, the IRM is also able to deal with extensional relationships between ontologies through the use of so-called *transformer functions*.

The user submits a query to the query processor in its own component node (in fact, each component node has a query processor). The query processor first uses the local ontology server to translate the query into queries on the local data repositories and then execute them, after which the user can choose to incrementally increase the query to multiple ontologies. The query processor then uses the IRM to translate the query into terms used by the other component ontologies and retrieve the results from the ontology servers at the other component nodes.

Querying in OBSERVER consists of the following three steps:

1. The user *constructs the query* using terms from the user's ontology.
2. The query processor uses the ontology server to *access the underlying data* at the user's node. The query is executed against the local data repositories.
3. In a process of *controlled query expansion to new ontologies* the user can specify whether he/she is satisfied with the query results or whether the query should be expanded to other component ontologies. In this case, the inter-ontology relationships retrieved from the IRM are used to rewrite queries and to transform query results.

The ontology server can be queried in two ways. Information about the ontology itself can be retrieved and the ontology server can answer queries formulated over an ontology using the mappings to the different data sources and the wrappers available for each data source. The query capabilities of each data source are consulted by the ontology server, which creates a query plan and invokes the wrappers to retrieve the data from the sources.

In principle, only the local ontology server is queried initially. The user can then choose to incrementally expand the query over multiple ontologies in order to retrieve more results for the query.

Ontology Languages Ontologies, as well as mappings between ontologies are specified using the Description Logic system CLASSIC.

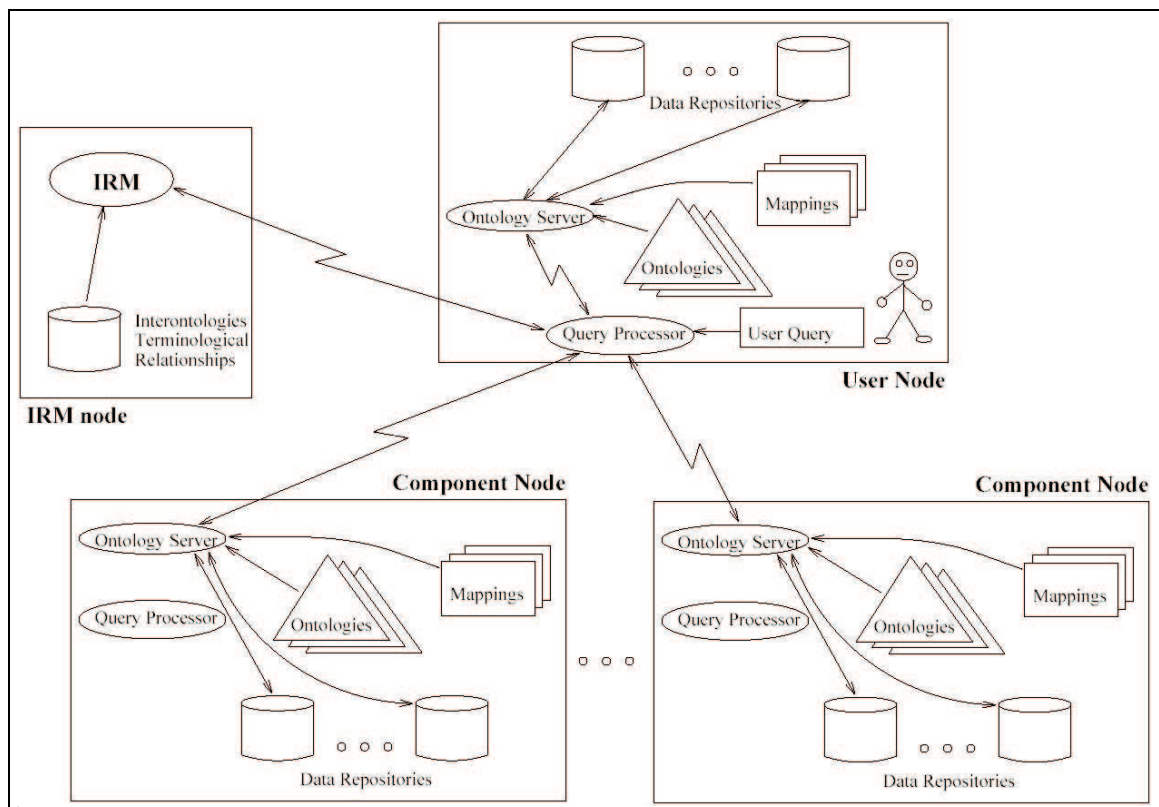


Figure 4.19: The general OBSERVER architecture [MIKS00]

Mapping Language In OBSERVER, there exist two types of mappings, namely the mappings between data repositories and ontologies and the mappings between the ontologies. We will first describe the mappings between data repositories and ontologies, after which we describe the specification of the inter-ontology relationships.

A data source is seen as consisting of entities and attributes (in the Entity-Relationship [Che79] sense of the terms). Mapping between data sources and the ontology is represented by associating each term in the ontology with a number of Extended Relation Algebra (ERA) expressions. ERA is used as an intermediate language between the Description Logic expressions of the ontology and the underlying data repositories. The wrapper is responsible for the translation between ERA and the data repository itself, which is straightforward if the data source is a relational database. It is interesting to note that in the mapping to roles in the ontology, ERA allows functions, which typically represent value transformations.

OBSERVER deals with a number of types of inter-ontology relationships in order to enable inter-operability:

- **Synonym.** Two synonymous terms have the same semantics, i.e. the same *intended* meaning. This does not guarantee that they have the same extension.
- **Hyponym.** A term is a hyponym of another term if it is less general, i.e. a term in one ontology subsumes a term in another ontology.
- **Hypernym.** A term is a hypernym of another term if it is more general, i.e. a term in one ontology is subsumed by a term in another ontology.
- **Overlap.** This means the two terms have an overlap, i.e. a non-empty intersection. In OBSERVER, the overlap between terms is usually indicated with a percentage, which can be used to estimate the loss of information in a query translation.
- **Disjoint.** This means there is no intersection between the two terms.
- **Covering.** When a term in one ontology corresponds to a union of terms in the other ontology, i.e. the meaning of the term in one ontology is *covered* by the union of terms from the other ontology. There does not exist an object represented by the term in the one ontology, which is not represented by the union of the given children terms.

The above mentioned inter-ontology relationships explicate the intentional relationship between terms in two ontologies. However, when an intentional relationship between terms is true, it does not mean that this relationship holds also for the extensions (i.e. sets of instances) of the ontologies. For the extensional level, a set of *transformer functions* between roles in different ontologies is used. These functions are used for both instance transformation and instance unification.

The Inter-ontology Relationship Manager can be used to discover sets of related component ontologies, to retrieve related terms between ontologies and to perform value transformations from one ontology representation to the other.

Automation Support There seems to be no automation in creating the mappings between ontologies and/or data sources. However, known matching algorithms could be easily used to identify similarities between ontologies.

The query processing, on the other hand, is completely automated with the exception that the user is still required in the incremental querying process to specify whether other component nodes need to be queried and he/she might also intervene to set some parameters like the bias towards precision or recall in the computation of information loss and the threshold for information loss in a query translation. As already mentioned, if the user desires the query to be answered by accessing other components as well, the query is translated into the ontology language(s) of the other components. If there is no exact translation, there might be alternative translations which are generated and evaluated by the system and the optimal one, which does not surpass the threshold set by the user for information loss, is chosen.

Applicability to Use Cases Instance transformation and instance unification are both performed in the querying process. The query processor is responsible for transforming and correlating query results from the target ontology into the user ontology.

The emphasis in OBSERVER is really on query rewriting. The relationships between the ontologies, expressed using Description Logics, are used to rewrite the queries from the user's ontology to the component ontology.

OBSERVER is a data integration system and as such provides no explicit support for ontology merging, although in the query processing ontologies are automatically integrated based on the inter-ontology relationships retrieved from the IRM.

Implementation The OBSERVER architecture, depicted in Figure 4.19, consists of a number of component nodes and the IRM node. A component node contains an *Ontology Server* which provides for the interaction with the ontologies and the data sources. It uses a repository of mappings to relate the ontologies and the data sources and to be able to translate queries on the ontology to queries on the underlying data sources. The architecture contains one Inter-Ontology Relationship Manager (IRM), which enables semantic inter-operation between component nodes by maintaining the relationships between the ontologies.

OBSERVER has been implemented as a prototype for the access of distributed heterogeneous data sources in the area of bibliographic data.

Experiences The authors have reused different existing ontologies in the area of bibliographic data and represented them in CLASSIC for integration in the OBSERVER architecture. Real-life ontologies and data repositories were used in the prototype. It turned out that the time required to access the distributed data repositories was the bottleneck for the prototype.

4.2.4 MOMIS

Summary MOMIS (Mediator envirOnment for Multiple Information Sources) approach [BCV99, BCVB01] is an approach to the integration of heterogeneous data sources using a global ontology, which is the result of a merge of the local data schemas.

The goal of MOMIS is to give the user a global virtual view (cf. [Hul97]) of the information coming from heterogeneous information sources. MOMIS creates a global mediation schema (ontology) for the structured and semi-structured heterogeneous data sources, in order to provide to the user a uniform query interface to these sources.

The first step in the creation of the global mediation schema is the creation of the Common Thesaurus from the disparate data sources. To do this, first a wrapper is created for each data source in the ODL_{I^3} language. ODL_{I^3} is an object-oriented language with an underlying Description Logic language OLC_D, which enables making inferences (e.g. subsumption) about the classes expressed in that language.

Using the disparate schemas, a Common Thesaurus is created, which describes intra- and inter-schema knowledge about ODL_{I^3} classes and attributes of source schemas. The Common Thesaurus is built in an incremental process in which relationships (between classes) are added based on the structure of the source schemas, lexical properties of the source classes and attributes (e.g. WordNet [Fel99] can be used to identify possible synonyms), relationships supplied by the designer, and relationships inferred by the inference engine.

Once the Common Thesaurus has been created, a tree of affinity clusters is created, in which concepts are clustered based on their (name and structural) affinity. The name affinity coefficient is calculated based on the terminological relationships between two classes. The structural affinity coefficient between two classes is calculated based on the level of matching of attribute relationships in the Common Thesaurus. The sum of these two coefficients is the global affinity coefficient, which is used to construct the affinity tree, in which concepts with a high affinity are clustered together.

For each cluster in the affinity tree, a global class is (interactively) created. For each global class a mapping (expressed in ODL_{I^3}) is maintained to all the source classes.

If we compare MOMIS with OBSERVER we can say that OBSERVER takes a minimalist approach to the specification of inter-ontology relationships, specifying only the bare minimum required for query processing, whereas MOMIS tries to identify all possible relationships between a set of ontologies, integrating them in one global ontology.

Ontology Languages A *wrapper* translates each data schema to the ODL_{I^3} representation. MOMIS also deals with semi-structured data by extracting *object patterns*, which are used as schema information for the source to generate the corresponding ODL_{I^3} description.

ODL_{I^3} is an object-oriented language with a translation to the OLCD Description Logic languages in order to allow inferencing. OLCD is a KL-ONE [BS85] like ontology language, which allows classes (types), binary roles (attributes), disjunction, negation and also has a number of base data types (integer, string, Boolean, real).

Mapping Language Source schemas and object patterns are first translated into ODL_{I^3} descriptions. This translation is performed by a wrapper. Then, a *Common Thesaurus* is created based on the ODL_{I^3} class descriptions and attributes. The Common Thesaurus consists of four kinds of relationships, which are added to the thesaurus in the following phases:

1. *Schema-derived relationships* In this phase, only intra-schema relationships are considered. Relationships within one particular schema are extracted, e.g. by exploiting foreign and primary key relationships in order to infer related and broader/narrower term relationships.
2. *Lexical-derived relationships* Lexical relationships between names in different schemas are exploited to extract inter-schema relationships. WordNet [Fel99] is used to extract synonyms and hypo/hypernyms. Furthermore, synonymous terms are also extracted from attributes with similar names.
3. *Designer-supplied relationships* In this phase, the designer can supply relationships between schemas. A Description Logic reasoner, such as ODB, can be used to check the relationships for inconsistency.
4. *Inferred relationships* Description Logic reasoning is used to infer new relationships between ODL_{I^3} classes, based on relationships specified in the previous phases.

In each of the phases, intentional relationships are added to the Common Thesaurus. The designer can strengthen these intentional relationships by creating extensional relationships, thereby enabling subsumption reasoning and consistency checking. An intentional relationship can be seen as saying “there exists in general a relationship between these terms”, whereas an extensional relationship can be seen as saying “this relationship holds for these particular data sources”.

MOMIS employs hierarchical clustering based on an affinity measure, which indicates the similarity between classes in a cluster. The affinity measure is based on both the name and structural similarity.

The clusters in the hierarchy are used to interactively create new classes for the merged ontology. Generally, a union is taken of all classes in a particular cluster in order to come up with the new global class. During the process of creating the global class, mapping rules between the attributes in the local classes and the global class are established and stored in the global ontology for later use. Because the global ontology is created on the basis of the local ontologies, MOMIS takes the global-as-view approach [Lev00], which means that the global schema is created as a *view* over the local schemas and all queries to the global schema can be easily translated to the local representation because of the presence of mapping rules.

It is not clear how new sources can really be integrated once the system is in place. It seems that the global schema has to be re-created from the local schemas, although the computed affinity clusters can of course be reused and if the classes in the new schema to be integrated fit inside the existing affinity clusters, only the mapping rules between the new schema and the global schema need to be created.

Automation Support Automation support in the ontology merging task is provided by the ARTEMIS tool [CdA99]. The ARTEMIS tool provides support in the matching task by determining the (name and structural) affinity between terms in the ontologies.

Applicability to Use Cases In MOMIS, the Query Manager does query rewriting based on the mapping rules in the global ontology. The wrapper of each data source rewrites the query from its ODL_{I^3} representation to the representation of the data source and also transforms the query results back to the ODL_{I^3} representation. The query manager then uses the mapping rules to translate the query results back to the global representation in order to present the results to the user. It is not exactly clear if and how the Query Manager fuses the query results from the different sources in case of overlap in the result sets.

Implementation A number of components are used to enable the MOMIS architecture. These components are (see Figure 4.20, taken from [BCVB01]):

- A *wrapper* performs the translation of the individual data source into the ODL_{I^3} language (and translates the queries back).
- The *mediator* consists of the query manager (QM) and the global schema builder (GSB). The QM component breaks up global ODL_{I^3} queries into sub-queries for the different data sources. Therefore, the GSB is an offline component, which aids in ontology merging and the QM is a run-time component, which performs the queries.
- The ARTEMIS tool environment performs the classification (affinity and synthesis) of classes for the synthesis of the global classes.

- The *ODB-tools engine* performs schema validation and inferences for the generation of the Common Thesaurus, as well as query optimization for the Query Manager.

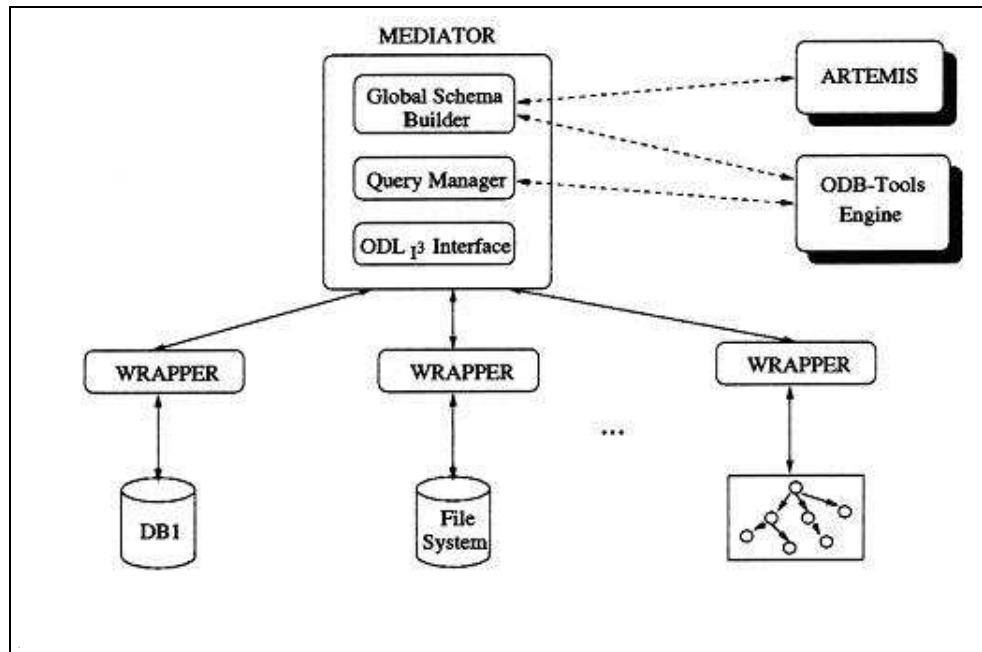


Figure 4.20: Architecture of the MOMIS system [BCVB01]

The architecture in Figure 4.20 shows the main tools used to support the overall architecture. A disadvantage is that there is no integrated tool environment. Any data source can be connected to the architecture, as long as an ODL_{I^3} wrapper is created.

Experience As far as we are aware, MOMIS has not been used in any major industrial project and is mainly an academic research activity, with toy examples. However, [CAV01] reports the application of ARTEMIS in a research project in the Italian Public Administration domain.

4.2.5 INFOMIX

INFOMIX [LGR⁺05b, LGR⁺05a, LLR02] is a system that supports information integration by utilizing advanced reasoning capabilities. The INFOMIX system is built in co-operation with RODAN systems¹⁸ which is a commercial database management system developer. It uses the DLV reasoning system [LPF⁺05] for the reasoning tasks. The DLV system is a disjunctive datalog reasoning system and has been developed independently

¹⁸The web site of RODAN systems can be found at <http://www.rodan.pl/en/>.

from INFOMIX. However, INFOMIX uses the DLV system for solving its reasoning tasks.

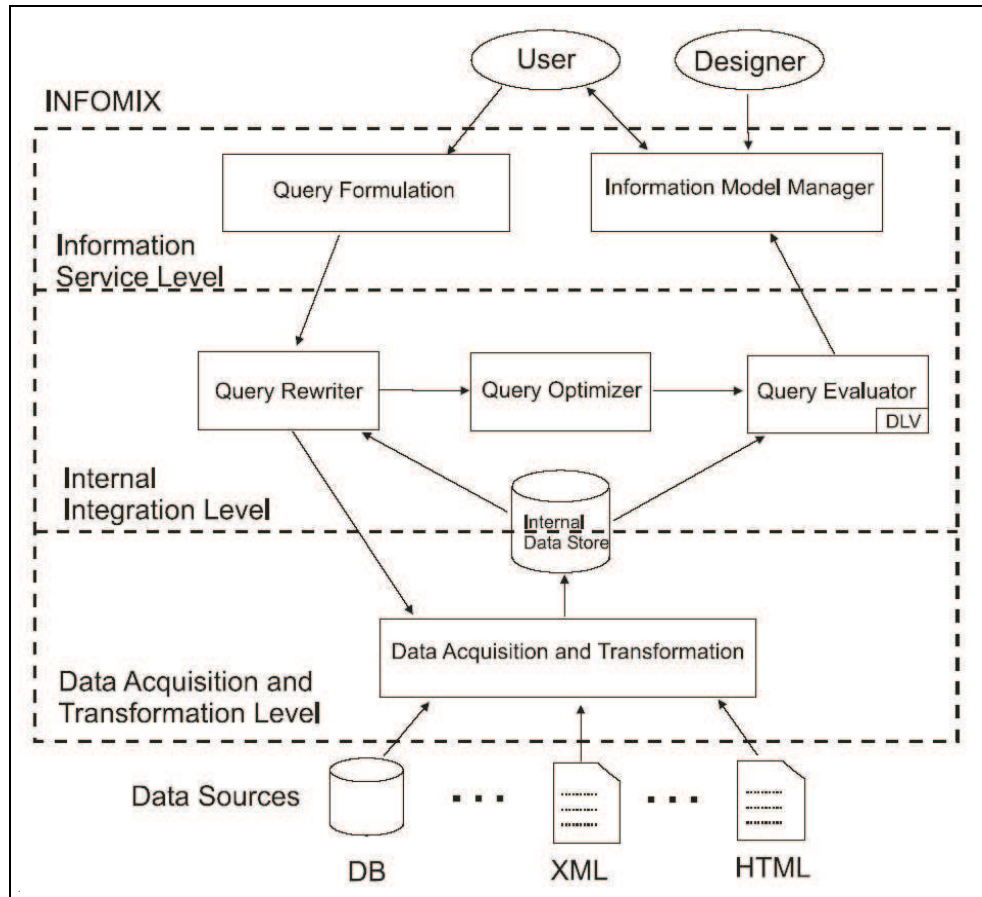


Figure 4.21: Architecture of the INFOMIX system [LGR⁺05b]

The INFOMIX architecture is depicted in Figure 4.21. It is divided into three levels and supports two modes:

- *a design mode*
Here, the global schema, the source schema and the mapping between them are specified. Also, the wrappers for the data sources are created or imported. The data sources consist of relational and XML data.
- *a query mode*
Here, query answering facilities are provided at run time, including data acquisition, integration, answer computation and presentation to the user.

In both modes, the INFOMIX system is divided into three levels:

- **Information Service Level**
Here, a direct interface to the user is provided at run time and to the designer at

design time. This level deals with global data and provides the interfaces that are necessary (e.g. for global schema definition and for query formulation).

This level comprises two modules: the Information Model Manager and the Query Formulation module:

- The Information Model Manager handles the definition of the global schema and the local schemas, as well as the mapping. User-friendly interfaces for these tasks, including schema browsers, are provided. Automatic support for the verification of coherency, redundancy and adequacy of the application specification is given. Finally, this module presents query results in a suitable form to the user.
- The Query Formulation module provides a graphical, user-friendly interface for query formulation over the global schema and query validation facilities. The query validation facilities check the interactions between the user query and global integrity constraints to guarantee that query answering is always decidable.

- **Internal Integration Level**

This level is based on computational logic and deductive database technology. It is composed by three modules:

- **The Query Rewriter** reformulates the user query according to global integrity constraints. It makes use of a submodule to verify data consistency. This submodule exploits the mapping and unfolds the user query over the source relations and activates the corresponding wrappers to retrieve relevant data. Afterwards, the submodule checks whether there are integrity constraint violations. If no violations occur, the reformulation produced by the rewriter is a simple (disjunction free) Datalog program. Otherwise, a suitable disjunctive Datalog program is generated that performs automatic repair of data, in a way such that cautious answers to this program evaluated over the data sources correspond to the certain answers to the query.
- **The Query Optimizer** provides several optimization strategies which enhance the efficiency of the system. In particular, the module exploits some focusing techniques which are able to isolate the portion of the source database that is relevant to answer a user query. This is done by pushing constants in the query towards the sources, i.e. the part of the query that contains constants is processed first. To this aim, an optimized (possibly disjunctive) Datalog program is generated by applying advanced binding propagation techniques similar to the magic sets algorithms that are well-known in the database area.
- **The Query Evaluator** takes the optimized program and first loads data from the Internal Data Store and then invokes the DLV system in order to com-

pute consistent answers. The results are then sent to the Information Model Manager for suitable presentation to the user.

- **Data Acquisition and Transformation Level**

This level provides access to external data sources. INFOMIX has an architecture which allows for the integration of heterogeneous types of data sources. The primary types of data sources are relational, XML, HTML, and object-oriented data sources. However, it is claimed that arbitrary other types of data sources can be incorporated easily. All data sources are conceptually transformed into a uniform source data format, which is a fragment of XML Schema. Data encoded in this uniform source data format can be browsed. The acquisition and transformation of data is done by wrappers. A query plan for executing suitable wrappers is generated, which load data into the Internal Data Store. Constants are pushed to the query wrappers whenever it is possible in order to reduce the amount of data retrieved.

Currently, INFOMIX offers three classes of wrappers, which provide different levels of support for query formulation and wrapper code generation:

- **Code wrappers** are basically a definition of an API and the code implementing it. The internals and characteristics of code wrappers are therefore inaccessible to INFOMIX.
- **Query wrappers** propagate queries to external data sources and treat the result as a logical data source.
- **Visual wrappers** support interactive development of wrappers at design time. Currently, there is support for developing LiXto wrappers and pipes as well as for Rodans Data Extractor.

Ontology Languages The INFOMIX system supports the following kinds of external data sources:

- Relational data in the ODBC standard, i.e. especially SQL, is supported.
- Any object-oriented database that conforms to the ODMG standard (cf <http://www.odmg.org/>) and which is accessible by the ODMG 2.0 Java binding will be supported. In particular, the object-oriented database must be accessible by OQL queries (more specifically, by the subset of OQL supported by the ODMG Java binding). However, no multimedia data types will be supported although usually supported by object-oriented database management systems. Currently the support of object-oriented databases is not available.
- Pure (semi-structured) XML data (according to the XML 1.0 standard) as well as XML data structured according to a corresponding XML Schema specification is supported. However, the INFOMIX system does not check the validity of structured XML data. It also does not take DTD specifications into account.

- Valid web pages conforming to the HTML 2.0, 3.2 and 4.1 standards are supported. However, the INFOMIX system does not check the validity of HTML data. Note that HTML rendered versions of embedded data formats like comma separated values (CVS), Microsoft Word (planned), Excel and Powerpoint (planned) documents as well PDF documents are supported. INFOMIX does not support multi-media data such as image files, sound or movie data.

Data extraction and preprocessing of the data sources is done by LiXto [BFG01] wrappers and tools.

The data is internally stored in a language called ISDF (INFOMIX source data format) and has an XML-like syntax. It is specified by a subset of XML Schema. Essentially, ISDF amounts to a small extension of the relational format, by allowing nesting of data and multiple attribute values. This format is translated by the query reformulator into disjunctive Datalog programs extended by aggregate functions.

Mapping Language The mapping is a Global-As-View (GAV) mapping. I.e. there is a global schema G and a source schema S which comprises the schemas of all the sources to be integrated. It consists of a set of logical implications in Datalog. Such an implication has a conjunction of atoms in S as antecedent and as consequent a relation of G . All free variables of the body of the implication are quantified. This means that each global relation is associated with a union of conjunctive queries. Clearly, this mapping language is not as general as our mapping language which can be grounded to different kinds of logics and is much more expressive.

Mapping Patterns There are a user-friendly interface and schema browsers for the global schema, the local schemas and the mappings. However, there does not exist a concept of mapping patterns. We are not aware of planned extensions to use mapping patterns.

Automation Support There is no automation support in the creation of mappings between the local schemas and the global schema. However, the query rewriting and the data fusion is completely automated based on the mappings between the local schemas and the global schemas as well as the wrapper specifications between the original data source formats and the internal storage format.

Applicability to Use Cases Instance transformation and unification can be performed by the INFOMIX system. Query rewriting is done by the INFOMIX system as well naturally.

The INFOMIX system does not perform ontology merging. However, it translates the relevant data into a uniform data format. This can be viewed in some respect as ontology

merging.

Implementation The kernel of the system is be developed in C++ to achieve better performance. The user interfaces and data-extraction modules are based on Java components.

Experience There is no experience report available yet.

4.2.6 AutoMed

AutoMed [BKL⁺04] is an integrated system that incorporates the BAV (both as view) approach. This means that it does not adhere to either the LAV (local as view) approach or the GAV (global as view) or GLAV (global local as view) approach. BAV subsumes the expressive power of all these mentioned approaches.

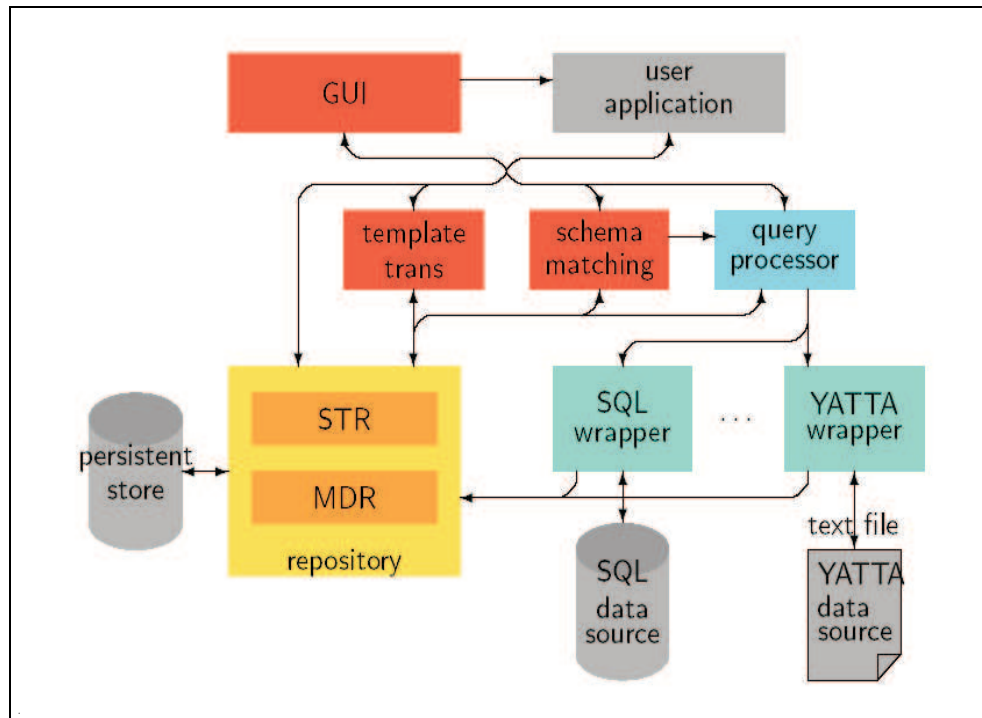
Figure 4.22 shows the architecture of the AutoMed system. The meta data repository can be viewed as the core and other components of the software architecture are implemented upon it. When a data source is wrapped, a definition of the schema for that data source is added to the repository. The schema matching tool can then identify related objects in various data sources by means of the query processor that retrieves data from schema objects. The template transformation tool is used to generate transformations between the data sources. A GUI is available for these components.

The repository has two logical components:

- **The model definitions repository (MDR)** defines how a data modelling language is represented (in the hypergraph data model (HDM)). It can be used to configure AutoMed to handle a particular data modelling language.
- **The schema transformation repository (STR)** defines schemas in terms of the data modelling concepts in the MDR and transformations that will be specified between those schemas. Most tools and users will be concerned with editing this repository when new databases are added to the repository.

The MDR and STR may be held in the same or in separate persistent storages.

Figure 4.23 gives an overview of the key objects in the repository. The STR contains a set of Schema descriptions. Each such Schema description contains a set of SchemaObject instances. Each such SchemaObject instance must be based on a Construct instance that exists in the MDR. This Construct describes how the SchemaObject can be constructed in terms of strings and references to other schema objects, and the relationship of the construct to the HDM. Schemas are thus easily translatable into HDM which is the common underlying representation of all the data modelling languages handled in AutoMed. Note that each Schema may contain SchemaObjects from more that one data

Figure 4.22: Architecture of the AutoMed system [BKL⁺04]

modelling language. In this way, mappings between different data modelling languages can be described. Schemas may be related to each other using transformation instances.

For this deliverable the schema matching architecture is most relevant and most important. Figure 4.24 shows the architecture of the schema mapping module.

Within AutoMed, five types of semantic relationships between schema objects are distinguished based on the comparison of their intentional domains, i.e. the sets of real-world entities represented by the schema objects. These are: *equivalence*, *subsumption*, *intersection*, *disjointness* and *incompatibility*. Transformation rules that should be performed on the schemas based on the discovered semantic relationships are defined in [Riz03]. In AutoMed, the automatic discovery of the semantic relationships are performed by means of a bidirectional comparison of schema objects.

The relationship discovery process is implemented by means of the architecture in Figure 4.24. The architecture consists of several modules that exploit different types of information in order to compute bidirectional similarity degrees of schema objects. The currently implemented modules compare schema object names, instances, statistical data over the instances, data types, value ranges and lengths. There are two types of modules: **relationship identification modules** attempt to discover compatible pairs of schema objects, and **relationship clarification modules** attempt to specify the type of the semantic relationship in each compatible pair. Initially in the schema matching process, the bidirectional similarity degrees produced by the modules are combined by a Filter that uses the

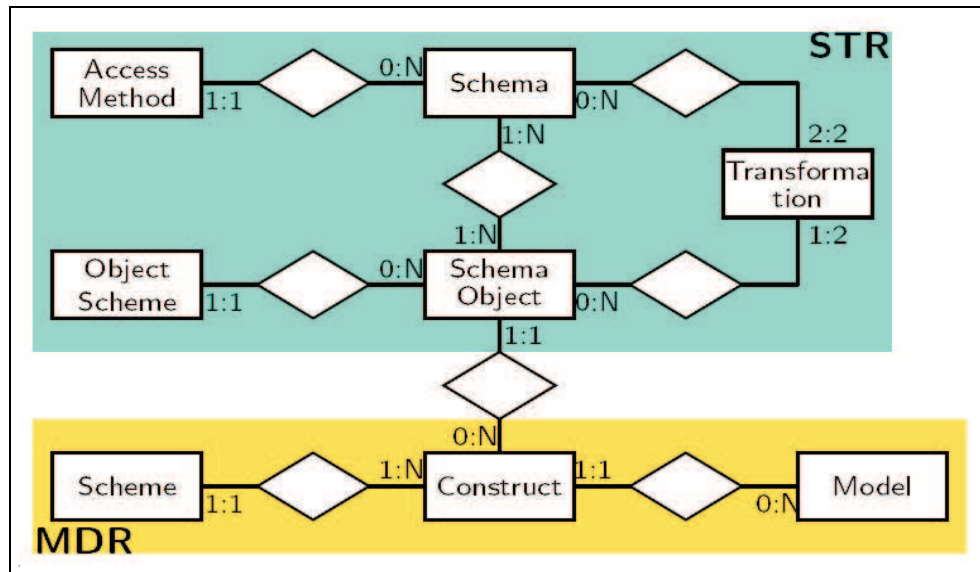


Figure 4.23: Architecture of the AutoMed repository schema [BKL⁺04]

average aggregation strategy in order to separate the compatible from the incompatible pairs of schema objects. Afterwards, the Aggregator component combines the similarity degrees of the compatible schema objects with the product aggregation strategy and indicates their semantic relationships. The output of the Aggregator becomes the input of the Degree Combinator, which is based on the relationship clarification modules and on the values of the similarity degrees. The Degree Combinator has as output the semantic relationships it has discovered. The user can then validate or reject the discovered relationships and step ahead to the data integration process.

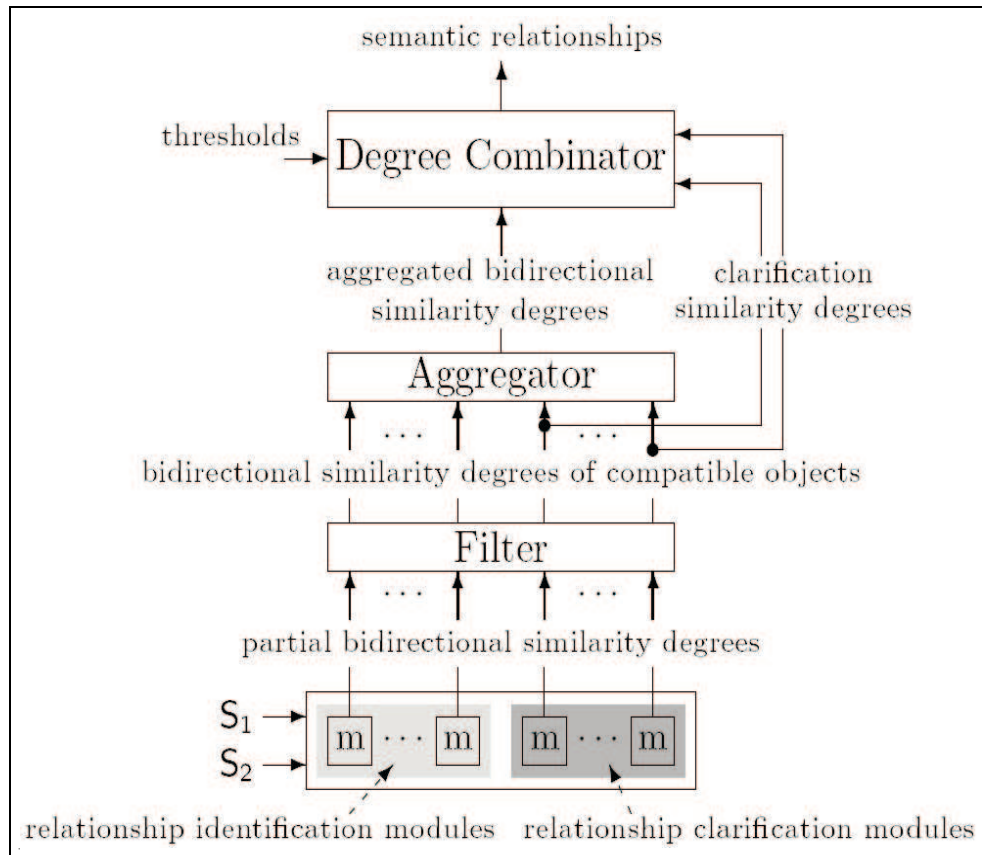
Ontology Languages The AutoMed system can use different Ontology languages. It supports RDF and RDFS as well as XML and common relational Databases formats.

Mapping Language The mapping language is the hypergraph data model (HDM). This is the internal language that is used.

Mapping Patterns Mapping Patterns are not mentioned.

Automation Support There is automatic support for the creation of mappings. The mappings creation architecture shown in Figure 4.24 displays the components that are used to create automatic mappings.

Applicability to Use Cases All use cases can be applied.

Figure 4.24: Architecture of the AutoMed mapping system [BKL⁺04]

Implementation The AutoMed system has been implemented in Java. The different components of the architecture have been implemented in Java as well.

Experience The authors are not aware of experience reports of the AutoMed system.

4.3 Specific Techniques

In this section we briefly describe some specific techniques for ontology merging such as FCA-Merge, OntoMorph, and QOM.

4.3.1 FCA-Merge

FCA-Merge [SM01] is a method for merging ontologies based on *Formal Concept Analysis* [GW99]. The FCA-Merge approach is a bottom-up approach, which means that it is based on application-specific instances of the two ontologies that need to be merged.

A set of documents that are relevant to both ontologies are provided as input. Using linguistic analysis, instances are extracted from the documents for both ontologies. Now a pruned concept lattice is generated using the similarity in instances for both ontologies. These first two steps (lexical analysis and generating the concept lattice) are carried out fully automatically. In the third and last step of the method, the merged target ontology is created interactively (i.e. semi-automatically).

4.3.2 OntoMorph

The OntoMorph system aims to facilitate ontology merging and the rapid generation of knowledge base translators [Cha00]. It combines two powerful mechanisms to describe KB transformations. The first of these mechanisms is syntactic rewriting via pattern-directed rewrite rules that allow the concise specification of sentence-level transformations based on pattern matching, and the second mechanism involves semantic rewriting which modulates syntactic rewriting via semantic models and logical inference. The integration of ontologies can be based on any mixture of syntactic and semantic criteria.

In the syntactic rewriting process, input expressions are first tokenized into lexemes and then represented as syntax trees, which are represented internally as flat sequences of tokens and their structure only exists logically. OntoMorph's pattern language and execution model is strongly influenced by Plisp [Smi90]. The pattern language can match and de-structure arbitrarily nested syntax trees in a direct and concise fashion. Rewrite rules are applied to the execution model.

For the semantic rewriting process, OntoMorph is built on top of the PowerLoom¹⁹ knowledge representation system, which is a successor to the Loom system. Using semantic import rules, the precise image of the source KB semantics can be established within PowerLoom (limited only by the expressiveness of first-order logic).

4.3.3 QOM Quick Ontology Mapping

The QOM tool [ES04a] represents an approach that considers both the quality of mapping results as well as the run-time complexity. The hypothesis is that mapping algorithms may be streamlined such that the loss of quality (compared to a standard baseline) is marginal, but the improvement of efficiency is so tremendous that it allows for the ad-hoc mapping of large-size, light-weight ontologies. To substantiate the hypothesis, a number of practical experiments were performed.

The goal is to present an efficient mapping algorithm. The outcome is QOM — Quick Ontology Mapping. It is defined by the steps of a process model as shown in Figure 4.25. Mapping one ontology onto another means that for each entity (concept C , relation R , or

¹⁹<http://www.isi.edu/isd/LOOM/PowerLoom/>

instance I) in ontology O_1 , one tries to find a corresponding entity, which has the same intended meaning, in ontology O_2 .

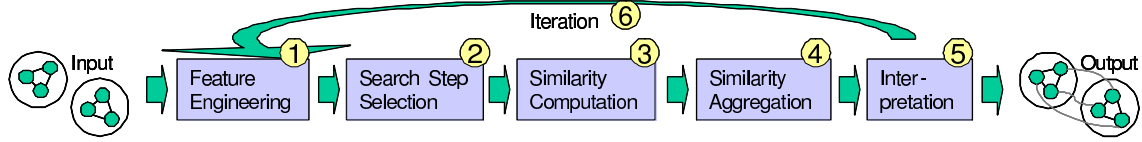


Figure 4.25: QOM Mapping Process

1. Firstly, QOM uses RDF triples as features.
2. Second, instead of comparing all entities of the first ontology with all entities of the second ontology, QOM uses heuristics to lower the number of candidate mappings, which is a major problem for run-time complexity. In this dynamic programming approach we only choose promising candidate mappings.
3. The actual similarity computation is done by using a wide range of similarity functions [ES04b]. An entity is described by the kind of appearance that is found to hold for this entity for characteristics like: identifiers such as URIs, RDF/S primitives such as subclass and instance relations, or domain specific features e.g. a *hashcode-of-file* in a file sharing domain. These features of ontological entities are compared using *String Similarity* and *SimSet* for set comparisons. For efficiency reasons the similarity computation was disburdened by removing extremely costly feature-measure combinations such as the comparison of all subclasses of two concepts.
4. These individual measures are all input to the similarity aggregation. Instead of applying linear aggregation functions, QOM applies a sigmoid function, which emphasizes high individual similarities and de-emphasizes low individual similarities.
5. From the similarity values we derive the actual mappings. A threshold to discard spurious evidence of similarity is applied. Further mappings are assigned based on a greedy strategy that starts with the largest similarity values first.
6. Through several iteration rounds the quality of the results rises considerably. Eventually, the output returned is a mapping table representing the relation map_{O_1, O_2} . The table is represented in a proprietary format, but can easily be transformed into a standardized format.

The evaluation was very promising. Depending on the scenario QOM reaches high quality mapping levels very quickly. QOM is on a par with other good state-of-the-art algorithms concerning the quality of proposed mappings, while outperforming them with respect to efficiency — in terms of run-time complexity ($O(n \cdot \log(n))$ instead of $O(n^2)$) and in terms of the experiments that have been performed on the QOM tool (by a factor of 10 to 100).

Chapter 5

Comparison of the Methods

5.1 Ontology Languages

We compare the ontology languages supported by the approaches included in this survey. Because many of the systems included in the survey are database integration systems, the ontology language is not the only language that counts. For these systems, it also matters which database schema languages are supported.

In these integration systems, the data schema is often lifted to the ontology level before the actual integration takes place. Therefore, for the core mapping task we are usually only concerned with the ontology languages. This lifting process often employed in database integration systems indicates the need for a comparison of lifting methods employed by the different systems. This small sub-comparison evaluates to what extent the schema is actually translated to the ontology. In other words, what we need to know is: (1) is the translation sound?, i.e. is the translation from the schema to the ontology correct and semantics preserving? and (2) is the translation complete?, i.e. is the schema translated completely?

Table 5.1 enumerates the ontology languages supported by the various approaches. It turns out that integration systems typically do not focus on inter-operability with other ontology tools. This makes sense, because all tasks (mainly querying) are performed in a closed environment. In a Semantic Web setting, use of standards is very important to enable inter-operability. We can see that tools created especially for the Semantic Web (e.g. MAFRA, RDFT, PROMPT, OntoMap, OMEN) support RDFS and PROMPT also supports OWL. On the other hand, matchers such as GLUE and S-Match use their own internal representation. Usually it is not a problem to convert an ontology in any language to such a representation, which is typically done in more comprehensive tool environments such as PROMPT.

Approach	Ontology Language	Remarks
<i>Methods and Tools</i>		
MAFRA	RDFS	
RDFT	RDFS	
PROMPT	Protégé-2000 supported	Includes support for RDFS, OWL, etc.
GLUE	taxonomies	
S-Match	DAGs	
OntoMap	proprietary language similar to OWL Lite ⁻	supports export to RDFS
RDFDiff	RDF	
OMEN	proprietary language similar to RDF Schema	
WSMX Mediation	Flora (soon WSMML based)	Integrated in the Web Services Modeling Toolkit.
DOMÉ Mediation	WSML	Based on an abstract mapping language
<i>Integration Systems</i>		
InfoSleuth	OKBC	wrappers are used to integrate data sources
ONION	Directed labeled graphs and Horn rules	source schemas are translated using wrappers
OBSERVER	Description Logics (CLASSIC)	The ontology server maintains mappings between data schemas and ontologies
MOMIS	ODL _I ³	relational and semi-structured sources are translated to ODL _I ³ using custom wrappers
INFOMIX	a set of logical implications in Disjunctive Datalog	INFOMIX has a very rich support for different interfaces and languages
AutoMed	HDM	XML, RDF(S) and relational databases are supported

Table 5.1: Ontology Languages

5.2 Mapping Language

Table 5.2 enumerates the mapping languages used by the different approaches. In mapping languages we can see three general approaches:

- *The ontology language and the mapping language are the same.* This is the case in MOMIS, to some extent in OntoMap and OMEN, to some extent in OBSERVER and to some extent in ONION.
- *The mapping language is different from the ontology language.* This is the case in MAFRA, RDFT and AutoMed and to some extent in OBSERVER and INFOMIX. MAFRA and RDFT both use a meta-ontology to describe types of bridges, which explicate the relationship between the ontologies. These types of bridges can be seen as the vocabulary for the mapping languages.

OBSERVER uses to some extent the same languages for the specification of both the ontologies and the mappings. However, the mapping also allows transformer functions, which are beyond the ontology language and the mappings between the ontologies and the data schemas are specified using ERA (Extended Relational Algebra).

- *There is no real mapping language.* The output of the tool is a similarity measure between concepts in the ontologies. This is the case for matchers, such as GLUE and S-Match. The purpose of these matchers is not to create an ontology mapping as such, but to discover similarities between the ontologies.

MAFRA, RDFT and OntoMap describe an ontology of bridges, called SBO (Semantic Bridge Ontology), RDFT (RDF-Transformations) and OntoMapO (OntoMap Ontology), respectively. These bridges are instantiated in the actual ontology mapping in order to realize the actual mapping specification. In this context, MAFRA has the most elaborate bridge ontology, i.e. MAFRA has the most expressive mapping language. However, MAFRA does not support mappings between classes and instances, which is supported by OntoMap. The specification of such a bridge ontology has many advantages; the main advantage is that it makes the type of mappings clearer and more understandable to the user and it allows the user to more easily find suitable mappings between ontologies.

We have not mentioned PROMPT and RDFDiff yet. PROMPT is not used to create a mapping between ontologies, but to merge ontologies. Therefore, the output of the tool is not a mapping specification, but a merged ontology, which, in this case, can be exported to any ontology language supported by Protégé, such as RDFS or OWL. RDFDiff returns as its output not the mapping between two different ontologies, but the structural difference between two versions of an ontology in the form of changed, added and deleted triples in the RDF document.

Approach	Mapping Language	Remarks
<i>Methods and Tools</i>		
MAFRA	Semantic Bridge Ontology (SBO)	SBO is a meta-ontology of semantic bridges. It allows arbitrary mappings between concepts, relations, and attributes, as well as conditional mappings and procedural transformations
RDFT	RDFT	RDFT is a meta-ontology, which describes types of mappings (bridges). Only allows class-to-class and property-to-property bridges
PROMPT	-	not applicable; PROMPT merges ontologies
GLUE	similarity measures	
S-Match	set-based (equal, disjoint, subset, superset)	
OntoMap	OntoMapO	OntoMapO allows specification of relationships between classes and also between classes and instances.
RDFDiff	changed, added, deleted	RDFDiff detects changed, added and deleted triples between versions of an RDF document
OMEN	Bayesian Network with individual pairs of matches as nodes	
WSMX Mediation	SEKT Abstract Mapping Language	
DOME	SEKT Abstract Map[ping Language	
<i>Integration Systems</i>		
InfoSleuth	wrappers	no ontology mapping; just mapping data schemas to ontologies
ONION	Articulation rules	
OBSERVER	Extended Relational Algebra for mapping ontology-DB and DL and <i>transformer functions</i> for mapping between ontologies	
MOMIS	ODL _I ³	wrappers are used to integrate data sources
INFOMIX	Disjunctive Datalog	wrappers are used to integrate data sources
AutoMed	HDM	wrappers are used to integrate data sources

Table 5.2: Mapping Language

5.3 Mapping Patterns

None of the approaches in this survey uses mapping patterns in the way proposed by [PGM98]. However, the types of mappings often present in specific mapping languages (e.g. the bridges in RDFT and MAFRA and articulation rules in ONION) can be seen as elementary mapping patterns. One could combine a number of these bridges to create more complex mapping patterns. However, there is no explicit support for such combinations in existing approaches.

5.4 Automation Support

Table 5.3 enumerates the automation support provided by the different approaches. Both ontology matchers (GLUE and S-Match) are completely automated, in the sense that after inputting two ontologies, the similarities between concepts in the ontology are returned without any user interactions. However, the matching of ontologies is just one step in the overall mapping process (see Section 1.2). Therefore, these approaches automate only part of the mapping process.

Mapping (or merging) ontologies is often an interactive process (e.g. in PROMPT), where the tool suggests mapping or merging actions to the user and the user can choose to either perform the suggested action, to discard it or to perform a different action. After the user interaction, the tool has more information to come up with more accurate suggestions. It is not clear if and how such one-shot matchers as GLUE and S-Match could fit in such an interactive process.

The integration systems ONION and MOMIS use specific tools (SKAT and ARTEMIS, respectively) for the discovery of similarity between ontologies. These tools are typically integrated in the system, which allows user interaction in the mapping process.

In the context of both MAFRA and RDFT, techniques were described to do ontology matching. In this context, MAFRA exploits the terms and the structure of the ontologies for the matching and RDFT exploits the instance descriptions associated with the ontologies to find similarities.

5.5 Applicability to Use Cases

The integration systems (InfoSleuth, MOMIS, OBSERVER and ONION) in this survey typically support all instance mediation use cases presented in Chapter 2. This is because the typical use case for data integration systems is the integrated querying of multiple data sources using a unified view (ontology). The querying of a unified view can be decomposed into query rewriting (the query in terms of the global ontology has to be

Approach	Automation Support	Remarks
<i>Methods and Tools</i>		
MAFRA	lexical and structural matching and semi-automatic creation of mappings	
RDFT	discovery of similarities based on instance data	
PROMPT	name & structural matching	semi-automatic ontology merging, where merge actions are suggested based on similarities
GLUE	multi-strategy machine learning approach	
S-Match	matching based on synsets from thesauri, using a SAT solver	
OntoMap	-	automation is supported when two ontologies are mapped to a common ontology
RDFDiff	changes between versions are detected automatically	
OMEN	mappings that depend on other mappings can be inferred automatically	
WSMX Mediation	String Matching, Wordnet similarity and real time user feedback	
DOME	No specific automation support. The use of the Mapping Language enable the possibility to use automation tools exporting in this language.	
<i>Integration Systems</i>		
InfoSleuth	-	
ONION	term and structural matching using SKAT	
OBSERVER	-	
MOMIS	name and structural matching using ARTEMIS	affinities computed by ARTEMIS are used to identify candidates for classes in the global ontology
INFOMIX	-	
AutoMed	bidirectional similarity degrees	

Table 5.3: Automation Support

rewritten in terms of the local data source), instance transformation (query results need to be translated from the local representation to the global representation) and instance unification (duplicates and redundancy have to be removed from the query results when the results from different data sources are combined).

Both MAFRA and RDFT have specific support for *instance transformations*. In MAFRA, it is possible to attach an executable piece of code to a Semantic Bridge. In RDFT, it is possible to associate an XPath expression with a bridge. Because RDFT is used for transforming XML documents and RDF documents in their XML representation, the XPath language can be used to express such transformations.

A form of *ontology merging* is performed in both the ONION and the MOMIS systems. In both systems, a global ontology is created, based on the local ontologies and database schemas. The global ontology is, in both cases, a virtual view over the underlying data sources; the local sources remain and mapping rules between the global ontology and the local ontologies are specified inside the global ontology.

PROMPT, on the other hand, is a pure ontology merging tool. The outcome of the PROMPT tool is a merged version of the source ontologies; no mappings between the sources and the merged ontology are created; the merged ontology is assumed to replace the original ontologies.

5.6 Implementation

Table 5.4 enumerates the type of implementations that have been made for the different approaches.

As we can see from the table, most approaches have only been implemented as academic prototypes. For most approaches we are not aware of any planned further development of the tools. Exceptions are PROMPT, which is currently under active development and has recently been adapted for the Protégé OWL plugin; S-Match, which is currently being extended to take the different semantics of different relations in the ontologies into account and to optimize the performance of the implementation; and RDFDiff, which will be further developed in the course of the SEKT project.

5.7 Experiences

Table 5.5 enumerates the experiences with the various approaches reported in the literature.

Most of the experiences reported in the literature are really toy problems; we feel that real experience with ontology mapping and ontology-based information integration is lacking. A cause of this problem is that the Semantic Web has not gained any real

Approach	Implementation	Remarks
<i>Methods and Tools</i>		
MAFRA	Two prototypes have been implemented	
RDFT	Research Prototype	
PROMPT	Version 2.1.1	PROMPT is still undergoing active development
GLUE	Research Prototype	
S-Match	First prototype	Work is still under way to improve the implementation
OntoMap	Prototype; under development since 2001	
RDFDiff	Research Prototype	The tool will be further developed in the course of the SEKT project
OMEN	first research prototype	
WSMX Mediation	Research Prototype	
DOMÉ	Research Prototype	
<i>Integration Systems</i>		
InfoSleuth	Project Prototype	
ONION	Research Prototype	
OBSERVER	Research Prototype	
MOMIS	Research Prototype	
INFOMIX	INFOMIX IST Project	
AutoMed	AutoMed project, LSDI project, ISPIDER project, RoDEX project	

Table 5.4: Implementation

Approach	Experience	Remarks
<i>Methods and Tools</i>		
MAFRA	Toy Problems	
RDFT	Mapping product classification schemes in GoldenBullet project	
PROMPT	HPKB project; evaluation using example ontologies	
GLUE	Toy problems	
Semantic Matching	Toy problems	Was evaluated against other matchers
OntoMap	Applied to most upper-level ontologies	
RDFDiff	Toy problems	
OMEN	Toy problems	
WSMX Mediation	DIP European Project	
DOME	DIP European project	
<i>Integration Systems</i>		
InfoSleuth	Two elaborate case studies	
ONION	Toy Examples	
OBSERVER	Prototype with real-life bibliographic data	
MOMIS	Toy Examples	ARTEMIS (part of MOMIS) has been applied in the domain of Italian Public Administration
INFOMIX	-	
AutoMed	some case studies	

Table 5.5: Experience

momentum as yet. Therefore, there are currently not so many ontologies on the Web, although there is some experience with real-life data sources, such as bibliographic sources.

Chapter 6

Conclusions

In this survey we have evaluated and compared several approaches to ontology mapping, ontology matching, ontology merging and data integration.

Comparing different types of approaches in this survey has made it clear that none of the approaches exactly fits all our criteria for ontology mediation on the Semantic Web. The purposes of the approaches in this survey tend to vary. The integration systems such as MOMIS [BCVB01] and ONION [MWK00, MW01] have the aim of providing query answering services over multiple data sources to the user. Matchers such as GLUE [DMDH04] and S-Match [GS04] have the more specific goal of finding similarities between schemas or ontologies. Integration systems often use matchers for the discovery of mappings (e.g. ONION uses SKAT; MOMIS uses ARTEMIS).

MAFRA [MMSV02] and RDFT [Ome02b] provide meta-ontologies for the specifications of mappings between ontologies. These mappings can be used for instance transformations. In fact, both approaches include specific means to enable transformation of instances between different representations. However, whereas InfoSleuth provides methods for the fusion of query results (instance unification), neither MAFRA nor RDFT handle this situation. Query rewriting is also not explicitly handled, but this should not be a problem because bridges between entities in the ontologies are explicitly present and can be readily used for query rewriting.

This state-of-the-art survey has made clear that work still needs to be done in the area of ontology mediation on the Semantic Web. We can learn from the data integration systems, which provide services for query answering over distributed heterogeneous data sources. However, the current setting of these integration systems is inside the enterprise, which is still a more-or-less controlled area. On the Web, not much control over the use of ontologies can be expected and the global integration scenario is not expected to scale, because eventually different organizations will use different ontologies and will not want to commit to a new ontology. However, the one-to-one integration approach is also not expected to scale, because it would require the maintenance of too many mappings

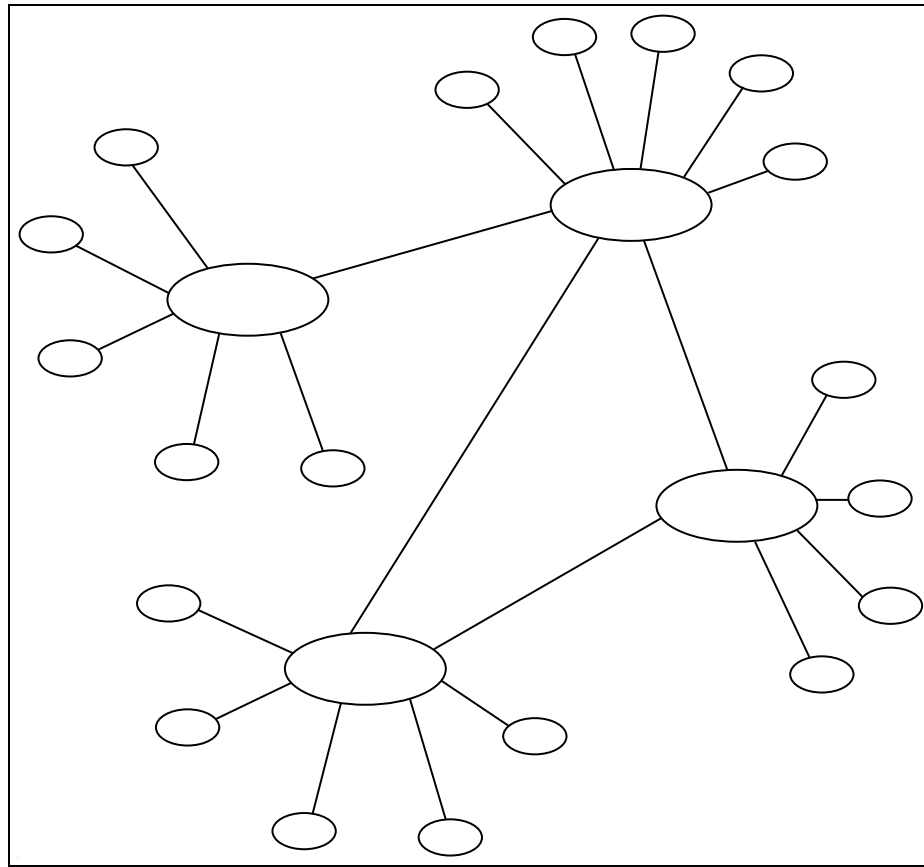


Figure 6.1: Ontology “islands”: large ellipses depict locally global ontologies; small ellipses depict locally local ontologies

between ontologies. Therefore, we expect a hybrid approach will appear, where we have several “islands” around influential domain ontologies, where within the island there is a form of global integration; one ontology would be the global ontology of the islands and a number of local ontologies are mapped to this global ontology. Then, there would be mappings between the islands, as illustrated in Figure 6.1. In the ontology mediation solution to be developed within SEKT, we need to take this into account and we need to combine global integration approaches, such as the ones supported by MOMIS and ONION, with one-to-one mappings, which are supported by, for example, MAFRA.

From the mapping process (Section 1.2) we can already see that we need different types of methods and tools for its realization. Most notably, we need an ontology matcher in order to identify similarities between ontologies and we need a mapping language and a mapping tool for the specification of ontology mapping. Examples of matchers are GLUE and S-Match. MAFRA and OntoMap provide mapping languages for the specification of mappings between ontologies.

From the comparison in Chapter 5 we can see that current approaches to ontology mapping have mostly been applied to toy problems and cannot be expected to scale both

in the number of ontologies to be mapped and the number of instances to be transformed at execution time. Within SEKT we need to take into account the different approaches that are out there and especially look into database integration approaches, which can overcome some of the scalability issues with large sets of instances.

Within SEKT the Ontology Mediation plugin called OntoMap is being developed. Note that OntoMap shares the name with the mediation system developed by Sirma (cf. Section 4.1.6), but is a completely different system. The OntoMap plugin uses F-Logic as Ontology Mapping Language and is integrated in OntoStudio which yields an integrated system. The Ontobroker reasoner is used for reasoning with the ontologies, mediation rules and data. OntoStudio has support for different kinds of databases and source formats and thus is competitive with the presented integrated systems for mediation. It also provides facilities for Query Answering. Furthermore, the following use cases can be implemented by OntoMap:

- Instance Transformation
- Instance Unification
- Ontology Merging

OntoMap supports Mapping Patterns which is not available in any of the systems that have been surveyed in this state-of-the-art survey.

Thus, OntoMap as a plugin for Ontostudio, can be considered at least competitive with the presented systems.

Bibliography

- [AC01] Maurizio Lenzerini, Andrea Cal, Giuseppe De Giacomo. Models for information integration: turning local-as-view into global-as-view. In *Proc. of the Workshop on Foundation of Models for Information Integration (FMII 2001)*, pages 270–284, 2001.
- [BCM⁺03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
- [BCV99] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record Special Issue on Semantic Interoperability in Global Information*, 28(1), March 1999.
- [BCVB01] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Ben-eventano. Semantic integration of heterogeneous information sources. *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering*, 36(1):215–249, 2001.
- [Bec03] Dave Beckett. RDF/XML syntax specification (revised). Recommendation 10 February 2004, W3C, 2003.
- [BFG01] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with LiXto. In *Proceedings of VLDB*, 2001.
- [BG04] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C, 2004. Available from <http://www.w3.org/TR/rdf-schema/>.
- [BKL⁺04] M. Boyd, S. Kittivoravithkul, P. Lazanitis, P.J. McBrien, and N. Rizopoulos. Automated: A data integration system for heterogeneous data sources. In *Proceedings of CAiSE04*, 2004.
- [BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

- <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam>.
- [BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
- [CA_dV01] Silvana Castano, Valeria De Antonellis, and Sabrina De Capitani di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, 2001.
- [CD99] James Clark and Steve DeRose. XML path language (XPath) version 1.0. Recommendation 16 November 1999, W3C, 1999.
- [CdA99] Silvana Castano and Valeria de Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *Proceedings of the 1999 International Symposium on Database Engineering & Applications*. IEEE Computer Society, 1999.
- [CDGL01] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A framework for ontology integration. In *Proc. of the 2001 Int. Semantic Web Working Symposium (SWWS 2001)*, pages 303–316, 2001.
- [CFF⁺98] 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.
- [Cha00] Hans Chalupsky. OntoMorph: A translation system for symbolic knowledge. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference*, pages 471–482, Breckenridge, Colorado, USA, 2000. Morgan Kaufmann Publishers.
- [Che79] P. Chen. The entity relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1979.
- [Cla99] James Clark. XSL transformations (XSLT) version 1.0. Recommendation 16 November 1999, W3C, 1999.
- [dB04] Jos de Bruijn. Semantic integration of disparate data sources in the cog project. In *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS2004)*, Porto, Portugal, 2004.
- [dBP04] Jos de Bruijn and Axel Polleres. Towards and ontology mapping specification language for the semantic web. Technical Report DERI-2004-06-30, DERI, 2004.

- [dBPF04] Jos de Bruijn, Axel Polleres, and Dieter Fensel. OWL lite⁻. Deliverable d20v0.1, WSMML, 2004.
Available from <http://www.wsmo.org/2004/d20/v0.1/>.
- [DKO⁺02] Ying Ding, M. Korotkiy, Borys Omelayenko, V. Kartseva, V. Zykov, Michel Klein, Ellen Schulten, and Dieter Fensel. GoldenBullet: Automated classification of product data in e-commerce. In Withold Abramowicz, editor, *Proceedings of BIS 2002*, Poznan, Poland, 2002.
- [DMDH02] AnHai Doan, Jazant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the World-Wide Web Conference*, 2002.
- [DMDH04] AnHai Doan, Jazant Madhavan, 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.
- [DMQ02] 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.
- [DR02] Hong-Hai Do and Erhard Rahm. COMA – a system for flexible combination of schema matching approaches. In *Proceedings of the VLDB'02*, pages 610–621, 2002.
- [DR05] H. Lausen (eds.) D. Roman, U. Keller. Web service modeling ontology. Technical report, DERI, 2005.
- [DS04] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004.
- [EA] P.McBrien E.Jasper, N.Tong and A.Poulovassilis. Generating and optimising views from both as view data integration rules. In *Proc. 6th Baltic Conference on Database and Information Systems (DBIS'04)*, Riga, June 2004.
- [ES04a] Marc Ehrig and Steffen Staab. QOM - quick ontology mapping. Hiroshima, Japan, November 2004.
- [ES04b] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In *Proceedings of the First European Semantic Web Symposium*, Heraklion, Greece, May 2004.
- [Fe199] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1999.

- [Fen03] Dieter Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer-Verlag, Berlin, 2003.
- [FMvH⁺03] Dieter Fensel, Enrico Motta, Frank van Harmelen, V. Richard Benjamins, Stefan Decker, Mauro Gaspari, Rix Groenboom, William Grosso, Mark A. Musen, Enric, Guus Schreiber, Rudi Studer, and Bob Wielinga. The unified problem-solving method development language upml. *Knowledge and Information Systems(KAIS) journal*, 5(1), 2003.
- [FNPB99] Jerry Fowler, Marian Nodine, Brad Perry, and Bruce Bargmeyer. Agent-based semantic interoperability in infosleuth. *SIGMOD Record*, 28(1), 1999.
- [GGM98] John H. Gennari, William Grosso, and Mark A. Musen. A method-description language: An initial ontology with examples. In *Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop*, Banff, Canada, 1998.
- [Gru94] T. Gruber. An ontology for engineering mathematics. Technical report, Knowledge Systems Laboratory, Stanford University, 1994. Report KSL-94-18.
- [GS04] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. *The Knowledge Engineering Review*, 18(3):265–280, 2004.
- [GSY04] 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.
- [GW99] Bernhard Ganter and Rudolph Wille. *Formal concept analysis: Mathematical Foundations*. Springer, Berlin-Heidelberg, 1999.
- [Hen05] Jan Henke. Omwg d8.2 editing and browsing design. Technical report, 2005.
- [HM93] Joachim Hammer and Dennis McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous, database systems. *International Journal on Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.
- [Hul97] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *ACM Symposium on Principles of Database Systems*, pages 51–61, Tuscon, Arizona, USA, 1997.
- [Ker05] M. Kerrigan. Web service modeling toolkit (wsmt). Technical report, WSMX Working Draft, 2005.

- [KFNM] Holger Knublauch, Ray W. Ferguson, Natalya F. Noy, and Mark A. Musen. The protégé owl plugin: An open development environment for semanticweb applications. In J. Allenet et al., editors, *Third International Semantic Web Conference - ISWC 2004, Hiroshima, Japan*.
- [KKOF02] Michel Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Finding and characterizing changes in ontologies. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, Tampere, Finland, 2002.
- [Kle01] 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.
- [Kle04] Michel Klein. *Change Management for Distributed Ontologies*. PhD thesis, Free University of Amsterdam, 2004.
- [KS00] Atanas Kiryakov and Kiril Iv. Simov. Mapping of eurowordnet top ontology to upper cyc ontology. In *Proceedings of Ontologies and Text workshop, EKAW 2000*, Juan-les-Pins, French Riviera, 2000.
- [KSD01a] Atanas Kiryakov, Kiril Iv. Simov, and Marin Dimitrov. Ontomap: The upper-ontology portal. In *Proceedings of "Formal Ontology in Information Systems"*, Ogunquit, Maine, 2001.
- [KSD01b] Atanas Kiryakov, Kiril Iv. Simov, and Marin Dimitrov. Tr1. ontomap: The upper-ontology portal. revision 2, Ontotext, 2001.
- [Len] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS 2002*, pages 233–246.
- [Lev00] Alon Y. Levy. *Logic-Based Techniques in Data Integration*, pages 575–595. Kluwer Publishers, 2000.
- [LGR⁺05a] N. Leone, G. Gottlob, R. Rosati, G. Greco, G. Ianni, V. Lio, V. Terracina, T. Eiter, W. Faber, M. Fink, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis. Data integration by logic programming: The INFOMIX system. In *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 2005.
- [LGR⁺05b] N. Leone, G. Gottlob, R. Rosati, G. Greco, G. Ianni, V. Lio, V. Terracina, T. Eiter, W. Faber, M. Fink, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 915–917, 2005.

- [LLR02] Domenico Lembo, Maurizio Lenzerini, and Ricardo Rosati. Functional specification of the infomix system. Technical Report D2.1, University of Rome, 2002.
- [LPF⁺05] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dl_v system for knowledge representation and reasoning. In *ACM Transactions on Computational Logic*, 2005.
- [LS99] Ora Lassila and Ralph R. Swick. Resource description framework (RDF) model and syntax specification. W3c recommendation, W3C, 1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>.
- [MBR01] 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.
- [MFRW00] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. An environment for merging and testing large ontologies. In *Proc. 7th Intl. Conf. On Principles of Knowledge Representation and Reasoning (KR2000)*, Colorado, USA, April 2000.
- [MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of ICDE*, pages 117–128, 2002.
- [MIKS00] Eduardo Mena, Arantza Illarramendi, Vipul Kashyap, and Amit P. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperability across pre-existing ontologies. *Distributed and Parallel Databases*, 8(2):223–271, 2000.
- [MMSV02] Alexander Maedche, Boris Motik, Nuno 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.
- [MNJ04] P. Mitra, N. F. Noy, and A. R. Jaiswal. Omen: A probabilistic ontology mapping tool. In *Workshop on Meaning coordination and negotiation at the Third International Conference on the Semantic Web (ISWC)*, Hiroshima, Japan, 2004.
- [Moc05] A. Mocan. Wsmx data mediation. Technical report, WSMX Working Draft, 2005.
- [MP03] P.J. McBrien and A. Poulouvasilis. Data integration by bi-directional schema transformation rules. In *Proceedings of ICDE03*, pages 227 – 238. IEEE, 2003.

- [MRB03] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Developing metadata-intensive applications with rondo. *Journal of Web Semantics*, 1(1), December 2003.
- [MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. Recommendation 10 February 2004, W3C, 2004. Available from <http://www.w3.org/TR/owl-features/>.
- [MW01] Prasenjit Mitra and Gio Wiederhold. An algebra for semantic interoperability of information sources. In *IEEE International Conference on Bioinformatics and Biomedical Engineering*, pages 174–182, 2001.
- [MWJ99] Prasenjit Mitra, Gio Wiederhold, and Jan Jannink. Semi-automatic integration of knowledge sources. In *Proceedings of Fusion 99*, Sunnydale, California, USA, July 1999.
- [MWK00] 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.
- [MZ05] M. Moran Michal Zaremba. Wsmx architecture. Technical report, WSMO Working Draft, 2005.
- [NFK⁺00] Marian H. Nodine, Jerry Fowler, Tomasz Ksiezyk, Brad Perry, Malcolm Taylor, and Amy Unruh. Active information gathering in infosleuth. *International Journal of Cooperative Information Systems*, 9(1-2):3–28, 2000.
- [NM99] Natalya F. Noy and Mark A. Musen. Smart: Automated support for ontology merging and alignment. Technical Report SMI-1999-0813, Stanford Medical Informatics, 1999.
- [NM00a] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, USA, 2000.
- [NM00b] 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.
- [NM03a] Natalya F. Noy and Mark A. Musen. Ontology versioning as an element of an ontology-management framework. To be published in *IEEE Intelligent Systems*, 2003.

- [NM03b] 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.
- [OF01] Borys Omelayenko and Dieter Fensel. A two-layered integration approach for product information in B2B e-commerce. In *Proceedings of the Second International Conference on Electronic Commerce and Web Technologies (EC WEB-2001)*, Munich, Germany, 2001. Springer-Verlag.
- [Ome02a] Borys Omelayenko. Integrating vocabularies: Discovering and representing vocabulary maps. In *Proceedings of the First International Semantic Web Conference (ISWC2002)*, Sardinia, Italy, 2002.
- [Ome02b] Borys Omelayenko. RDFT: A mapping meta-ontology for business integration. In *Proceedings of the Workshop on Knowledge Transformation for the Semantic Web (KTSW 2002) at the 15-th European Conference on Artificial Intelligence*, pages 76–83, Lyon, France, 2002.
- [PGM98] 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.
- [RB01] 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.
- [Riz03] N. Rizopoulos. Bav transformations on relational schemas based on semantic relationships between attributes. Technical Report Technical Report 22, Imperial College London, 2003.
- [SaR03a] Nuno Silva and Jo ao Rocha. Ontology mapping for interoperability in semantic web. In *Proceedings of the IADIS International Conference WWW/Internet 2003 (ICWI'2003)*, Algarve, Portugal, 2003.
- [SaR03b] 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.
- [SdB05] François Scharffe and Jos de Bruijn. A language to specify correspondences between ontologies. In *Proceedings of the IEEE Signal Image and Internet Based Systems (SITIS-05)*, 2005.
- [Sil02] N. Silva. Discovering mappings between distributed ontologies. Internal Report July, University of Karlsruhe, 2002.

- [SM01] 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.
- [Smi90] D.C. Smith. *Plisp Users Manual*. Apple Computers, august, 1990 edition, 1990.
- [Usc00] 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.
- [VC98] 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.
- [VJBCS97] Pepijn R. S. Visser, Dean M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.
- [Wie94] 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.