Ontology alignment
An ontology management perspective[*]

Jérôme Euzenat[†] Adrian Mocan[‡] François Scharffe[†]

September 30, 2007

Abstract

Relating ontologies is very important for many ontology-based applications and more important in open environments like the semantic web. The relations between ontology entities can be obtained by ontology matching and represented as alignments. Hence, alignments must be taken into account in ontology management. This chapter establishes the requirements for alignment management. After a brief introduction to matching and alignments, we justify the consideration of alignments as independent entities and provide the life cycle of alignments. We describe the important functions of editing, managing and exploiting alignments and illustrate them with existing components.
Keywords: ontology matching; ontology alignment; alignment management; alignment server; ontology mediation; mapping.

1 Relating ontologies: from ontology islands to continent

In many applications, ontologies are not used in isolation. This can be because several ontologies, representing different domains have to be used within the same application, e.g., an ontology of books with an ontology of shipping for an on-line bookstore, or because different ontologies are encountered dynamically, e.g., different ontologies from different on-line bookstores to choose from.

These ontologies must be related together for the ontology-based application to work properly. In the context of ontology management, these relations may be used for composing at design time the different ontology parts that will be used

[*]Slightly revised version of [Euzenat *et al.*, 2007].

[†]INRIA Rhône-Alpes & LIG, Monbonnot, France, `Jerome.Euzenat@inrialpes.fr`

[‡]Innsbruck Universität, Austria, `{adrian.mocan,francois.scharffe}@deri.org`

by the applications (either by merging these ontologies or by designing data integration mechanisms), for dealing with different versions of ontologies that may be found together at design time, or for anticipating the need for dynamically matching encountered ontologies at run time.

We call "ontology matching" the process of finding the relations between ontologies and we call "alignment" the result of this process expressing declaratively these relations. In an open world in which ontologies evolve, managing ontologies requires using alignments for expressing the relations between ontologies. We have defended elsewhere the idea that for that purpose the use of alignments is preferable to using directly mediators or transformations [Euzenat, 2005]. We go one step further here by proposing that ontology management involves alignment management.

In the remainder we first briefly present what ontology matching is and where it is used (§2). Then, we consider some requirements and functions for alignment management addressing the alignment life cycle (§3). Following this life cycle we present in more details how to address these requirements in what concerns alignment editing (§4), alignment storing and sharing (§5) and finally alignment processing (§6). We then consider existing systems that feature to some extent ontology management capabilities (§7).

2 Ontology matching and alignments

We present in deeper details what is meant by an alignment and provide some vocabulary as it will be used in this chapter (§2.1). Then we discuss the different applications that can take advantage of matching ontologies (§2.2). We identify some characteristics of these applications in terms of exploitation of the alignments. Finally, we provide an overview of the various matching techniques available (§2.3). Complete coverage of these issues can be found in [Euzenat and Shvaiko, 2007].

When we talk about ontologies, we include database schemas and other extensional descriptions of data which benefit from matching as well.

2.1 Alignments for expressing relations

The ontology matching problem may be described in one sentence: given two ontologies each describing a set of discrete entities (which can be classes, properties, rules, predicates, or even formulas), find the correspondences, e.g., equivalence or subsumption, holding between these entities. This set of correspondences is called an alignment.

Given two ontologies o and o', alignments are made of a set of correspondences (called mappings when the relation is oriented) between (simple or complex) entities belonging to o and o' respectively. A correspondence is described as a quadruple $\langle e, e', r, n \rangle$ such that:

e and e' are the entities, e.g., formulas, terms, classes, individuals, between which a relation is asserted by the correspondence.

r is the relation declared to hold between e and e' by the correspondence. This relation can be a simple set-theoretic relation (applied to entities seen as sets or their interpretation seen as sets), a fuzzy relation, a probabilistic distribution over a complete set of relations, a similarity measure, etc.

n is a degree of confidence associated with that correspondence (this degree does not refer to the relation r, it is rather a measure of the trust in the fact that the correspondence is appropriate – "I trust 70% the fact that the correspondence is correct, reliable, etc." – and can be compared with the certainty measures provided by meteorological agencies). The trust degree can be computed in many ways, including user feedback or log analysis.

So, the simplest kind of correspondence (level 0) is:

$$URI_1 = URI_2$$

while a more elaborate one could be:

$$employee(x, y, z) \Leftarrow_{.85} empno(x, w) \wedge name(w, concat(y, "", z))$$

The first one expresses the equivalence ($=$) of what is denoted by two URIs (with full confidence). These URI can be the denotations of classes, properties or instances. The second one is a Horn-clause expressing that if there exists a w such that $empno(x, w)$ – w's identifier is x – and $name(w, concat(y, "", z))$ – the name of w is the result of the concatenation of string y, "" and z – are true in one ontology then $employee(x, y, z)$ must be true in the other one (and the confidence is here quantified with a degree equal to .85). Of course, in this last example, functions and predicates can also be identified by URIs.

As can be observed from these two examples, alignments in themselves are not tied to a particular language. But in order to use complex alignments like the second one, systems must be able to understand the language in which formulas and relations are expressed. This is supported through the definition of a particular subtype of alignment.

Since everyone does not share the same terminology, we define below, according to [Euzenat and Shvaiko, 2007], the various terms used in this chapter:

alignment is the result of the matching task: it is a set of correspondences;

bridge axioms are formulas in an ontology language that expresses the relations as assertions on the related entities. They are used when merging ontologies.

correspondence is the relation holding (or supposed to hold according to a particular matching algorithm or individual) between two entities of different ontologies. These entities can be as different as classes, individuals, properties or formulas. Some authors use the term "mapping" or "mapping rule" that will not be used here;

matching is the task of comparing two ontologies and finding the relationships between them;

mediator a mediator is a software module [Wiederhold, 1992], providing interoperability between heterogeneous knowledge sources. In query applications, it is a dual pair of translations that transforms the query from one ontology to another and that translate the answer back.

merging ontologies consists of creating a new ontology out of two or more ontologies. Ontology merging first involves the definition of an alignment between the ontologies to be merged.

transformation is a program that transforms an ontology from one ontology expression language to another;

translation is a program that transforms formulas with regard to some ontology into formulas with regard to another ontology (translation can be implemented by a set of translation rules, an XSLT stylesheet or a more classical program).

2.2 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. See Chapter 5 of this book [De Leenheer and Mens, 2007].

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 about 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 area of ontology, 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 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.

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. They should be supported in manipulating alignments during the whole ontology life cycle (see Chapter 3 of this book [Waterfeld *et al.*, 2007]).

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.

2.3 Matching ontologies

The matching operation determines the alignment A' for a pair of ontologies o and o'. There are some other parameters that can extend the definition of the matching process, namely:

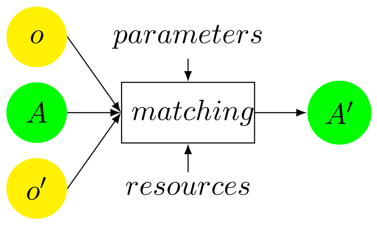

Figure 1: The ontology matching process: it establishes an alignment (A') from two ontologies (o and o') and optionally an input alignment (A), parameters and external resources.

(i) the use of an input alignment A, which is to be completed by the process;
(ii) the matching parameters, p, e.g., weights, thresholds; and
(iii) external resources used by the matching process, r, e.g., common knowledge or domain specific thesauri.

So, the matching process can be seen as a function f which, from a pair of ontologies o and o', an input alignment A, a set of parameters p and a set of resources r, returns an alignment A' between these ontologies:

$$A' = f(o, o', A, p, r)$$

There have already been many reviews of ontology matching algorithms [Rahm and Bernstein, 2001; Wache *et al.*, 2001; Kalfoglou and Schorlemmer, 2003; Euzenat and Shvaiko, 2007][1] so we will be brief and refer the reader to these presentations.

Ontology matching consists of generating an alignment from two (or more) ontologies. There are many different features of ontologies that are usually used for providing matching:

terminological techniques are based on the text found within ontologies for identifying ontology entities (labels), documenting them (comments) or other surrounding textual sources (related element labels). These techniques come from natural language processing and information retrieval. They can use the string structure themselves, e.g., string distances, the ontology as corpus, e.g., statistical measures based on the frequency of occurrence of a term, or external resources, such as dictionaries.

[1]In fact, the ontology matching builds on previous research done in databases and information integration [Genesereth *et al.*, 1997].

structural techniques are based on the relations between ontology entities. These can be relations between entities and their attributes, including constraints on their values, or relations with other entities. These techniques take advantage of type comparison techniques or more elaborate graph techniques, e.g., tree distances, path matching, graph matching.

extensional techniques compare the extension of entities. These extensions can be made of other entities, e.g., instances, as well as related resources, e.g., indexed documents. They differ depending on if the two ontologies share resources, e.g., they index the same set of documents, or not (in which case a similarity between the extensions may be established). These techniques can come from data analysis and statistics.

semantic techniques are based on the semantic definition of ontologies. They use extra formalised knowledge and theorem provers for finding consequences of a particular alignment. This can be used for expanding the alignment or, on the contrary, for detecting conflicting correspondences.

Of course, most of the systems combine several techniques in order to improve their results. The techniques can be combined by aggregating distance results [van Hage *et al.*, 2005], by using selection functions for choosing which one to use in the present case [Jian *et al.*, 2005; Tang *et al.*, 2006], or by deeply involving them all in global distance computation [Euzenat and Valtchev, 2004; Melnik *et al.*, 2002].

Moreover, there is a difference when training sets are available or not (this is most often useful when a matching algorithm is needed for recognising instances). When available, one can apply machine learning techniques such as Bayes learning, vector support machines or decision trees.

As a conclusion, many applications need ontology matching for many different purposes. Ontology matching can, in turn, be obtained by many different techniques that can be combined in many different ways. Currently, matching systems are not usable automatically on real scale ontologies. They loss in accuracy as the ontologies gain in size, complexity and heterogeneity. They are usable in particular contexts such as databases for which common identifiable data exists or evolutionary versions of ontologies. Consequently, matching systems are currently used interactively or semi-automatically so that users control and improve the quality of the result. In this context, the help of matching algorithms is as powerful as the ontologies grow in size and complexity.

Current scale of using such systems is not known otherwise than from their providers. However, some commercial systems are available, especially in the area of database and directory integration showing serious interest. A good way to approach the performances of matching algorithms is to follow the yearly Ontology Alignment Evaluation Initiative campaigns[2].

This difficulty of obtaining usable alignments calls for proper alignment management beside ontology management. We consider this in the next section.

3 Towards alignment management

We first identify why alignments should be considered in isolation (§3.1). We then present what should be an alignment life cycle from the standpoint of ontology management (§3.2) and elicit the requirements for supporting this life cycle (§3.3). Finally we describe a set of services and tools that can be provided for fulfilling these requirements (§3.4). The further sections will present in more details possible implementations of these services.

3.1 Why supporting alignments?

The reasons for supporting alignments have been provided in §2: many applications use them for different purposes using various matching algorithms combined in multiple ways.

As heterogeneous ontologies are a global problem for many applications, this calls for an infrastructure able to help these different applications to deal with it. In such a way, the effort of interoperating ontologies does not need to be solved for each kind of use.

Moreover, given the difficulty of the matching task, there are few algorithms available and when good alignments are available, they are worth sharing.

Supporting alignments has notable advantages over supporting other kind of matching results such as transformations, mediator implementations or merged ontologies. There are several reasons for this:

Sharing matching algorithms: Many different applications have matching needs. It is thus appropriate to share the solutions to these problems, the matching algorithms and systems, across applications.

Sharing alignments: Alignments are quite difficult to provide. 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.

[2]http://oaei.ontologymatching.org

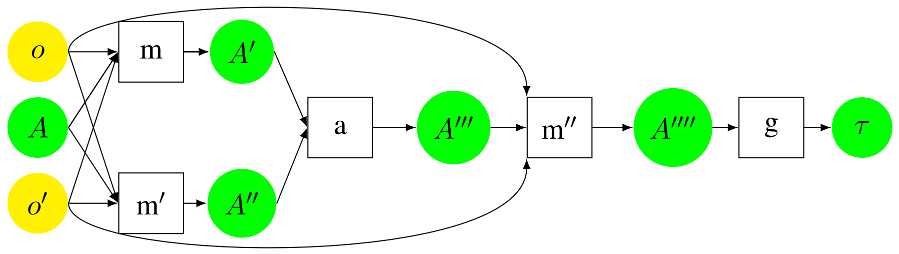

Figure 2: Alignment passing from tools to tools. Two matchers (m and m') are first run in parallel from the given ontologies, their resulting alignments are aggregated (a) resulting in another alignment which will be improved by another method (m'') before generating (g) a transformation program from it.

Sharing exploitation means: Matching results, once expressed as alignments, may be used for different purposes. Hence, a good matching algorithm does not have to be reimplemented for merging ontologies or for transforming new data: the same implementation will be reused together with mediator generators for exploiting the alignment in different mediation scenarios.

Combining matchers: If one wants to combine several matching systems in a particular application, this is easier if all the systems can exchange their results in a pivot language. This is illustrated in Figure 2.

So, considering ontology alignments as first class citizens, has several benefits:

– from a software engineering point of view, as alignments can be passed from a program to another.
– from an ontology engineering and management point of view, as they will evolve together with the ontology life cycle.

3.2 The alignment life cycle

Like ontologies, alignments have their own life cycle (see Figure 3). 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

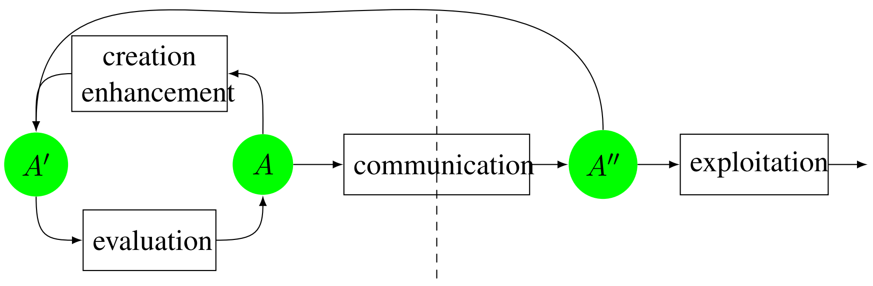

Figure 3: The ontology alignment life cycle.

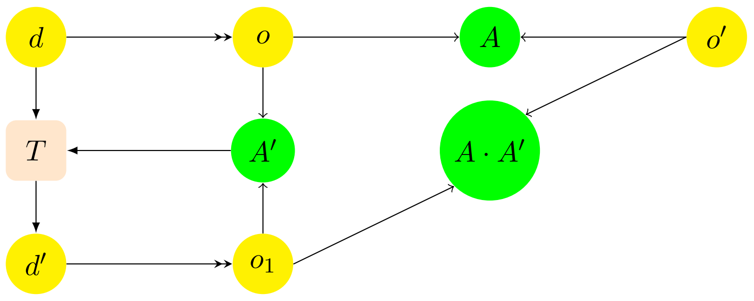

Figure 4: Evolution of alignments. When an ontology o evolves into a new version o_1, it is necessary to update the instances of this ontology (d) and the alignments (A) it has with other ontologies (o'). To that extent, a new alignment (A') between the two versions can be established and it can be used for generating the necessary instance transformation (T) and updated alignments ($A \cdot A'$).

worth publishing, then it can be stored and communicated to other parties interested in such an alignment. Finally, the alignment is transformed into another form or interpreted for performing actions like mediation or merging.

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 4).

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.

3.3 Requirements for alignment support

Ontology alignments, like ontologies, must be supported during their life cycle phases by adequate tools. These required functions can be implemented by services. The most notable services are:

Matching two ontologies possibly by specifying the algorithm to use and its parameters (including an initial alignment).

Storing an alignment in persistent storage.

Retrieving an alignment from its identifier.

Retrieving alignment metadata from its identifier can be used for choosing between specific alignments.

Suppressing an alignment from the current alignment pool.

Finding (stored) alignments between two specific ontologies.

Editing an alignment by adding or discarding correspondences (this is typically the result of a graphic editing session).

Trimming alignments over a threshold.

Generating code implementing ontology transformations, data translations or bridge axioms from a particular alignment.

Translating a message with regard to an alignment.

Finding a similar ontology is useful when one wants to align two ontologies through an intermediate one.

For instance, someone wanting to translate a message expressed in ontology o to ontology o'' can ask for matching the two ontologies and for a translation of the message with regard to the obtained alignment. A more extreme scenario involves (1) asking for alignments between o and o'', maybe resulting in no alignment, (2) asking for an ontology close to o'' which may result in ontology o', (3) asking for the alignments between o and o', which may return several alignments a, a' and a'', (4) asking for the metadata of these alignments and (5) choosing a' because it is certified by a trusted authority, (6) matching o' and o'' with a particular algorithm, (7) trimming the result over a reasonable threshold for this algorithm, (8) editing the results so that it seems correct, (9) storing it in the server for sharing it with other people, (10) retrieving alignment a' and this latter one as data translators, (11) finally applying these two translations in a row to the initial message.

Most of these services correspond to primitives provided by the Alignment API [Euzenat, 2004]. They require, in addition, several features extending traditional matching frameworks:

– The ability to store alignments, whether they are provided by automatic means or by hand;
– Their proper annotation 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 can be extended to the justifications for correspondences that can be used in agent argumentation);
– The ability to generate knowledge processors such as mediators, transformations, translators, rules as well as to apply these processors if necessary;
– The possibility to find similar ontologies and to contact other such services in order to ask them for operations that the current service cannot provide by itself.

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. This kind of information can however be stored together with the alignment in order for the client to be able to discriminate among them.

3.4 Example scenario: data mediation for semantic web services

The remainder of this chapter presents in more depth the functions of editing (§4), communicating (§5) and processing (§6) alignments. We will neither consider the alignment creation which has been the subject of much literature, nor the evaluation. Each of these functions will be illustrated through a common example related to 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.

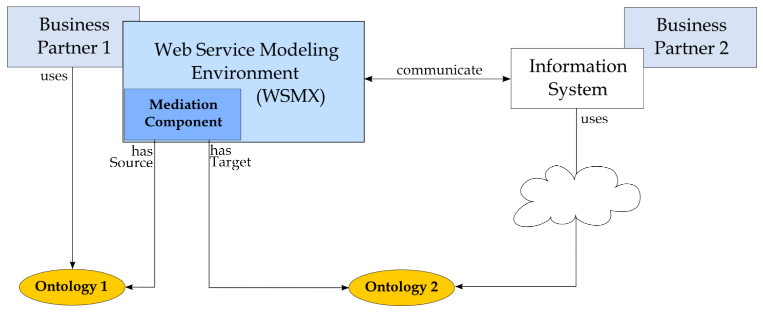

Figure 5: Instance transformation scenario.

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 5). 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. Also, due to the fact that 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 4 and Section 6; Section 5 will consider the management of the alignments between these two phases.

4 Design time alignment support

The first place where ontology heterogeneity can be found is while designing an application. Ontology management environments (see Chapter 3 of this book [Waterfeld *et al.*, 2007]) must support users in obtaining alignments and manipulating them. We provide some requirements for such an environment and detail further the Web Service Modeling Toolkit from this point of view.

4.1 Requirements

Design time alignment support requires first the ability to obtain an alignment between two ontologies. This can be achieved by retrieving an existing alignment, running a matching algorithm or creating an alignment manually.

Retrieving an alignment requires that alignments are stored and accessible somewhere. This can be done within the current ontology management environment, either from the local disk or from a remote server. If alignments are to be of good quality, it is preferable that the environment provides access to remote servers storing alignments. We will come back to this point in Section 6.

Running a matching algorithm requires the availability of such an algorithm. Having several such algorithms available in an ontology management environment seems highly desirable. Some tools provide support for finding the correspondences, like Protégé through the Prompt suite [Noy and Musen, 2003].

An often overlooked functionality of matching algorithms is their ability to provide explanation for the provided alignments. Explanations can be obtained by interacting with the matcher or by accessing metadata about a stored alignment. [Shvaiko *et al.*, 2005] explores the first alternative.

These alignments may also need to be manipulated. Most common manipulations involve trimming correspondences under a threshold or aggregating several alignments obtained on the same two ontologies.

Finally, creating an alignment manually requires an alignment editor. The same alignment editor can be used for manipulating more precisely the obtained alignments. They should provide a convenient display of the currently edited alignments and the opportunity to discard, modify or add correspondences. Ideally, from the alignment editor, all the design time functions should be available. Since ontologies and alignments can be very large, it is very challenging to offer intuitive alignment editing support.

The VisOn tool, developed by University of Montréal, is such a tool that can be used for editing alignments in the Alignment API format. Prompt also offers such facilities. Other tools developed for database schema matching could be adapted.

The Web Service Modeling Toolkit is an Integrated Development Environment (IDE) for Semantic Web services which also provides ontology engineering capabilities. Among other capabilities, WSMT offers a set of tools for creating, editing and storing ontology alignments. In the following section these WSMT features will be described in more details.

4.2 Example design-time tool: Web Service Modeling Toolkit

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: It 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.

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 6 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 assuring that all the problem-related entities have been explored.

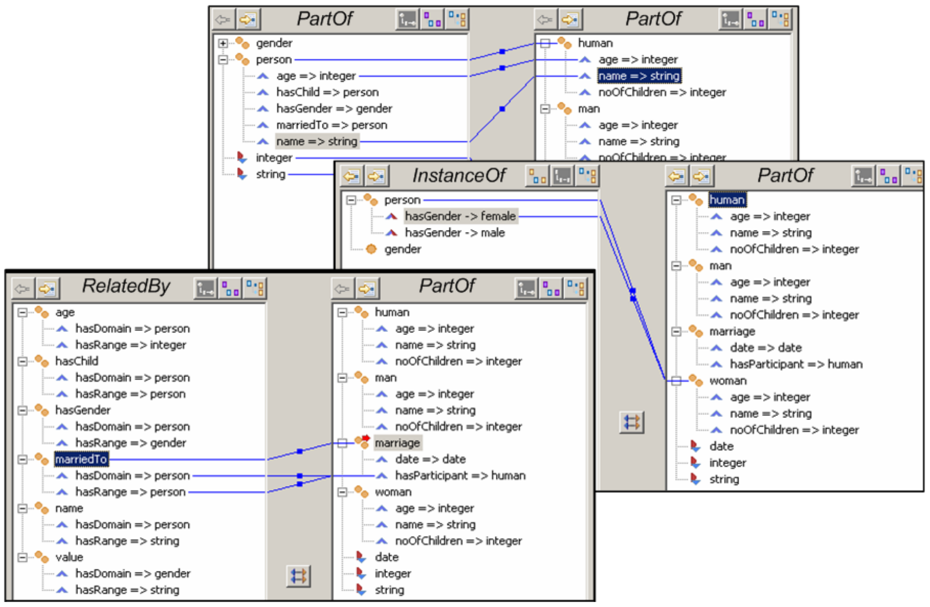

Figure 6: Mapping views in the AML View-Based Editor.

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.

5 Ontology alignment management and maintenance

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

Alignment support can be implemented either as a component of an ontology management tool and even being specific to each particular workstation (see Section 7). 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.

5.1 Alignment server for storing

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 Section 3 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 be able to exchange the alignments they found and select them on various criteria. This can 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.

5.2 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.

5.3 Evolving and maintaining ontology alignments

Like ontologies, alignments are not cast in stone once and for all. In particular, as ontologies evolve, it is necessary to evolve alignments accordingly. However, it can be quite hard for the engineer to be aware of the effects that these constant changes have. It is thus particularly important to provide support for alignment evolution and maintenance in alignment management environments.

Some tools, such as PrompDiff [Noy and Musen, 2003], are already particularly good at finding alignments between versions of ontologies. When such an alignment is made available, it is possible, as displayed in Figure 4, to provide by composition new versions of the alignment tied to the previous version and to migrate data.

WSMT offers a MUnit Testing View for the Abstract Mapping Language which gives the engineer support to ensure that instances are being correctly transformed. Users can define pairs of sources and targets, specifying that the result of transforming the sources, using the existing alignments, should be the targets. These tests can then be incrementally run by engineers when alignment validation is required.

6 Alignment processing

Finally, once alignments are obtained, either using a graphical tool, as the output of a matching algorithm, or retrieved from an alignment store, they can be processed in concrete mediation scenarios.

The following techniques all require an alignment between the source and target ontologies in order to be achieved.

Query rewriting: a query addressed to a source ontology needs to be rewritten in terms of a query for a target ontology.

Instance transformation: a set of instances described under a source ontology needs to be transformed into terms of a target ontology.

Ontology merging: a set of source ontologies need to be merged into a one ontology.

The scenario determines the operation that must be processed: a web service data mediator, as the one presented in Section 3.4, requires transformation of instances, while on-line catalog integration may require query rewriting in order to query the various catalogs.

When applying instance transformation or query rewriting, the resulting sets of instances may contain duplicates. For example, two similar products sold by different vendors. In the case of ontology merging, it might also be necessary to merge instances described by the merged ontologies. Again, duplicates have to be identified in order to avoid their duplication in the newly created ontology.

The technique of merging similar instances is known as instance identification and unification.

We describe these techniques in detail in the remaining of this section. Their application often requires preprocessing of the alignment in order to make it executable for the mediation system. Section 6.3 presents how alignments are transformed between various formats, motivating the use of a common alignment format for exchange between applications, algorithms and tools.

6.1 Query rewriting and instance transformation

Applying query rewriting techniques consists, as the name suggests, of rewriting a query in terms of a source ontology o into terms of a target ontology o'. The rewriting engine takes as input the original query q, the alignment between o and o', and returns a query q' in terms of o'. Figure 7 illustrates this process. 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 transformed to instances of o in order to be further processed by the system.

Instance transformation is done by taking a set of instances described under a source ontology o, and transforming it to instances of a target ontology o' using the alignment between the two ontologies. 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 7, is used in the example scenario in Section 3.4.

The two former techniques result in two sets of instances described according to a single 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 separated). Instance unification techniques are used to merge similar instances by analyzing their attributes values, as well as the relations they share with other instances.

Instance unification is also necessary after two ontologies have been merged into one. Instances of the source ontologies then also need to be merged, and duplicates removed. The next section presents the ontology merging technique.

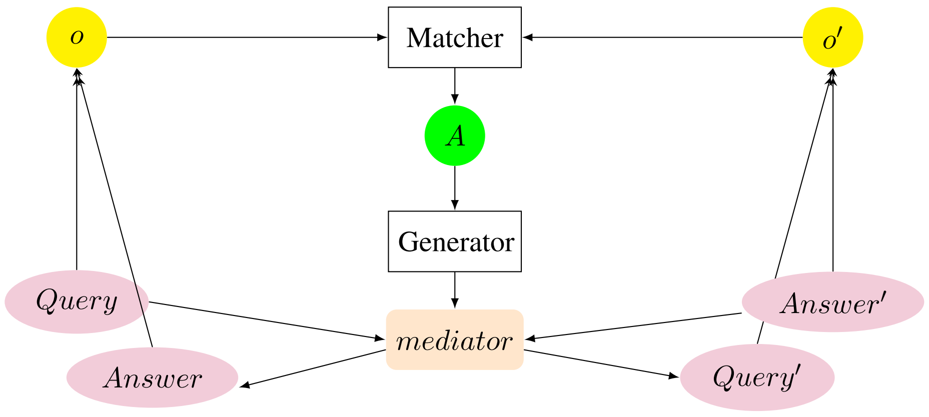

Figure 7: Query mediation (from [Euzenat and Shvaiko, 2007]). From two matched ontologies o and o', resulting in alignment A, a $mediator$ is generated. This allows the transformation of queries expressed with the entities of the first ontology into a query using the corresponding entities of a matched ontology and the translation back of the results from the second ontology to the first one.

6.2 Merging

There are cases where 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 catalog 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. The ontology merging process can be fully automatised if an adequate alignment is provided [Scharffe, 2007], but usually requires human intervention in order to solve conflicts and choose a merging strategy. Figure 8 illustrates the ontology merging process.

The techniques presented in the previous two subsections require only the alignment as an input (they interpret it). As we will see in the next section, this alignment may require a further step in order to be usable. This step is tightly linked to the format in which the alignment is expressed.

6.3 Semantic data mediation

The mediation of the heterogeneous semantic data can be achieved through instance transformation. Data represented by ontology instances has to be transformed either by the sender or transparently by a third party in the format required

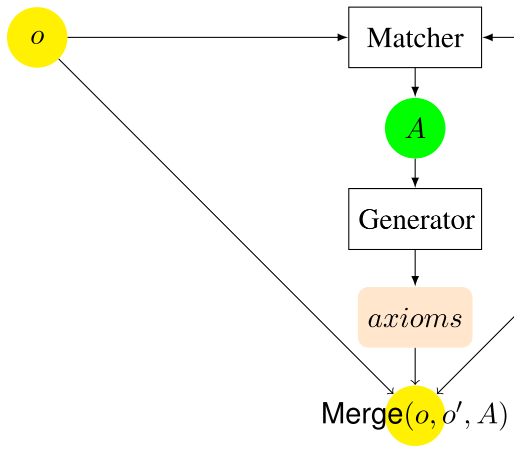

Figure 8: Ontology merging (from [Euzenat and Shvaiko, 2007]). From two matched ontologies o and o', resulting in alignment A, articulation $axioms$ are generated. This allows the creation of a new ontology covering the matched ontologies.

by the receiver, i.e., instances expressed in the target ontology.

In order to accommodate such a mediation scenario, the alignments generated by using the techniques described in Section 4 have to be processed by an engine able to perform instance transformation. If the alignments are expressed in an abstract form, e.g., using AML, 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 9 shows how such an instance transformation engine (the Data Mediation Run-Time Component in WSMX) can be deployed and used in various scenarios. A straightforward way is to integrate it in an Information System (in this case WSMX) which needs mediation 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 transformed in target in-

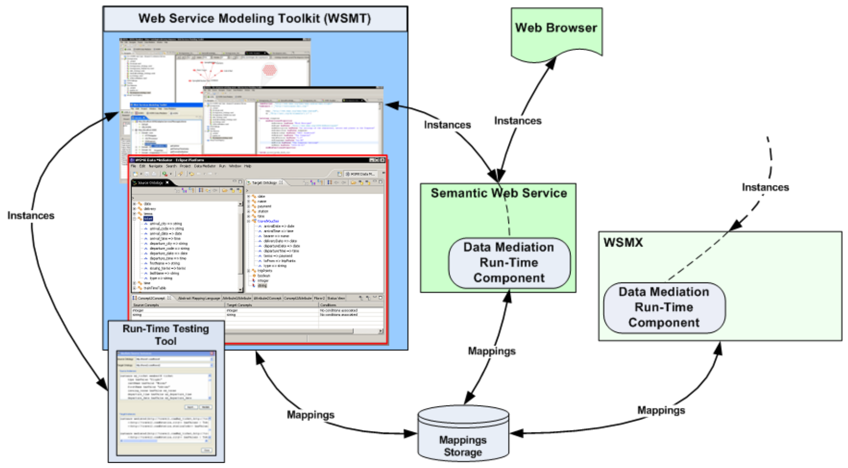

Figure 9: Run-time Data Mediator Usage Scenario (from [Mocan and Cimpian, 2007]).

stances.

7 Software and tools

Most of the work on general organisation of alignments is tied to some kind of application, e.g., C-OWL for peer-to-peer applications, WSMX for web services, Edutella for emerging semantics. There are, however, a few systems which are autonomous enough for being used as independent alignment management support.

Model management 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. Rondo[3] is such a system [Melnik *et al.*, 2003]. It offers operators for generating the alignments, composing them and applying them as data transformation. It is currently a standalone program with no editing functions.

MAFRA[4] [Mädche *et al.*, 2002] proposes an architecture for dealing with "semantic bridges" that offers many functions such as creating, manipulating, storing and processing such bridges. MAFRA has transformations associated with bridges: it does not record alignments in a non processable format. MAFRA does not offer editing or sharing alignments.

[3]http://infolab.stanford.edu/ modman/rondo/
[4]http://mafra-toolkit.sourceforge.net

Protégé is an ontology edition environment (see Chapter 3 of this book [Waterfeld *et al.*, 2007]) that offers design time support for matching. In particular it features Prompt[5] [Noy and Musen, 2003], an environment that provides some matching methods and alignment visualisation. Since alignments are expressed in an ontology, they can be stored and shared through the Protégé server mode. Prompt can be extended through a plug-in mechanism.

Foam[6] [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 Protégé plug in and is integrated in the KAON2 ontology management environment. COMA++ is another standalone (schema) matching workbench that allows integrating and composing matching algorithms. It supports matching, evaluating, editing, storing and processing alignments.

The Alignment Server, associated with the Alignment API[7] [Euzenat, 2004], offers matching ontologies, manipulating, storing and sharing alignments as well as processor generation. It can be accessed by clients through API, web services, agent communication languages ot HTTP. It does not support editing.

WSMT[8], which has been taken as example within these pages is a design time alignment creator and editor. It manipulates the AML format and can generate WSML rules. It also works as a standalone system.

The NeOn[9] project ambitions to produce a toolkit for ontology management which features run time and design time ontology alignment support.

8 Further readings

The topic of alignment management is relatively new so there is no specifically dedicated publications. A recent extensive reference on ontology matching is [Euzenat and Shvaiko, 2007]. ontologymatching.org is a web site collecting information about ontology matching.

9 Conclusions

Applications using ontologies face the problem of ontology heterogeneity whenever they want to communicate with each others or evolve. Hence, ontology man-

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

[6]http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/

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

[8]http://wsmt.sourceforge.net

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

agement must take ontology heterogeneity into account. Dealing with ontology heterogeneity involves finding the alignments, or sets of correspondences, existing between ontology entities and using them for reconciling the ontologies.

Because, this problem occurs in many applications and is solved in many different ways, it is better dealt with in a general way. This involves managing alignments together with ontologies.

We have presented alignment management through the life cycle of alignments and the associated support functions: creating, selecting, editing, maintaining, sharing and processing alignments. We have presented a few systems which implement part of this alignment support and in particular the notion of alignment server which can be used for storing and sharing alignment at both run time and design time.

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. Another challenge is the easy finding of available alignments. 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.

10 Acknowledgements

This work has been partly supported by the European network of excellence Knowledge Web (IST-2004-507482). The first author has also been supported by the European integrated project NeOn (IST-2005-027595) and the RNTL project WebContent.

The first author thanks Pavel Shvaiko for many fruitful discussions related to this chapter.

References

[De Leenheer and Mens, 2007] 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), 2007.

[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, 2007] Marc Ehrig. *Ontology alignment: bridging the semantic gap*. Springer, New-York (NY US), 2007.

[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.*, 2007] Jérôme Euzenat, Adrian Mocan, and Fraçois Scarffe. 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), 2007.

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference*, 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.

[Genesereth *et al.*, 1997] Michael Genesereth, Arthur Keller, and Oliver Duschka. Infomaster: An information integration system. In *Proc. International Conference on Management of Data (SIGMOD) Prototype Demonstration*, pages 539–542, Tucson (AZ US), 1997.

[Jian *et al.*, 2005] Ningsheng Jian, Wei Hu, Gong Cheng, and Yuzhong Qu. Falcon-AO: Aligning ontologies with Falcon. In *Proc. K-CAP Workshop on Integrating Ontologies*, pages 87–93, Banff (CA), 2005.

[Kalfoglou and Schorlemmer, 2003] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[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.

[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.

[Mädche *et al.*, 2002] Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 235–250, Siguenza (ES), 2002.

[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.*, 2003] Sergey Melnik, Erhard Rahm, and Philip Bernstein. Rondo: A programming platform for model management. In *Proc. ACM SIGMOD*, pages 193–204, San Diego (CA US), 2003.

[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.

[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.

[Palma and Haase, 2005] Raúl Palma and Peter Haase. Oyster: Sharing and reusing 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.

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

[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.

[Shvaiko *et al.*, 2005] Pavel Shvaiko, Fausto Giunchiglia, Paulo Pinheiro da Silva, and Deborah McGuinness. Web explanations for semantic heterogeneity discovery. In *Proc. 2nd European Semantic Web Conference (ESWC)*, volume 3532 of *Lecture notes in computer science*, pages 303–317, Hersounisous (GR), May 2005.

[Tang *et al.*, 2006] Jie Tang, Juanzi Li, Bangyong Liang, Xiaotong Huang, Yi Li, and Kehong Wang. Using Bayesian decision for ontology mapping. *Journal of Web Semantics*, 4(1):243–262, 2006.

[van Hage *et al.*, 2005] Willem Robert van Hage, Sophia Katrenko, and Guus Schreiber. A method to combine linguistic ontology-mapping techniques. In *Proc. 4th International Semantic Web Conference (ISWC)*, volume 3729 of *Lecture notes in computer science*, pages 732–744, Galway (IE), 2005.

[Wache *et al.*, 2001] Holger Wache, Thomas Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner. Ontology-based integration of information – a survey of existing approaches. In *Proc. IJCAI Workshop on Ontologies and Information Sharing*, pages 108–117, Seattle (WA US), 2001.

[Waterfeld *et al.*, 2007] 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), 2007.

[Wiederhold, 1992] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.