Analysis of knowledge transformation and merging techniques and implementations

François Scharffe, Jérôme Euzenat, Chan Le Duc, Pavel Shvaiko

D2.2.7: Analysis of knowledge transformation and merging techniques and implementations

Coordinator: François Scharffe (U. Innsbruck)
Jérôme Euzenat, Chan Le Duc (INRIA), Adrian Mocan (U. Innsbruck), Pavel Shvaiko (U. Trento)

Abstract.
Dealing with heterogeneity requires finding correspondences between entities of ontologies and using these correspondences for performing some action such as merging ontologies, transforming ontologies, translating data, mediating queries and reasoning with the aligned ontologies. This deliverable considers this problem through the introduction of an alignment life cycle which also identifies the need for manipulating, storing and sharing the alignments before processing them. In particular, we also consider support for run time and design time alignment processing.
Keyword list: ontology alignment, alignment life cycle, alignment edition, ontology merging, ontoloy transformation, data translation, query mediation, reasoning, alignment support.

Document Identifier	KWEB/2004/D2.2.7/0.8
Project	KWEB EU-IST-2004-507482
Version	0.8
Date	December 1, 2007
State	final
Distribution	public

Knowledge Web Consortium

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Work package participants

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

Centre for Research and Technology Hellas
Ecole Polytechnique Fédérale de Lausanne
Free University of Bozen-Bolzano
Institut National de Recherche en Informatique et en Automatique
National University of Ireland Galway
Universidad Politècnica de Madrid
University of Innsbruck
University of Karlsruhe
University of Manchester
University of Sheffield
University of Trento
Vrije Universiteit Amsterdam
Vrije Universiteit Brussel

Changes

Version	Date	Author	Changes
0.1	13.02.2006	Jérôme Euzenat	creation
0.2	22.04.2007	Jérôme Euzenat	inclusion of Chapter 10 from the Ontology Matching book
0.3	18.07.2006	François Scharffe	inclusion of Ontology management book material
0.4	24.09.2007	Jérôme Euzenat	reorganised outline
0.5	22.10.2007	Franọis Scharffe	rewritten some parts
0.6	06.11.2007	Jérôme Euzenat	completed biblio
0.7	23.11.2007	Franọis Scharffe	Implemented corrections from Pavel
0.8	28.11.2007	Franọis Scharffe	Added system descriptions

Executive Summary

In Knowledge web, we have considered the heterogeneity problem as a two steps problem: (i) matching ontologies to determine alignments and (ii) processing alignments according to application needs. So far, most of the work has been dedicated to the ontology matching problem.

In this deliverable, we introduce an alignment life cycle, tied to the ontology life cycle. This life cycle identifies five different operations applied to alignments: creation, evaluation, enhancement, storing and sharing, and finally processing. Since we covered elsewhere the two first operations, this deliverable focusses on the three last ones.

In particular, this mandates supporting applications to deal with alignments at design time and at run time. Indeed, the fact that some applications require high quality alignments (or high quality programs to apply) makes that it is impossible to compute these at run time. They must be either stored somewhere and retrieved dynamically or computed definitively at design time. On the other hand, some applications evolve in such a dynamic world that this is impossible and they have to compute alignments dynamically.

So, we consider alignment support at design time and in particular frameworks that enables "scriptable" alignment manipulation and editors which facilitate manual manipulation. We also consider alignment support at run time which helps storing and sharing alignments between applications. Both design time and run time environments need to effectively process the alignments, so we review the operations that use alignments for dealing with heterogeneity. This includes merging ontologies, transforming ontologies, translating data, mediating queries and reasoning with aligned ontologies. For each kind of operations, we show how it can be developed from alignments.

There usually exist a few such systems in each category, most of them tied to a particular matching strategy and thus closely related to particular applications. These systems cannot take advantage of more general alignment support. In contrast, tools developed with the two step strategy can take advantage of the results of the many matching systems available and provide executable input for any of the processing operations. We mention some of these tools and, in particular, those developed by Knowledge web partners such as the Alignment API and WSMT.

Alignment management is not as advanced as ontology management and much remains to be developed for fully supporting the alignment life cycle on a wide scale. New projects are building on Knowledge web results for providing such an integrated support.

Contents

Chapter 1

Introduction

In Knowledge web, we have taken a two steps view on reducing semantic heterogeneity: (i) matching entities to determine alignments and (ii) processing alignments according to application needs.

So far, we more specifically investigated the first step through defining alignments (D2.2.1), proposing alignment formats (D2.2.6, D2.2.10) and semantics (D2.2.5) and evaluating matching results (D2.2.2, D2.2.4, D2.2.9). Many Knowledge web partners have produced ontology matching systems.

In this deliverable, we present how the alignments can be specifically used by applications, thus focusing on the alignment processing step. We also consider the need for considering alignments in the long term, i.e., alignment management. We present the broad classes of alignment use and the tools for implementing these usages.

The remainder of this introduction is as follows: first we recall the kinds of applications which need matching and we elicit their requirements about the matching operation and the use of alignements (§1.1); then, we illustrate the use of alignment processing in data mediation (§1.2); finally, we replace the ontology matching operation within the broader view of what we call the alignment life cycle (§1.3).

Some parts of this deliverable have been published in [Euzenat and Shvaiko, 2007] and [Euzenat *et al.*, 2008].

1.1 Applications

Several classes of applications can be considered (they are more extensively described in [Euzenat and Shvaiko, 2007], we only summarise them here). They are the following:

Ontology evolution uses matching for finding the changes that have occurred between two ontology versions [De Leenheer and Mens, 2008].

Schema integration uses matching for integrating the schemas of different databases under a single view;

Catalog integration uses matching for offering an integrated access to on-line catalogs;

Data integration uses matching for integrating the content of different databases under a single database;

P2P information sharing uses matching for finding the relations between ontologies used by different peers;

Web service composition uses matching between ontologies describing service interfaces in order to compose web services by connecting their interfaces;

Multiagent communication uses matching for finding the relations between the ontologies used by two agents and translating the messages they exchange;

Context matching in ambient computing uses matching of application needs and context information when applications and devices have been developed independently and use different ontologies;

Query answering uses ontology matching for translating user queries between heterogeneous data sources on the web.

Semantic web browsing uses matching for dynamically (while browsing) annotating web pages with partially overlapping ontologies.

It is clear, from the above examples, that matching ontologies is a major issue in ontology related activities. It is not circumscribed to one application area, but applies to any application that communicates through ontologies.

These kinds of applications have been analysed in order to establish their requirements with regard to matching systems. The most important requirements concern:

- the type of available input a matching system can rely on, such as schema or instance information. There are cases when data instances are not available, for instance due to security reasons or when there are no instances given beforehand. Therefore, these applications require only a matching solution able to work without instances (here a schema-based method).
- some specific behaviour of matching, such as requirements of (i) being automatic, i.e., not relying on user feed-back; (ii) being correct, i.e., not delivering incorrect matches; (iii) being complete, i.e., delivering all the matches; and (iv) being performed at run time.
- the use of the matching result as described above. In particular, how the identified alignment is going to be processed, e.g., by merging the data or conceptual models under consideration or by translating data instances among them.

In particular, there is an important difference between applications that need alignments at design time and those that need alignments at run time. Ontology evolution is typically used at design time for transforming an existing ontology which may have instances available. It requires an accurate, i.e., correct and complete, matching, but can be performed with the help of users. Schema, catalogue and data integration are also performed off-line but can be used for different purposes: translating data from one repository to another, merging two databases or generating a mediator that will be used for answering queries. They also will be supervised by a human user and can provide instances. Other applications are rather performed at run time. Some of these, like P2P information sharing, query answering and semantic web browsing are achieved in presence of users who can support the process. They are also less demanding in terms of correctness and completeness because the user will directly sort out the results. On the other hand, web-service composition, multiagent communication and context matching in ambient computing

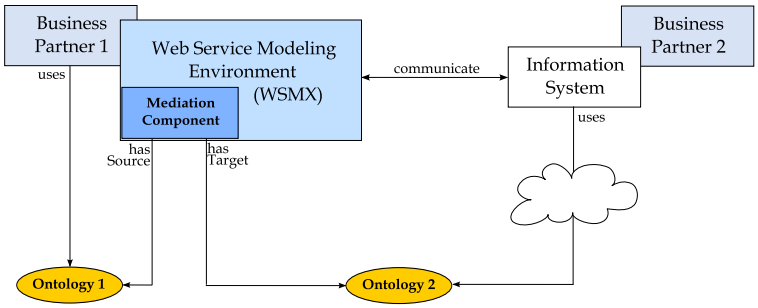

Figure 1.1: Instance transformation scenario.

require matching to be performed automatically without assistance of a human being. Since, the systems will use the result of matching for performing some action (mediating or translating data) which will be fed in other processes, correctness is required. Moreover, usually these applications do not have instance data available.

1.2 Example: data mediation for semantic web services

Web services represent one of the areas where data mediation is the most required. Services are resources usually developed independently which greatly vary from one provider to another in terms of the used data formats and representation. By adding semantics to web services, heterogeneity problems do not disappear but require more intelligent dynamic and flexible mediation solutions. Ontologies which carry most of these explicit semantics become the crucial elements to support the identification and capturing of semantic mismatches between models.

Web Services Execution Environment (WSMX) is a framework that enables discovery, selection, invocation and interoperation of Semantic Web services [Mocan *et al.*, 2006]. Ontology-based data mediation plays a crucial role in enabling all the above mentioned service operations. Different business actors use ontologies to describe their services internal business logic, and, more importantly in this case, their data. Each of these actors uses its own information system, e.g., WSMX, and tries to interact with other actors, part of other (probably more complex) business processes (Figure 1.1). A specialised component or service is needed to transform the data expressed in terms of a given ontology (the source ontology) in the terms of another ontology (target ontology), allowing the two actors to continue using their own data representation formats. Being part of a run time process the data, i.e., instances, transformation has to be performed completely automatically. In addition, as soon as such a mediator has to act in a business environment, the result of the mediation process has to be correct and complete at all time.

In order to achieve these three requirements (automation, correctness and completeness), the whole process is split in two phases: a design time phase which covers the correctness and completeness by involving the human domain expert and the run time phase when the mediation is performed in an automatic manner based on the alignments established at design time.

We will provide further details on these two phases in Section 2 and Section 6; Section 8 will consider the management of the alignments between these two phases.

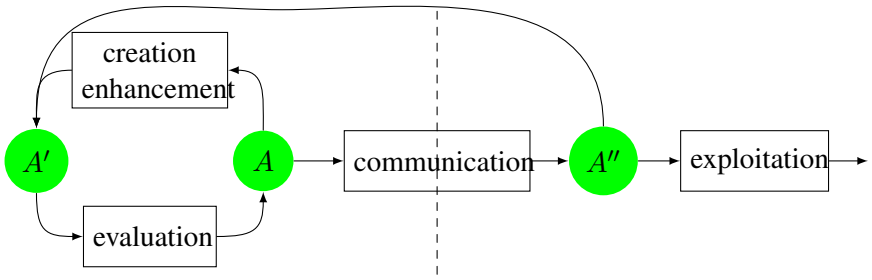

Figure 1.2: The ontology alignment life cycle.

1.3 The alignment life cycle

As the example shows, the difference between design time and run time is very relevant to ontology management. On the one hand, if alignments are required at design time, then ontology developers will need support in creating, manipulating and using these alignments. On the other hand, if alignments are required at run time, then one way of ensuring timely and adequate response may be to find some existing alignment in an alignment store. Alignments stored there should be carefully evaluated and certified alignments. They thus require alignment management on their own.

Like ontologies, alignments have their own life cycle (see Figure 1.2) and users should be supported in manipulating alignments during their life cycle. They are first created through a matching process (which may be manual). Then they can go through an iterative loop of evaluation and enhancement. Again, evaluation can be performed either manually or automatically, it consists of assessing properties of the obtained alignment. Enhancement can be obtained either through manual change of the alignment or application of refinement procedures, e.g., selecting some correspondences by applying thresholds. When an alignment is deemed worth publishing, then it can be stored and communicated to other parties interested in it. At this stage, users feedback might lead to modification of the alignment. Finally, the alignment is transformed into another form or interpreted for performing actions like mediation or merging. This last step is the specific topic of this deliverable. However, it cannot be considered independently of the previous steps.

To this first independent cycle is added the joint life cycle that can tie ontologies and alignments. As soon as ontologies evolve, new alignments have to be produced for following this evolution. This can be achieved by recording the changes made to ontologies and transforming these changes into an alignment (from one ontology version to the next one). This can be used for computing new alignments that will update the previous ones. In this case, previously existing alignments can be replaced by the composition of themselves with the ontology update alignment (see Figure 1.3).

Taking seriously ontology management requires to involve alignment management with ontology management. However, so far very few tools offer support for alignment management, let alone, joint ontology-alignment support.

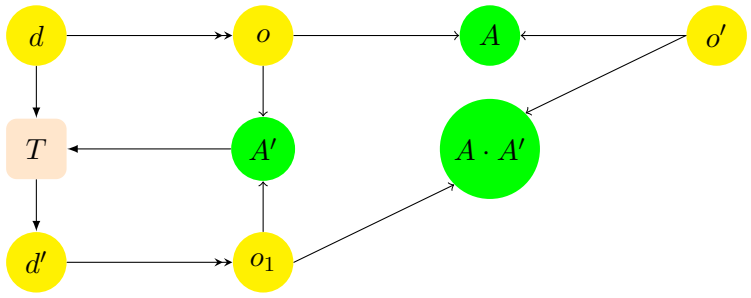

Figure 1.3: Evolution of alignments. When an ontology o evolves into a new version o_1, it is necessary to update the instances of this ontology (d) and the alignments (A) it has with other ontologies (o'). To that extent, a new alignment (A') between the two versions can be established and it can be used for generating the necessary instance transformation (T) and updated alignments ($A \cdot A'$).

1.4 Conclusion

We have identified several types of use for ontology matching results in solving heterogeneity problems. These are determined by the kind of operation to perform, e.g., merging ontologies, transforming data, and the context in which this performed, i.e., run time or design time. We have also replaced these operations within an alignment life cycle whose term is broader than one shot matching-processing. This has led to identify four distinct areas:

matching has been the subject of the work done in this work package so far and considered in other deliverable (D2.2.3);

evaluating has also be largely covered by this work package through deliverables D2.2.2, D2.2.4, and D2.2.9.

enhancing requires tools specifically designed to manipulate alignments; these tools are used mostly at design time and will be considered in Section 2;

storing and sharing requires tools to store and share alignments that can be used at design and run time; they will be presented in Section 8;

processing requires tools to process alignments that can be used at desing and run time; they will be presented in Section 3 through Section 7;

Chapter 2

Design time support for ontology matching

The first place where ontology heterogeneity can be found is while designing an application. Ontology management environments [Waterfeld *et al.*, 2008] must support users in obtaining alignments and manipulating them.

There exist infrastructures which use alignments as one of their components. The goal of such infrastructures is to enable users to perform high-level tasks which involve generating, manipulating, composing and applying alignments within the same environment. These infrastructures usually rely on the implementation of specific operations (merging, transforming, etc.) which will be considered in further chapters.

We consider here two types of infrastructures: frameworks which usually encompass matching and processing alignments (including sophisticated manipulations of these alignments) and editors which allow end users to control and modify alignments before processing them.

We illustrate the various frameworks supporting design time matching and alignment exploitation (§2.1) and alignment editors (§2.2).

2.1 Frameworks

Frameworks are environments providing support for alignment management. They usually provide formats and API so that developers can introduce matching algorithms and/or alignment processors in the framework.

2.1.1 Model management

Model management [Bernstein *et al.*, 2000; Madhavan *et al.*, 2002; Melnik, 2004] has been promoted in databases for dealing with data integration in a generic way. It offers a high-level view to the operations applied to databases and their relations. It aims at providing a metadata manipulation infrastructure in order to reduce the amount of programming required to build metadata driven applications.

Model management deals with *models* which can be related by *mappings*. A *model* is an information structure, such as an XML schema, a relational database schema, a UML model (by extrapolation, we will consider that it could be an ontology). Similarly, *mappings* are oriented

alignments from one model into another. Technically, a key idea of generic model management is to solve metadata intensive tasks at a high level of abstraction using a concise script. It is generic in the sense that a single implementation should be applicable to the majority of data models and scenarios, e.g., data translation, data integration. However, it is primarily targeted at databases. It provides an algebra to manipulate models and mappings. In [Melnik *et al.*, 2005], the following operators are defined:

- $\mathsf{Match}(m, m')$ which returns the mapping a between models m and m';
- $\mathsf{Compose}(a, a')$ which composes mappings a and a' into a new one a'', given that the range of a' is the domain of a;
- $\mathsf{Confluence}(a, a')$ which merges alignments by union of non conflicting correspondences, provided as a and a' that have the same domain and range;
- $\mathsf{Merge}(a, m, m')$ which merges two models m and m' according to mapping a;
- $\mathsf{Extract}(a, m)$ which extracts the portion of model m which is involved in mapping a;
- $\mathsf{Diff}(a, m)$ which extracts the portion of model m which is not involved in mapping a.

A mapping in this context is a function from m to m'. [Melnik *et al.*, 2005] also provides axioms governing these operations. For instance, the merge operation between two models m' and m'' through a mapping a, returns a new model $m = \mathsf{Domain}(a') \cup \mathsf{Domain}(a'')$ and a pair of surjective mappings a' and a'' from m to m' and m'' respectively, such that: $a = \mathsf{Compose}(\mathsf{Invert}(a'), a'')$.

A typical example of the model management script is as follows:

```
A1 := Match( O1, O2 );
A2 := Match( O2, O3 );
O4 := Diff( O1, A1 );
A3 := Compose( A1, A2 );
O5 := Merge( Extract( O1, A1), O3, A3 );
O6 := Merge( O4, O5, ∅ );
```

The above example operates with three ontologies. It merges the first one and the last one on the basis of the composition of their alignment with the intermediate one. Finally, it adds the part of the first one that was not brought in the first merge.

There are some model management systems available. In particular, Rondo[1] is a programming platform implementing generic model management [Melnik *et al.*, 2003b; Melnik *et al.*, 2003a]. It is based on conceptual structures which constitute the main Rondo abstractions:

Models such as relational schemas, XML schemas, are internally represented as directed labelled graphs, where nodes denote model elements, e.g., relations and attributes. Each such element is identified by an object identifier (OID).

Morphisms are binary relations over two, possibly overlapping, sets of OIDs. The morphism is typically used to represent a mapping between different kinds of models. Morphisms can always be inverted and composed.

Selectors are sets of node identifiers from a single or multiple models. These are denoted as S. A selector can be viewed as a relation with a single attribute, $S(V : OID)$, where V is a unique key.

[1] http://infolab.stanford.edu/ modman/rondo/

The operators presented above, e.g., match, merge, are implemented upon these conceptual structures. Match is implemented in Rondo by using the Similarity flooding algorithm [Melnik *et al.*, 2002]. Rondo is currently a standalone program with no editing functions.

Another system, called Moda, is described in [Melnik *et al.*, 2005] in which correspondences are expressed as logical formulas. This system is more expressive than Rondo. Examples of some other model management systems include: GeRoMe [Kensche *et al.*, 2005] and ModelGen [Atzeni *et al.*, 2005; Atzeni *et al.*, 2006].

2.1.2 COMA++ (University of Leipzig)

COMA++ [Do and Rahm, 2002; Do, 2005] is another standalone (schema) matching workbench that allows integrating and composing matching algorithms. It supports matching, evaluating, editing, storing and processing alignments.

It is built on top of COMA [Do and Rahm, 2002] and provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. COMA++ enables importing, storing and editing schemas (or models). It also allows various operations on the alignments among which compose, merge and compare. Finally, alignments can be applied to schemas for transforming or merging them.

Contrary to Rondo, the matching operation is not described as atomic but rather described as workflow that can be graphically edited and processed. Users can control the execution of the workflow in a stepwise manner and dynamically change execution parameters. The possibility of performing iterations in the matching process assumes interaction with users who approve obtained matches and mismatches to gradually refine and improve the accuracy of match (see Figure 2.1). The matching operation is performed by the Execution engine based on the settings provided by the Match customiser, including matchers to be used and match strategies.

The data structures are defined in a homogeneous proprietary format. The Schema pool provides various functions to import and export schemas and ontologies and save them to and from the internal Repository. Similarly, the Mapping pool provides functions to manipulate mappings. COMA++ can also export and import the matching workflows as executable scripts (similar to those manipulated in Rondo).

Finally, according to [Do, 2005], there are some other tools built on top of COMA++. For example, the CMC system provides a new weighting strategy to automatically combine multiple matchers [Tu and Yu, 2005], while the work of [Dragut and Lawrence, 2004] has adapted COMA to compute correspondences between schemas by composing the correspondences between individual schemas and a reference ontology.

2.1.3 MAFRA (Instituto Politecnico do Porto and University of Karlsruhe)

MAFRA[2] (MApping FRAmework) is an interactive, incremental and dynamic framework for matching distributed ontologies [da Silva, 2004; Mädche *et al.*, 2002]. It proposes an architecture for dealing with "semantic bridges" that offers functions such as creating, manipulating, storing and processing such bridges. MAFRA does not record alignments in a non processable format but associates transformations with bridges. MAFRA does not offer editing or sharing alignments.

[2] http://mafra-toolkit.sourceforge.net

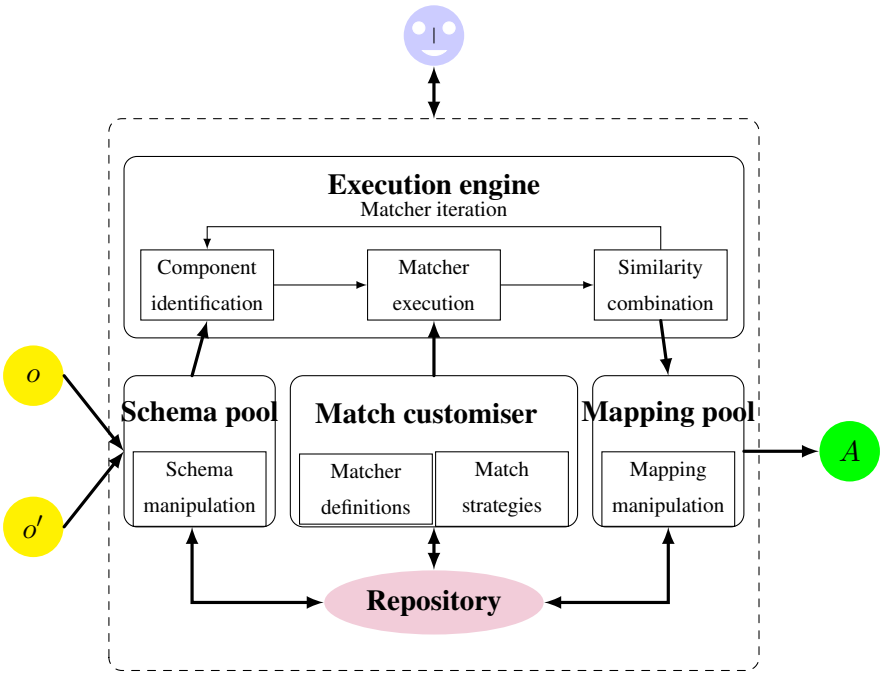

Figure 2.1: COMA++ architecture (adapted from [Do, 2005]).

The framework consists of horizontal and vertical dimensions. The horizontal dimension covers the mapping process. It is organised according to the following components:

– *Lift and Normalisation.* This module handles syntactic, structural, and language heterogeneity. In particular, the lifting process includes translation of input ontologies into an internal knowledge representation formalism, which is RDF Schema. Normalisation, in turn, includes (i) tokenisation of entities, (ii) elimination of stop words, (iii) expansion of acronyms.

– *Similarity.* This module calculates similarities between ontology entities by exploiting a combination of multiple matchers. First, *lexical similarity* between each entity from the source ontology and all entities from the target ontology is determined based on WordNet and altered Resnik measure. Second, the *property similarity* is computed. This measures similarity between concepts based on how similar the properties they are involved in are. Finally, *bottom-up* and *top-down similarities* are computed. For example, bottom-up matchers take as input the property (dis)similarity and propagate it from lower parts of the ontology to the upper concepts, thus yielding an overall view of similarity between ontologies.

– *Semantic Bridging.* Based on the similarities determined previously, the correspondences (bridges) between the entities of the source and target ontologies are established. Bridges, in turn, can be executed for the data translation task.

– *Execution.* The actual processing of bridges is performed in the execution module. This module translates instances from the source ontology to the target ontology. This translation

can either be performed off-line, i.e., one time transformation, or on-line, i.e., dynamically, thus taking into account the 'fresh' data, if any.

– *Post-processing.* This module is in charge of the analysis and improvement of the transformation results, for instance, by recognising that two instances represent the same real-world object.

Components of the vertical dimension interact with horizontal modules during the whole mapping process. There are four vertical components. The *Evolution* module, in a user-assisted way, synchronises bridges obtained with the Semantic Bridging module according to the changes in the source and target ontologies. The *Cooperative Consensus Building* module helps users to select the correct mappings, when multiple mapping alternatives exist. The *Domain Constraints and Background Knowledge* module stores common and domain specific knowledge, e.g., WordNet, precompiled domain thesauri, which are used to facilitate the similarity computation. Finally, a graphical user interface assists users in accomplishing the matching process with a desired quality.

2.1.4 Alignment API (INRIA Rhône-Alpes)

The Alignment API and implementation[3] [Euzenat, 2004] offer matching ontologies, manipulating, storing and sharing alignments as well as processor generation. It also features an Alignment Server (see Section 8.3) which can be accessed by clients through API, web services, agent communication languages ot HTTP. It does not support editing alignments.

The Alignment API manipulates alignments in the Alignment format. It can be used for can be used for implementing this format and linking to alignment algorithms and evaluation procedures. It defines a set of interfaces and a set of functions that they can perform.

Classes

The OWL API is extended with the org.semanticweb.owl.align package which describes the Alignment API. This package name is used for historical reasons. In fact, the API itself is fully independent from OWL or the OWL API.

The Alignment API is essentially made of three interfaces:

Alignment describes a particular alignment. It contains a specification of the alignment and a list of cells.

Cell describes a particular correspondence between entities.

Relation does not mandate any particular feature.

To these interfaces implementing the Alignment format, are added several of other interfaces:

AlignmentProcess extends the Alignment interface by providing an align method. So this interface is used for implementing matching algorithms (Alignment can be used for representing and manipulating alignments independently of algorithms).

Evaluator describes the comparison of two alignments (the first one could serve as a reference). Each implemented measure must provide the eval method.

An additional AlignmentException class specifies the kind of exceptions that are raised by alignment algorithms and can be used by alignment implementations.

[3] http://alignapi.gforge.inria.fr

Functions

The Alignment API provides support for manipulating alignments. As in [Bechhofer *et al.*, 2003], these functions are separated in their implementation. It offers the following functions:

Parsing and serialising an alignment from a file in RDF/XML (AlignmentParser.read(), Alignment.write());

Computing the alignment, with input alignment (Alignment.align(Alignment, Parameters));

Thresholding an alignment with threshold as argument (Alignment.cut(double));

Hardening an alignment by considering that all correspondences whose strength is strictly greater than the argument are converted to $\top$, while the others are converted to $\bot$ (Alignment.harden(double));

Comparing one alignment with another (Evaluator.eval(Parameters)) and serialising the result (Evaluator.write());

Outputting alignments in a particular format, e.g., SWRL, OWL, XSLT, RDF. (Alignment.render(visitor));

Matching and evaluation algorithms accept parameters. These are put in a structure that allows storing and retrieving them. The parameters can be various weights used by some algorithms, some intermediate thresholds or the tolerance of some iterative algorithms. There is no restriction on the kind of parameters to be used.

The Alignment API has been implemented in Java. This implementation has been used for various purposes: on-line alignment [Zhdanova and Shvaiko, 2006] and Evaluation tool in the Ontology Alignment Evaluation Initiative [Euzenat *et al.*, 2004; Stuckenschmidt *et al.*, 2005; Shvaiko *et al.*, 2007]. Also many extensions use it for implementing matching algorithms, such as oMap [Straccia and Troncy, 2005], FOAM [Ehrig, 2007], and OLA [Euzenat and Valtchev, 2004].

2.1.5 FOAM (University of Karlsruhe)

FOAM[4] [Ehrig *et al.*, 2005; Ehrig, 2007] is a framework in which matching algorithms can be integrated. It mostly offers matching and processor generation. It does not offer on-line services nor alignment editing, but is available as a Prompt plug-in (see Section 2.2.4) and is integrated in the KAON2 ontology management environment.

FOAM is a general tool for processing similarity-based ontology matching. It follows a general process, presented in Figure 2.2, which is made of the following steps:

Feature engineering selects the features of the ontologies that will be used for comparing the entities.

Search step selection selects the pairs of elements from both ontologies that will be compared.

Similarity computation computes the similarity between the selected pairs using the selected features.

Similarity aggregation combines the similarities obtained as the result of the previous step for each pair of entities.

Interpretation extracts an alignment from the computed similarity.

Iteration iterates this process, if necessary, taking advantage of the current computation.

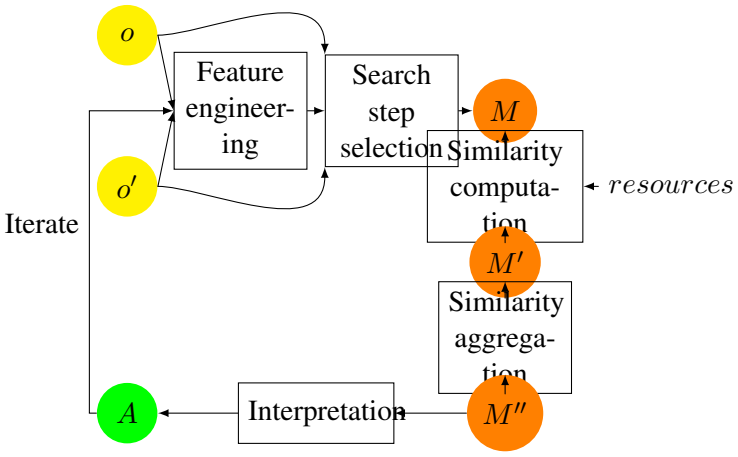

Figure 2.2: FOAM architecture (adapted from [Ehrig, 2007]).

The FOAM framework bundles several algorithms and strategies developed by its authors. Within this framework have been cast matching systems such as NOM [Ehrig and Sure, 2004], QOM [Ehrig and Staab, 2004], and APFEL [Ehrig *et al.*, 2005]. More systems can be integrated simply by changing any of the modules above. The global behaviour of the system can be parameterised through different scenarios, e.g., data integration, ontology merging, ontology evolution, query rewriting and reasoning. FOAM offer default parameters adapted to these tasks.

FOAM itself is based on the KAON2 [Oberle *et al.*, 2004] suite of tools and accepts ontologies in the OWL-DLP fragment. It offers a web-based interface. Finally, it also offers translation tools from and to the Alignment format [Euzenat, 2004] and other formats.

Platforms for integrating matchers and alignment manipulation operations are relatively new, however, they constitute a promising perspective to knowledge engineers and application developers. Another, type of useful alignment manipulation systems are alignment editors which offer human users the opportunity to be involved in the matching process.

2.2 Ontology editors with alignment manipulation capabilities

Other tools for dealing with ontology matching are ontology edition environments provided with support for matching and importing ontologies. These tools are primarily made for creating ontologies, but they also provide tools for comparing ontologies and relating them.

2.2.1 Web Service Modeling Toolkit (DERI, Austria)

WSMT[5] is a design time alignment creator and editor. It manipulates the AML [Scharffe and de Bruijn, 2005] format and can generate WSML rules [de Bruijn, 2007]. It also works as a standalone system.

[4]http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
[5]http://wsmt.sourceforge.net

As mentioned above, data mediation within a semantic environment such as WSMX is a semi-automatic process where alignments between two ontologies are created at design time and then applied at run time in order to perform instance transformation in an automatic manner. Approaches for automatic generation of ontology alignments do exist but their accuracy is usually unsatisfactory for business scenarios and it is necessary for business to business integration to have an engineer involved in creating and validating the correspondences between ontologies. This is a non-trivial task and the user should be guided through the process of creating these alignments and ensuring their correctness.

Web Service Modeling Toolkit (WSMT) [Kerrigan *et al.*, 2007] is a semantic web service and ontology engineering toolkit, also featuring tools capable of producing alignments between ontologies based on human user inputs. It offers a set of methods and techniques that assist domain experts in their work such as different graphical perspectives over the ontologies, suggestions of the most related entities from the source and target ontology, guidance throughout the matching process [Mocan *et al.*, 2006]. The tools and the domain expert work together in an iterative process that involves cycles consisting of suggestions from the tool side and validation and creation of correspondences from the domain expert side.

Within WSMT, alignments are expressed by using the Abstract Mapping Language (AML) [Scharffe and de Bruijn, 2005] which is a formalism-neutral syntax for ontology alignments. WSMT includes several tools and editors meant to offer all the necessary support for editing and managing such ontology alignments:

Alignment validation: WSMT provides validation for the AML syntax useful especially when alignments created in various tools need to be integrated into the same application.

Alignment text editor: This editor provides a text editor for the human readable syntax of AML. It provides similar features to that of a programming language editor, e.g., a Java editor, including syntax highlighting, in line error notification, content folding and bracket highlighting. This editor enables the engineer to create or modify correspondences through textual descriptions. Such a tool is normally addressed to experts familiar with both the domain and the alignment language.

Alignment view-based editor: The View-based Editor provides graphical means to create correspondences between ontologies. Such a tool is addressed to those experts that are capable of understanding the problem domain and who can successfully align the two heterogeneous ontologies but they are not specialists in logical languages as well. Additionally, even if domain experts have the necessary skills to complete the alignment by using a text editor, a graphical mapping tool would allow them to better concentrate on the heterogeneity problems to be solved and, in principle, to maximise the efficiency of the overall mapping process. All the advantages described above, have been acknowledged by other approaches as well [Mädche *et al.*, 2002; Noy and Musen, 2003]. The View-based Editor includes some of well-established classical methods, e.g., lexical and structural suggestion algorithms, iterative alignment creation processes. Additionally, this particular approach provides several new concepts and strategies aiming to enhance the overall automation degree of the ontology matching tool [Mocan and Ciampian, 2005]. Three of the most important features of this tool (views, decomposition and contexts) are presented below.

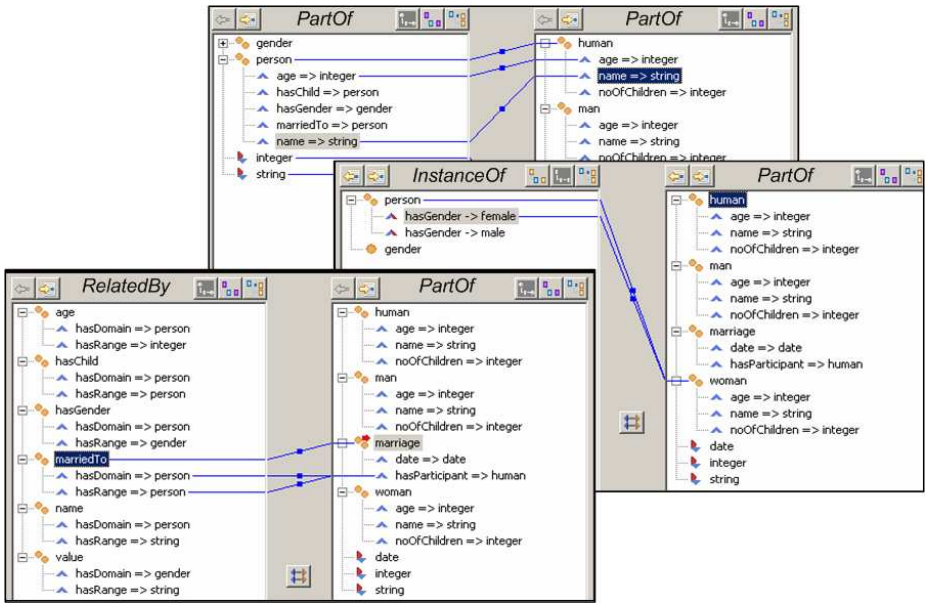

Figure 2.3: Mapping views in the AML View-Based Editor.

A *view* (also referred to as a perspective in [Mocan *et al.*, 2006]) represents a viewpoint in displaying the entities defined in a particular ontology; each view displays entities from the ontology in a two-level tree structure. The graphical viewpoint adopted to visualise the source and the target ontologies is important to simplify the design of the correspondences according to their type. By switching between combinations of these views on the source and the target ontologies, certain types of correspondences can be created using the same operations, combined with mechanisms for ontology traversal and contextualised visualisation strategies.

Each view specifies what ontological entities should appear as roots or as children in these trees, by switching the focus between various relationships existing in the ontology. Views can be defined and grouped in pairs in such a way to solicit specific skill sets, offering support for users profiling. Currently, three types of views are available, namely PartOf (concepts as roots and their attributes as children), InstanceOf (concepts as roots and their attributes together with the values they can take as children) and RelatedBy (attributes as roots and their domain or range as children); Figure 2.3 illustrates the creation of alignments by using combinations of these perspectives.

Decomposition is the process of bringing into focus the descriptive information of the root items presented in the view tree by exploring their children. A successful decomposition is followed by a context update. That is, instead of displaying the whole ontology at a time, only a subset (the one determined by decomposition) can be presented. Such subsets form the source and target contexts. If views can be seen as a vertical projection over ontologies, contexts can be seen as a horizontal projection over views. Decomposition and contexts aims to improve the effectiveness of the matching process by keeping the domain expert focused on the exact heterogeneity problem to be solved and by ensuring that all the problem-related entities have been explored.

Mappings Views: The Mappings Views provide a light overview on the alignment created either by using the Text Editor or the View-based Editor. Instead of seeing the full description of an

alignment (as quadruples in AML syntax or grounded rules in an ontology language) the domain expert can choose to see a more condensed version of this information: which are the entities in the source and in the target that are matched and if there are some special conditions associated with them.

Once a satisfying alignment has been designed, it can be stored and managed so that it is available to whoever needs it.

2.2.2 Chimaera (Stanford University)

Chimaera is a browser-based environment for editing, merging and testing (diagnosing) large ontologies [McGuinness *et al.*, 2000]. It aims to be a standard-based and generic tool. Users are provided with a graphical user interface (the Ontolingua ontology editor) for editing taxonomy and properties. They also can use various diagnosis commands, which provide a systematic support for pervasive tests and changes, e.g., tests for redundant super classes, slot value or type mismatch. Matching in the system is performed as one of the subtasks of a merge operation. Chimaera searches for merging candidates as pairs of matching terms, with terminological resources such as term names, term definitions, possible acronym and expanded forms, names that appear as suffixes of other names. It generates name resolution lists that help users in the merging task by suggesting terms which are candidates to be merged or to have taxonomic relationships not yet included in the merged ontology. The suggested candidates can be names of classes or slots. The result is output in OWL descriptions. Chimaera also suggests taxonomy areas that are candidates for reorganisation. These edit points are identified by using heuristics, e.g., looking for classes that have direct subclasses from more than one ontology.

2.2.3 The Protégé Prompt Suite (Stanford University)

Protégé[6] is an ontology edition environment that offers design time support for matching. In particular it features Prompt[7] [Noy and Musen, 2003], an interactive framework for comparing, matching, merging, maintaining versions, and translating between different knowledge representation formalisms [Noy and Musen, 2003; Noy, 2004]. The Prompt suite includes: an alignment editor (see Figure 2.4), an interactive ontology merging tool, called iPrompt [Noy and Musen, 2000] (formerly known as Prompt), an ontology matching tool, called Anchor-Prompt [Noy and Musen, 2001], an ontology-versioning tool, called PromptDiff [Noy and Musen, 2002], and a tool for factoring out semantically complete subontologies, called PromptFactor.

Since alignments are expressed in an ontology, they can be stored and shared through the Protégé server mode. Similarly to Protégé, Prompt can be extended through plug-ins. For example, there is a Prompt plug-in for FOAM. As a recent extension, Prompt offers a new functionality which allows to edit alignments graphically. The alignments can be considered as suggestions from which users may select appropriate correspondences. The graphical alignment editor is named CogZ [Falconer and Storey, 2007]. CogZ allows to create alignments by hand through drag-and-drop, to visualise the alignments and selectively filter some of the correspondences.

Figure 2.5 shows an example of usage of the graphical mapping editor in Prompt.

[6] http://protege.stanford.edu/

[7] http://protege.stanford.edu/plugins/prompt/prompt.html

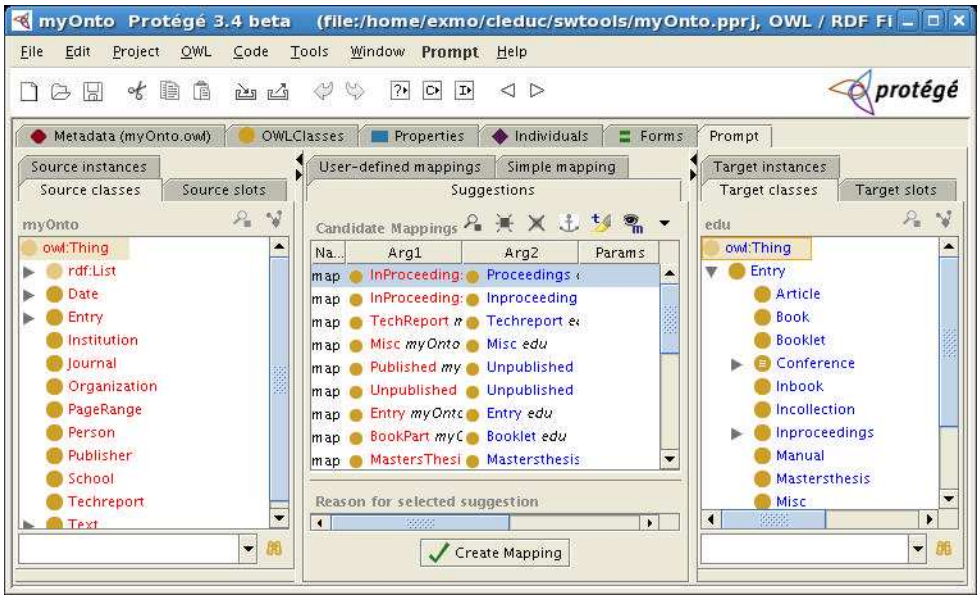

Figure 2.4: Prompt alignment editor: represents and synchronises two ontologies through their sets of correspondences.

Figure 2.5: Protégé CogZ alignment editor: curved lines connecting entities within two ontologies are correspondences that are created by providing subsumption relationships between the entities.

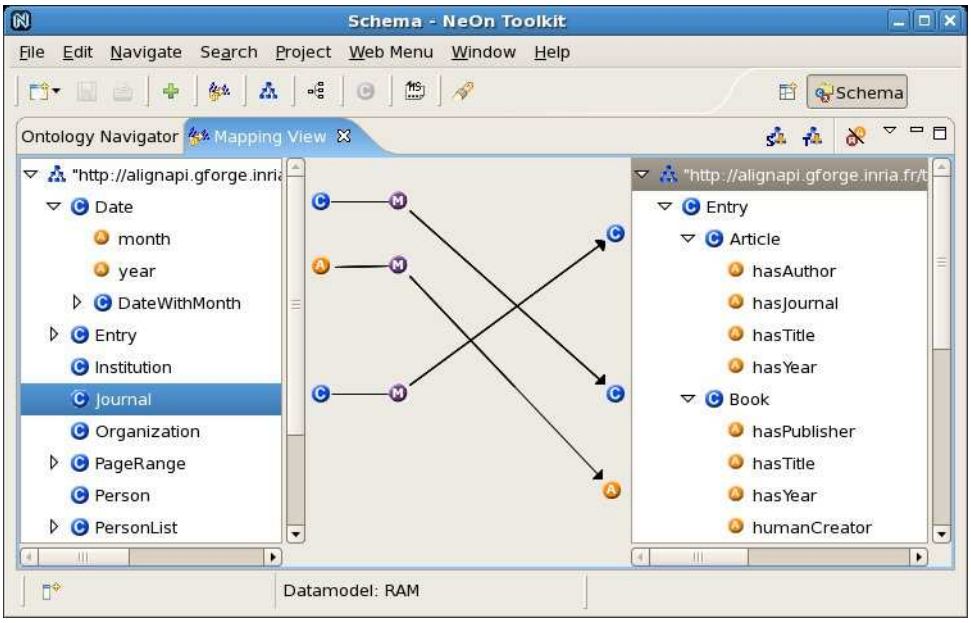

Figure 2.6: NeOn toolkit OntoMap editor: it allows to manipulate alignments between concepts and attributes and to visualise them.

2.2.4 The NeOn toolkit (NeOn Consortium)

The NeOn toolkit is an environment for managing networked ontologies developed within the NeOn project[8]. It is developed as a plug-in for managing ontologies under Eclipse and extends previous products such as KAON2 [Oberle *et al.*, 2004] which is the basis for a tool called OntoStudio [Waterfeld *et al.*, 2008].

The NeOn toolkit features run time and design time ontology alignment support. It can be extended through a plug-in mechanism, so it can be customized to the user's needs. As a development environment for ontology management, the NeOn Toolkit supports the W3C recommendations OWL and RDF as well as F-Logic for processing rules. With the support of the integrated mapping-tool, named OntoMap, heterogeneous data sources, e.g., databases, file systems, UML diagrams, can be connected to ontologies quickly and easily. Thus it provides a single view through the ontology to all connected data sources. For example, the Neon Toolkit can import external database schemes and convert them into ontologies. Connected by rules, relations of content from different databases can be created.

In addition, the Neon Toolkit offers a graphical editor for alignments, called OntoMap, in which the user can create, delete and store alignments. OntoMap allows for the creation of different types of alignments: from concept to concept (CC), from attribute to attribute (AA), etc. Each alignment can be edited by defining a filter that gives the user the possibility to limit the matched instances over their characteristic values. Figure 2.6 shows the OntoMap editor that has created CC and AA mappings.

[8] http://www.neon-project.org

2.3 Conclusion

This chapter has presented tools for dealing with ontology alignment at design time. These environments are controled by a human person and rely for their implementation on various operations. We have detailed operations on alignments such as editing or trimming. These frameworks also offers operations which process alignments on actual ontologies or databases. Such operations involve:

- merging ontologies (§3);
- transforming ontologies (§4);
- translating data and instances (§5);
- mediating queries and answers (§6);
- reasoning on aligned ontologies (§7).

These operations can either be applied at run time or at design time. This is the reason why they have not been presented in detail and will be presented in individual chapters. They will be completed by the functions of storing and sharing ontologies (§8) which are not alignment processing per se, but are useful at both run time and design time.

Chapter 3

Ontology merging

There are cases in which the ontologies are not kept separate but need to be merged into a single new ontology. As an example, we can consider the case of one vendor acquiring another, their catalogs will probably be merged into a single one. Ontology merging is achieved by taking the two ontologies to be merged and an alignment between these two ontologies. It results in a new ontology combining the two source ontologies.

3.1 Specification

Ontology merging is a first natural use of ontology matching. As depicted in Figure 3.1, it consists of obtaining a new ontology o'' from two matched ontologies o and o' so that the matched entities in o and o' are related as prescribed by the alignment. Merging can be presented as the following operator:

$$\mathsf{Merge}(o, o', A) = o''$$

The ideal property of a merge would be that ($\models$ being the consequence relation such as defined in [Euzenat and Shvaiko, 2007] or [Bouquet *et al.*, 2004]):

$$\mathsf{Merge}(o, o', A) \models o$$
$$\mathsf{Merge}(o, o', A) \models o'$$
$$\mathsf{Merge}(o, o', A) \models \alpha(A)$$

if $\alpha(A)$ is the alignment expressed in the logical language of $\mathsf{Merge}(o, o', A)$, and

$$o, o', A \models \mathsf{Merge}(o, o', A)$$

The former set of assertions means that the merge preserves the consequences of both ontologies and of the relations expressed by the alignment. The latter assertion means that the merge does not entail more consequences than specified by the semantics of alignments [Zimmermann and Euzenat, 2006]. Of course, this is not restricted to the union of the consequences of o, o' and A.

When the ontologies are expressed in the same language, merging often involves putting the ontologies together and generating bridge or articulation axioms.

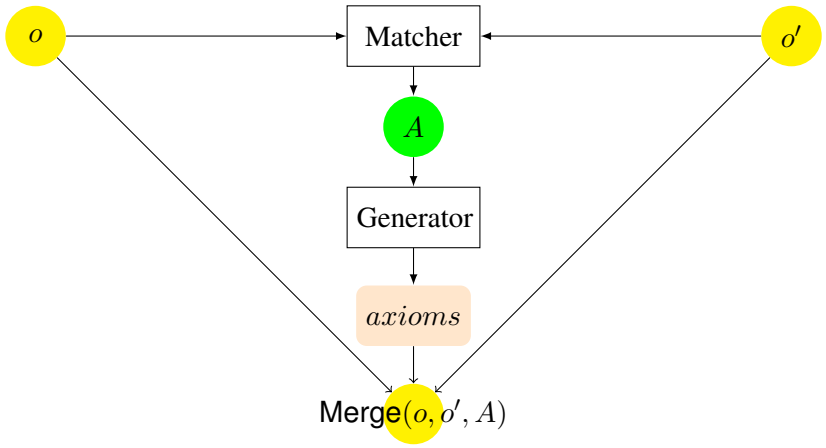

Figure 3.1: Ontology merging (from [Euzenat and Shvaiko, 2007]). From two matched ontologies o and o', resulting in alignment A, articulation $axioms$ are generated. This allows the creation of a new ontology covering the matched ontologies.

Merging does not usually require a total alignment: those entities which have no corresponding entity in the other ontology will remain unchanged in the merged ontology.

The ontology merging process can be fully automatic if an adequate alignment is provided [Scharffe, 2007], but usually requires human intervention in order to solve conflicts and choose a merging strategy. Figure 3.1 illustrates the ontology merging process.

Ontology merging is especially used when it is necessary to carry out reasoning involving several ontologies. It is also used when editing ontologies in order to create ontologies tailored for a particular application. In such a case, it is most of the time followed by a phase of ontology reengineering, e.g., suppressing unwanted parts from the obtained ontology.

3.2 Systems

Protégé [Noy and Musen, 2003; Noy, 2004] and Rondo [Melnik *et al.*, 2002] offer independent operators for ontology merging. The Alignment API [Euzenat, 2004] can generate axioms in OWL or SWRL for merging ontologies.

OntoMerge [Dou *et al.*, 2005] is a system fully dedicated to merging ontologies. Merging two ontologies is performed by taking the union of the axioms defining them. Bridge axioms or bridge rules are then added to relate the terms in one ontology to the terms in the other. They can be expressed using the full power of predicate calculus. It is assumed that bridge rules are provided by domain experts, or by other matching algorithms, which are able to discover and interpret them with clear semantics (OntoMerge does not offer matching).

Other systems are able to match ontologies and merge them directly: FCA-merge [Stumme and Mädche, 2001], SKAT [Mitra *et al.*, 1999], DIKE [Palopoli *et al.*, 2003], HCONE [Kotis *et al.*, 2006]. OntoBuilder [Modica *et al.*, 2001] uses ontology merging as an internal operation: the system creates an ontology that is mapped to query forms. This ontology is merged with the global ontology so that queries can be directly answered from the global ontology.

Chapter 4

Ontology transformation

Ontology transformation is used for connecting an ontology to another ontology. It computes the difference between the ontologies for obtaining an ontology corresponding to their merge but preserving only the specific part of one of the two ontologies (it is thus usually smaller than the merge).

4.1 Specification

Ontology transformation, from an alignment A between two ontologies o and o', consists of generating an ontology o'' expressing the entities of o with respect to those of o' according to the correspondences in A. It can be denoted as the following operator:

$$\mathsf{Transform}(o, A) = o''$$

Contrary to merging, ontology transformation, and the operators to follow, are oriented. This means that the operation has an identified source and target and from an alignment it is possible to generate two different operations depending on source and target.

The result is rather an ontology featuring only the elements of the source ontology which are not equivalent (according to the alignment) to an element of the target ontology. What is expected is that:

$$\mathsf{Merge}(o, o', A) \equiv \mathsf{Transform}(o, A) \cup o'$$

4.2 Example

Consider an ontology o' defining concepts e and e' as well as property a. Consider ontology o defining concepts c, c', c'' and c''' as well as property b and individual i and i'. The ontology o also contains the following axioms:

$$\begin{aligned} c'' &= (\text{and } c\ c'\ (\text{all } b\ 3)) \\ i &\in c'' \\ c''' &\leq c'' \\ i' &\in c''' \end{aligned}$$

Let us consider the alignment defined by:

$$e = c$$
$$e' = c'$$
$$a = b$$

Then the transformation of o with regard to this alignment is:

$$i \in e$$
$$i \in e'$$
$$i \in (\text{all } a\ 3)$$
$$c''' \leq (\text{and } e\ e'\ (\text{all } a\ 3))$$
$$i' \in c'''$$

As explained before, the transformation has replaced elements of o by equivalent elements of o'. So the resulting ontology is expressed with regard to o'. This allows answering directly queries expressed with regard to o'.

4.3 Systems

Ontology transformation is not well supported by tools. It is useful when one wants to express one ontology with regard to another one. This can be particularly useful for connecting an ontology to a common upper level ontology, for instance, or local schemas to a global schema in data integration. This is rather a design time operation.

Chapter 5

Data translation

A very common operation after matching is data translation that allows to export data to another ontology. In fact data translation occurs in ontology merging (partially) and in query mediation.

5.1 Specification

Data translation, presented in Figure 5.1, consists of translating instances from entities of ontology o into instances of connected entities of matched ontology o'. This can be expressed by the following operator:

$$\mathsf{Translate}(d, A) = d'$$

Data translation usually involves generating some transformation program from the alignment.

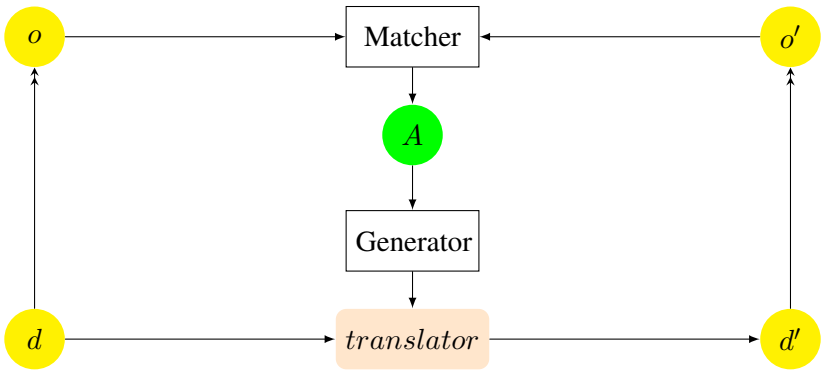

Figure 5.1: Data translation. From two matched ontologies o and o', resulting in alignment A, a $translator$ is generated. This allows the translation of the instance data (d) of the first ontology into instance data (d') for the second one.

Data translation requires a total alignment if one wants to translate all the extensional information of the source ontology. Non total alignments risk loosing instance information in the translation (this can also be acceptable if one does not want to import all the instance information).

Data translation is used for importing data under another ontology without importing the ontology itself. This is typically what is performed by database views in data integration, in multiagent

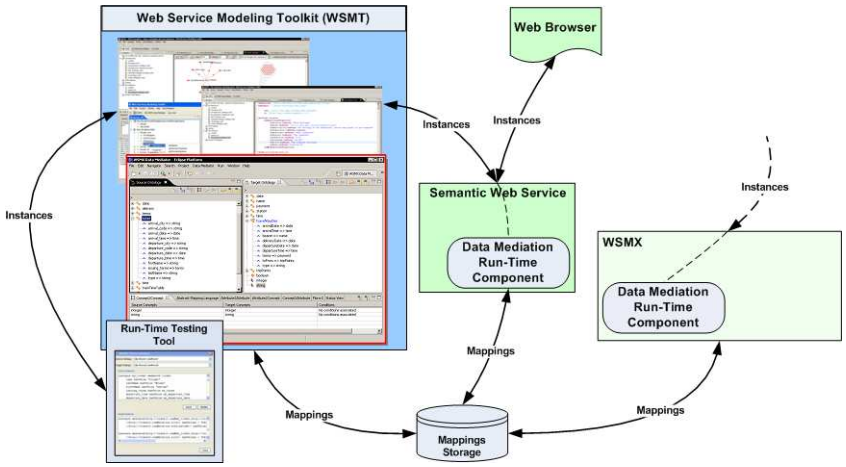

Figure 5.2: Run time data mediator usage scenario (from [Mocan and Cimpian, 2007]).

communication for translating messages, and in semantic web services for translating the flow of information in data mediators.

5.2 Example of instance translation

The mediation of the heterogeneous semantic data can be achieved through instance translation. Data represented by ontology instances has to be translated either by the sender or transparently by a third party in the format required by the receiver, i.e., instances expressed in the target ontology. In order to accommodate such a scenario, the alignments generated by using any ontology matching technique have to be processed by an engine able to perform instance translation. If the alignments are expressed in an abstract form, e.g., using AML or the Alignment format, an extra step has to be performed: the correspondences in the alignment must be expressed in a concrete ontology specification language which can be interpreted.

Figure 5.2 shows how such an instance translation engine (the Data Mediation Run Time Component in WSMX) can be deployed and used. A straightforward way is to integrate it in an information system (in this case WSMX) which needs translation support in order to facilitate the exchange of heterogeneous data. Another possibility is to encapsulate this engine in a (semantic) web service and to allow external calls having as inputs the source instances and optionally the alignments to be applied. As output, the corresponding target instances are returned. Additionally, such an engine can be used for testing the correctness of the alignments been produced, either by using it as a test module in the design time matching tool (see the WSMT MUnit) or by providing a web interface that would allow domain experts to remotely send source instances to be translated in target instances.

5.3 Instance identification

Data translation results in various sets of instances described according to the same ontology. The different origin of these instances may lead to duplicates. For instance, in a web application integrating various on-line catalogs, each described as an ontology, once the catalogs queried and the results adapted to the reference ontology, it is likely that some products are sold by many vendors. Similar products have to be identified in order to be presented under the same one (eventually with the different prices kept separate).

Instance unification techniques are used to merge similar instances by analyzing their attributes values, as well as the relations they share with other instances. This method is usable when one knows that the instances are the same. This works, for example, when integrating, two human resource databases of the same company, but does not apply for those of different companies or for databases of events which have no relations. A first natural technique for identifying instances is to take advantage of keys in databases. Keys can be either internal to the database, i.e., generated unique surrogates, in which case they are not very useful for identification, or external identification, in which case there is high probability that these identification keys are present in both data sets (even if they are not present as keys). In such a case, if they are used as keys, we can be sure that they uniquely identify an individual (like isbn). When keys are not available, or they are different, other approaches to determine property correspondences use instance data to compare property values. In databases, this technique has been known as record linkage [Fellegi and Sunter, 1969; Elfeky *et al.*, 2002] or object identification [Lim *et al.*, 1993]. They aim at identifying multiple representations of the same object within a set of objects. They are usually based on string-based and internal structure-based techniques used in ontology matching (see [Euzenat and Shvaiko, 2007]). If values are not precisely the same but their distributions can be compared, it is possible to apply global statistical techniques (see [Euzenat and Shvaiko, 2007]).

Instance identification is also necessary after two ontologies have been merged into one (see Chapter 3). Instances of the source ontologies then also need to be merged, and duplicates removed.

5.4 Systems

Rondo (§2.1.1) provides tools for data translation. The Alignment API (§2.1.4) can generate translations in XSLT or C-OWL. Many tools developed for data integration can generate translators under the form of SQL queries. Drago [Serafini and Tamilin, 2005] is an implementation of C-OWL, which can process alignments expressed in C-OWL for transferring data from one ontology to another one.

Some tools provide their output as data translation or process themselves the translation. These include: Clio [Miller *et al.*, 2000], ToMAS [Velegrakis *et al.*, 2003], TransScm [Milo and Zohar, 1998], MapOnto [An *et al.*, 2006], COMA [Do and Rahm, 2002], SKAT [Mitra *et al.*, 1999], and sPLMap [Nottelmann and Straccia, 2005].

Meanwhile, most of the commercially available ontology integration tools focus on automation of alignment processing, by opposition to matching. They are very often specialised in a particular segment of the matching space. Altova MapForce[1] and Stylus Studio XSLT Mapper[2]

[1] http://www.altova.com/products/mapforce/data_mapping.html

[2] http://www.stylusstudio.com/xslt_mapper.html

are specialised in XML integration. They integrate data from XML sources as well as databases or other structured sources. Microsoft BizTalk Schema Mapper[3] is targeted at the business process and information integration, using the proprietary BizTalk language. Ontoprise SemanticIntegrator[4] offers ontology-based integration of data coming from databases or ontologies. There are unfortunately no scholar references describing these systems in depth and URLs change so often that we refer the reader to www.ontologymatching.org for accurate and up to date information.

The matching operation itself is not automated within these tools, though they facilitate manual matching by visualising input ontologies (XML, database, flat files formats, etc.) and the correspondences between them. Once the correspondences have been established it is possible to specify, for instance, data translation operations over the correspondences such as adding, multiplying, and dividing the values of fields in the source document and storing the result in a field in the target document.

[3] http://www.microsoft.com/biztalk/
[4] http://ontoedit.com

Chapter 6

Mediation

In this chapter, we consider a mediator as an independent software component that is introduced between two other components in order to help them interoperate.

6.1 Specification

There are many different forms of mediators, including some acting as brokers or dispatchers. We concentrate here on query mediators. Query mediation consists of rewriting a query q in terms of a source ontology o into terms of a target ontology o' (according to some alignment A). This corresponds to performing the following operation (Figure 6.1 illustrates this process):

$$\mathsf{TransformQuery}(q, A) = q'$$

TransformQuery is a kind of ontology transformation (see Chapter 4) which transforms a query expressed using ontology o into a query expressed with the corresponding entities of a matched ontology o'. Query rewriting has been largely studied in database integration [Duschka and Genesereth, 1997].

Once the rewritten query addressed to the target ontology, the instances eventually returned are described in terms of o'. They might have to be translated to instances of o in order to be further processed by the system. This consists of applying a data translation operation as described in Chapter 5:

$$\mathsf{Translate}(a', \mathsf{Invert}(A)) = a$$

Instance translation is done by taking instances described under a source ontology o', and translating them to instances of a target ontology o using the alignment between the two ontologies. In this case, the alignment is taken in the reverse direction as in the query transformation operation. New instances of o classes are described, and attribute values are transformed [Scharffe and de Bruijn, 2005] according to the alignment. This process may lead to the creation of multiple target instances for one source instance, or, inversely, to combine some source instances into one target instance. Instance transformation, illustrated in Figure 6.1, is used in the example scenario in Section 1.2.

The Translate operation performs data translation on the answer of the query if necessary. This process is presented in Figure 6.1.

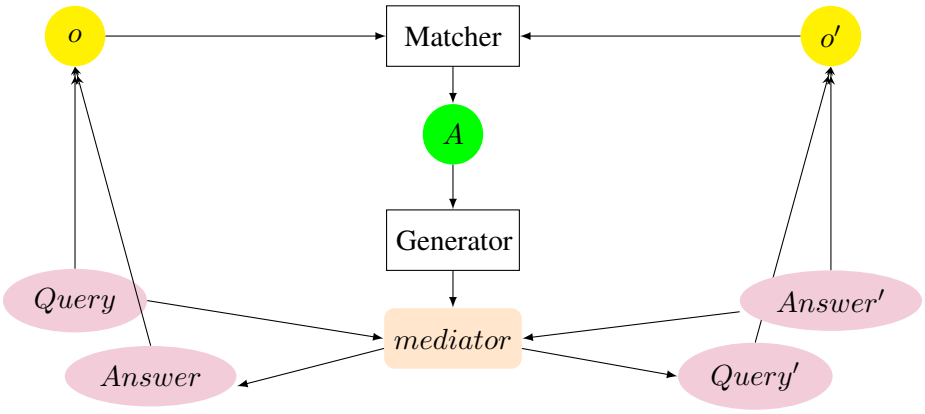

Figure 6.1: Query mediation (from [Euzenat and Shvaiko, 2007]). From two matched ontologies o and o', resulting in alignment A, a $mediator$ is generated. This allows the transformation of queries expressed with the entities of the first ontology into a query using the corresponding entities of a matched ontology and the translation back of the results from the second ontology to the first one.

Translating the answers requires the possibility of inverting the alignments (Invert operator). The generated functions should be compatible, otherwise the translated answer may not be a valid answer to the initial query. Compatibility can be expressed as follows:

$$\forall e \in o, \mathsf{TransformQuery}(\mathsf{TransformQuery}(e, A), \mathsf{Invert}(A)) \sqsubseteq e$$

Here we use a subsumption relation ($\sqsubseteq$), but it can be replaced by any suitable relation ensuring that the answer is compatible.

However, it is not always necessary to translate the answers, since they can be objects independent from the ontologies, e.g., URI, files, strings.

6.2 Systems

Query mediation is mainly used in data integration and peer-to-peer systems. When the mediator content is expressed as SQL view definitions, many database systems can process them. The Alignment API (§2.1.4) can behave as a SPARQL query mediator from simple alignments.

Some systems directly generate mediators after matching, such as Wise-Integrator [He *et al.*, 2004], DIKE [Palopoli *et al.*, 2003], Artemis [Castano *et al.*, 2000]. Clio [Haas *et al.*, 2005] can serialise alignments into different query languages, e.g., SQL, XSLT, XQuery, thus enabling query answering.

Chapter 7

Reasoning

Reasoning consists of using an alignment as axioms for reasoning with the two matched ontologies. Bridge axioms used for merging can also be viewed as such axioms.

Two kinds of reasoning can be considered. The first one reasons with the ontologies for improving them. This kind of tools provides a support for ontology matching. The second one uses the ontology and the alignments for finding their consequences, e.g., for querying in a semantic peer-to-peer system.

We present below these two types of reasoning. They are both based on the possibility, given an alignments and ontologies, to decide what are their consequences (denoted by $\models$). This assumes that the alignment is expressed in some kind of logic. For that purpose, we will consider a TransformAsRules operation which builds a set of axioms from an alignment:

$$\mathsf{TransformAsRules}(A) = o$$

Here the set of rules is represented as an ontology o which must be written in an ontology language supporting rules or the expression of bridge axioms (in OWL, C-OWL, SWRL,r F-logic, etc.).

We present below the techniques for improving alignments and reasoning with alignments.

7.1 Reasoning with alignments

Recently, several logics have been implemented providing semantics to ontology with alignments: DDL [Borgida and Serafini, 2003], $\mathcal{E}$-connections [Grau *et al.*, 2006], equalising semantics [Zimmermann and Euzenat, 2006]. These semantics provide the consequence relations for their semantics. They are thus the necessary basis for reasoning with alignments. This consist of asking to a reasoner implementing this semantics if a particular formula is a consequence of the ontologies with alignments and can be used in various applications where an assertion has to be evaluated with regard to alignments (semantic peer-to-peer systems, agent systems).

7.2 Alignment enhancement

We distinguish here between two techniques for improving alignments: detecting inconsistencies in alignments and removing them or finding a reduced or expanded form for alignments.

7.2.1 Inconsistency recovery

Inconsistency recovery consists of detecting inconsistencies caused by alignments which are discovered by heuristic or syntax-based methods. In particular, such alignments often contain wrong and redundant correspondences. Several works [Meilicke *et al.*, 2006; Meilicke *et al.*, 2007] exploit successfully the reasoning algorithm implemented in DRAGO to improve or repair ontology alignments which are created by different matching systems without any human intervention. The proposed method can be used to check (automatically created) alignments for formal consistency and determine deduced correspondences that have not explicitly been represented. The basic assumption is that *a correspondence that correctly states the semantic relations between ontologies should not cause inconsistencies in any of the ontologies*. This means that by diagnosing inconsistencies in (matched) local ontologies, we would discover incorrect sets of correspondences to be removed.

As suggested in [Meilicke *et al.*, 2006], the improving process consists of several steps :

1. *Correspondence creation*. In order to support automatic repair of inconsistent correspondences later on (step 3), the matching algorithms chosen should ideally not only return a set of correspondences but also a level of confidence in the correctness of a correspondence.
2. *Diagnosis*. With the help of an inference engine we can identify unsatisfiable concepts, and correspondences which are responsible for any unsatisfiability. We assume that the initial ontologies do not contain unsatisfiable concepts. If we now observe an unsatisfiable concept in the target ontology this implies that is caused by some correspondences in the alignment. Considering the unsatisfiable concept as a symptom, this step then tries to identify and repair the cause of this unsatisfiability. For this purpose, an irreducible conflict set for the identified unsatisfiable concept has to be computed. An irreduccible conflict set is a set of correspondences that makes the concept unsatisfiable and removing a correspondence from this set makes the concept satisfiable again.
3. *Heuristic debugging*. From the irreducible conflict set of correspondences computed, this step will decide which correspondence to remove. For this purpose, several approaches can be used: (i) removing the correspondence whose level of confidence is the smallest (automatically), (ii) if there are several correspondences that have the same level of confidence, displaying whole conflict set and leaving the decision to the user.

7.2.2 Expanding and reducing alignments

Expanding and reducing alignments consists of computing their deductive closure or reduction defined as:

$$Cn(A) = \{\alpha; o, o', A \models \alpha\}$$

and $Red(A)$ can be defined algebraically by:

1. $Red(A) \models A$
2. $A \models Red(A)$
3. $\forall A'; A' \models A$ and $A \models A \Rightarrow A' \not\subset Red(A)$

Contrary to the deductive closure, the reduction is usually not unique. This operation may be interesting for removing redundancies from an alignment. The idea is that if a correspondence is entailed from an alignment that does not contain that correspondence, then it is logically redundant

with respect to that alignment. As a consequence, such correspondences can be removed from the alignment without changing semantically the alignment. Reducing alignments by this way can be used as the last step in approaches to improving alignments. It can also be useful for presenting alignments to users in a minimal way or for more easily comparing alignments.

7.2.3 Consistent alignment merge

Merging alignments combines several available alignments between two ontologies. A consistent merge, does this so that the result is consistent by checking if they are partly or completely compatible together. If adding to an alignment A a correspondence α picked from another does not cause any inconsistency then A can be expanded by α. This task may be interesting when an alignment-based application needs a *maximal consistent alignment merge* that is built from several available alignments created by different methods. That alignment can be a candidate for a compromising solution that allows to reuse the most possible knowledge from available alignments.

Definition 1 (Expansion)**.** *Let $S = \langle o, o', \{A_1\}\rangle$ be consistent. Let A_2 be another alignment of o, o'. We say $A_1 \cup \{\alpha\}$ is an expansion of A_1 by $\alpha \in A_2$ iff $\langle o, o', \{A_1 \cup \{\alpha\}\}\rangle$ is consistent.*

Note that in Definition 1 it is not necessary that S entails α since adding α to A_1 may reduce the set of models of S. From Definition 1, we can give a formal definition for *maximal consistent alignment merge* as follows.

Definition 2 (Maximal consistent alignment merge)**.** *Let $S = \langle o, o', \mathbf{A}\rangle$ such that $\langle o, o', \{A\}\rangle$ is consistent for all $A \in \mathbf{A}$. A_M is a maximal consistent alignment merge iff*

1. $A_M \subseteq \bigcup_{A\in\mathbf{A}} A$,
2. $\langle o, o', \{A_M\}\rangle$ *is consistent, and*
3. $\langle o, o', \{A_M \cup \{\alpha\}\}\rangle$ *is inconsistent for all* $\alpha \in \bigcup_{A\in\mathbf{A}} A \setminus A_M$.

Conditions 1 and 2 guarantee that A_M is an admissible alignment constructed from the available alignments. Finally, the maximality of A_M is ensured by Condition 3. There may exist several maximal consistent alignment merges of $S = \langle o, o', \mathbf{A}\rangle$ since the expansion of an alignment by a correspondence as described in Definition 1 is nondeterministic.

A procedure for computing a maximal consistent alignment merge of $S = \langle o, o', \mathbf{A}\rangle$ can be directly devised from Definition 1 and 2. Such a procedure would consist of the following steps:

1. For each $A \in \mathbf{A}$,
2. For each $\alpha \in \bigcup_{A\in\mathbf{A}} A \setminus A_M$,
3. If $A_M \cup \{\alpha\}$ is consistent then $A_M := A_M \cup \{\alpha\}$. Repeat step 2.

Since alignments contain a finite number of correspondences, this procedure terminates. We can verify that A_M obtained from this procedure is a maximal consistent alignment merge according to Definition 2. However, this procedure would be particularly inneficient.

7.3 Systems

The inference engines Pellet [Sirin *et al.*, 2007] and DRAGO [Serafini and Tamilin, 2005] implement algorithms to reason on OWL ontologies with alignments. DRAGO is able to reason with the C-OWL language [Bouquet *et al.*, 2003] while Pellet can deal with OWL "modules" based on $\mathcal{E}$-connection.

More generaly, it is possible to retain the unique domain semantics and consider using the merge of both ontologies with a representation of the alignments for reasoning with them. Any transformation of the alignments under a form suitable for reasoning, such as SWRL, OWL, or F-Logic can be used by inference engines for these languages, such as Pellet [Sirin *et al.*, 2007], FaCT++, Racer, or Flora. The Alignment API can transform simple alignments into set of such rules. In OntoMerge (mentioned in §3), once the merged ontology is constructed, inferences can be carried out either in a demand-driven (backward-chaining) or data-driven (forward chaining) way with the help of a first-order theorem prover, called *OntoEngine*.

The inconsistency recovery technique has also been implemented in the ASMOV system [Jean-Mary and Kabuka, 2007].

Chapter 8

Run time services for storing and sharing alignments

There are several reasons why applications using ontology matching could benefit from sharing ontology matching techniques and results:

- *Each application can benefit from more algorithms*: Many different applications have comparable needs. It is thus appropriate to share the solutions to these problems. This is especially true as alignments are quite difficult to provide.
- *Each algorithm can be used in more applications*: Alignments can be used for different purposes and must be expressed as such instead of as bridge axioms, mediators or translation functions.
- *Each individual alignment can be reused by different applications*: There is no magic algorithm for quickly providing a useful alignment. Once high quality alignments have been established – either automatically or manually – it is very important to be able to store, share and reuse them.

For that purpose, it is useful to provide an alignment service able to store and share existing alignments as well as to generate new alignments on-the-fly. This kind of service should be shared by the applications using ontologies on the semantic web. They should be seen as a directory or a service by web services, as an agent by agents, as a library in ambient computing applications, etc. Operations that are necessary in such a service include:

- the ability to store alignments and retrieve them, disregarding whether they are provided by automatic means or manually;
- the proper annotation of alignments in order for the clients to evaluate the opportunity to use one of them or to start from it (this starts with the information about the matching algorithms and the justifications for correspondences that can be used in agent argumentation);
- the ability to produce alignments on-the-fly through various algorithms that can be extended and tuned;
- the ability to generate knowledge processors, such as mediators, transformations, translators and rules as well as to run these processors if necessary;
- the possibility to discover similar ontologies and to interact with other such services in order to ask them for operations that the current service cannot provide by itself.

Such a service would require a standardisation support, such as the choice of an alignment format or at least of metadata format. There have been proposals for providing matching systems and alignment stores that can be considered as servers [Euzenat, 2005; Zhdanova and Shvaiko, 2006], but they need a wider availability (to agents, services, etc.) and achieving a critical mass of users to really be helpful.

Alignment support can be implemented either as a component of an ontology management tool and even being specific to each particular workstation (see Chapter 2). However, in order to optimise sharing, which is an important benefit of using alignments, it is better to store the alignments in an independent alignment server. Such a server can be either used for sharing alignments among a particular organisation or open to the semantic web at large.

8.1 Storing alignments

If alignments between widely accepted ontologies are required, they will have to be found over and over again. Hence, as mentioned in the requirements, the alignments should be stored and shared adequately. An infrastructure capable of storing the alignments and of providing them on demand to other users would be useful.

Alignment servers are independent software components which offer a library of matching methods and an alignment store that can be used by their clients. In a minimal configuration, alignment servers contribute storing and communicating alignments. Ideally, they can offers all the services identified in Chapter 1 and in particular alignment manipulation.

Alignment servers serve two purposes: for design time ontology matching, they will be components loosely coupled to the ontology management environment which may ask for alignments and for exploiting these alignments. For run time matching, the alignment servers can be invoked directly by the application. So, alignment servers will implement the services for both design time and run time matching at once.

These servers are exposed to clients, either ontology management systems or applications, through various communication channels (agent communication messages, web services) so that all clients can effectively share the infrastructure. A server may be seen as a directory or a service by web services, as an agent by agents, as a library in ambient computing applications, etc.

Alignment servers must be found on the semantic web. For that purpose they can be registered by service directories, e.g., UDDI for web services. Services or other agents should be able to subscribe some particular results of interest by these services. These directories are useful for other web services, agents, peers to find the alignment services.

In addition, servers can be grouped into an alignment infrastructure which supports them in communicating together. They can exchange the alignments they found and select them on various criteria. This may be useful for alignment servers to outsource some of their tasks. In particular, it may happen that:

- they cannot render an alignment in a particular format;
- they cannot process a particular matching method;
- they cannot access a particular ontology;
- a particular alignment is already stored by another server.

In these events, the concerned alignment server will be able to call other servers. This is especially useful when the client is not happy with the alignments provided by the current server, it is then

possible to either deliver alignments provided by other servers or to redirect the client to these servers.

Moreover, this opens the door to value-added alignment services which use the results of other servers as a pre-processing for their own treatments or which aggregates the results of other servers in order to deliver a better alignment.

8.2 Sharing alignments

The main goal of storing alignments is to be able to share them among different applications. Because these applications have diverse needs and various selection criteria, it is necessary to be able to search and retrieve alignments on these criteria. Alignment metadata used for indexing alignments are thus very important. So far, alignments contain information about:

- the aligned ontologies;
- the language in which these ontology are expressed;
- the kind of alignment it is (1:1 or n:m for instance);
- the algorithm that provided it (or if it has been provided by hand);
- the confidence in each correspondence.

This information is already very precious and helps applications selecting the most appropriate alignments. It is thus necessary that ontology matchers be able to generate and alignment servers be able to store these metadata. Oyster [Palma and Haase, 2005], a peer-to-peer infrastructure for sharing metadata about ontologies that can be used in ontology management, has been extended for featuring some metadata about alignments.

However, metadata schemes are extensible and other valuable information may be added to alignment format, such as:

- the parameters passed to the generating algorithms;
- the properties satisfied by the correspondences (and their proof if necessary);
- the certificate from an issuing source;
- the limitations of the use of the alignment;
- the arguments in favor or against a correspondence [Laera *et al.*, 2007].

All such information can be useful for evaluating and selecting alignments and thus should be available from alignment servers.

8.3 The Alignment server

The Alignment server has been proposed in [Euzenat, 2005] and implemented as reported in [Euzenat *et al.*, 2007] in order to suit the purpose of storing and sharing alignments and methods for finding alignments. Such a server enables matching ontologies, storing the resulting alignment, storing manually provided alignments, extracting merger, transformer, mediators from these alignments.

The Alignment Server is built around the Alignment API developed by INRIA (see Figure 8.1). It thus provides access to all the features of this API (see Table 8.1). The server ensures the persistence of the alignments through the storage of these in a relational database.

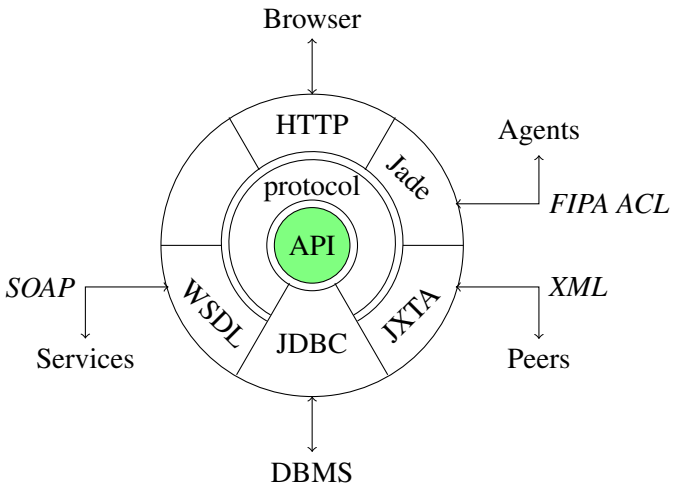

Figure 8.1: The Alignment Server is built on the Alignment API that is seated on top of a relational database repository for alignment and is wrapped around a simple protocol. Each access method is a plug-in that interacts with the server through the protocol. Currently, HTML, agent and web service plug-ins are available.

The server is available for applications at design time and/or at run time. There is no constraint that the alignments are computed on-line or off-line, i.e., they are stored in the alignment store, or that they are processed by hand or automatically.

Access to the API is achieved through a protocol which extends the one designed in [Euzenat *et al.*, 2005]. Plug-ins allow the remote invocation of the alignment server (see Figure 8.1). At the time of writting, three plug-ins are available for the server:

– HTTP/HTML plug-in for interacting through a browser;
– JADE/FIPA ACL for interacting with agents;
– HTTP/SOAP plug-in for interacting as a web service.

The components of the Alignment Server as well as the connected clients can be distributed in different machines. Several servers can share the same databases (the server works in write once mode: it never modifies an alignment but always creates new ones; not all the created alignments being stored in the database *in fine*). Applications can reach the Alignment server by any way they want, e.g., starting by using Jade and then turning to web service interface.

Alignment services must be found on the semantic web. For that purpose they can be registered by service directories, e.g., UDDI for web services. These directories are useful for other web services, agents, peers to find the alignment services. They are even more useful for alignment services to basically outsource some of their tasks. In particular, it may happen that:

– they cannot render an alignment in a particular format;
– they cannot process a particular matching method;
– they cannot access a particular ontology;
– a particular alignment is already stored by another service.

In these events, the concerned alignment service will be able to call other alignment services. This is especially useful when the client is not happy with the alignments provided by the current

Service	Syntax
Finding a similar ontology	$o' \Leftarrow Match(o, t)$
Align two ontologies	$A' \Leftarrow Align(o, o', A, p)$
Thresholding	$A' \Leftarrow Threshold(A, V)$
Generating code	$P \Leftarrow Render(A, language)$
Translating a message	$m' \Leftarrow Translate(m, A)$
Storing alignment	$n \Leftarrow Store(A, o, o')$
Suppressing alignment	$Delete(n)$
Finding (stored) alignments	$\{n\} \Leftarrow Find(o, o')$
Retrieving alignment	$\langle o, o', A\rangle \Leftarrow Retrieve(n)$

Table 8.1: Services provided by the alignment service and corresponding API primitives (o denotes an ontology, A an alignment, p parameters, n an index denoting an alignment, P a program realising the alignment and t and m some expressions, namely, terms to be matched and messages to be translated).

service, it is then possible to either deliver alignments provided by other services or to redirect the client to these services.

Like the Alignment API, the Alignment server can always be extended. In particular, it is possible to add new matching algorithms and mediator generators that will be accessible through the API. They will also be accessible through the alignment services. Services can thus be extended to new needs without breaking the infrastructure.

Chapter 9

Conclusion

Dealing with ontology heterogeneity involves finding the alignments, or sets of correspondences, existing between ontology entities and processing them for reconciling the ontologies. We have reviewed techniques for processing alignments and systems implementing these techniques. Alignments can be used in different ways (merging, transformation, translation, mediation) and there are different languages adapted to each of these ways (SWRL, OWL, C-OWL, XSLT, SQL, etc.). Beyond alignment processing, we also have identified the need for alignment management at both design time and run time.

So far there are only a few systems able to process alignments in such diverse ways. Several matching systems process directly their results in one of these operation, while others deliver alignments. Unfortunately, most often, the delivered alignments are in a format that cannot be exploited by other systems and operator generators, thus requiring additional efforts to embed them into the new environments. This is not particularly useful for alignment management.

Useful alignments are such a scarce resource that storing them in an independent format such as those seen in deliverable D2.2.6 and D2.2.10 is very important. It would allow sharing and processing them in different ways independently form the applications. This would give more freedom to application developers to choose the best suited algorithm and to process alignments adequately.

The small number of systems implementing these techniques with regard to the large number of systems for ontology matching validate the two-steps approach for dealing with heterogeneity used in Knowledge web. No system implements all the proposed features, though those implemented by Knowledge web partners, namely the Alignment API from INRIA and WSMT from the University of Innsbruck, are the closest to this, by implementing the two-steps approach.

Alignment management is not as advanced as ontology management and much remains to be developed for fully supporting and sharing alignments on a wide scale. Challenges for alignment management include adoption challenges and research problems. The important challenge is to have a natural integration of alignment management with most of the ontology engineering and ontology management systems. If alignment sharing and management is to become a reality, then there should not be one proprietary format with each tool that cannot be handled by other tools. Knowledge web has worked towards this by providing expressive formats for alignments. Another challenge is the easy retrieving of available alignments. To some extent, we pursue this effort within other projects (e.g., NeOn). For this purpose, proper alignment metadata and web-wide search support have to be set up.

There remains difficult research problems in the domain of alignment management such as:

- The identification of duplicate alignments or evolutions from a particular alignment;
- Aggregating, composing and reasoning usefully with a massive number of alignments;
- The design of ever better user interaction systems for both interacting with matching systems and editing alignments.

Appendix A

Overview of processing systems

Here are the systems reviewed in [Euzenat and Shvaiko, 2007] which can process alignments (operation).

Table A.1: Various ontology matching systems offering alignment processing.

System	Input	Needs	Output	Operation
TransScm [Milo and Zohar, 1998]	SGML, OO	Semi	Translator	Data translation
SKAT [Mitra *et al.*, 1999]	RDF	Semi	Bridge rules	Data translation
COMA & COMA++ [Do, 2005]	Relational schema, XML schema, OWL	User	Alignment	Data translation
ToMAS [Velegrakis *et al.*, 2003]	Relational schema, XML schema	Query, Alignment	Query, Alignment	Data translation
MapOnto [An *et al.*, 2006]	Relational schema, XML schema, OWL	Alignment	Rules	Data translation
sPLMap [Nottelmann and Straccia, 2005]	Database schema	Auto, Instances, Training	Rules	Data translation
Clio [Miller *et al.*, 2000]	Relational schema, XML schema	Semi, Instances (opt.)	Query transformation	Data translation
DIKE [Palopoli *et al.*, 2003]	ER	Semi	Merge	Query mediation
Artemis [Castano *et al.*, 2000]	Relational schema, OO, ER	Auto	Views	Query mediation
oMap [Straccia and Troncy, 2005]	OWL	Auto, Instances (opt.), Training	Alignment	Query answering
H-Match [Castano *et al.*, 2006]	OWL	Auto	Alignment	P2P query mediation
Tess [Lerner, 2000]	Database schema	Auto	Rules	Version matching
OntoBuilder [Modica *et al.*, 2001]	Web form, XML schema	User	Mediator	Ontology merging
Anchor-Prompt [Noy and Musen, 2001]	OWL, RDF	User	Axioms (OWL/RDF)	Ontology merging
OntoMerge	OWL	Alignment	Ontology	Ontology merging
HCONE [Kotis *et al.*, 2006]	OWL	Auto, Semi, User	Ontology	Ontology merging
FCA-merge	Ontology	User,	Ontology	Ontology merging

Table A.1: Various ontology matching systems offering alignment processing (continued).

System	Input	Needs	Output	Operation
[Stumme and Mädche, 2001]		Instances		
DCM	Web form	Auto	Alignment	Data integration
[He and Chang, 2006]				
Wang & al.	Web form	Instances	Alignment	Data integration
Wise-Integrator	Web form	Auto	Mediator	Data integration

Bibliography

[An *et al.*, 2006] Yuan An, Alexander Borgida, and John Mylopoulos. Discovering the semantics of relational tables through mappings. *Journal on Data Semantics*, VII:1–32, 2006.

[Atzeni *et al.*, 2005] Paolo Atzeni, Paolo Cappellari, and Philip Bernstein. Modelgen: Model independent schema translation. In *Proc. 21st International Conference on Data Engineering (ICDE)*, pages 1111–1112, Tokyo (JP), 2005.

[Atzeni *et al.*, 2006] Paolo Atzeni, Paolo Cappellari, and Philip Bernstein. Model-independent schema and data translation. In *Proc. 10th Conference on Extending Database Technology (EDBT)*, volume 3896 of *Lecture notes in computer science*, pages 368–385, München (DE), 2006.

[Bechhofer *et al.*, 2003] Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the semantic web with the OWL API. In *Proc. 2nd International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture notes in computer science*, pages 659–675, Sanibel Island (FL US), 2003.

[Bernstein *et al.*, 2000] Philip Bernstein, Alon Halevy, and Rachel Pottinger. A vision of management of complex models. *ACM SIGMOD Record*, 29(4):55–63, 2000.

[Borgida and Serafini, 2003] Alexander Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *Journal on Data Semantics*, I:153–184, 2003.

[Bouquet *et al.*, 2003] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL – contextualizing ontologies. In *Proc. 2nd International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture notes in computer science*, pages 164–179, Sanibel Island (FL US), 2003.

[Bouquet *et al.*, 2004] Paolo Bouquet, Marc Ehrig, Jérôme Euzenat, Enrico Franconi, Pascal Hitzler, Markus Krötzsch, Luciano Serafini, Giorgos Stamou, York Sure, and Sergio Tessaris. Specification of a common framework for characterizing alignment. Deliverable D2.2.1, Knowledge web NoE, 2004.

[Castano *et al.*, 2000] 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, 2000.

[Castano *et al.*, 2006] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching ontologies in open networked systems: Techniques and applications. *Journal on Data Semantics*, V:25–63, 2006.

[da Silva, 2004] Nuno Alexandre Pinto da Silva. *Multi-dimensional service-oriented ontology mapping*. PhD thesis, Universidade de Trás-os-Montes e Alto Douro, Villa Real (PT), 2004.

[de Bruijn, 2007] Jos de Bruijn. The web service modeling language WSML. Technical Report 16.1, WSMO, 2007.

[De Leenheer and Mens, 2008] Pieter De Leenheer and Tom Mens. Ontology evolution; state of the art and future directons. In Martin Hepp, Pieter De Leenheer, Aldo De Moor, and York Sure, editors, *Ontology management: semantic web, semantic web services, and business applications*, chapter 5, pages 131–176. Springer, New-York (NY US), 2008.

[Do and Rahm, 2002] Hong-Hai Do and Erhard Rahm. COMA – a system for flexible combination of schema matching approaches. In *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, pages 610–621, Hong Kong (CN), 2002.

[Do, 2005] Hong-Hai Do. *Schema matching and mapping-based data integration*. PhD thesis, University of Leipzig, Leipzig (DE), 2005.

[Dou *et al.*, 2005] Dejing Dou, Drew McDermott, and Peishen Qi. Ontology translation on the semantic web. *Journal on Data Semantics*, II:35–57, 2005.

[Dragut and Lawrence, 2004] Eduard Dragut and Ramon Lawrence. Composing mappings between schemas using a reference ontology. In *Proc. 3rd International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 3290 of *Lecture notes in computer science*, pages 783–800, Larnaca (CY), 2004.

[Duschka and Genesereth, 1997] Oliver Duschka and Michael Genesereth. Infomaster – an information integration tool. In *Proc. KI Workshop on Intelligent Information Integration*, Freiburg (DE), 1997.

[Ehrig and Staab, 2004] Marc Ehrig and Steffen Staab. QOM – quick ontology mapping. In *Proc. 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture notes in computer science*, pages 683–697, Hiroshima (JP), 2004.

[Ehrig and Sure, 2004] Marc Ehrig and York Sure. Ontology mapping – an integrated approach. In *Proc. 1st European Semantic Web Symposium (ESWS)*, volume 3053 of *Lecture notes in computer science*, pages 76–91, Hersounisous (GR), May 2004.

[Ehrig *et al.*, 2005] Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping ontology alignment methods with APFEL. In *Proc. 4th International Semantic Web Conference (ISWC)*, volume 3729 of *Lecture notes in computer science*, pages 186–200, Galway (IE), 2005.

[Ehrig, 2007] Marc Ehrig. *Ontology alignment: bridging the semantic gap*. Semantic web and beyond: computing for human experience. Springer, New-York (NY US), 2007.

[Elfeky *et al.*, 2002] Mohamed Elfeky, Ahmed Elmagarmid, and Vassilios Verykios. Tailor: A record linkage tool box. In *Proc. 18th International Conference on Data Engineering (ICDE)*, pages 17–28, San Jose (CA US), 2002.

[Euzenat and Shvaiko, 2007] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, Heidelberg (DE), 2007.

[Euzenat and Valtchev, 2004] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proc. 15th European Conference on Artificial Intelligence (ECAI)*, pages 333–337, Valencia (ES), 2004.

[Euzenat *et al.*, 2004] Jérôme Euzenat, Marc Ehrig, and Raúl García Castro. Specification of a benchmarking methodology for alignment techniques. Deliverable D2.2.2, Knowledge web NoE, 2004.

[Euzenat *et al.*, 2005] Jérôme Euzenat, Loredana Laera, Valentina Tamma, and Alexandre Viollet. Negotiation/argumentation techniques among agents complying to different ontologies. Deliverable 2.3.7, Knowledge web NoE, 2005.

[Euzenat *et al.*, 2007] Jérôme Euzenat, Antoine Zimmermann, Marta Sabou, and Mathieu d'Aquin. Matching ontologies for context. deliverable 3.3.1, NeOn, 2007.

[Euzenat *et al.*, 2008] Jérôme Euzenat, Adrian Mocan, and François Scharffe. Ontology alignment: an ontology management perspective. In Martin Hepp, Pieter De Leenheer, Aldo De Moor, and York Sure, editors, *Ontology management: semantic web, semantic web services, and business applications*, chapter 6, pages 177–206. Springer, New-York (NY US), 2008.

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture notes in computer science*, pages 698–712, Hiroshima (JP), 2004.

[Euzenat, 2005] Jérôme Euzenat. Alignment infrastructure for ontology mediation and other applications. In *Proc. International Workshop on Mediation in Semantic Web Services (MEDIATE)*, pages 81–95, Amsterdam (NL), 2005.

[Falconer and Storey, 2007] Sean Falconer and Margaret-Anne Storey. A cognitive support framework for ontology mapping. In *Proc. 6th International Semantic Web Conference, Busan (KR)*, pages 114–127, 2007.

[Fellegi and Sunter, 1969] Ivan Fellegi and Alan Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[Grau *et al.*, 2006] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining OWL ontologies using $\mathcal{E}$-connections. *Journal of web semantics*, 4(1):40–59, 2006.

[Haas *et al.*, 2005] Laura Haas, Mauricio Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *Proc. 24th International Conference on Management of Data (SIGMOD)*, pages 805–810, Baltimore (MD US), 2005.

[He and Chang, 2006] Bin He and Kevin Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. *ACM Transactions on Database Systems*, 31(1):1–45, 2006.

[He *et al.*, 2004] Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Automatic integration of web search interfaces with WISE-Integrator. *The VLDB Journal*, 13(3):256–273, 2004.

[Jean-Mary and Kabuka, 2007] Yves Jean-Mary and Mansur Kabuka. Asmov results for oaei 2007. In *Proc. 2nd Ontology matching workshop, Busan (KR)*, pages 141–150, 2007.

[Kensche *et al.*, 2005] David Kensche, Christoph Quix, Mohamed Amine Chatti, and Matthias Jarke. GeRoMe: A generic role based metamodel for model management. In *Proc. 4th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 3761 of *Lecture notes in computer science*, pages 1206–1224, Agia Napa (CY), 2005.

[Kerrigan *et al.*, 2007] Mike Kerrigan, Adrian Mocan, Martin Tanler, and Dieter Fensel. The web service modeling toolkit - an integrated development environment for semantic web services. In *Proc. 4th European Semantic Web Conference (ESWC) System Description Track*, pages 303–317, Innsbruck (AT), 2007.

[Kotis *et al.*, 2006] Konstantinos Kotis, George Vouros, and Konstantinos Stergiou. Towards automatic merging of domain ontologies: The HCONE-merge approach. *Journal of Web Semantics*, 4(1):60–79, 2006.

[Laera *et al.*, 2007] Loredana Laera, Ian Blacoe, Valentina Tamma, Terry Payne, Jérôme Euzenat, and Trevor Bench-Capon. Argumentation over ontology correspondences in MAS. In *Proc. 6th International conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1285–1292, Honolulu (HA US), 2007.

[Lerner, 2000] Barbara Staudt Lerner. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems*, 25(1):83–127, 2000.

[Lim *et al.*, 1993] Ee-Peng Lim, Jaideep Srivastava, Satya Prabhakar, and James Richardson. Entity identification in database integration. In *Proc. 9th International Conference on Data Engineering (ICDE)*, pages 294–301, Wien (AT), 1993.

[Mädche *et al.*, 2002] Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proc. 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, volume 2473 of *Lecture notes in computer science*, pages 235–250, Siguenza (ES), 2002.

[Madhavan *et al.*, 2002] Jayant Madhavan, Philip Bernstein, Pedro Domingos, and Alon Halevy. Representing and reasoning about mappings between domain models. In *Proc. 18th National Conference on Artificial Intelligence (AAAI)*, pages 122–133, Edmonton (CA), 2002.

[McGuinness *et al.*, 2000] Deborah McGuinness, Richard Fikes, James Rice, and Steve Wilder. An environment for merging and testing large ontologies. In *Proc. 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 483–493, Breckenridge (CO US), 2000.

[Meilicke *et al.*, 2006] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Improving automatically created mappings using logical reasoning. In *Proc. 1st ISWC International Workshop on Ontology Matching (OM)*, pages 61–72, Athens (GA US), 2006.

[Meilicke *et al.*, 2007] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Repairing ontology mappings. In *Proc. 22nd National conference on artificial intelligence (AAAI), Vancouver (CA)*, pages 1408–1413, 2007.

[Melnik *et al.*, 2002] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proc. 18th International Conference on Data Engineering (ICDE)*, pages 117–128, San Jose (CA US), 2002.

[Melnik *et al.*, 2003a] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Developing metadata-intensive applications with Rondo. *Journal of Web Semantics*, 1(1):47–74, 2003.

[Melnik *et al.*, 2003b] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Rondo: A programming platform for model management. In *Proc. 22nd International Conference on Management of Data (SIGMOD)*, pages 193–204, San Diego (CA US), 2003.

[Melnik *et al.*, 2005] Sergey Melnik, Philip Bernstein, Alon Halevy, and Erhard Rahm. Supporting executable mappings in model management. In *Proc. 24th International Conference on Management of Data (SIGMOD)*, pages 167–178, Baltimore (MD US), 2005.

[Melnik, 2004] Sergey Melnik. *Generic Model Management Concepts and Algorithms*. Springer, Heidelberg (DE), 2004.

[Miller *et al.*, 2000] Renée Miller, Laura Haas, and Mauricio Hernández. Schema mapping as query discovery. In *Proc. 26th International Conference on Very Large Data Bases (VLDB)*, pages 77–88, Cairo (EG), 2000.

[Milo and Zohar, 1998] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. 24th International Conference on Very Large Data Bases (VLDB)*, pages 122–133, New York (NY US), 1998.

[Mitra *et al.*, 1999] Prasenjit Mitra, Gio Wiederhold, and Jan Jannink. Semi-automatic integration of knowledge sources. In *Proc. 2nd International Conference on Information Fusion*, pages 572–581, Sunnyvale (CA US), 1999.

[Mocan and Ciampian, 2005] Adrian Mocan and Emilia Ciampian. Mappings creation using a view based approach. In *Proc. of the 1st International Workshop on Mediation in Semantic Web Services (Mediate)*, pages 97–112, Amsterdam (NL), 2005.

[Mocan and Cimpian, 2007] Adrian Mocan and Emilia Cimpian. An ontology-based data mediation framework for semantic environments. *International journal on semantic web and information systems*, 3(2):66–95, 2007.

[Mocan *et al.*, 2006] Adrian Mocan, Emilia Cimpian, and Mick Kerrigan. Formal model for ontology mapping creation. In *Proc. 5th International Semantic Web Conference (ISWC)*, volume 4273 of *Lecture notes in computer science*, pages 459–472, Athens (GA US), 2006.

[Modica *et al.*, 2001] Giovanni Modica, Avigdor Gal, and Hasan Jamil. The use of machine-generated ontologies in dynamic information seeking. In *Proc. 9th International Conference on Cooperative Information Systems (CoopIS)*, volume 2172 of *Lecture notes in computer science*, pages 433–448, Trento (IT), 2001.

[Nottelmann and Straccia, 2005] Henrik Nottelmann and Umberto Straccia. sPLMap: A probabilistic approach to schema matching. In *Proc. 27th European Conference on Information Retrieval Research (ECIR)*, pages 81–95, Santiago de Compostela (ES), 2005.

[Noy and Musen, 2000] Natalya Noy and Mark Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proc. 17th National Conference on Artificial Intelligence (AAAI)*, pages 450–455, Austin (TX US), 2000.

[Noy and Musen, 2001] Natalya Noy and Mark Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proc. IJCAI Workshop on Ontologies and Information Sharing*, pages 63–70, Seattle (WA US), 2001.

[Noy and Musen, 2002] Natalya Noy and Mark Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Proc. 18th National Conference on Artificial Intelligence (AAAI)*, pages 744–750, Edmonton (CA), 2002.

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

[Noy, 2004] Natalya Noy. Tools for mapping and merging ontologies. In Steffen Staab and Rudi Studer, editors, *Handbook on ontologies*, chapter 18, pages 365–384. Springer Verlag, Berlin (DE), 2004.

[Oberle *et al.*, 2004] Daniel Oberle, Raphael Volz, Steffen Staab, and Boris Motik. An extensible ontology software environment. In Steffen Staab and Rudi Studer, editors, *Handbook on ontologies*, chapter 15, pages 299–319. Springer Verlag, Berlin (DE), 2004.

[Palma and Haase, 2005] Raúl Palma and Peter Haase. Oyster: Sharing and re-using ontologies in a peer-to-peer community. In *Proc. 4th International Semantic Web Conference (ISWC)*, volume 3729 of *Lecture notes in computer science*, pages 1059–1062, Galway (IE), 2005.

[Palopoli *et al.*, 2003] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. DIKE: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Software–Practice and Experience*, 33(9):847–884, 2003.

[Scharffe and de Bruijn, 2005] François Scharffe and Jos de Bruijn. A language to specify mappings between ontologies. In *Proc. IEEE Conference on Internet-Based Systems (SITIS)*, Yaounde (CM), 2005.

[Scharffe, 2007] François Scharffe. Dynamerge: A merging algorithm for structured data integration on the web. In *Proc. DASFAA 2007 International Workshop on Scalable Web Information Integration and Service (SWIIS)*, Bangkok (TH), 2007.

[Serafini and Tamilin, 2005] Luciano Serafini and Andrei Tamilin. DRAGO: Distributed reasoning architecture for the semantic web. In *Proc. 2nd European Semantic Web Conference (ESWC)*, volume 3532 of *Lecture notes in computer science*, pages 361–376, Hersounisous (GR), May 2005.

[Shvaiko *et al.*, 2007] Pavel Shvaiko, Jérôme Euzenat, Heiner Stuckenschmidt, Malgorzata Mochol, Fausto Giunchiglia, Mikalai Yatskevich, Paolo Avesani, Willem Robert van Hage, Ondrej Svab, and Vojtech Svatek. Description of alignment evaluation and benchmarking results. deliverable 2.2.9, Knowledge web NoE, 2007.

[Sirin *et al.*, 2007] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: a practical OWL-DL reasoner. *Journal of Web Semantics*, 5, 2007. To appear.

[Straccia and Troncy, 2005] Umberto Straccia and Raphaël Troncy. oMAP: Combining classifiers for aligning automatically OWL ontologies. In *Proc. 6th International Conference on Web Information Systems Engineering (WISE)*, pages 133–147, New York (NY US), 2005.

[Stuckenschmidt *et al.*, 2005] Heiner Stuckenschmidt, Marc Ehrig, Jérôme Euzenat, Andreas Hess, Robert van Hage, Wei Hu, Ningsheng Jian, Gong Chen, Yuzhong Qu, George Stoilos, Giorgo Stamou, Umberto Straccia, Vojtech Svatek, Raphaël Troncy, Petko Valtchev, and Mikalai Yatskevich. Description of alignment implementation and benchmarking results. deliverable 2.2.4, Knowledge web NoE, 2005.

[Stumme and Mädche, 2001] Gerd Stumme and Alexander Mädche. FCA-Merge: Bottom-up merging of ontologies. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 225–234, Seattle (WA US), 2001.

[Tu and Yu, 2005] Kewei Tu and Yong Yu. CMC: Combining multiple schema-matching strategies based on credibility prediction. In *Proc. 10th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 3453 of *Lecture notes in computer science*, pages 888–893, Beijing (CN), 2005.

[Velegrakis *et al.*, 2003] Yannis Velegrakis, Renée Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *Proc. 29th International Conference on Very Large Data Bases (VLDB)*, pages 584–595, Berlin (DE), 2003.

[Waterfeld *et al.*, 2008] Walter Waterfeld, Moritz Weiten, and Peter Haase. Ontology management infrastrutures. In Martin Hepp, Pieter De Leenheer, Aldo De Moor, and York Sure, editors, *Ontology management: semantic web, semantic web services, and business applications*, chapter 3, pages 39–89. Springer, New-York (NY US), 2008.

[Zhdanova and Shvaiko, 2006] Anna Zhdanova and Pavel Shvaiko. Community-driven ontology matching. In *Proc. 3rd European Semantic Web Conference (ESWC)*, volume 4011 of *Lecture notes in computer science*, pages 34–49, Budva (ME), 2006.

[Zimmermann and Euzenat, 2006] Antoine Zimmermann and Jérôme Euzenat. Three semantics for distributed systems and their relations with alignment composition. In *Proc. 5th International Semantic Web Conference (ISWC)*, volume 4273 of *Lecture notes in computer science*, pages 16–29, Athens (GA US), 2006.

Related deliverables

A number of Knowledge web deliverables are clearly related to this one:

Project	Number	**Title** and relationship
KW	D2.2.1	**Specification of a common framework for characterizing alignment** provided the framework for us to define ontology matching and alignment processing.
KW	D2.2.3	**State of the art on ontology alignment** provides a panorama of many of the techniques that can be used for matching ontologies, they provided some input considered in this deliverable.
KW	D2.2.6	**Specification of the delivery alignment format** compares several alignment formats that can be used for expressing the alignments that will be processed and for sharing them.
KW	D2.2.10	**Expressive alignment language and implementation** proposes a join between the Abstract Mapping Language (or OMWG Mapping Language) and the Alignment format that is the most expressive candidate for expressing the alignments.
KW	D2.3.7	**Negotiation/argumentation techniques among agents complying to different ontologies** specifies part of the protocol used in the Alignment servers.