



NeOn-project.org

NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — “Semantic-based knowledge and content systems”

D3.3.2: Matching ontologies for context: The NeOn Alignment plug-in

Deliverable Co-ordinator: Chan Le Duc

Deliverable Co-ordinating Institution: INRIA

Other Authors: Mathieu d’Aquin (OU), Jesus Barrasa (UPM), Jérôme David (INRIA), Jérôme Euzenat (INRIA), Raul Palma (UPM), Rosario Plaza (UPM), Marta Sabou (OU), Boris Villazón-Terrazas (UPM)

This deliverable presents the software support provided by the NeOn toolkit for matching ontologies, and in particular, recontextualise them. This support comes through the NeOn Alignment plug-in which integrate the Alignment API and offer access to Alignment servers in the NeOn toolkit. We present the NeOn Alignment plug-in as well as several enhancement of the Alignment server: the integration of three matching methods developed within NeOn, i.e., Semantic Mapper, OLA and Scarlet, as well as the connection of the Alignment servers with Oyster.

Document Identifier:	NEON/2008/D3.3.2/v1.0	Date due:	February 29th, 2008
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	February 29th, 2008
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of the NeOn research project funded by the IST Programme of the Commission of the European Communities by the grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	Universität Karlsruhe – TH (UKARL) Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB Englerstrasse 11 D-76128 Karlsruhe, Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	Software AG (SAG) Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Jesús Contreras E-mail address: jcontreras@isoco.com	Institut 'Jožef Stefan' (JSI) Jamova 39 SL-1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier, France Contact person: Jérôme Euzenat E-mail address: jerome.euzenat@inrialpes.fr	University of Sheffield (USFD) Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield, United Kingdom Contact person: Hamish Cunningham E-mail address: hamish@dc.shef.ac.uk
Universität Koblenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	Consiglio Nazionale delle Ricerche (CNR) Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 00100 Rome Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org
Atos Origin S.A. (ATOS) Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parietelobo@atosorigin.com	Laboratorios KIN, S.A. (KIN) C/Ciudad de Granada, 123 08018 Barcelona Spain Contact person: Antonio López E-mail address: alopez@kin.es

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 the writing of this document or its parts:

- Jožef Stefan Institute
- Open university
- iSoCo
- Universität Karlsruhe
- INRIA
- Ontoprise
- CNR
- Universidad Politécnica de Madrid

Change Log

Version	Date	Author	Changes
0.1	15.11.2007	Chan Le Duc	initial version
0.2	10.12.2007	Jérôme Euzenat	structuring
0.3	19.12.2007	Chan Le Duc	first description of plug-in
0.4	30.01.2008	Chan Le Duc	Integrated comments
0.5	02.02.2008	Chan Le Duc	Integrated UPM description
0.6	15.02.2008	Chan Le Duc	Described Scarlet and OLA connection
0.7	19.02.2008	Jérôme Euzenat	Added the part about Oyster connection
0.8	29.02.2008	Chan Le Duc	Added missing parts
0.9	21.03.2008	Chan Le Duc	Accounted for quality control comments
1.0	15.04.2008	Jérôme Euzenat	Accounted for quality assessment comments

Executive Summary

In the NeOn project, context can be expressed within a network of ontologies through relationships between ontologies. Such relationships would provide the missing axioms that specify the meaning of knowledge further and help considering them in their context.

These relations must be established from the initial resources which will be linked to contextual resources. This can be achieved through the use of ontology matching techniques. In order to help networked ontology designers to contextualise ontologies, we integrate within the NeOn toolkit a plug-in for finding alignments between ontologies and importing them within the network of ontologies.

The present deliverable aims at presenting the NeOn Alignment plug-in which can be used either offline, when users want to compute alignments between two ontologies, or online, when they want to obtain the alignment from an Alignment server (presented in D3.3.1). In addition, it is possible to import the resulting alignments under the form of a merged OWL ontology. The NeOn Alignment plug-in offers access to the Alignment server functions, i.e.:

- Finding alignments available in the servers;
- Computing and representing alignments;
- Piping alignments algorithms (for improving an existing alignment);
- Manipulating (trimming and hardening) and combining (merging, composing) alignments;
- Generating “mediators” (transformations, axioms, rules in format such as XSLT, SWRL, OWL, C-OWL, WSMML).

In this case it can take advantage of extra matching methods available from the server.

So, in addition, the deliverable features several developments that have been made during the past year and that are made available through the NeOn Alignment plug-in:

- integrating the OWL-Lite Aligner (OLA) within the Alignment server;
- integrating the SemanticMapper matcher within the Alignment server;
- integrating the Scarlet matcher within the Alignment server (see Deliverable D3.3.1);
- organising the cooperation between Oyster and the Alignment server for sharing alignments;

Contents

1	Context through alignments within networked ontologies	8
1.1	Contextualising ontologies through matching	8
1.2	Support for matching ontologies	9
1.3	The NeOn Alignment plug-in and Alignment server extensions	10
1.4	Outline of the deliverable	11
2	The NeOn Alignment plug-in	12
2.1	Principles	12
2.2	Implementation	12
2.3	How to use the NeOn Alignment plug-in	13
2.4	Stepwise examples	14
3	Integration of OLA	19
3.1	Presentation	19
3.2	Specific requirements	19
3.3	Integration	19
3.4	Results	20
4	Integration of the Semantic mapper	24
4.1	Presentation	24
4.2	Specific requirements	24
4.3	Integration	25
4.4	Results	25
5	Integration of Scarlet	28
5.1	Presentation	28
5.2	Specific requirements	28
5.3	Integration	28
5.4	Results	28
6	Connection with Oyster	29
6.1	Presentation	29
6.2	Specific requirements	29
6.3	Integration	29
6.4	Results	30
7	Conclusions	31

7.1	Current state of the prototypes	31
7.2	Further developments	31
A	Installing the NeOn Alignment plug-in	32
A.1	Get the NeOn toolkit	32
A.2	Get and install the NeOn Alignment plug-in	32
A.3	Run the NeOn toolkit with the plug-in	32
A.4	Use the plug-in with the Alignment server	32
B	Installing and extending the Alignment server	34
B.1	Get and install the Alignment server	34
B.2	Install WordNet	34
B.3	Connect to the Oyster system	34
B.4	Add new matching algorithms	35
B.5	Launching the alignment server	35
C	Semantic mapper	36
C.1	Introduction	36
C.2	Automatic mapping discovery	37
	Bibliography	58

Chapter 1

Context through alignments within networked ontologies

In this chapter, we recast the intuition presented in deliverables D3.1.1 and D3.3.1 in the context of networked ontologies. In particular we explain in what sense context are networks of ontologies (§1.1), what are the support developed within NeOn for supporting ontology matching and alignment sharing (§1.2) and how this integrates within the NeOn architecture (§1.3).

1.1 Contextualising ontologies through matching

Domain ontologies are designed to be applied in a particular context and use terms in a sense that is relevant to this domain, e.g., *Ontology* in computer science, and which may not be related to similar concepts in other domains. They do not fully specify concepts because part of their specification is implicit from the context. We will use here the word “constraint” for this specification in a broad sense: any axiom constrains the meaning of the terms it uses.

NeOn has in its core the ambitious scenario that ontologies are developed in an open environment in a distributed fashion. This means that ontologies will be used in settings that are not those that have led to their design. In order to use them properly, it is necessary to be able to identify this context so that appropriate actions can be taken: either not using them or adapting them to the new context.

The context of some knowledge or ontology is given by additional knowledge in which or in the perspective of which this knowledge has been elaborated. This knowledge can vary in nature and expression. For instance, in work package 2, the context of elaboration of ontologies is expressed as argumentative structures about ontology design rationale. In work package 3 [Haase *et al.*, 2006], the context of ontologies is prominently placed in the framework of networked ontologies: the context of an ontology is given by the network of other ontologies with which it is related.

From this definition, a pair of dual operations can be associated with context (see Figure 1.1): contextualisation and decontextualisation. Contextualisation or recontextualisation is the action of finding the relations of an ontology with other ontologies which express its context. Decontextualisation, as the opposite, extracts one particular ontology from its context. These operation can be combined, for instance, if someone wants to transfer one ontology from a domain to another one, by first decontextualising it and recontextualising it to another domain.

As can be considered from this brief description, contextualising an ontology is a matter of matching it to other ontologies which will provide its context. For that purpose ontology matching technologies can be used. So we would like to provide support for contextualising ontologies through ontology matching. Moreover, recontextualising ontologies can be succesfully used for helping the process of matching an ontology into another one. Thus, we also aim at taking into account the contextualisation operation for matching.

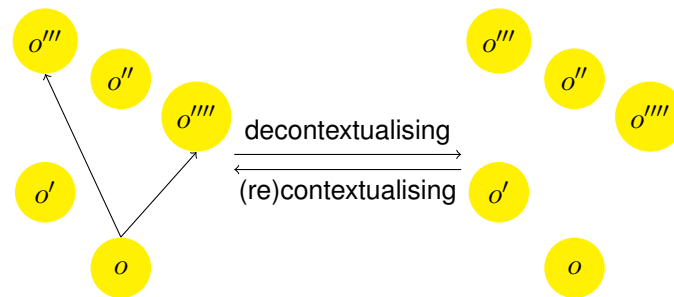


Figure 1.1: Contextualising/Decontextualising ontologies in the framework of networked ontologies.

1.2 Support for matching ontologies

In Deliverable D3.3.1, we have considered expressing context with two main constructions from the NeOn model for networked ontologies: Ontologies and mappings (or alignments). The NeOn model already offers the metamodel for these two kinds of entities. The Alignment API can be considered as an implementation of this metamodel.

The Alignment API [Euzenat, 2004] has been designed to help developing applications based on alignments. It has been developed in the aim of manipulating a standard alignment format for sharing among matching systems, but it provides the features required for sharing them more widely. The API is a JAVA description of tools for accessing alignments in the format presented above.

The Alignment API can be used in conjunction with an ontology language API (the OWL-API is currently available, but other instantiation could be based on totally different languages). This implementation offers the following services:

- Computing and representing alignments;
- Piping alignments algorithms (for improving an existing alignment);
- Manipulating (trimming and hardening) and combining (merging, composing) alignments;
- Generating “mediators” (transformations, axioms, rules in format such as XSLT, SWRL, OWL, C-OWL, WSML);
- Comparing alignments (like computing precision and recall or a symmetric distance with regard to a particular reference alignment).

The API also provides the ability to compose matching algorithms and manipulating alignments through programming. The API can be used for producing transformations, rules or bridge axioms independently from the algorithm that produced the alignment. Since its definition, several matching systems have been developed within this API (OLA, oMap) and more of them are able to generate its format (FOAM, Prompt, Falcon, etc.).

Within task 3.3 of NeOn we have developed an Alignment sever (see Deliverable 3.3.1) which allows sharing alignments. The Alignment server is built around the Alignment API. The server architecture is made of three layers (shown in Figure 1.2):

A storage system that allows persistent storage and retrieval of alignments. It implements only basic storage and runtime memory caching functions. The storage is made through a DBMS interface and can be replaced by any database management system as soon as it is supported by jdbc.

A protocol manager which handles the server protocol. It accepts the queries from plug-in interfaces and uses the server resources for answering them. It uses the storage system for caching results.

Protocol plug-ins which accept incoming queries in a particular communication system and invoke the protocol manager in order to satisfy them. These plug-ins are ideally stateless and only translator for the external queries.

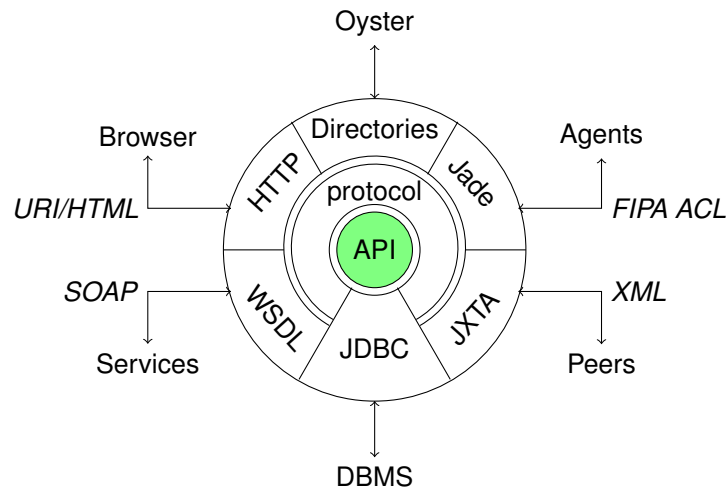


Figure 1.2: The Alignment server is built on the Alignment API that is seated on top of a relational database repository for alignment and is wrapped around a simple protocol. Each access method is a plug-in that interacts with the server through the protocol. Currently, HTML, agent and web service plug-in are available.

Currently, 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.

Services that are provided by the Alignment server are:

- storing alignments, whether they are provided by automatic means or by hand;
- storing annotations in order for the clients to evaluate them and to decide to use one of them or to start from it (this starts with the information about the matching algorithms, the justifications for correspondences that can be used in agent argumentation, as well as properties of the alignment);
- producing alignments on the fly through various algorithms that can be extended and parametrised;
- manipulating alignments by inverting them, applying thresholds;
- generating knowledge processors such as mediators, transformations, translators, rules as well as to process these processors if necessary.

There is no constraint that the alignments are computed online 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. For applications, the Alignment server can be available:

at design time through invocation by design and engineering environments: It can be integrated within the development environment.

at run time through the web service access of the server (or any other available plug-in).

1.3 The NeOn Alignment plug-in and Alignment server extensions

The NeOn Alignment plug-in provides design time support for alignment management from the NeOn toolkit. It is a NeOn toolkit plug-in that takes advantage of both the Alignment API and Alignment servers.

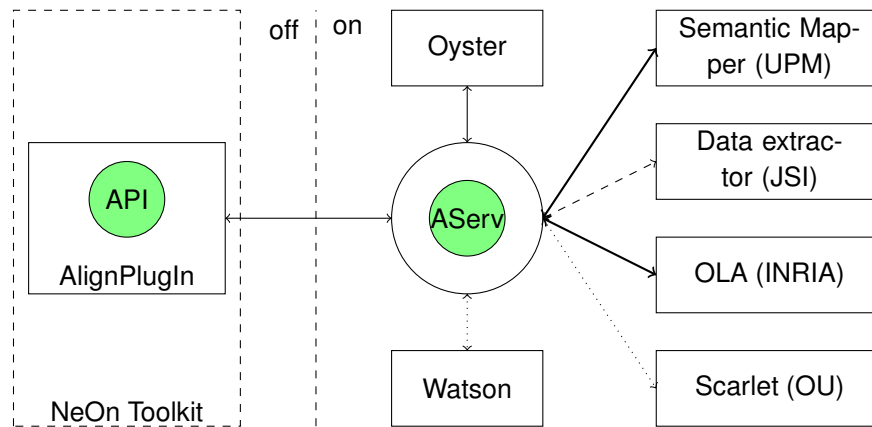


Figure 1.3: The integration of the Alignment server within the NeOn architecture (solid line corresponds to connection presented in this deliverable, dashed lines to a connection to be made in D3.3.3 and dotted lines to future connections).

Figure 1.3 presents the design-time support provided in NeOn in relation with Alignment servers. The NeOn Alignment plug-in can work in stand alone mode thanks to its embedding of the Alignment API or in connection with an Alignment server. In the latter case, it will benefit from new matchers integrated within the Alignment server and alignment sharing.

Hence the goal of the work presented in this deliverable is to (1) offer design time alignment manipulation support within the NeOn toolkit through the Alignment API and (2) offer access to Alignment servers from the NeOn toolkit.

1.4 Outline of the deliverable

The present deliverable is a prototype report that describes the work that has been carried out for integrating matching, contextualising and decontextualising to the NeOn toolkit. This work is presented through the development of an Alignment plug-in integrated the NeOn toolkit (Chapter 2). Three algorithms dedicated to matching networked ontologies developed within NeOn have been integrated to the Alignment API and server so that they are available to the NeOn Alignment plug-in. These are OLA (Chapter 3), the Semantic Mapper (Chapter 4) and Scarlet (Chapter 5). Finally, the Alignment server has also been integrated with another infrastructure tool developed within NeOn: Oyster for publicising the availability of alignments (Chapter 6).

This work takes advantage of the previous work in work packages 1 and 3. It shows a practical instantiation of the NeOn networked ontology model using the ontology and alignment (a.k.a. mapping) components. This will allow us to consider OWL ontologies and alignments as networked ontologies and to offer software for dealing with contexts in the framework of networked ontologies.

Chapter 2

The NeOn Alignment plug-in

The NeOn Alignment plug-in aims to bring the functions implemented in the Alignment server to the NeOn toolkit. From the NeOn toolkit environment in which the NeOn Alignment plug-in is integrated, users can exploit the Alignment server for computing and managing alignments between ontologies. An architecture and features of the Alignment server were introduced in the Deliverable 3.3.1.

In this chapter, we begin by presenting working principles and the functions of the NeOn Alignment plug-in. Then, we illustrate the usage of the NeOn Alignment plug-in through a step-by-step example.

2.1 Principles

The NeOn Alignment plug-in is designed so that it is able to work in two modes, namely *off-line* and *on-line*, according to the availability of resources:

- In the off-line mode, the Alignment API is integrated within the plug-in, and thus, allows the plug-in to perform all functions implemented in this API. For instance, the plug-in in the offline mode is able to match local ontologies which are fed by the NeOn toolkit environment or to trim an alignment resulting from an ontology matching.
- In the on-line mode, the NeOn alignment plug-in provides functions related to managing and accessing alignments on a server, in addition to the functions offered by the off-line mode. In order to do so, the plug-in opens connections with an Alignment server for either computing an alignment, finding an existing alignment or submitting an alignment it has computed.

The NeOn Alignment plug-in in the current state offers all main functions of the Alignment server developed in INRIA : ontology matching and alignment trimming, finding, storing. The alignment retrieve function is performed through exporting an alignment to the Ontology Navigator as an OWL Ontology. It remains two functions of the server to be integrated into the NeOn toolkit : alignment invert and load. This task will be carried out in a future version of the NeOn Alignment plug-in.

2.2 Implementation

Communication between the NeOn Alignment plug-in and the Alignment server relies on the web service Interface of the server. Queries from the plug-in and results from the server are expressed as SOAP messages.

The NeOn toolkit itself is a plug-in in Eclipse environment and can be extended thanks to plug-in mechanism. Through its modular design, the NeOn toolkit can be enriched with self-developed modules and be customized to the user's personal needs. The integration of the NeOn Alignment plug-in into the NeOn toolkit is based on these features.

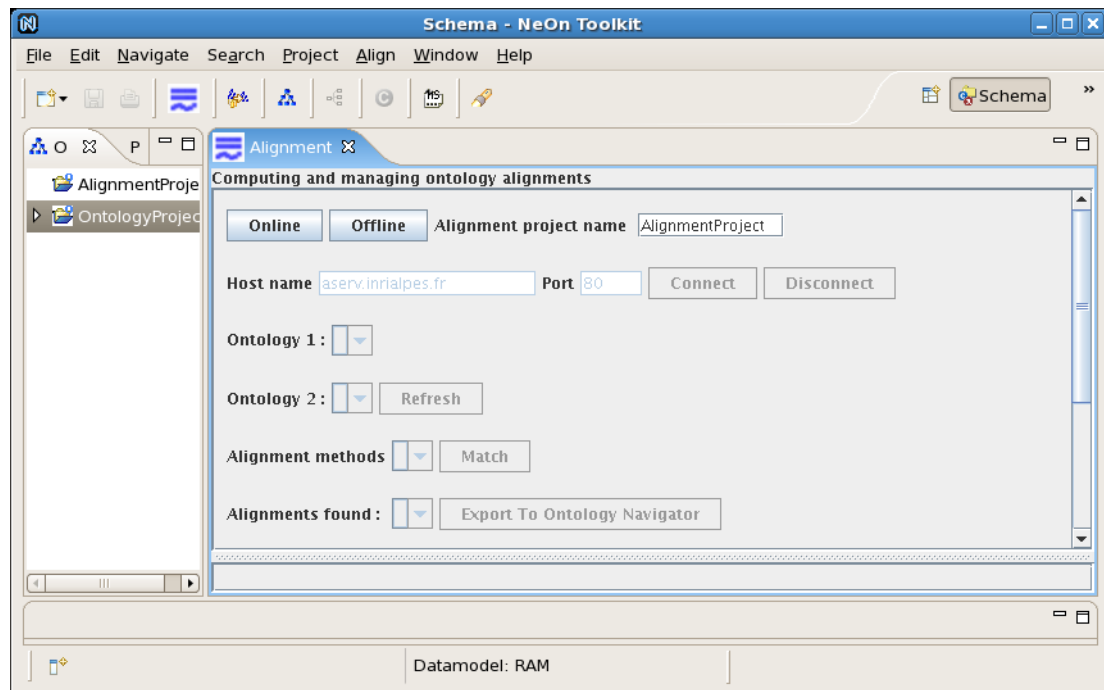


Figure 2.1: Two working modes in the NeOn Alignment plug-in.

More precisely, the NeOn Alignment plug-in can get the working ontologies from the Ontology Navigator of the NeOn toolkit and call functions from an Alignment server with these ontologies as input. If the result of a function is an alignment, the plug-in can display it directly as a set of correspondences or export it to the Ontology Navigator of NeON Toolkit as an OWL ontology.

2.3 How to use the NeOn Alignment plug-in

As a plug-in integrated into the NeOn toolkit, the NeOn Alignment plug-in can access to data models implemented in the NeOn toolkit and manipulate them. More precisely, the NeOn Alignment plug-in can get URIs of opened ontologies from the Ontology Navigator in the NeOn toolkit. Since alignments provided by matchers must be able to refer to stable and identifiable ontologies, we require that these ontologies be identifiable outside of the NeOn toolkit and thus have a URI.

Additionally, the NeOn Alignment plug-in is able to add an alignment as an OWL ontology to the Ontology Navigator. This may provide a way to exploit and share alignments, which are computed by the NeOn Alignment plug-in, among different plug-ins in the NeOn toolkit environment.

The NeOn Alignment plug-in can work in two modes : offline and online.

Offline mode

The offline mode is the default mode and can be activated by clicking on button "Offline". In this mode, the NeOn Alignment plug-in can access to NeOn toolkit ontologies (i.e. opened ontologies in the Ontology Navigator) and match any pair of them (with locally available methods). Resulting alignments are stored as local system files and exported to the Ontology Navigator as OWL ontologies (or to some Alignment server when in online mode).

Online mode

The online mode is activated by clicking on the "Online" button and providing the address of an Alignment server. In this mode, the NeOn Alignment plug-in provides all functions from the Alignment server. Resulting alignments are stored on the server and exported to the Ontology Navigator as OWL ontologies. This allows NeOnToolkit users, with help of the Ontology Editor, to use, share or edit alignments.

2.4 Stepwise examples

2.4.1 Starting a session in off-line mode

1. *Show the Ontology Navigator view.* From the menu bar of the NeOn toolkit, perform "Window -> Show view -> Ontostudio-> Ontology Navigator", and create a project, for example "OntologyProject", in this view.
2. *Open working OWL ontologies.* From the menu bar of the NeOn toolkit, open working OWL ontologies in the project "OntologyProject" created above by "File -> Import... -> OntoStudio -> FileSystem Import Wizard".
3. *Activate the NeOn Alignment plug-in.* From the menu bar or the button, activate the NeOn Alignment plug-in and a view for the plug-in will be opened.
4. *Define an alignment project.* From the view for the plug-in, give a name in the field "Alignment Project Name". This name will be used for an Ontology Navigator project which includes all alignments created by the NeOn Alignment plug-in.

2.4.2 Matching ontologies

1. *Fetch the ontologies* in the current workspace from the Ontology Navigator by clicking on the button "Refresh". The available ontologies will be added to two lists "Ontology 1" and "Ontology 2".
2. *Choose two ontologies* to match from the two lists.
3. *Choose an alignment method* from the list "Alignment methods".
4. *Click on "Match"* to match these two ontologies with the method chosen (see Figure 2.2).
5. The resulting alignment will be added to the list "Local alignments" (or "Alignment found").

2.4.3 Manipulating alignments

1. *Exporting an alignment.* Choose an alignment from one of the lists "Local alignments" or "Alignment server" (or "Alignments found") and click on "Export". The alignment chosen will be exported as an OWL ontology to the alignment project which was created in the Ontology Navigator. The result is presented in Figure 2.3.
2. *Trim an alignment.* Choose an alignment from one of the lists "Local alignments" or "Alignment server" (or "Alignments found"), then, define a threshold (see Figure 2.7). A new alignment created by the trim function will be added to the list "Local alignments" (or "Alignments found").
3. *Finding an alignment for two ontologies.* Choose two ontologies from two lists and click on the button "Find an alignment for ontologies". The found alignments are visualized in one of the lists "Local alignments" or "Alignment server" (or "Alignments found").

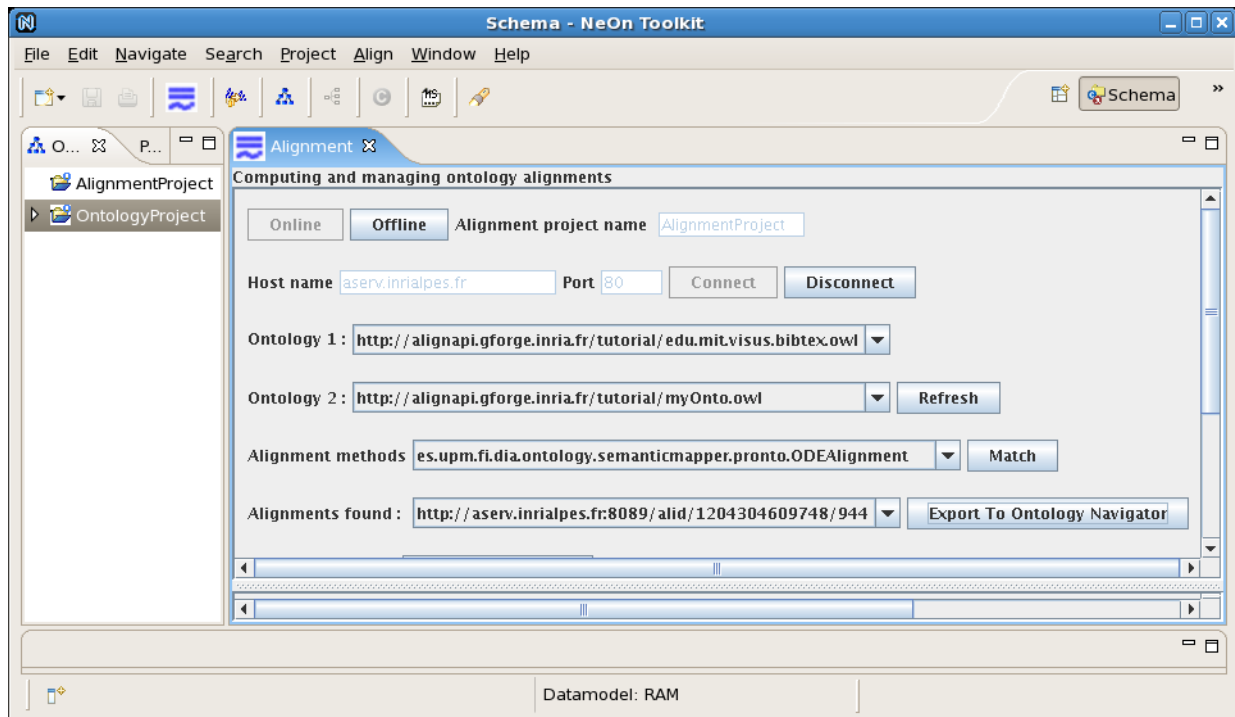


Figure 2.2: Matching two ontologies.

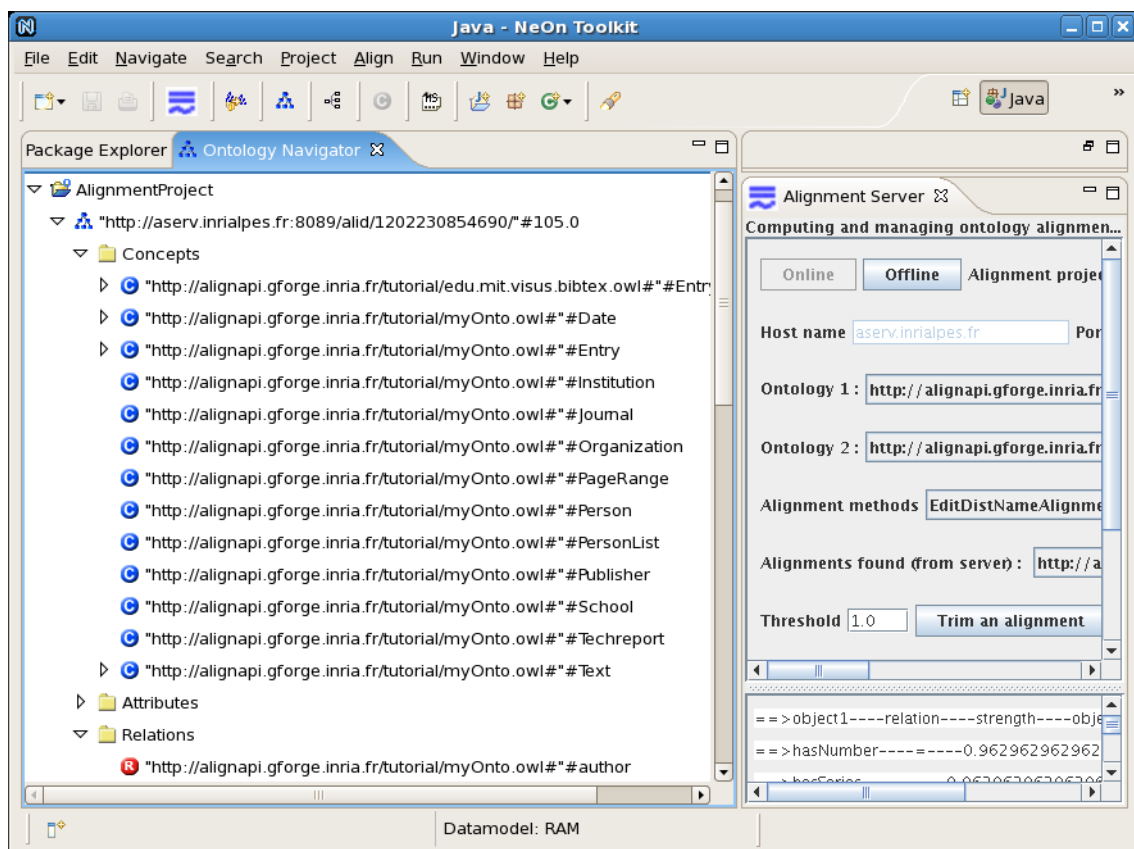


Figure 2.3: Exporting an alignment as an OWL ontology.

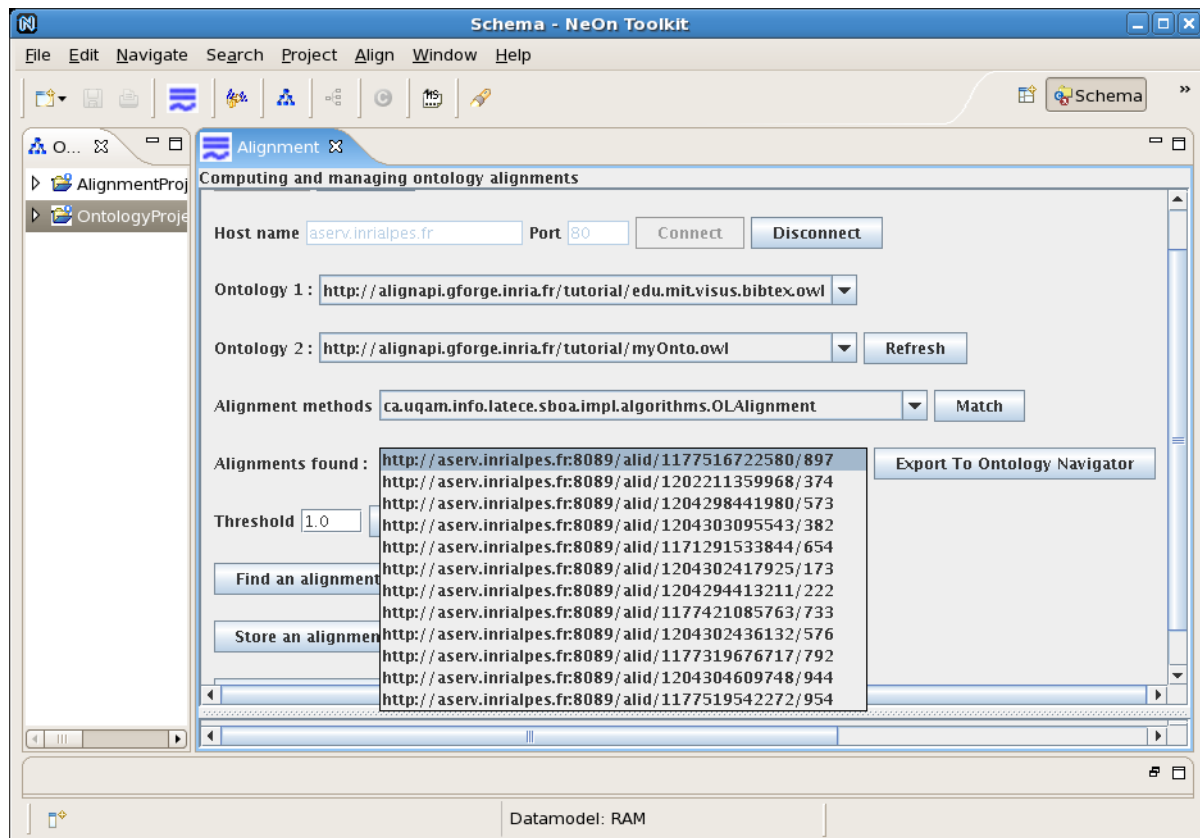


Figure 2.4: Listing all alignments available on the Alignment server.

4. *Store an alignment.* Choose an alignment to store from the list and click on "choose".
5. *Fetch available alignments.* This function allows users to obtain a list of all alignments available on the Alignment server.

2.4.4 Activating the on-line mode

1. *Activate the on-line mode.* From the view for the plug-in, activate the on-line mode by clicking on the button "Online" and the button "Connect" becomes enabled.
2. *Connection* To connect to the INRIA's Alignment server from the NeOn Alignment plug-in, you must enter "aserv.inrialpes.fr" for hostname and "80" for port. (see Figure 2.5)

If the connection is successful, we can see the buttons "Refresh", "Match", "Export to Ontology Navigator", "Trim", "Find an alignment...", "Store an alignment..." and "Fetch available alignments..." enabled. In particular, a list of available alignment methods is visible at "Alignment methods". In addition, an Ontology Navigator project, whose name was defined above, is automatically created for alignments.

2.4.5 Reverting to off-line mode

1. Perform steps 1-4 as described in Section 2.4.1, and then activate the offline mode by clicking on the button "Offline". After activating the offline mode, the buttons "Refresh", "Match", "Import to Ontology Navigator" and "Trim" become enabled.

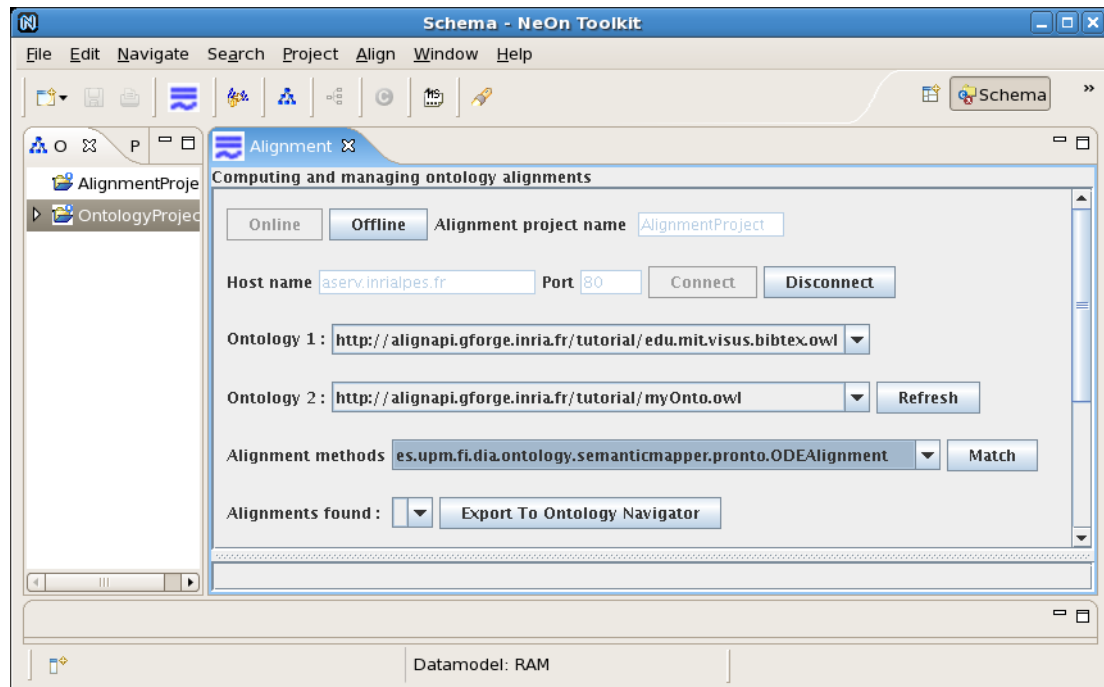


Figure 2.5: Connection to the Alignment server.

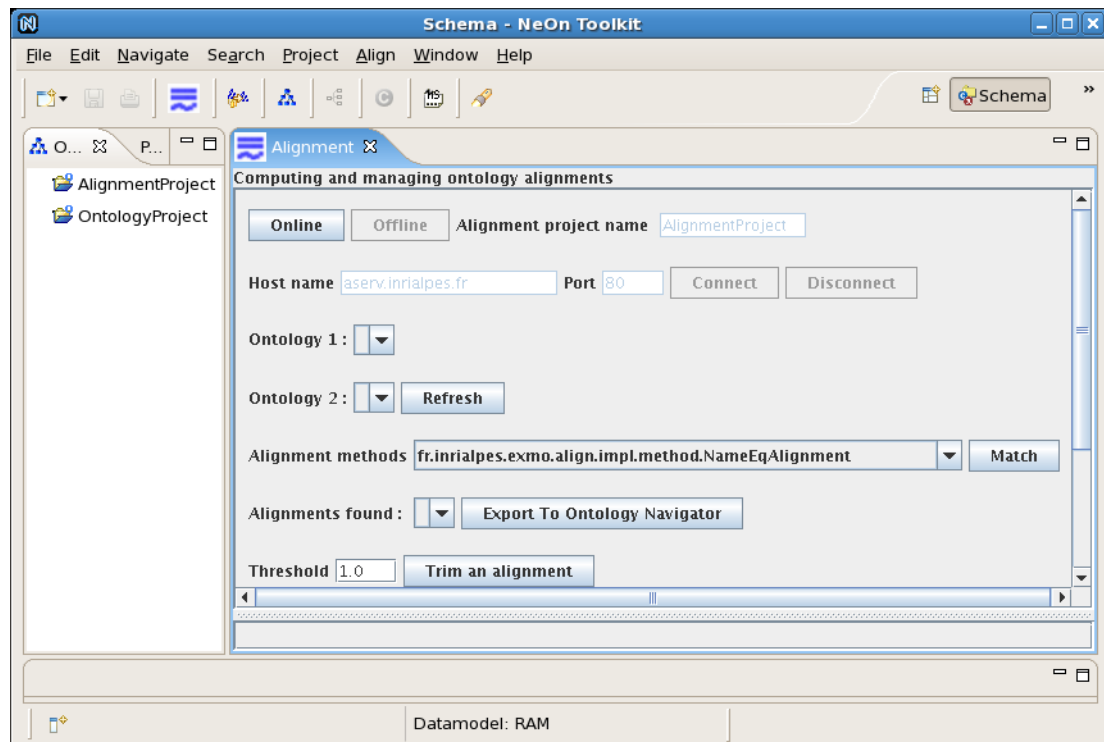


Figure 2.6: Functions in the offline mode.

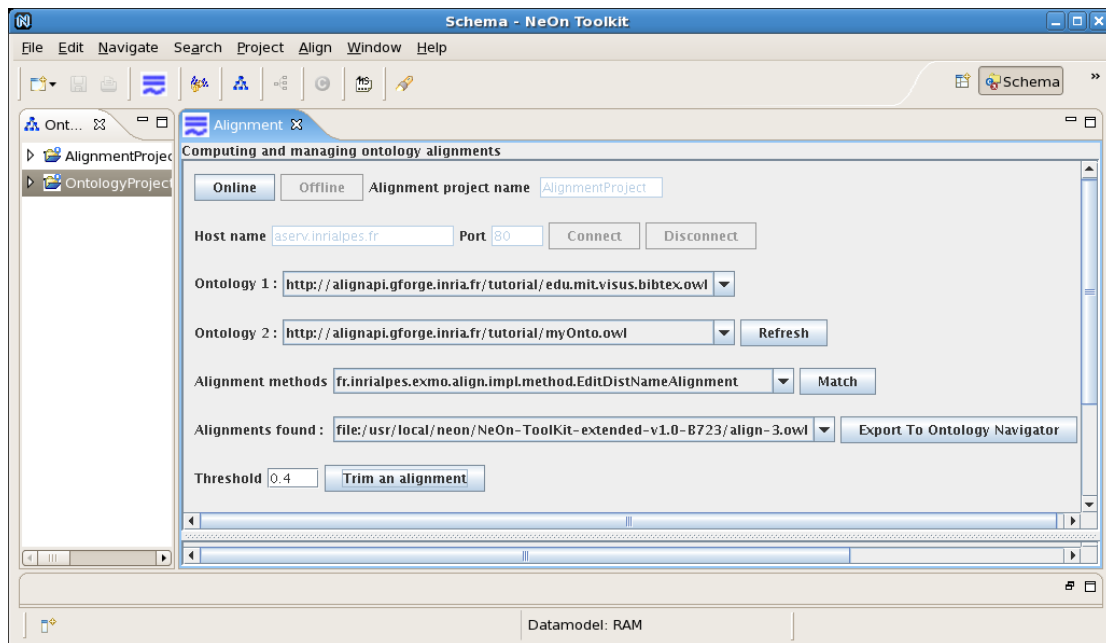


Figure 2.7: Trimming function in the offline-mode.

2. In the offline mode, the functions : matching two ontologies, exporting and trim an alignment, can be performed as described in Section 2.4.2 and 2.4.3 . For instance, the trim function is illustrated in Figure 2.7.

Chapter 3

Integration of OLA

3.1 Presentation

OLA is a matching algorithm for ontologies written in OWL. OLA relies on a universal measure for comparing the entities of two ontologies that combines in a homogeneous way all the knowledge used in entity descriptions: it deals with external data types, internal structure of classes as given by their properties and constraints, external structure of classes as given by their relationships to other classes and the availability of individuals. This is an improvement over other methods that usually take advantage of only a subpart of the language features.

The proposed method does not only compose linearly individual methods for assessing the similarity between entities, it uses an integrated similarity definition that makes them interact during computation [Euzenat and Valtchev, 2004]. OLA is based on the definition of a distance between entities of two ontologies as a system of equations that has to be solved in order to extract an alignment. One-to-many relationships and circularity in entity descriptions constitute the key difficulties in this context: These are respectively dealt with through local matching of entity sets and iterative computation of recursively dependent similarities which produces subsequent approximations of the target solution. So doing, OLA copes with the unavoidable circularities that occur within ontologies.

These equations are parameterized by a number of weights corresponding to the respective importance of different components of ontologies. We introduced a preprocessing step which considers the ontologies to match and evaluate the availability of the corresponding features in order to choose the corresponding weights [Euzenat *et al.*, 2005].

The integrated version of OLA, OLA₂, is a full reimplementations of the initial system based on matrix computation. OLA had remarkable results at the last OAEI evaluation [Euzenat *et al.*, 2007]: it was among the best participants in the benchmark tests and was the best participant in the directory test.

3.2 Specific requirements

OLA may require WordNet to work properly. However, it uses the WordNet support provided with the Alignment API so beside installing WordNet, this is not a particularly specific requirement.

OLA can take from several seconds to several hours to process depending on the size of the ontologies, so the asynchronous call that has been developed for Scarlet is most welcomed for using OLA.

3.3 Integration

The integration has been quite natural since OLA fits from the beginning within the Alignment API framework. It suffices to have OLA libraries within those of the server to use OLA.

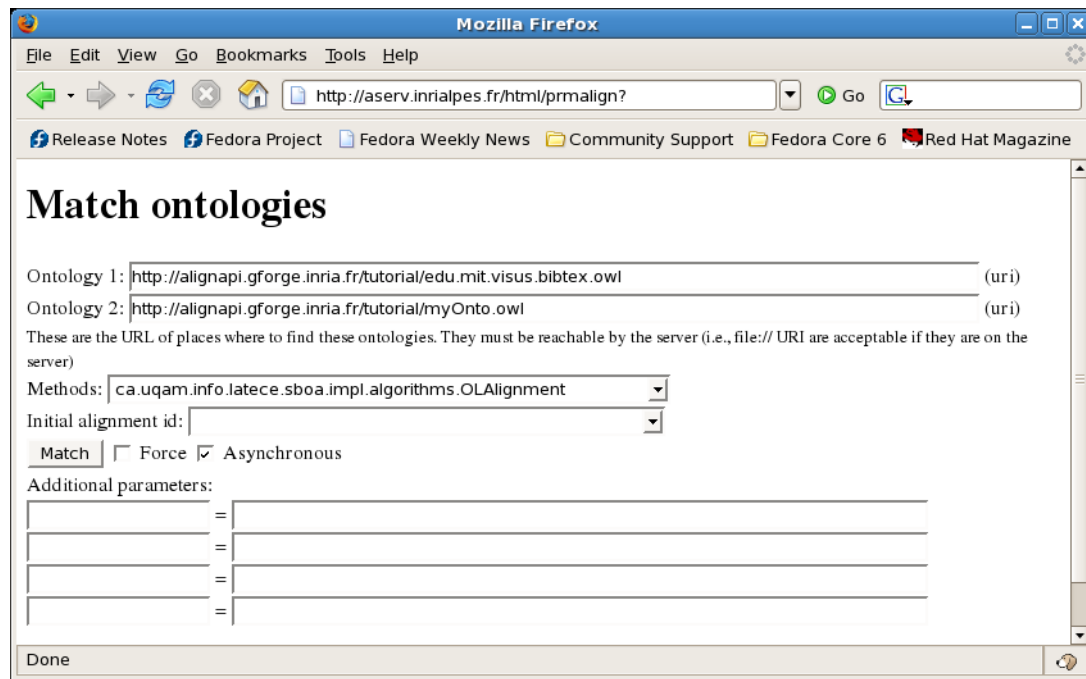
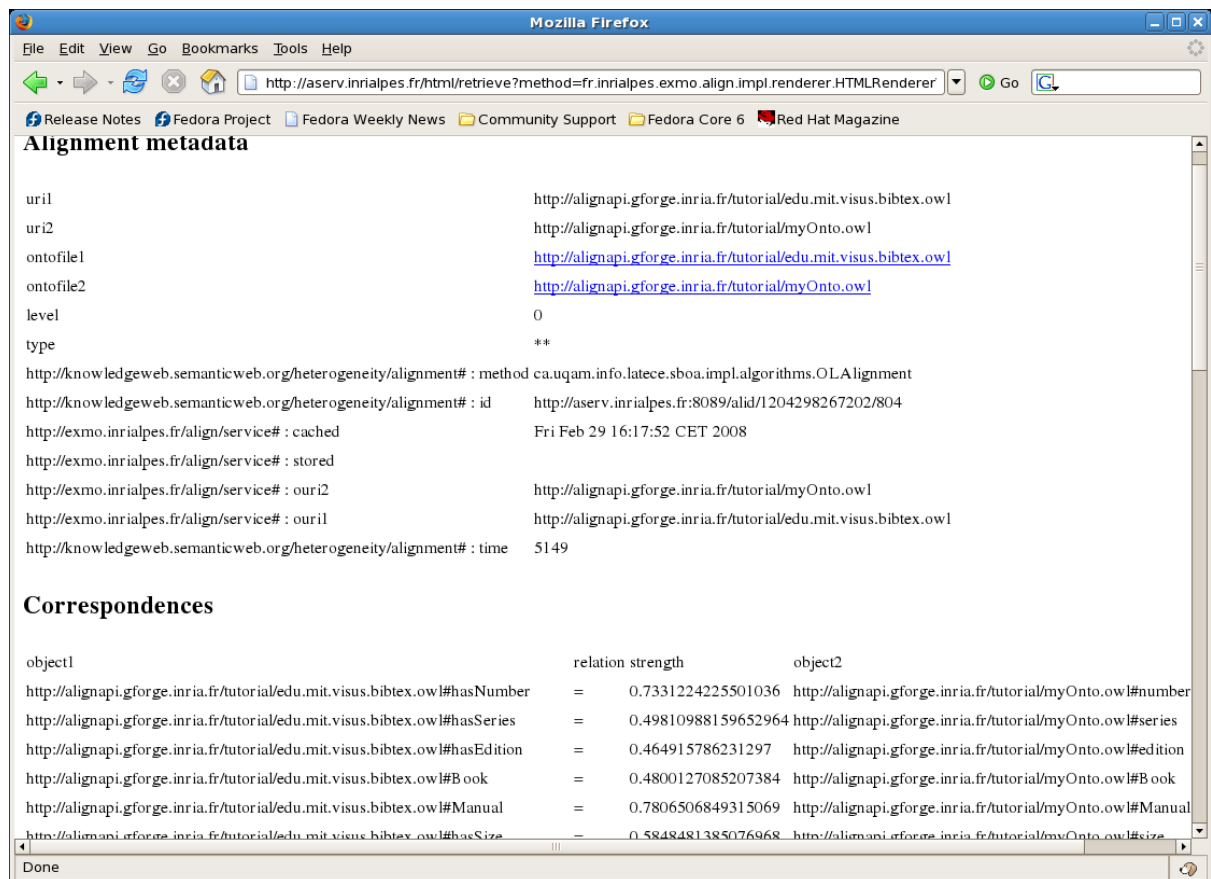


Figure 3.1: OLA match call from the HTML interface.

3.4 Results

OLA is visible under the HTML interface (Figure 3.1) as well as the NeOn Alignment plug-in (Figure 3.3) and can be used for matching ontologies (Figure 3.2 and 3.4).



Alignment metadata

uri1	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
uri2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
ontofile1	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
ontofile2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
level	0
type	**
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : method ca.uqam.info.latece.sboa.impl.algorithms.OLAAlignment	
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : id	http://aserv.inrialpes.fr:8089/alid/1204298267202/804
http://exmo.inrialpes.fr/align/service# : cached	Fri Feb 29 16:17:52 CET 2008
http://exmo.inrialpes.fr/align/service# : stored	
http://exmo.inrialpes.fr/align/service# : ouri2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
http://exmo.inrialpes.fr/align/service# : ouri1	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : time	5149

Correspondences

object1	relation strength	object2
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	= 0.7331224225501036	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#number
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasSeries	= 0.49810988159652964	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#series
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasEdition	= 0.464915786231297	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#edition
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#Book	= 0.4800127085207384	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#Book
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#Manual	= 0.7806506849315069	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#Manual
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasSize	= 0.5848481385076968	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#size

Done

Figure 3.2: OLA result from the HTML interface.

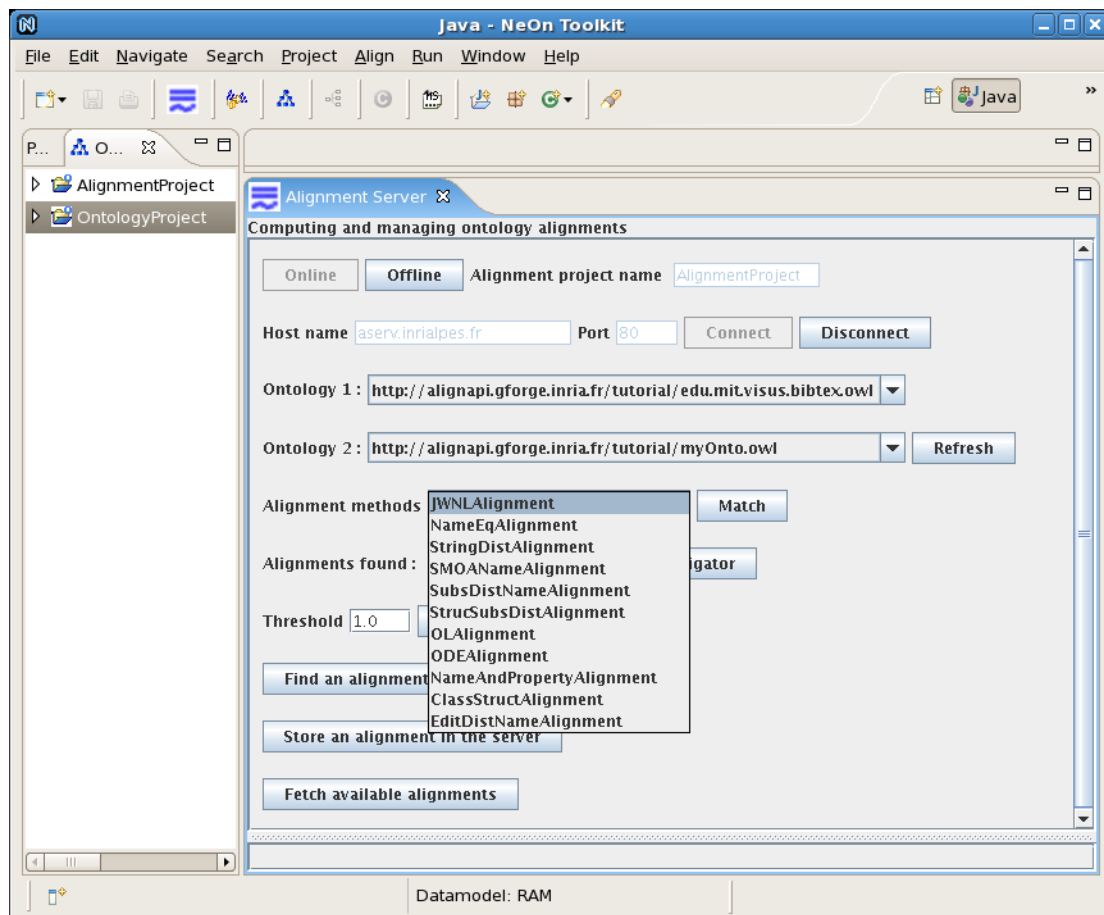


Figure 3.3: OLA match call from the NeOn toolkit.

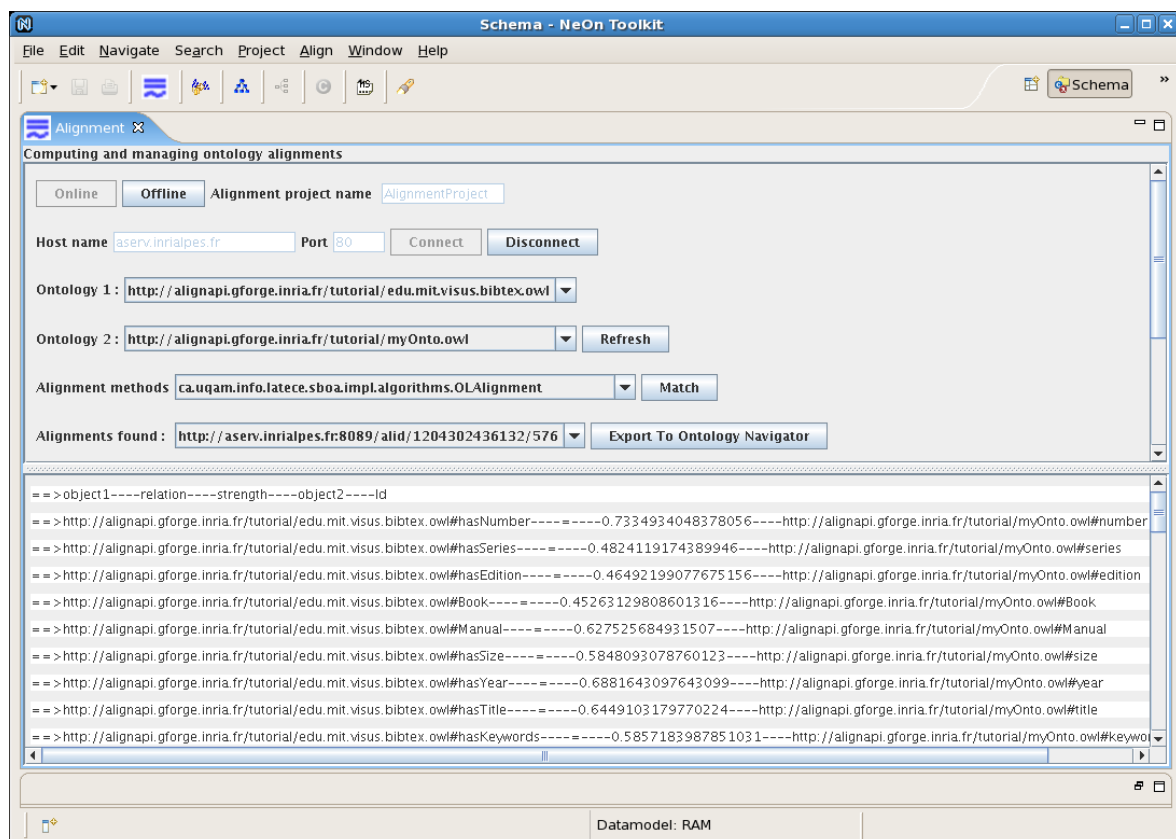


Figure 3.4: OLA results from the NeOn toolkit.

Chapter 4

Integration of the Semantic mapper

4.1 Presentation

The Semantic Mapper is an algorithm capable of discovering mappings between ontology elements and relational model elements. The Semantic Mapper generates automatically concept specifications, attributes and relations of a conceptual model, an ontology, according to the elements of a relational model. Although this process is automatic, the results will not be completed or directly exploitable; on the other hand, because of its heuristic nature, it should be supervised and validated by any user with a deep knowledge of the semantics underlying the models. The Semantic Mapper has two main components: a linguistic component and a structural one.

- The **linguistic component** detects lexical relations between the terms used to identify the elements of both models so as to infer semantic relations from them. This is, in fact, a comparison between elements, though they are taken in isolation, regardless of the context in which they occur and of their relation with other elements of the same model. To detect this type of relations we will use a series of linguistic resources (glossaries, thesauri, lexical databases, etc.) whose quality and suitability guarantee the success of the discovery process.
- The **structural component** is in charge of applying the patterns described in Appendix C to the results of the linguistic component in order to generate concept, attribute and relation specifications. Unlike in the previous case, the structural component takes into account the elements in the context of the model in which they are defined and also their structure and their relation with other elements.

The first implementation of the algorithm includes only the first component, lexical-semantic correlation (linguistic component), while the application of heuristics (structural component) has been left for subsequent developments. The implementation is exclusively based on lexical relations provided by WordNet.

The Semantic Mapper is presented in more details in [Barrasa, 2007] and Appendix C.

4.2 Specific requirements

The Semantic mapper requires WordNet to work properly. Instead of taking advantage of portable JWN interface (or the Alignment API support for WordNet), it uses a native platform interface from Java and thus requires to be linked by dynamic libraries. We have found that in practice, this library has to be recompiled for each server.

The Semantic mapper is usually quite long to process (several minutes) so the asynchronous call that has been developed for Scarlet is most welcomed for using the Semantic mapper as well.

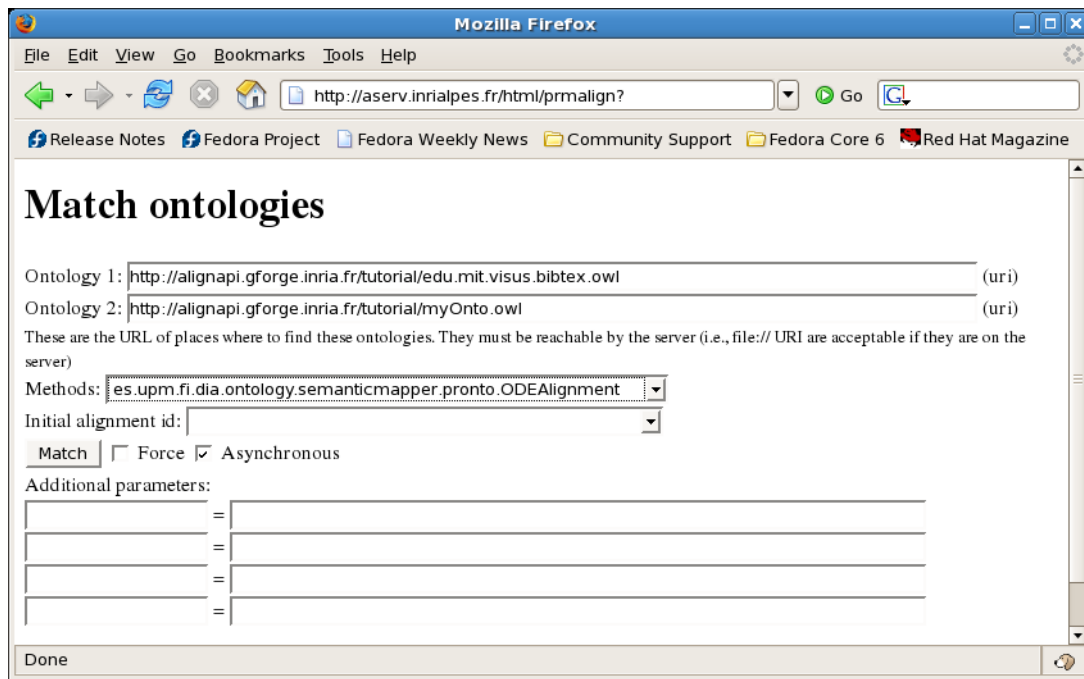


Figure 4.1: Semantic Mapper match call from the HTML interface.

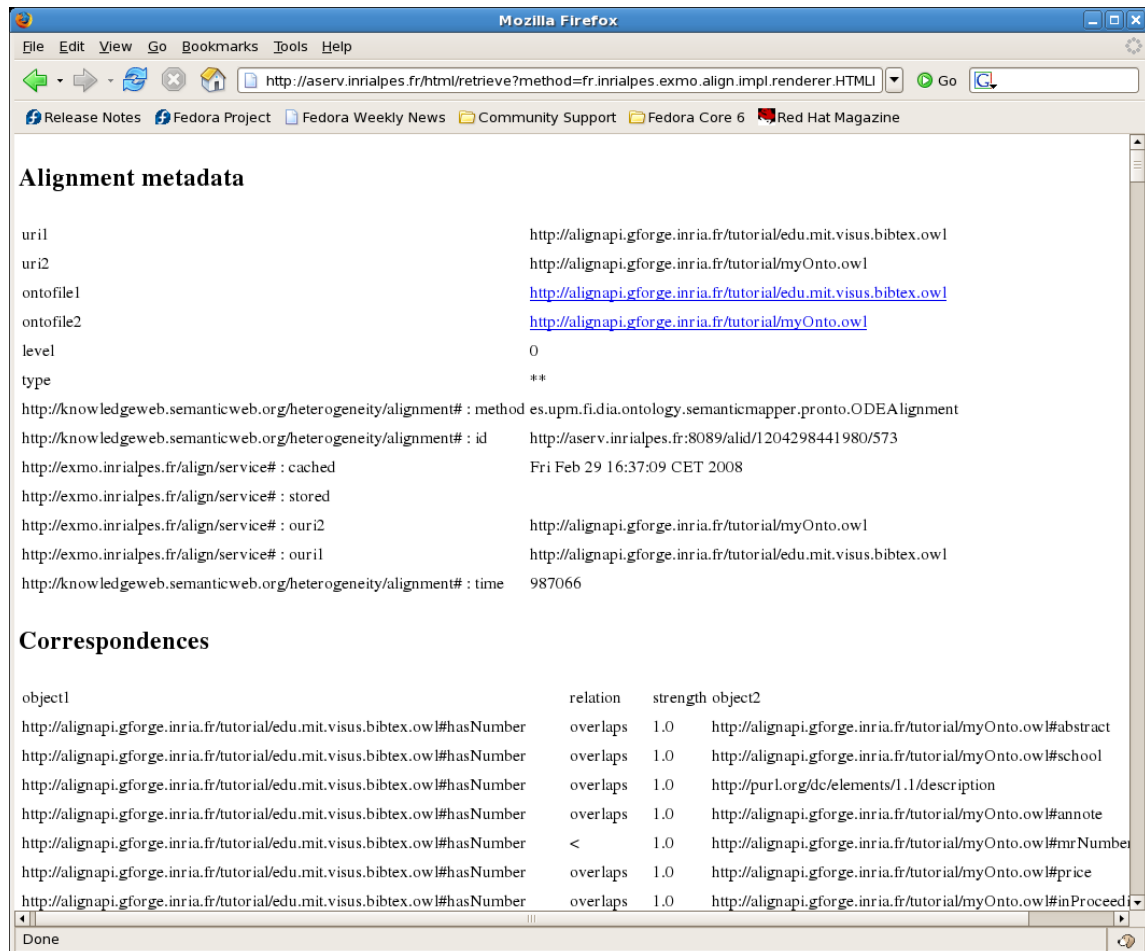
4.3 Integration

The Semantic mapper has not been designed for matching ontologies. It was rather a component made for matching database schemata. So it had to be integrated within the framework of the Alignment API. This is a relatively simple step since it suffices to subclass the `BasicAlignment` class and to implement the `align()` method. In this case, the `OWLAPIAlignment` class has been extended. This class provides all the services for loading the ontologies, simplifying the implementation of `align()`. The new class, called `ODEAlignment` asks the Semantic Mapper the relation it finds between each pair of entities of both ontologies.

An additional problem has been the fact that the algorithm was not expected to deal with URIs while the Alignment API, as a semantic web component, was expecting URIs, so the Semantic Mapper has been reengineered for accepting URIs.

4.4 Results

The Semantic Mapper is integrated in the Alignment server as is OLA. The following figures shows the Semantic Mapper in action both from the HTML interface and the NeOn Alignment plug-in. Then the Semantic Mapper will be visible under the HTML interface (Figure 4.1) as well as the NeOn Alignment plug-in (Figure 3.3) and can be used for matching ontologies (Figure 4.2 and 4.3).



Alignment metadata

uril	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
uri2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
ontofile1	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
ontofile2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
level	0
type	**
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : method	es.upm.fi.dia.ontology.semanticmapper.pronto.ODEAlignment
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : id	http://aserv.inrialpes.fr:8089/alid/1204298441980/573
http://exmo.inrialpes.fr/align/service# : cached	Fri Feb 29 16:37:09 CET 2008
http://exmo.inrialpes.fr/align/service# : stored	
http://exmo.inrialpes.fr/align/service# : ouri2	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl
http://exmo.inrialpes.fr/align/service# : ouri1	http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl
http://knowledgeweb.semanticweb.org/heterogeneity/alignment# : time	987066

Correspondences

object1	relation	strength	object2
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#abstract
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#school
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://purl.org/dc/elements/1.1/description
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#annote
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	<	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#mrNumber
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#price
http://alignapi.gforge.inria.fr/tutorial/edu.mit.visus.bibtex.owl#hasNumber	overlaps	1.0	http://alignapi.gforge.inria.fr/tutorial/myOnto.owl#inProceed

Figure 4.2: Semantic Mapper results call from the HTML interface.

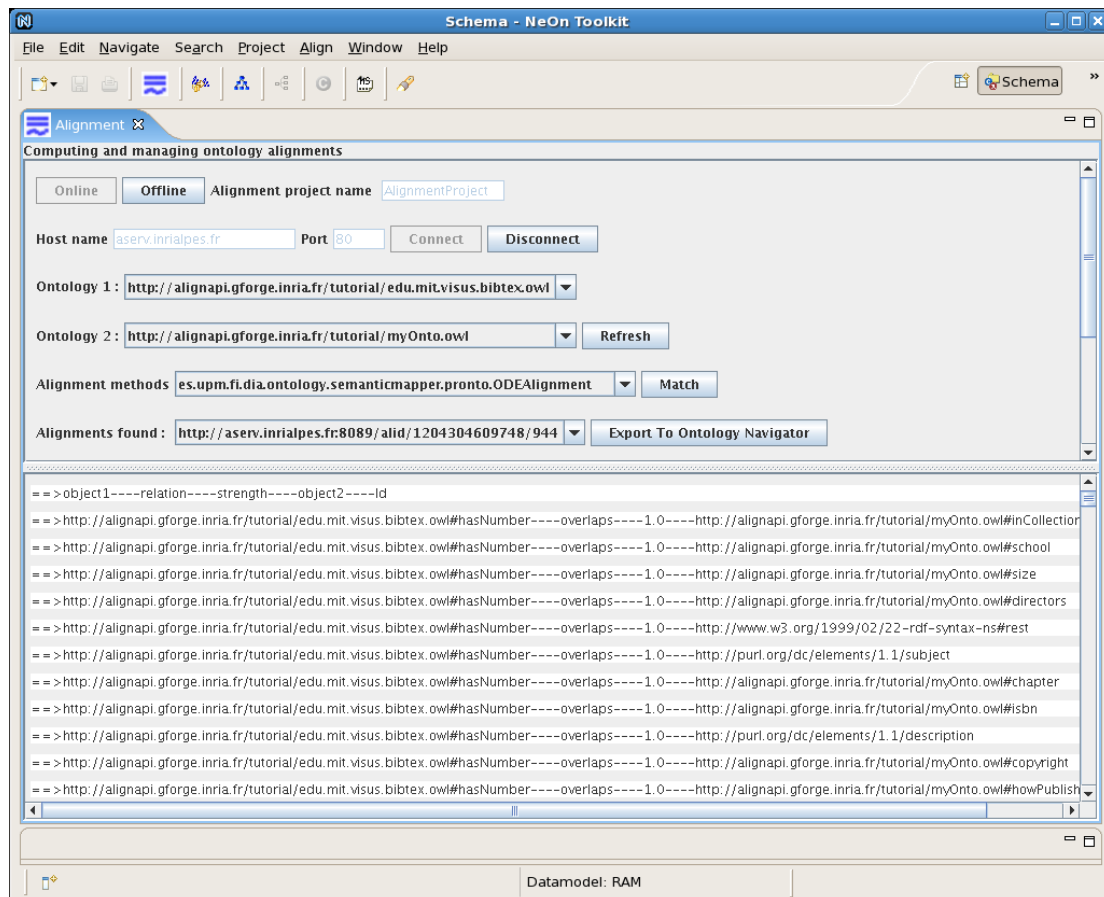


Figure 4.3: Semantic Mapper match call from the NeOn toolkit.

Chapter 5

Integration of Scarlet

5.1 Presentation

Scarlet is a system for matching ontologies by using ontologies found on the web. It has been fully described in Deliverable D3.3.1 (though the “Scarlet” name has been given to the system afterward).

Scarlet finds ontologies on the web (thanks to a search engine like Swoogle or Watson) and finds correspondences between terms in ontologies (and even in less structured resources like thesaurus) depending on the relations the corresponding concepts have in the found ontologies.

5.2 Specific requirements

Scarlet is generally long to process (several hours) so we developed an asynchronous option for the match call in the Alignment server. This works by allocating a URI for the requested alignment and returning the URI immediately, i.e., before the alignment has been computed. The computation of the alignment is run in a separate thread of the Alignment server and once finished it is assigned the URI. Until this process has ended, requesting the alignment from the server returns an error, but once computation has reached its end, the alignment is returned and can be manipulated normally.

This has required some reengineering of the Alignment server cache which used to generate the URI at the end of the process. The asynchronous mode is visible in the interfaces through the “Asynchronous” checkbox. It has been useful for all the other matchers.

Scarlet needs an ontology search engine for working. So far it has been based on the Swoogle search engine. Unfortunately, the version we were using has stopped working before a few weeks before delivery. Scarlet is thus currently reengineered to be used with Watson (this was planned but not so early because Watson evolves a lot).

5.3 Integration

Consequently, Scarlet is not yet available in the server.

5.4 Results

Scarlet integrated within the Alignment server and the NeOn Alignment plug-in can be used in the same way as the Semantic Mapper and OLA.

Chapter 6

Connection with Oyster

6.1 Presentation

Oyster is a Peer-to-Peer application (described in Deliverable 1.4.1) that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies. The goal is a decentralized knowledge sharing environment using Semantic Web technologies that allows developers to easily share ontologies and alignments. To achieve this, Oyster implements the proposal for a metadata standard, so called Ontology Metadata Vocabulary (OMV) as the way to describe ontologies.

In the context of NeOn we have extended the OMV so that it can deal with alignment metadata. The goal is to be able to share metadata about alignments and ultimately to be able to find a particular alignment.

Oyster is an implementation of the registry services of NeOn. Further it provides an API and a Web Service interface to query, create, and manipulate ontology metadata information according to the OMV model.

Oyster offers a user driven approach where each peer has its own local registry of ontology metadata and also has access to the information of others registries, thus creating a virtual decentralized ontology metadata registry.

The goal of connecting the Alignment server to Oyster is to be able, for other Oyster peer, to obtain information about stored alignments in the servers and to the Alignment servers to find which alignments involve particular ontologies.

6.2 Specific requirements

For the integration of Oyster and the Alignment server, we have used the Oyster API. We could have used the Oyster web service interface. Unfortunately it was not completed for the OMV alignment metadata extension. The Oyster API requires that a Kaon2 server be running on the server machine. This operation is documented in the Alignment server documentation and does not present any difficulty.

6.3 Integration

Each Alignment server can thus be a peer of Oyster. As such, it has a metadata repository about the alignments it permanently stores. Each time the server has to store a new alignment in its own database, this alignment metadata is stored in by the Oyster peer metadata registry. The peer regularly connects its master in order to communicate its activity and the new content it has.

We developed a particular abstraction of registry (that we call `Directory`) within the Alignment server, that is called at critical points (when another server is sought, when an alignment is stored, at launch time). Oyster is seen by the server as such a directory.

The metadata generated towards the Oyster network is currently minimal with regard to the OMV extension.

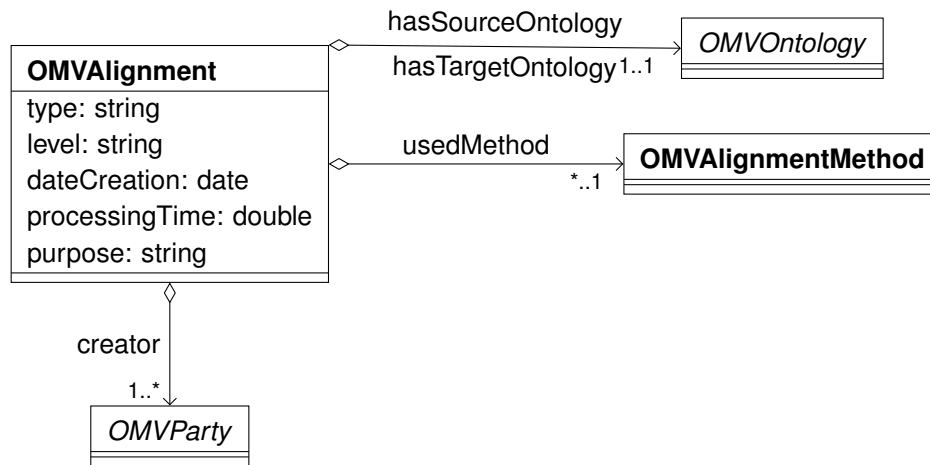


Figure 6.1: UML characterization of the common core between OMV and the Alignment format.

It is displayed in Figure 6.1.

6.4 Results

The Alignment server-Oyster connection is fully functional. It is not currently possible to illustrate this through screendumps.

The next step will be that the Alignment servers use Oyster to find other alignments and ontologies that are close to another ontology.

Chapter 7

Conclusions

This deliverable has presented software developed in NeOn for providing matching support within, and outside, the NeOn toolkit. This matching support is meant, in particular, to contribute recontextualising ontologies. It can be used for other purposes. One of the notable add-on that we are considering is the generation of modules (as described in Deliverable 1.1.3) from aligned ontologies.

The provided software integrates the Alignment server presented in Deliverable 3.3.1 with the NeOn toolkit through a dedicated plug-in. This plug-in can work either independently of any Alignment server, and thus rely on local matching algorithms, or take advantage of Alignment servers and use both their embedded matchers and their stored alignments. The interface between the NeOn Alignment plug-in and the Alignment server uses SOAP over HTTP.

We also have enhanced the Alignment server by integrating several matching algorithms developed by NeOn partners (OLA, Semantic Mapper and Scarlet) and connected it to the Oyster metadata sharing system. We also have improved the alignment server, in particular with asynchronous matching (many other improvements have been made since Deliverable D3.3.1, but they are not relevant in this deliverable).

7.1 Current state of the prototypes

All prototypes described here are working systems (the exception of Scarlet should be solved soon). Most of them, beside the Semantic Mapper, are available as a free download. Appendix A and B explain how to set them up. They are also available from the public demonstration server <http://aserv.inrialpes.fr>.

7.2 Further developments

There are several lines of development that will follow this first integration within the NeOn toolkit. We discuss them briefly:

- Improving the integration of the plug-in within the NeOn toolkit: the NeOn toolkit and its interfaces evolve. We thus have to make the plug-in evolve for taking the best advantage of it.
- Graphic interface. There is currently no way to graphically visualise or edit the returned alignments. We are currently investigating two ways to deal with this: integrating the plug-in with the Ontoprise OntoMap plug-in (closed source) or taking advantage of display developed by JSI.
- Watson integration. Watson allows to find ontologies. We would like to integrate better with Watson so that it is possible to find ontologies aligned with a particular ontology or to find ontologies close to another one.
- New matchers. The Alignment server is always open to the integration of new matchers. In particular, Deliverable D3.3.3 will present the integration of the Data extractor developed specifically by JSI.
- Matcher evaluation. Finally, it is not sufficient to add matchers to the Alignment server offer. We will proceed to an evaluation of these matchers in the context of the NeOn applications.

Appendix A

Installing the NeOn Alignment plug-in

A.1 Get the NeOn toolkit

A version of the NeOn toolkit is available at <http://www.neon-toolkit.org>.

From this site, you download a zipped NeOn toolkit file according to your Operating System (OS). After unzipping this file, a new directory is created, and within it you have a directory "plugins" and an executable file.

For instance, for Linux, download the zipped file "NeOnToolkit-1.0-extended-B723-Linux.zip".

These descriptions have been made with the version 1.0 of the open source version of the NeOn toolkit¹

A.2 Get and install the NeOn Alignment plug-in

The last version of the NeOn Alignment plug-in can be obtained from the <http://alignapi.gforge.inria.fr> web site. The plug-in is currently in `plugins/neon/lib/neonalign.jar` of the last release of the Alignment API.

The `neonalign.jar` jar-file must be put it in the "plugins" directory of the NeOn toolkit.

The version described by this deliverable is version 3.2.

A.3 Run the NeOn toolkit with the plug-in

To run the NeOn toolkit with the NeOn Alignment plug-in, launch the NeOn toolkit with the executable file (e.g. under Linux, run the "NeonToolKit.sh" script). This will bring an interface screen like that of Figure A.1.

A.4 Use the plug-in with the Alignment server

To activate the NeOn Alignment plug-in from the NeOn toolkit Menu, click on the "Align" menu or the button on Toolbar, a view "Alignment Server" for the plug-in will be opened.

Connection To connect to the INRIA's Alignment server from the NeOn Alignment plug-in, you have to type "aserv.inrialpes.fr" for hostname and "80" for port (see Figure A.2).

¹We are currently working towards using the subsequent versions of the NeOn toolkit.

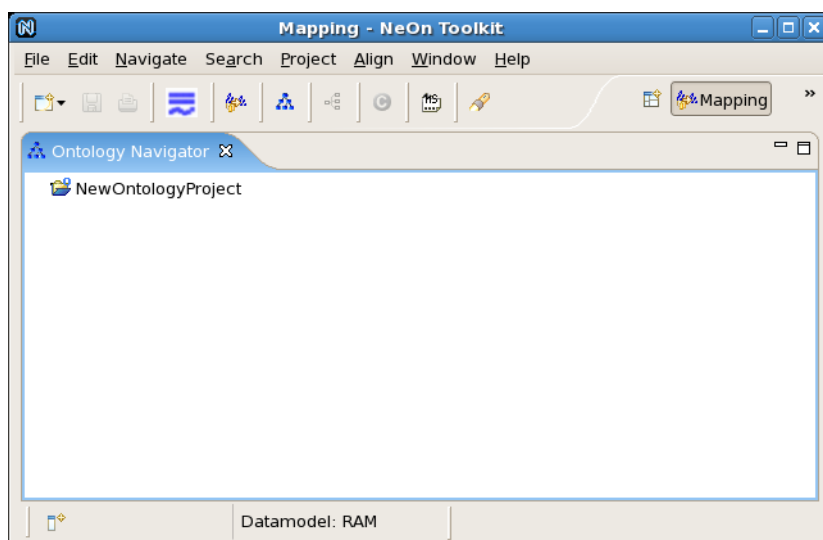


Figure A.1: The NeOn toolkit (version 1.0) with embedded NeOn Alignment plug-in.

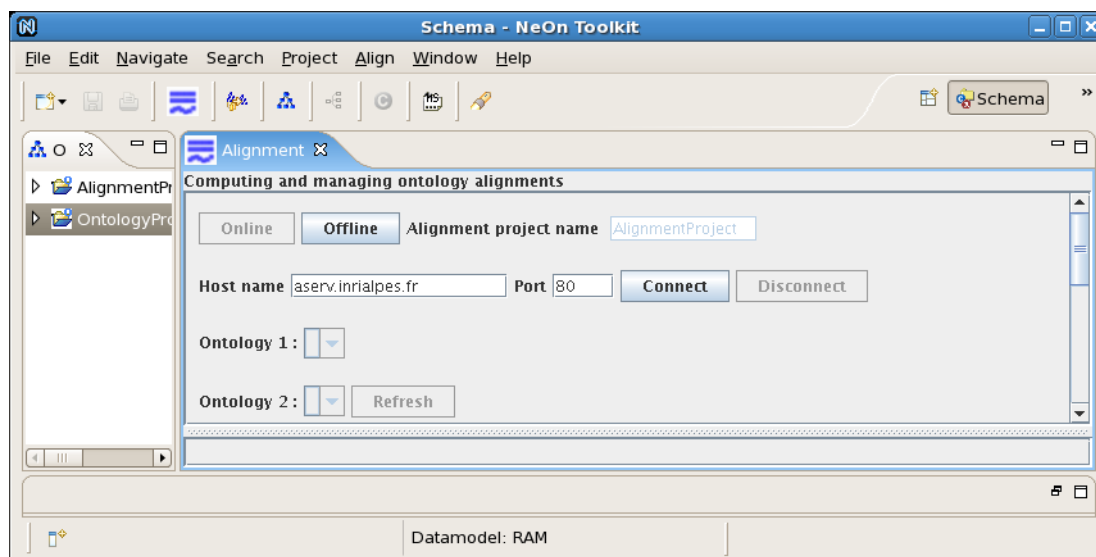


Figure A.2: Connecting to the INRIA Alignment server.

Appendix B

Installing and extending the Alignment server

B.1 Get and install the Alignment server

The last version of the Alignment server can be obtained from the <http://alignapi.gforge.inria.fr> web site. Download and unzip the zip file corresponding to the desired version.

The version described by this deliverable is version 3.2.

Running the Alignment server without extensions requires opening some TCP ports and running a DBMS server. This has been described in Appendix A of Deliverable D3.3.1 and in Appendix A of the Alignment API and server documentation. We concentrate here on extensions.

We assume that the directory `$LIB` contains the jar files that come with the server release (they can be moved anywhere).

B.2 Install WordNet

Wordnet is required by several algorithms: JWNAlignment, OLA and Semantic mapper. Therefore it may be necessary to install wordnet. It can be obtained from <http://wordnet.princeton.edu/>.

We assume that it will be installed in a directory denoted by variable `$WNHOME`.

The installation goes through a classic:

```
$ ./configure
$ make
$ make install
```

In addition, the Semantic mapper requires the installation of the WordNet Java Native library. This library uses native code (supposed faster) to access WordNet from Java.

We suggest that you follow the instructions from <http://wnjn.sourceforge.net> and you recompile the wnjn native code so that it is compatible with your machine. This is achieved simply by:

```
$ ant -Dtarget.wnx="30"
```

"30" is the version of WordNet used.

We assume that the resulting dynamic library (either for Linux or Windows) is installed in a directory denoted by variable `$WNJNLIB` (by default `/usr/local/lib/wnjn`).

B.3 Connect to the Oyster system

Oyster is freely available from <http://ontoware.org/projects/oyster2/>.

Just put in `$LIB` the libraries:

- oyster2.jar
- kaon2.jar
- org.eclipse.jface_3.1.0.jar
- org.eclipse.equinox.common_3.2.0.v20060603.jar

Oyster requires that a Kaon2 repository containing the metadata be running. This repository can easily be launched by:

```
$ java -cp $LIB/kaon2.jar org.semanticweb.kaon2.server.ServerMain
    -registry -rmi -ontologies O2serverfiles
```

such that `O2serverfiles` is the directory in which Kaon 2 will find its ontologies and store its annotations (this directory comes with the Oyster release).

B.4 Add new matching algorithms

B.4.1 Adding OLA

OLA is available from <http://ola.gforge.inria.fr>. We have used a version made from the subversion repository which goes beyond version 2.0.

Just put in `$LIB` the libraries:

- mtj.jar
- ontographs.jar
- sboatools.jar
- sboaalgorithms.jar

Do not forget the `-Dwndict=$WNHOME` argument to the server.

B.4.2 Adding Semantic mapper

Semantic Mapper is available from UPM.

Just put in `$LIB` the libraries:

- semanticmapper-0.2.jar
- wnjn-1.0.jar

Do not forget to add the `-Djava.library.path=$WNJNLIB/dict` Java switch when launching the alignment server.

B.5 Launching the alignment server

We have added in the build file for the alignment server a target `aserv` which regenerates a proxy jar file `lib/aserv.jar` listing the extension libraries that have to be embedded within the server. The server will find out the embedded matchers from this list of libraries. So, it may be necessary to modify the build file (more precisely the `manifest` declaration of the `aserv` target) to embed new matchers.

The full sequence for launching the server is:

```
$ ant aserv
$ java -Djava.library.path=$WNJNLIB -jar $LIB/aserv.jar -Dwndict=$WNHOME/dict -O -H -W &
```

Appendix C

Semantic mapper

This appendix describes the Semantic Mapper algorithm, an algorithm capable of discovering and generating automatically concept specifications, attributes and interrelations of a conceptual model –an anthology– according to the elements of a relational model. The main purpose of the Semantic Mapper algorithm is to help build a conceptual coverage, expressed in R_2O , over a relational model M based on a given ontology O . Although this process is automatic, the results will not be completed or directly exploitable; on the other hand, because of its heuristic nature, it should be supervised and validated by any user with a deep knowledge of the semantics underlying the models. The first implementation of the algorithm includes only the lexical-semantic correlation, while the application of heuristics has been left for a subsequent prototype.

In the introduction section we describe the analysis of the problem, that is, the disparity of models; this disparity makes necessary to establish correspondences.

C.1 Introduction

When the models need to capture and represent any kind of knowledge, they are usually conditioned by a set of implicit suppositions (previous experience, or the designers' subjective knowledge) or of explicit and/or pragmatic suppositions (purpose of the model). These suppositions provoke disparity or difference between any two models, and even between models that model the same domain.

This disparity adopts different forms, which in literature have been organized into types or levels. We briefly describe here the different kinds of disparity and have organized them into levels according to the characterizations.

Euzenat [Euzenat, 2001] and [Euzenat *et al.*, 2004] organize heterogeneity into four levels: the **lexical-syntactic level**, which depends on the representation language selected for the models to be compared; the **terminological level**, which depends on the names selected to refer to the different entities described in the model; the **conceptual level**, which is related to the entities selected to model a domain and which can include differences in coverage, granularity or perspective; and finally, the **semiotic /pragmatic level**, which is originated because different users or communities interpret the same model differently.

Corcho [Corcho, 2005] distinguishes four levels: lexical, syntactical, semantic and pragmatic. The first two levels are equivalent to the lexical-syntactic levels defined in [Euzenat *et al.*, 2004]; the semantic level focuses on the different expressivity that the existing knowledge representation paradigms have; and the pragmatic level is equivalent to the pragmatic level defined in [Euzenat *et al.*, 2004].

[Dou *et al.*, 2003] distinguishes two levels of disparity between models: **syntactical** (because of the language in which those models are expressed) and **semantic** (because of the different decisions taken by the experts).

Visser *et al.* [Visser *et al.*, 1997] identify two large levels, and they refer to the differences that take place within each one as non-semantic (because of the language or representation paradigm) and semantic (because of the set of "implicit suppositions" of conceptual type).

Klein [Klein, 2001] distinguishes the same two levels that Visser et al. do, but he refers to them as the **language level or meta-model**, which includes expressivity differences and modes of knowledge representation of languages, and as the **ontology level or model level**, which includes the different ways in which a domain can be modelled.

Tamma [Tamma, 2001] distinguishes two levels: syntactical and semantic, and both correspond with Visser's [Visser et al., 1997].

Chalupski [Chalupsky, 2000] identifies three classes of disparity: **modelling conventions**, which are differences encountered in the design decisions; differences of **coverage and granularity**, which can be included generically as semantic differences according to [Euzenat et al., 2004]; and finally, differences in the **representation paradigm**, which are different theories used for specific parts of the domain, such as the representation of time or space.

Grosso et al. [Grosso et al., 1998] analyze the differences between two models of knowledge and classify the differences into groups; they also emphasize **the fundamental differences** which they attribute to the differences in the primitives that each model provides.

Hamer and McLeod [Hammer and McLeod, 1993] focus on a heterogeneity spectre based on four levels of abstraction: the **meta-data language level**, which uses different languages of specification of the conceptual model; the **meta-data specification level**, which includes the different ways of modelling a domain; the **object comparability level**, which groups together terminological differences; and finally, the **data format level**, which refers to the different ways in which atomic values can be stored in two different models.

Kim and Seo [Kim and Seo, 1991] also present two levels: the **structural level**, which refers to the different structures in which the information systems organize their data, and the **semantic level**, which refers to the content and meaning of each item of information.

Figure C.1 compiles the proposed classifications by relating them with each other.

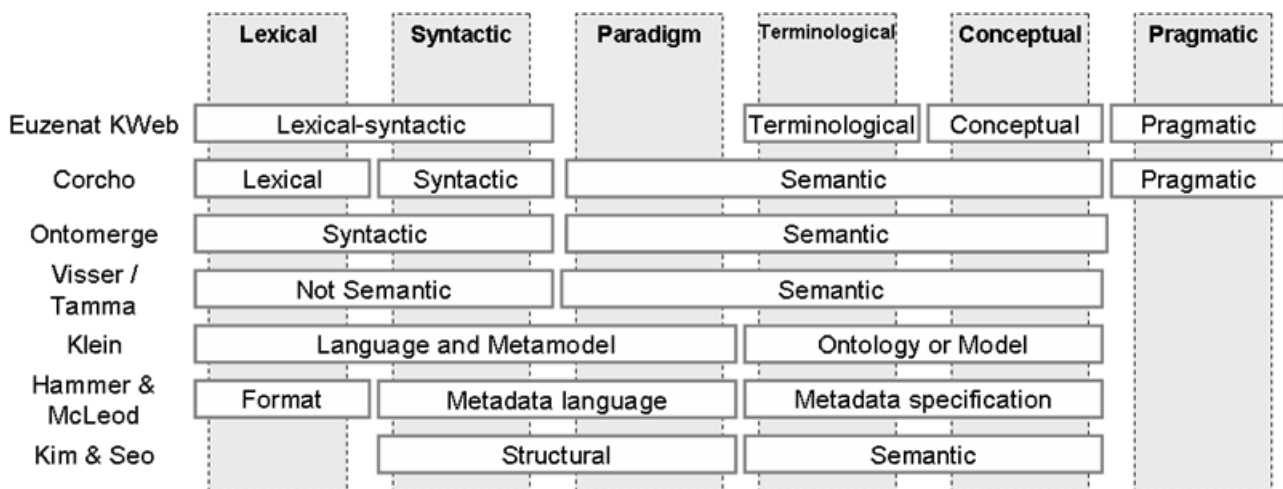


Figure C.1: Different types of disparity organized into levels according to the characterizations appeared in the literature.

C.2 Automatic mapping discovery

This section describes the *Semantic Mapper* algorithm, an algorithm capable of discovering and generating automatically specifications of concepts, attributes and interrelations of a conceptual model –an ontology– according to the elements of a relational model. The main purpose of the Semantic Mapper algorithm is to help build a conceptual coverage, expressed in R_2O , over a relational model M based on a given ontology O . Although this process is automatic, the results will not be completed or directly exploitable; additionally, because of its heuristic nature it should be supervised and validated by any user who knows the

semantics underlying the models. The first implementation of the algorithm includes only the lexical-semantic correlation, while the application of heuristics has been left for a subsequent prototype.

C.2.1 Foundations of the mapping discovery process

The process of the automatic mapping discovery between ontologies and relational models is based on the following hypotheses:

1. The hypothesis of the **lexical-semantic correlation** between elements of two models. It is possible to forecast the semantic relation between elements pertaining to two models from the lexical relation existing between the terms that represent them. It is quite sensible to think that when designing a model (being this a relational model, an ontology or any other model), the elements that compose such a model should have comprehensible and meaningful names instead of random names or codes since the latter complicate the management and use of the model. In other words, the names of the elements of a model (relations, attributes, concepts, etc.,) have detectable semantic "clues" that permit inferring the semantic relations existing between them. For example, a synonymy relation between two terms would probably hide a conceptual relation of equivalence.
2. The hypothesis of the **confinement based on patterns**. A set of patterns is considered and according to these patterns the elements of a conceptual model (an ontology) can be presented confined in a relational model. Only these patterns are regarded in the algorithm described below.

According to what these hypotheses suggest, the mapping discovery method here proposed should have two main components: **a linguistic component** and **a structural one**.

1. The **linguistic component** will detect lexical relations between the terms used to identify the elements of both models so as to infer semantic relations from them. This is, in fact, a comparison between elements, though they are taken in isolation, regardless of the context in which they occur and of their relation with other elements of the same model. An example of the type of mappings that this component will detect is that which exists between a concept of the ontology O and an attribute of the relational model M because **both have the same name**. To detect this type of relations we will use a series of linguistic resources (glossaries, thesauri, lexical databases, etc.) whose quality and suitability will guarantee the success of the discovery process.
2. The **structural component** will be in charge of applying the described patterns to the results of the linguistic component in order to generate concept, attribute and relation specifications. Unlike in the previous case, the structural component takes into account the elements in the context of the model in which they are defined and also **their structure and their relation with other elements**. An example of the type of mapping that this component will detect is the mapping between an attribute A_O of the ontology O and an attribute A_M of the relational model M because A_O has as a domain a concept C_O , which is related in its turn to the relation R_M of M to which A_M belongs.

Both components exploit exclusively the meta-information of the models, that is, the information of the relational schema and the information of the ontology description; however, they do not exploit the information of the individuals that the components contain (in the case of the relational model, tuples, whereas in the case of the ontology, instances). Therefore, this is a *schema-only based* approach, according to the classification of [Rahm and Bernstein, 2001].

C.2.2 Specification of the mapping discovery process

The process of automatic mapping discovery that the Semantic Mapper algorithm implements has as its main goal –given an ontology O and a relational model M – to automatically generate a conceptual coverage of the relational model M based on the ontology O and the expression of this conceptual coverage through

the R₂O language. The following expression specifies declaratively the behaviour of the Semantic Mapper algorithm:

$$\Psi(O, M, \mathcal{R}) \rightsquigarrow \mathcal{RC}(O, M)$$

Where

- Ψ Represents the operation of the Semantic Mapper algorithm
- O and M are the ontology and the relational model to be aligned, respectively.
- \mathcal{R} is a set of resources used in the process of discovering, where the specific models of entry O and M are also used. Such resources can be of two kinds: own and external. The former (own resources) constitute the core of the algorithm and make up the "heuristic knowledge" (parameters, rules, etc.) on which this knowledge is based. The latter (external resources) are configurable to some extent and interchangeable; these resources are the thesauri, term glossaries, domain ontologies and any other linguistic resources used by the component with the same name.
- The covering $\mathcal{RC}(O, M)$ is the result of the Semantic Mapper algorithm.

More specifically, if we consider that a conceptual coverage of a relational model M based on an ontology O is composed of a set of concepts, attribute and interrelation specifications, then the discovery process carried out by the Semantic Mapper algorithm will consist in inferring and formalizing a set of specifications of elements of the ontology O according to the relational model M , using for this purpose the R₂O language. The operation carried out by the Semantic Mapper algorithm with respect to each pair of elements of the models will be formalized as follows:

$$\Psi_E(\mathcal{E}'_M(E_O), \mathcal{S}_O(E_O), \mathcal{S}_M(E_M), \mathcal{R}) \rightsquigarrow \mathcal{E}_M(E_O)$$

Where E_O and E_M are the elements trying to get in relation; the first element belongs to the terminology or signature of the ontology O , whereas the second belongs to the relational model M .

Additionally

- Ψ_E represents the Semantic Mapper operation at the element level.
- $\mathcal{E}'_M(E_O)$ represents an a priori specification for E_O , and is either the result of previous iterations of the Semantic Mapper algorithm or has been introduced manually by the user.
- $\mathcal{S}_O(E_O)$ and $\mathcal{S}_M(E_M)$ represent the structural information available in the model M for E_M and in the model O for E_O (obviously, the name of the concept is included)
- \mathcal{R} represents the set of own resources or external resources that the algorithm uses to infer its results (as it happens in the general case).
- The specification $\mathcal{E}_M(E_O)$ (in which E_M is involved) is the result of the Semantic Mapper operation and will be a revision of $\mathcal{E}'_M(E_O)$, proposed as entry argument to the algorithm operation. Note that in some cases $\mathcal{E}_M(E_O) = \mathcal{E}'_M(E_O)$, which is the same as saying that the semantic relation between the E_O and E_M elements does not add any information to the E_O specification.

To conclude, we can consider the global result of the operation carried out by the Semantic Mapper algorithm as the accumulation of its operations over each pair of elements of the two models.

$$\Psi(O, M, \mathcal{R}) = \bigcup_{\forall E_O \in O, E_M \in M} \Psi_E(\mathcal{E}'_M(E_O), \mathcal{S}_O(E_O), \mathcal{S}_M(E_M), \mathcal{R})$$

C.2.3 Lexical-semantic correlation

The lexical-semantic correlation is a property of the models which permits inferring the semantic relation between elements¹ pertaining to two different models from the lexical relation between the terms with which they are represented.

Therefore, if the relational model M_1 defines the relation *Work*, the ontology O_1 defines the concept *Job*, and both terms are synonyms, then we can infer the semantic relation between the elements that both terms represent _ in this case probably the relation of equivalence, though, in general, the semantic relation to be inferred will also depend on the lexical relation from which it is inferred and on the type of elements (concepts, relations, etc.)

It is important to note that lexical relations refer specifically to terms or expressions, and that they have an informal nature (for example, synonymy is the semantic relation that holds between two words that can express the same or very similar meaning), whereas semantic relations refer specifically to elements of a model (concepts, attributes, etc.) and are, therefore, formalizable logical relations and unambiguous (for example, the equivalence between two concepts supposes that every instance of the first concept is an instance of the second). The following expression formalizes the notion of the lexical-semantic correlation:

$$r_{lex}(term_A, term_B) \Rightarrow r_{sem}(A, B)$$

In this expression, $term_A$ is the term used to name element A , and $term_B$ is the term corresponding to element B . Given the lexical relation r_{lex} between $term_A$ and $term_B$, we can infer the semantic relation r_{sem} between elements A and B . The terms $term_A$ and $term_B$ do not have to be simple or atomic², as it could be expected from this expression, they can be complex terms composed of other simple terms. The expression below generalizes the previous one for any type of term.

$$\bigwedge_{\forall i \in 1..n, \forall j \in 1..m} r_{lex}^{i,j}(term_A^i, term_B^j) \Rightarrow r_{sem}(A, B)$$

$$term_A = term_A^1 \dots term_A^n$$

$$term_B = term_B^1 \dots term_B^m$$

In this expression, $term_A$ is a generic term composed of a sequence of simple terms $term_A = term_A^1 \dots term_A^n$, which is used to name the element A , whereas $term_B$ (with a similar structure $term_B = term_B^1 \dots term_B^m$) is used to name element B .

The **lexical relations** r_{lex} that will be considered for the pair of terms $term_A$ and $term_B$ are the standard ones: $r_{lex} \in \{sin, hiper, hipo, ant\}$ **synonymy** annotated as $syn(term_A, term_B)$, **hyponymy** annotated as $hypo(term_A, term_B)$, **hyperonymy** annotated as $hiper(term_A, term_B)$ and **antinomy** annotated as $ant(term_A, term_B)$.

The **semantic relations** r_{sem} between two elements A and B to be inferred from the previous lexical relations are the following: $r_{sem} \in \{\equiv, \sqsubseteq, \supseteq, \perp, \sqcap\}$ **annotated equivalence** $A \equiv B$; **more/less generic** (implication in both senses) $A \sqsubseteq B$ and $A \supseteq B$, **disjunction** (or empty intersection) $A \perp B$, and overlapping (or generic intersection) $A \sqcap B$.

The process of inferring the semantic relation $r_{sem}(A, B)$ from the lexical relation $r_{lex}(term_A, term_B)$ in the case of simple terms is based on the ideas put forward by [Bouquet *et al.*, 2003] and [Magnini *et al.*, 2003]. When dealing with complex terms, the semantic relation is inferred from the semantic relations existing between the basic terms of which they are composed. To carry out this process, we propose the **propagation algorithm**.

¹ Any element of a model: concepts, attributes, or relations for ontologies, and relations and attributes for a relational model

² They can not be decomposed into other terms; for example, Publication or Author are atomic or simple terms, whereas Scientific publication or Main author are complex terms

C.2.4 Pattern confinement

As mentioned in the previous section, a semantic relation between an element (concept, attribute or relation) pertaining to an ontology and other element pertaining to a relational model will be interpreted differently depending on the type of element in question. Thus, the equivalence between an ontology concept and a relation of a relational model will not be considered in the same fashion, nor will be a relation between a concept and a relational model. Below we describe each possible case.

The ideas commented in this section will be implemented through a set of heuristic rules.

Concept confinement

In [Barrasa, 2007] we describe how concepts of an ontology can appear confined to a relational model; therefore, in accordance to that description, it will be necessary to identify the relation or relations of the relational model of those concepts with the key attributes that will permit identifying exclusively the individuals of a concept. Additionally, (and if possible) they will also permit identifying the condition and gathering expressions.

A semantic relation between a concept and an element (relation or attribute) of a relational model inferred from the lexical-syntactical correlation between the terms that identify them should be interpreted as follows:

Interpretation of semantic relations between ontology concepts and relations of the relational models

The presence of a semantic relation ³ $r_{sem} (\equiv, \sqsubseteq, \text{etc.})$ between the concept C of the ontology O and the relation R of the relational model M will be interpreted within the context of the Semantic Mapper algorithm as follows:

Assume \mathcal{IM} is the interpretation $(\Delta^{\mathcal{IM}}, \mathcal{IM})$ of all the possible interpretations of the ontology O , which coincides with the purpose with which the ontology was built; and assume likewise the interpretation $\mathcal{IM}_R = (\Delta^{\mathcal{IM}}, \mathcal{IM}_R)$ that describes the semantics of the relational model M based on the *intended meaning* of the ontology (note that domain described on the interpretation is the same as the ontology domain); given that a semantic relation is established between the concept C of the ontology O and the relation R of the relational model M , then we can verify that

$$r_{sem}(C, R) \Rightarrow r_{conj}(C^{\mathcal{IM}}, R^{\mathcal{IM}_R})$$

Where r_{conj} is a relation between sets among equals, included, includes, disjoint ($r_{conj} \in \{=, \subseteq, \supseteq, \cap = \emptyset\}$). The mapping between r_{sem} and r_{conj} is given in table C.1

Semantic	Relation
Equivalence ($C \equiv R$)	$C \equiv R \Rightarrow C^{\mathcal{IM}} = R^{\mathcal{IM}_R}$
Implication ($C \sqsubseteq R$)	$C \sqsubseteq R \Rightarrow C^{\mathcal{IM}} \subseteq R^{\mathcal{IM}_R}$
Empty intersection ($C \perp R$)	$C \perp R \Rightarrow C^{\mathcal{IM}} \cap R^{\mathcal{IM}_R} = \emptyset$
Generic intersection ($C \sqcap R$)	$C \sqcap R \Rightarrow C^{\mathcal{IM}} \cap R^{\mathcal{IM}_R} \neq \emptyset$

Table C.1: Semantic relations between ontology concepts and relations of the relational model.

The following example clarifies this notion: The ontology O_1 includes the concept *FootballTeam*. The relational model M_1 in its turn includes the relation *EnglishFootballTeams* with the *Id*, *name* and *country* attributes.

³Obtained from a set of lexical relations (r_{lex} (*sin*, *hiper*, etc.) between the term $term_C$, used for naming the concept C of O , and the term $term_R$, used for naming the relation R de M

Relational Model M_1 (fragment) <i>EnglishFootballTeams</i> (<i>Id</i> , <i>name</i> , <i>country</i>)	Ontology O_1 (fragment) class-def <i>FootballTeam</i> slot-def <i>name</i> domain <i>FootballTeam</i>
---	---

Let's suppose a “more general than” semantic relation between the concept of *FootballTeam* O_1 , and the relation *EnglishFootballTeams* of M_1 , then we can verify that

$$\begin{aligned} \text{FootballTeam} \sqsubseteq \text{EnglishFootballTeams} &\Rightarrow \\ \text{FootballTeam}^{\mathcal{IM}} \supseteq \text{EnglishFootballTeams}^{\mathcal{IM}_{\mathcal{R}}} \end{aligned}$$

Interpretation of semantic relations between ontology concepts and relational model attributes

The presence of a semantic relation ⁴, used for naming the attribute A of M , r_{sem} , between a concept C of the ontology O and an attribute A of the relational model M will be interpreted also as a semantic relation between the concept C and the relation R_A formed as the projection of the relation R that contains it ($R_A = \{x.A | x \in R\}$) over the attribute A ; or it can be interpreted from the perspective of the relational algebra $R_A = \pi_A R$.

This type of relations reveals a “structuring disparity” situation or what is the same, a scenario in which a structured element of the ontology (concept) is confined to an element of the relational model that lacks structure (attribute).

Once we have clarified this point, we can add that its semantics is identical to the semantics of the above case. The following expression formalizes this idea:

$$r_{sem}(C, A) \rightsquigarrow r_{sem}(C, R_A) \Rightarrow r_{conj}(C^{\mathcal{IM}}, R_A^{\mathcal{IM}_{\mathcal{R}}}), \quad R_A = \{x.A | x \in R\}$$

Example: In the example the ontology O includes the concepts *Person* and *Address* associated through the ad-hoc relation *livesAt*; in its turn, the relational model M_1 includes the relation *Clients* with the attributes *ID*, *Name* and *Address*, among others.

Relational Model M_1 (fragment) <i>Clients</i> (<i>Id</i> , <i>name</i> , <i>address</i> , ...))	Ontology O_1 (fragment) class-def <i>Person</i> class-def <i>Address</i> slot-def <i>Name</i> domain <i>Person</i> range <i>Address</i>
---	---

In the ontology O_1 , the information relative to the address is modeled as a class (a concept) with its corresponding structure- which is organized in attributes- , but in the relational model M_1 , the same information is modeled as an attribute –whose internal structure is nil. Let's suppose a semantic relation of equivalence between the concept *Address* of O_1 and the attribute *Address* of M_1 , then we can verify that

$$\begin{aligned} \text{Address} \equiv \text{address} &\Rightarrow \text{Address}^{\mathcal{IM}} = R_{\text{address}}^{\mathcal{IM}_{\mathcal{R}}}, \text{ where} \\ R_{\text{address}} &= \{x.\text{address} | x \in \text{Clients}\} \end{aligned}$$

⁴ As in the above case, the semantic relation is inferred from a set of lexical relations $r_{lex}^{i,j}$ between the term $term_C$ (possibly complex), used for naming the concept C of O , and the term $term_A$ (possibly complex)

Attribute confinement

If we agree with how the attributes of an ontology can appear confined to a relational model, then we need to identify the concept to which the attributes belong (their domain) and from this, to identify the attribute or attributes of the relational model where the ontology attribute is and also the eventual transformation function that will permit calculating the first attributes from the second ones. Additionally, (if possible) the condition and gathering expressions that describe the attribute specification will also be identified.

Interpretation of the semantic relations between ontology attributes and relational model attributes

The presence of a semantic relation $r_{sem} (\equiv, \sqsubseteq, \text{etc.})$ ⁵ between a conceptual attribute A_O of the ontology O and a relational attribute A_R of the relational model M will be interpreted as the participation (not necessarily exclusive) of A_R in the specification of A_O . The following expression formalizes this notion:

$$r_{sem}(A_O, A_R) \Rightarrow A_O^{\mathcal{IM}} = C_{A_R}^{\mathcal{IM}_R}, \quad A_R \in C_{A_R} \\ r_{sem} \notin \{\perp, \sqcap\}$$

Let's assume two interpretations \mathcal{IM} and \mathcal{IM}_R of the same domain Δ (which coincides with the *intended meaning* of the ontology) for the ontology and the relational model respectively. The fact that there exists a semantic relation different from the disjunction or overlapping relation between A_O and A_R means that A_R forms part of the set of relational attributes of C_{A_R} equivalent to A_O , that is, that the interpretation functions $\cdot_{\mathcal{IM}}$ and $\cdot_{\mathcal{IM}_R}$ of \mathcal{IM} and \mathcal{IM}_R respectively assign the same subset of $\Delta \times \Delta$ to A_O and to the set C_{A_R} .

Example: The ontology O_1 includes the concept *Publication* and its attribute *MainAuthor*. The relational model M_1 in its turn includes the relation *Articles*, one of whose attributes is *Authors*.

Relational Model M_1 (fragment)
Articles(*Id*, *title*, *authors*)

Ontology O_1 (fragment)
class-def *Publication*
slot-def *MainAuthor*
domain *Publication*

In this case the set of attributes C_{A_R} contains exclusively the attribute *authors*.

Interpretation of the semantic relations between ontology attributes and relations of the relational model.

We can interpret the presence of a semantic relation r_{sem} ⁶ $r_{sem} (\equiv, \sqsubseteq, \text{etc.})$ between the attribute A_O of the ontology O_1 and the relation R of the relational model M_1 in the same way as in the aforementioned case, though now all the attributes of the relation R ($att(R)$) form part of the set C_{A_R} .

$$r_{sem}(A_O, R) \wedge A_O^{\mathcal{IM}} \Rightarrow C_{A_R}^{\mathcal{IM}_R}, \quad att(R) \subseteq C_{A_R} \\ r_{sem} \notin \{\perp, \sqcap\}$$

This kind of semantic mapping shows a *structuring disparity* or, what is the same, that a non-structured element of the ontology (attribute) is confined to a structured element (relation) of the relational model.

⁵ Obtained from the set of lexical relations $r_{lex} (syn, hiper, \text{etc.})$ between the terms $term_{A_O}$ and $term_{A_R}$, which are used to name the attribute A_O of O and the relational attribute A_R of M respectively.

⁶ Obtained from a set of lexical relations $Xr_{lex}^{i,j} (sin, hiper, \text{etc.})$ between the term $term_{A_O}$, used for naming the attribute A_O of O , and the term $term_R$, used for naming the relation R of M .

Example: The ontology O_1 includes the concept *DesktopEquipment* and the attribute *components* of such concept. The relational model M_1 in its turn includes the relation *PCs* and the relation *components*, which are linked to each other through the association relation *PCsComponents*.

Relational Model M_1 (fragment)	Ontology O_1 (fragment)
<i>PCs</i> (<i>Id</i> , <i>name</i> , <i>ReleaseDate</i>)	class-def <i>DesktopEquipment</i>
<i>Components</i> (<i>Id</i> , <i>name</i> , <i>manufacturer</i>)	slot-def <i>hasComponent</i>
<i>PCsComponents</i> (<i>IdPC</i> , <i>IdComponent</i>)	domain <i>DesktopEquipment</i>
fk <i>PCsComponents.IdPC</i>	
references <i>PCs.Id</i>	
fk <i>PCsComponents.IdComponent</i>	
references <i>Components.Id</i>	

Assume a semantic relation of equivalence between the attribute *hasComponents* of O_1 and the relation *Components* of M_1 –inferred obviously from the lexical relation of synonymy between the attribute name (term) *hasComponents* of O and the relation name (term) *Components* of M – then, the following expression will be verified:

$$hasComponents \equiv Components \Rightarrow hasComponents^{IM} = C_{A_R}^{IMR}, \\ \{Id, name, manufacturer\} \subseteq C_{A_R}$$

Confinement of relations

The Semantic Mapper algorithm does not use the linguistic component to discover the mappings between ontology interrelations and elements of the relational model (relations and/or attributes). This is so because the occasions in which a significant name is used for identifying the elements implementing such a name are scarce, as can be seen in the relation *PCsComponents* of the above example, which implements the relation existing between the relations *PCs* and *Components*.

To detect this kind of elements we should rely on the structural component, identifying the so-called “relational paths”, which are links between relations through integrity restrictions (foreign keys)

C.2.5 Description of the Semantic Mapper Algorithm

The algorithm, which will be executed sequentially, is composed of the following steps:

1. Detection of semantic relations between each pair of ontology elements and the relational model terminologies
2. Iterative application of the heuristic rules, until no new results are obtained. These heuristic rules will be in charge of constructing concept and attribute specifications
3. Application of the heuristic rules to explain the relations.

C.2.6 Discovery of semantic relations between terms

In this section we describe the part of the Semantic Mapper algorithm that puts into use the notions of the lexical-semantic correlation between models. The goal here is to discover the semantic relations between elements of an ontology (concepts and attributes) and elements of a relational model (relations and attributes). The procedure here described permits identifying the general semantic relation between two elements of each model.

The type of semantic relations to be discovered ($r_{sem} = \{\equiv, \sqsubseteq, \supseteq, \perp, \sqcap\}$) is more expressive than one degree of “abstract similarity” or of “semantic proximity”, defined as a numeric value in an in a given interval (generally (0,1)), as it occurs in most parts of the Schema Matching. Such expressivity can be exploited by the structural component for constructing specifications.

As it is mentioned in each section, some of the notions proposed in [Magnini *et al.*, 2003], [Bouquet *et al.*, 2003] and [Giunchiglia *et al.*, 2005] have been taken and adapted with the purpose of establishing mappings between relational databases and ontologies.

>From the methodological point of view, the process of discovering semantic relations between elements of two models (a process dealt with in the previous sections) will be implemented in an algorithm that is composed of three basic phases: pre-process, calculus of semantic relations between elements described with simple terms, and propagation of those relations to elements described with compound terms.

Pre-process

The goal of the process here described is to translate a set of relations of informal nature defined over terms into other formal relations and without ambiguity. To do this, we should first translate the terms – generally described in natural language – into logical expressions that permit reasoning over such expressions and, specifically, into a prepositional logic expression that will describe the “entity” (though quite often that prepositional logic expression will be referred to as a concept). This entity should be understood as an element of the model and not as a term.

The pre-process consists of four steps that are inspired in the implementation described in [Giunchiglia *et al.*, 2005]:

Step 1: Division of the terms into indivisible or atomic units. The division of a term (if possible/necessary) is carried out through the analysis of such term following a set of standard annotations. The annotations specifically evaluated are Camel or capital letters at the beginning of each word (example: PartTimeWorker), the use of separating signs, such as “-” or “.” (for example: part-time-worker), or the normal separation of words with blank spaces (for example, part time worker). Additionally, “*multiwords*” (in the Wordnet terminology) or units composed of more than one word can also be detected in this step.

Example: the term “*artificial-intelligent-book*” can be decomposed into two units: *artificial-intelligence* and *book* instead of three units, as we may deduce from analysing the term if we consider only the notation “separator”. - In the same way, “*Black and White movie*” will be decomposed into two units: *Black and White* and *movie*.

In the present implementation of the Semantic Mapper algorithm, the only linguistic resource used by the algorithm for this pre-process is WordNet; however, the effectiveness of its performance can be increased with thesauri, dictionaries or domain ontologies that permit detecting acronyms or *multiwords* specific to a particular domain (for example: “*OEPapers*” could be decomposed into “Ontology Engineering” and “*Papers*” if any of the auxiliary resources knew the acronym “*OE*”).

Step 2: Creation of simple concepts for each simple or atomic term detected in the previous step. As it was mentioned before, “concept” should not be understood in this context as an ontology concept, but as a logical expression that describes an entity or element of a model. Thus, a concept is built as the conglomeration formed by uniting all the synsets⁷ of the lemmas⁸ found in WordNet that correspond to each atomic term [Bouquet *et al.*, 2003].

Example: In the previous step, the term “*funding*” does not require to be decomposed since it is a simple or atomic term. The “concept” it originates (which will be annotated with brackets) is (*funding*) = Of the 8 senses that Wordnet proposes for funding, 2 appear as the noun “funding” and 6 appear as the verb “fund”)

Step 3. Construction of complex concepts taking into account the logical connectives that are deduced from the nexus or union atoms. A complex concept (here understood in the sense described above) is, put

⁷ Set of synonyms (Wordnet terminology)

⁸ The canonical form of a word

simply, the composition of more than one simple concept through logical connectives. Remember that this is a logical prepositional expression. Logical connectives can be obtained through the application of the following heuristic rules:

1. Simple or atomic terms such as *by*, *of*, *from*, *with*, etc. or the absence of a nexus atom are translated into connective or conjunction AND. **Example:** the term “*PicturesFromItaly*” would be decomposed into the atoms “*Pictures*”, “*From*” and “*Italy*”. Two of these atoms would yield the concepts (*Picture*) and (*Italy*) and the third one would serve as the link between the other two, and would be translated according to the previous rule into a connective type AND. The resultant complex concept of the pre-process of the term “*PicturesFromItaly*” would be the expression $[Picture] \wedge [Italy]$.
2. The simple or atomic terms *non*, *no* and *without* are translated into the operator NOT or negation. **Example:** the complex term “*Non Technical*” would be decomposed into the atoms “*Non*” and “*Technical*”; the first one would function as a logical operator translated according to the previous rule used in negation NOT (\neg). The complex concept resultant of the pre-process of the term “*Non Technical*” would be $\neg (Technical)$.
3. The simple or atomic terms *and* or *&* are translated into the operator OR or disjunction. **Example:** the complex term “*PapersAndDeliverables*” would be decomposed into its corresponding atomic terms “*Papers*”, “*And*” and “*Deliverables*”. Two of the atomic terms would yield the concepts (*Paper*) and (*Deliverable*) and the third one would function as a logical connective, which would be translated according to the previous rule of disjunction OR. The complex concept resultant of the pre-process of the term “*PapersAndDeliverables*” would be $(Paper) \vee (Deliverable)$.

The result of pre-processing each term is a “complex concept”, if we understand by this a logical expression (\wedge , \vee , \neg) applied to a set of predicates or “concepts”, being these concepts the conglomeration formed by the union of all the synsets of Wordnet for the lemmas of each of the atomic units, in which such term can be decomposed.

Calculus of the semantic relation between simple concepts

To calculate the semantic relations between simple “concepts” involves applying a set of rules for implementing the notion of the semantic-lexical correlation. For this purpose, the proposal described in [Bouquet *et al.*, 2003] can be used, though any other proposal could be equally valid⁹.

Equivalence (\equiv): Two “concepts” $[A]$ and $[B]$ are equivalent if at least one synset is common to $[A]$ and $[B]$. In other words, it is very likely that two elements are equivalent if their names are synonyms.

Less general (\sqsubseteq): a “concept” $[A]$ is less general than other concept $[B]$ if there is a relation of hyponymy (hypo) between any synset of $[A]$ and $[B]$. In other words, it is very likely that element $[A]$ is less general than element $[B]$ (there is implication between A and B) if the term used to name concept $[A]$ is a hyponym of the term used to name concept $[B]$ (and the same occurs for the opposite case).

More general (\sqsupseteq): A “concept” $[A]$ is more general than other concept $[B]$ if there exists a relation of hyperonymy between any synset of $[A]$ and any of $[B]$.

Disjuncts (\perp): Two concepts $[A]$ and $[B]$ are disjuncts ($[A] \wedge [B] = \perp$) if there is a relation of antonymy (ant) between any synset of $[A]$ and any of $[B]$. In other words, it is very likely that two elements whose names are antonyms be disjuncts.

⁹ The use of WordNet provides satisfying results when creating semantic relations between simple concepts; however, these relations can be equally implemented if they are based on a set of domain ontologies or on any other resources that permit generating such relations through the application of any type of heuristic.

Overlapping (\cap): This is the more general case, in which none of the previous relations can be confirmed because Wordnet does not provide any relation between two “concepts”. Two elements, whose names do not have any of the above relations, will usually have some degree of undetermined intersection (eventually empty).

Table C.2 provides the mappings between lexical and semantic relations described above. In the table we have used the following notation: a represents a synset of the concept $[A]$ and b represents a synset of concept B .

Lexical Rel.	Semantic Rel.
$\exists a a \in [A] \wedge a \in [B]$	$A \equiv B$
$\exists a, b a \in [A] \wedge b \in [B] \wedge hipo(a, b)$	$A \sqsubseteq B$
$\exists a, b a \in [A] \wedge b \in [B] \wedge hiper(a, b)$	$A \sqsupseteq B$
$\exists a, b a \in [A] \wedge b \in [B] \wedge ant(a, b)$	$A \perp B \text{ o } A \cap B = \emptyset$
–	$A \cap B$

Table C.2: Correspondences between lexical and semantic relations.

Calculus of the semantic relation between complex concepts. Propagation.

Once we know the semantic relations between the simple “concepts”¹⁰ that constitute complex terms, we have to infer from these relations the relation between the corresponding complex “concepts”. The following example clarifies this notion.

Example: Suppose the relation between *TemporaryJob* of the relational model MR_1 and the concept *PartTimeOccupations* of the ontology O_1 . To calculate the semantic relation between their respective “complex concepts” we should proceed as in the previous sections, decomposing both terms into their atomic elements¹¹ and calculating for each of them the “associated concept”; and thus we have

$$\begin{aligned} TemporaryJobs &\rightsquigarrow [Temporary] \wedge [Job] \\ PartTimeOccupations &\rightsquigarrow [PartTime] \wedge [Occupation] \end{aligned}$$

We have calculated the semantic relation between each concept using WordNet and following the steps described previously; the results obtained are the followings:

$$\begin{aligned} [Temporary] \cap [Job], \quad [Temporary] \cap [PartTime], \\ [Temporary] \cap [Occupation], \quad [Job] \cap [PartTime], \\ [Job] \equiv [Occupation], \quad [PartTime] \cap [Occupation] \end{aligned}$$

The next question will be, how should be combined the set of semantic relations with basic concepts to infer the semantic relation between $[Temporary] \wedge [Job]$ and $[PartTime] \wedge [Occupation]$?

To answer this question we propose **an algorithm that propagates semantic relations between simple concepts; such algorithm will calculate the corresponding complex concepts**. To start with, we should define a set of rules that should answer the following question: If we know the semantic relation between any pair of simple concepts X_A and X_B , which will be the semantic relation between its intersection $\bigwedge_{i=1}^n x_i$ and $\bigwedge_{j=1}^m x_j$?

¹⁰ We should insist once more that we use the term “concept” in this context because it is convenient for us; therefore, it should not be understood as a concept in a conceptual model but as an entity of any model (to differentiate the concept from the term that identifies it). In our case, it could be a concept or an attribute of an ontology or a relation or an attribute of a relational model.

¹¹ Note that the chain “Part Time” is here considered as one term as it appears in thesauri, domain ontologies and WordNet, and not as two independent words.

We should analyze one by one all the possible cases for $n = 2 \wedge m = 1$ and then, for $n = 1 \wedge m = 2$. We can observe that this proposal is applied to any of the values of m and n .

Case 1: Suppose that $[A]$, $[B]$ and $[C]$ are three concepts such that between $[A]$ and $[B]$ there is relation of semantic equivalence ($[A] \equiv [B]$) What can we say of the relation between $[A] \wedge [C]$ and $[B]$ for each of the possible semantic relations between $[C]$ and $[B]$?

The table below should be interpreted as follows: each cell contains the semantic relation between the concept represented in its row and the concept represented in its column. Recall that the order is important here because the semantic relations defined are not commutative.

	$[B]$	$[B]$	$[B]$	$[B]$	$[B]$
$[A]$	\equiv	\equiv	\equiv	\equiv	\equiv
$[C]$	\equiv	\perp	\sqsubseteq	\supseteq	\sqcap
$[AC]$	\equiv	\perp	\sqsubseteq	\supseteq	\sqcap

Each column (all headed with $[B]$) represents a possible relation between the concepts $[C]$ and $[B]$. (It is obviously a compiling table in which only one relation will be true each time). The third row (headed with $[AC]$)¹² lists each of the semantic relations between $[AC]$ and $[B]$ derived from the relations for each simple concept. Thus, each column should be interpreted as the set of expressions that follow:

The first:

$$([A] \equiv [B]) \wedge ([C] \equiv [B]) \Rightarrow [AC] \equiv [B]$$

The second:

$$([A] \equiv [B]) \wedge ([C] \perp [B]) \Rightarrow [AC] \perp [B]$$

The third:

$$([A] \equiv [B]) \wedge ([C] \sqsubseteq [B]) \Rightarrow [AC] \sqsubseteq [B]$$

The fourth:

$$([A] \equiv [B]) \wedge ([C] \supseteq [B]) \Rightarrow [AC] \supseteq [B]$$

The fifth:

$$([A] \equiv [B]) \wedge ([C] \sqcap [B]) \Rightarrow [AC] \sqcap [B]$$

Each expression is a tautology easily demonstrable in propositional logic¹³ though the most intuitive way to understand the propagation of the semantic relations is probably by establishing an analogy with the set theory. Each concept is compared with a set while the semantic relations are compared with their equivalent in the set theory.

Semantic Relation	Set operations
$[A] \equiv [B]$	$A = B$
$[A] \perp [B]$	$A \cap B = \emptyset$
$[A] \sqsubseteq [B]$	$A \subseteq B$
$[A] \supseteq [B]$	$A \supseteq B$
$[A] \sqcap [B]$	$A \cap B \neq \emptyset$

¹² Abbreviated notation for $([B] \wedge [C])$

¹³ For example, through its truth table associated. Take, for example, the first of the expressions : $Exp = [(a = b) \wedge (c = b)] \rightarrow [(a \wedge c) = b]$

a	b	c	$a = b$	$c = b$	$a = b \wedge c = b$	$a \wedge c$	$(a \wedge c) = b$	Exp
0	0	0	1	1	1	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	0	0	1
0	1	1	0	1	0	0	0	1
1	0	0	0	1	0	0	1	1
1	0	1	0	0	0	1	0	1
1	1	0	1	0	0	0	0	1
1	1	1	1	1	1	1	1	1

For brevity reasons, of each case studied we will show its graphical representation, using notation sets of only one of the expressions analyzed. Fig C.2 shows the situation described by the fifth of the expressions (which corresponds to the fifth column in the table). The sets A and B are coincidental because the concepts $[A]$ and $[B]$ are equivalent. The set C will have some intersection with B because the semantic relation between the concepts $[B]$ and $[C]$ is of overlapping. It is easy to prove that the relation between $A \cap C$ and B is of inclusion: $A \cap C \subseteq B$, which, when translated into semantic relations between concepts, is the same as to affirm that $[A] \wedge [C] \sqsubseteq [B]$ or $[AC] \sqsubseteq [B]$.

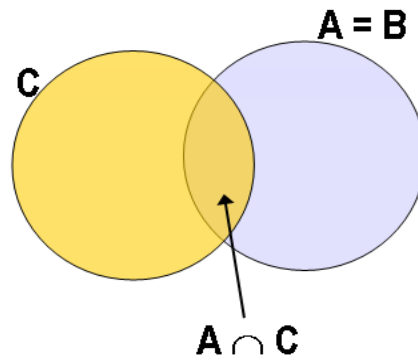


Figure C.2: Example using set notations of the propagation of semantic relations from equivalence.

Case 2. Suppose that $[A]$, $[B]$ and $[C]$ are three concepts such that between $[A]$ and $[B]$ there exists an overlapping semantic relation ($[A] \sqcap [B]$). What can be said of the relation between $([A] \wedge [C])$ and $[B]$ for each of the possible semantic relations between $[C]$ and $[B]$?

As in the other cases, each expression is a tautology easily demonstrable in propositional logic; however, we have wanted to make a more intuitive representation based on the set theory.

	$[B]$	$[B]$	$[B]$	$[B]$	$[B]$
$[A]$	\sqcap	\sqcap	\sqcap	\sqcap	\sqcap
$[C]$	\equiv	\perp	\sqsubseteq	\sqsupseteq	\sqcap
$[AC]$	\sqsubseteq	\perp	\sqsubseteq	\sqcap	\sqcap

Figure C.3 shows graphically the situation described by the second of the expressions (corresponding to the second column in the table). The sets A and B have non-disjunct intersection because between the concepts $[A]$ and $[B]$ there exists some semantic overlapping. The set C has not intersection with B because the semantic relation between the concepts $[B]$ and $[C]$ is of semantic disjunction (though nothing prevents C from having intersection with A in a more general case). It is easy to prove that $A \cap C$ and B are disjoint sets: $(A \cap C) \cap B = \emptyset$, which, when translated into semantic relations between concepts, means that $[A] \wedge [C] \perp [B]$ or $[AC] \perp [B]$.

Case 3: Suppose that $[A]$, $[B]$ and $[C]$ are three concepts such that **between** $[A]$ and $[B]$ **there exists a semantic relation of the less general type** $[A]$, $[B]$ What can be said of the relation between $[A] \wedge [C]$ and $[B]$ for each of the possible semantic relations between $[C]$ and $[B]$?

As in previous cases, the following table compiles each possible relation between the concepts $[B]$ and $[C]$. In the table, the last row (headed with $[AC]$) lists every semantic relation between $[AC]$ and $[B]$, which are derived from the relations for each simple concept by means of propagation rules.

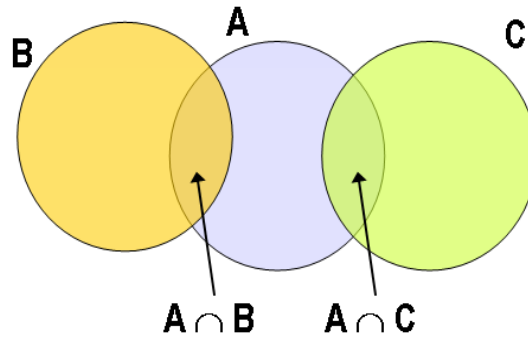


Figure C.3: The set theory is here used to show an example of the propagation of semantic relations starting from semantic overlapping.

	[B]	[B]	[B]	[B]	[B]
[A]	\sqsubseteq	\sqsubseteq	\sqsubseteq	\sqsubseteq	\sqsubseteq
[C]	\equiv	\perp	\sqsubseteq	\supseteq	\sqcap
[AC]	\sqsubseteq	\perp	\sqsubseteq	\sqsubseteq	\sqsubseteq

Thus, each of the rows can be interpreted as the set of expressions that follow next:

The first:

$$([A] \sqsubseteq [B]) \wedge ([C] \equiv [B]) \Rightarrow [AC] \sqsubseteq [B]$$

The second:

$$([A] \sqsubseteq [B]) \wedge ([C] \perp [B]) \Rightarrow [AC] \perp [B]$$

The third:

$$([A] \sqsubseteq [B]) \wedge ([C] \sqsubseteq [B]) \Rightarrow [AC] \sqsubseteq [B]$$

The fourth:

$$([A] \sqsubseteq [B]) \wedge ([C] \supseteq [B]) \Rightarrow [AC] \sqsubseteq [B]$$

The fifth:

$$([A] \sqsubseteq [B]) \wedge ([C] \sqcap [B]) \Rightarrow [AC] \sqsubseteq [B]$$

As in the other cases, each expression is a tautology easily demonstrable in propositional logic.

Figure C.4 shows graphically the situation described by the fourth of the expressions (which corresponds to the fourth column of the table). The set A is included in set B because between concepts $[A]$ and $[B]$ there is a semantic relation of the type “less general”. The set C is a superset of B because the semantic relation that exists between concepts $[C]$ and $[B]$ is of the type “more general”. It is easy to prove that $A \cap C$ is a subset of B : $(A \cap C) \subseteq B$, which, when translated into semantic relations between concepts, means that $[A] \wedge [C] \sqsubseteq [B]$ or $[AC] \sqsubseteq [B]$.

Case 4: Suppose that $[A]$, $[B]$ and $[C]$ are three concepts such that between $[A]$ and $[B]$ there is a semantic relation of the type “more general” ($[A] \supseteq [B]$). What can be said about the relation between $[A] \wedge [C]$ and $[B]$ for each of the possible semantic relations between $[C]$ and $[B]$?

As in the other cases, the table below compiles each possible relation between concepts $[C]$ and $[B]$. In the table the last row (headed with $[AC]$) lists every semantic relation between $[AC]$ and $[B]$ derived from the existing relations for each simple concept by means of propagation rules.

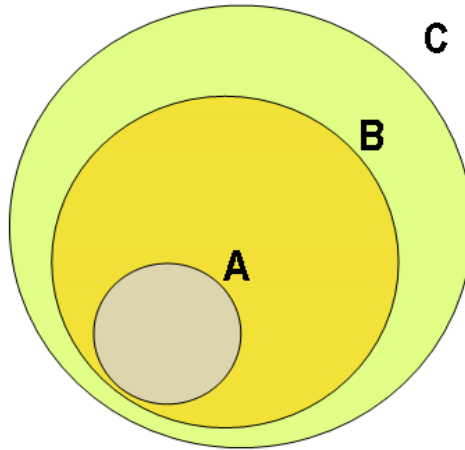


Figure C.4: Example using notation sets of the propagation of semantic relations from the “less general” semantic relation.

	[B]	[B]	[B]	[B]	[B]
[A]	\supseteq	\supseteq	\supseteq	\supseteq	\supseteq
[C]	\equiv	\perp	\sqsubseteq	\supseteq	\cap
[AC]	\equiv	\perp	\sqsubseteq	\supseteq	\cap

Thus each column should be interpreted as the set of expressions that follow:

The first:

$$([A] \supseteq [B]) \wedge ([C] \equiv [B]) \Rightarrow [AC] \equiv [B]$$

The second:

$$([A] \supseteq [B]) \wedge ([C] \perp [B]) \Rightarrow [AC] \perp [B]$$

The third:

$$([A] \supseteq [B]) \wedge ([C] \sqsubseteq [B]) \Rightarrow [AC] \sqsubseteq [B]$$

The fourth:

$$([A] \supseteq [B]) \wedge ([C] \supseteq [B]) \Rightarrow [AC] \supseteq [B]$$

The fifth:

$$([A] \supseteq [B]) \wedge ([C] \cap [B]) \Rightarrow [AC] \cap [B]$$

As in the other cases, each expression is a tautology easily demonstrable in propositional logic.

Fig. C.5 shows graphically the situation described by the fifth expression (corresponding to the fifth column of the table). The set A is a superset of B because between concepts $[A]$ and $[B]$ there exists a semantic relation of the “more general” type. The set C has no empty intersection with B because the semantic relation between concepts $[C]$ and $[B]$ is of overlapping. It is easy to prove that $A \cap C$ has no empty intersection with B , which, when translated into semantic relations, means that $([A] \wedge [C] \cap [B]) \supseteq [AC] \cap [B]$

Case 5: Suppose that $[A]$, $[B]$ and $[C]$ are three concepts such that between $[A]$ and $[B]$ there exists a disjunct semantic relation ($[A] \perp [B]$). What can be said of the relation between $[A] \wedge [C]$ and $[B]$ for each of the possible semantic relations between $[C]$ and $[B]$?

As in the other cases, the table below compiles each semantic relation possible between concepts $[C]$ and $[B]$. In this table, the last row (headed with $[AC]$) lists each of the semantic relations between $[AC]$ and $[B]$ derived from the relations for each simple concept by means of the propagation rules.

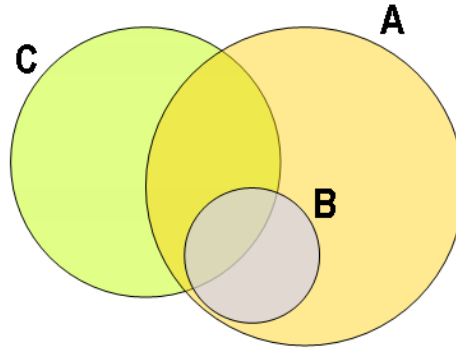


Figure C.5: Example using notation sets of the propagation of semantic relations from the “more general” semantic relation.

	[B]	[B]	[B]	[B]	[B]
[A]	\perp	\perp	\perp	\perp	\perp
[C]	\equiv	\perp	\sqsubseteq	\sqsupseteq	\sqcap
[AC]	\perp	\perp	\perp	\perp	\perp

Thus, each column should be interpreted as the following set of expressions:

The first:

$$([A] \perp [B]) \wedge ([C] \equiv [B]) \Rightarrow [AC] \perp [B]$$

The second:

$$([A] \perp [B]) \wedge ([C] \perp [B]) \Rightarrow [AC] \perp [B]$$

The third:

$$([A] \perp [B]) \wedge ([C] \sqsubseteq [B]) \Rightarrow [AC] \perp [B]$$

The fourth:

$$([A] \perp [B]) \wedge ([C] \sqsupseteq [B]) \Rightarrow [AC] \perp [B]$$

The fifth:

$$([A] \perp [B]) \wedge ([C] \sqcap [B]) \Rightarrow [AC] \perp [B]$$

As in the other cases, each expression is a tautology easily demonstrable in propositional logic.

Fig C.6 shows graphically the situation described by the fourth expression (corresponding to the fourth column of the table). The sets A and B are disjoint because between the concepts $[A]$ and $[B]$ exists a relation of semantic disjunction. The concept $[C]$ is a superset of $[B]$ because the semantic relation between the concepts $[C]$ and $[B]$ is of the “more general” type. It is easy to prove that $A \cap C$ and B are disjoint: $(A \cap C) \cap B = \emptyset$, which, when translated into semantic relations between concepts, means that $[A] \wedge [C] \perp [B]$ or $[AC] \perp [B]$

In table C.3 we can see the set of propagation rules of the semantic relations we have described so far. They are shown as a set of operations for combining semantic relations through the application of the intersections between the elements of the left side of the expression.¹⁴ The expression “vertical propagation” ($\downarrow \wedge$) is used

¹⁴The calculus table

	[B]
[A]	\perp
[C]	\equiv
[AC]	\perp

is equivalent to:

$$([A] \perp [B]) \wedge ([C] \equiv [B]) \Rightarrow [AC] \perp [B]$$

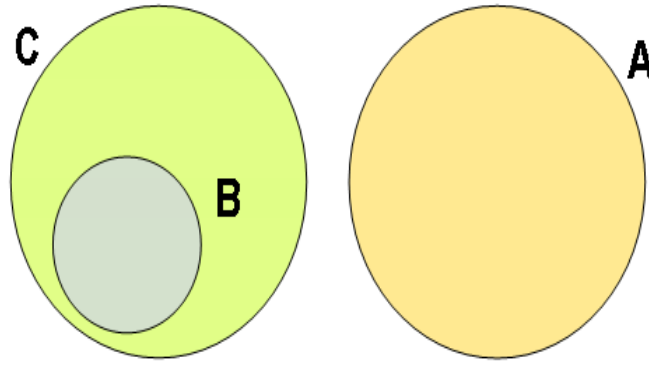


Figure C.6: Example using notation sets of the propagation of semantic relations from the disjunction semantic relation.

$\downarrow \wedge$	\sqcap	\perp	\sqsubseteq	\sqsupseteq	\equiv
\sqcap	\sqcap	\perp	\sqsubseteq	\sqcap	\sqsubseteq
\perp	\perp	\perp	\perp	\perp	\perp
\sqsubseteq	\sqsubseteq	\perp	\sqsubseteq	\sqsubseteq	\sqsubseteq
\sqsupseteq	\sqcap	\perp	\sqsubseteq	\sqsupseteq	\equiv
\equiv	\sqsubseteq	\perp	\sqsubseteq	\equiv	\equiv

Table C.3: Set of “vertical” propagation rules of the semantic relations.

to refer to the layout of the calculus table and in opposition to “horizontal propagations” (\rightarrow^{\wedge}) that come next. It is, in fact, a table¹⁵ -matrix- that, as it can be observed, is symmetric. This is so because the intersection operation has this property ($A \wedge B \Leftrightarrow B \wedge A$).

So far we have described all the possibilities existing for $n = 2 \wedge m = 1$ (*vertical propagations* \downarrow^{\wedge}); the same should be done for $n = 1 \wedge m = 2$ (*horizontal propagations* \rightarrow^{\wedge}) though for brevity reasons, the second block of propagation rules of the semantic relations will not be detailed but just synthesized in table C.4 as a set of operations that combine semantic relations by applying intersections to the elements on the right side of the expression “horizontal propagations” (\rightarrow^{\wedge}).

This table is symmetrical in the case of vertical propagations.

The following examples show the result of applying the propagation algorithm.

Example 1: Suppose that we want to calculate the relation between the terms *SecondHandAutomobile* and

¹⁵ The propagation matrix of the operations should not be confused with the calculus tables presented previously. In this case, the elements of the row and column headings are semantic relations, whereas in the previous tables they were simple concepts.

\rightarrow^{\wedge}	\sqcap	\perp	\sqsubseteq	\sqsupseteq	\equiv
\sqcap	\sqcap	\perp	\sqcap	\sqsupseteq	\sqsubseteq
\perp	\perp	\perp	\perp	\perp	\perp
\sqsubseteq	\sqcap	\perp	\sqsubseteq	\sqsupseteq	\equiv
\sqsupseteq	\sqsubseteq	\perp	\sqsubseteq	\sqsupseteq	\sqsubseteq
\equiv	\sqsubseteq	\perp	\equiv	\sqsubseteq	\equiv

Table C.4: Set of “horizontal” propagation rules of the semantic relations.

Car. To do this we build their associated concepts $([SecondHand] \wedge [Automobile])^{16}$. Then the semantic relations between their elements are calculated with the help of WordNet, which yield $[Automobile] \equiv [Car]$ (these are equivalent elements), $[SecondHand] \sqcap [Car]$. By applying the propagation of the semantic relations described in table C.3, we can deduce that the composition of the relations \equiv and \sqcap yields the relation \sqsubseteq as a result; therefore, $[SecondHand] \wedge [Automobile] \sqsubseteq [Car]$, or, what is the same, *Car* is more generic than *SecondHandAutomobile*.

Similarly, (and using now the propagation table C.4) it can be proved that the relation between *Automobile* and *PinkCar* is $[Automobile] \sqsupseteq [pink] \wedge [Car]$, and thus *Automobile* is **more generic** than *PinkCar*.

Example 2: The relation between *SoccerTeam* and *SportTeam* will be $[Soccer] \wedge [Team] \sqsubseteq [Sport] \wedge [Team]$ because $[Soccer] \sqsubseteq [Sport]$ and $([Team] \equiv [Team])$. It is easy to prove by means of the propagation algorithm that *SpanishSoccerTeam* is **less generic** than (\sqsubseteq) *SportTeam*; equally, *SoccerTeam* has some overlapping with (\cap) *AmericanSportTeam*.

C.2.7 Application of heuristics for constructing specifications

The semantic relations between elements of the ontology and the relational model discovered by the linguistic component will serve as a base for constructing specifications of ontology concepts and attributes.

For each case of semantic relation between elements of the ontology and elements of the relational model according to the type of element, the elements appear confined to a relational model if the following heuristics are defined:

Heuristics for relations between concepts of the ontology and relations of the relational model

Table C.5 describes the heuristics associated with each possible semantic relation between a concept C_O of an ontology O and a relation R_M of the relational model M .

Semantic Relation	Inferred elements for $\mathcal{E}_M(C_O)$	Desc.
Equivalence ($C_O \equiv R_M$)	$Key(\mathcal{E}_M(C_O)) = pk(R_M)$ $Rel(\mathcal{E}_M(C_O)) = \{R_M\}$ $e_{C_O}^{Id} := exp(pk(R_M))$	Desc. 1
Implication ($C_O \sqsubseteq R_M$)	$Key(\mathcal{E}_M(C_O)) = pk(R_M)$ $Rel(\mathcal{E}_M(C_O)) = \{R_M\}$ $e_{C_O}^{Id} := exp(pk(R_M))$ $e_{C_O}^{Cond} := exp(att(R_M))$ o bien $Key(\mathcal{E}_M(C_O)) \perp pk(R_M)$ $Rel(\mathcal{E}_M(C_O)) = \{R_M\}$ $e_{C_O}^{Id} := exp(att(R_M) \text{ not in } pk(R_M))$	Desc. 2
Implication ($C_O \sqsupseteq R_M$)	$pk(R_M) \subseteq Key(\mathcal{E}_M(C_O))$ $R_M \in Rel(\mathcal{E}_M(C_O))$	Desc. 3
Empty intersection ($C_O \perp R_M$)	$R_M \notin Rel(\mathcal{E}_M(C_O))$	Desc. 4

Table C.5: Heuristics associated with each possible semantic relation between concepts and relations.

1. If the ontology concept C_O is equivalent to the relation R_M of the relational model, then the concept C_O is probably confined exclusively to the relation R_M ; therefore, the key of the specification will

¹⁶ Recall that $[SecondHand] \wedge [Automobile]$ is the expanded notation for $[SecondHand \text{ Automobile}]$ and $[Car]$

coincide with the primary key of R_M and, consequently, the expression $\mathcal{E}_M(C_O)$ will be defined over the attributes that form such key.

2. If the ontology concept C_O is less generic than the relation R_M of the relational model, two situations can be encountered
 - The concept C_O is confined exclusively to relation R_M , but if the definition of the subset of the individuals of R_M that will produce instances of C_O is required, then a definition of $e_{C_O}^{Id}$ will be given.
 - The concept C_O is confined to parts of the attributes of R_M and uses a different key than the concept R_M does, because this concept probably defines a partial view of the relation.
3. If the ontology concept C_O is more generic than the relation R_M of the relational model, then all the individuals of the relation will yield instances of the concept; therefore, it is not necessary to define $e_{C_O}^{Cond}$, and R_M will be one of the relations where C_O is confined.
4. If the ontology concept C_O is disjunct with the relation R_M of the relational model, then such relation will not form part of the concept specification.

Heuristics for relations between concepts of the ontology and relations of the relational model

Table C.6 describes the heuristics associated with each possible semantic relation between a concept C_O of the ontology O and an attribute A_M of the relational model M .

Semantic Relation	Inferred elements for $\mathcal{E}_M(C_O)$	Desc.
Equivalence ($C_O \equiv A_M$)	$Key(\mathcal{E}_M(C_O)) = A_M$ $e_{C_O}^{Id} := exp(A_M)$	Desc. 1
Implication ($C_O \sqsubseteq A_M$)	$Key(\mathcal{E}_M(C_O)) = A_M$ $e_{C_O}^{Id} := exp(A_M)$ $e_{C_O}^{Cond} := exp(A_M)$	Desc. 2
Implication ($C_O \sqsupseteq A_M$)	$A_M \subseteq Key(\mathcal{E}_M(C_O))$	Desc. 3
Empty intersection ($C_O \perp A_M$)	$R_M \notin Rel(\mathcal{E}_M(C_O)) A_M \in att(R_M)$	Desc. 4

Table C.6: Heuristics associated with each possible semantic relation between concepts and relational attributes.

1. If the ontology concept C_O is equivalent to the attribute A_M of the relational model, then the concept C_O is probably confined to this attribute exclusively. Consequently, the only relation implied in the specification of C_O will be the R_M that contains A_M .
2. If the ontology concept C_O is less generic than the attribute A_M of the relational model, then we have a scenario similar to the previous one but with the difference that in this new scenario we need to define an expression of condition $e_{C_O}^{Cond}$ over the attribute A_M .
3. If the ontology concept C_O is more generic than the attribute A_M of the relational model, then all the individuals of the relation will produce instances of the concept, and thus it will be necessary to define $e_{C_O}^{Cond}$.
4. If the ontology concept C_O is disjunct with the attribute A_M of the relational model, such attribute will not be part of the concept specifications.

Heuristics for relations between ontology attributes and attributes of the relational model

Table C.7 compiles the heuristics associated with each possible relation between an attribute A_O of the ontology O and an attribute A_M of the relational model M .

Semantic relation	Inferred elements for $\mathcal{E}_M(A_O)$	Desc.
Equivalence ($A_O \equiv A_M$)	$e_{A_O}^{Cond} := (true)$ $e_{A_O}^{Trf} := (\mathbf{constant}(A_M))$	Desc. 1
Implication ($A_O \sqsubseteq A_M$) \circ ($A_O \sqsupseteq A_M$)	$e_{A_O}^{Cond} := (\mathbf{f}(A_M))$ $e_{A_O}^{Trf} := (\mathbf{f}(A_M, \dots))$	Desc. 2
Empty intersection ($A_O \perp A_M$)	A_M does not take part in $\mathcal{E}_M(A_O)$	Desc. 3

Table C.7: Heuristics associated with each possible relation between ontology attributes and relational attributes.

1. If the ontology attribute A_O is equivalent to the relational attribute A_M , then the transformation function will be **constant**, and the value A_M is left without modification.
2. If the ontology attribute A_O is more or less general than the relational attribute A_M , then A_O will require some transformation to obtain the A_O values.
3. If the ontology attribute A_O is disjunct with the relational attribute A_M , then A_M will not form part of the A_O specification

Heuristics for relations between ontology attributes and relations of the relational model

Table C.8 compiles the heuristics associated with each possible semantic relation between an attribute A_O of the ontology O and a relation A_M of the relational model M .

Relación semántica	Elementos inferidos para $\mathcal{E}_M(A_O)$	Desc.
Equivalence ($A_O \equiv R_M$)	$e_{A_O}^{Cond} := (true)$ $e_{A_O}^{Trf} := (\mathbf{f}(att(R_M)))$	Desc. 1
Implicación ($A_O \sqsubseteq R_M$) \circ ($A_O \sqsupseteq R_M$)	$e_{A_O}^{Cond} := (\mathbf{f}(att(R_M)))$ $e_{A_O}^{Trf} := (\mathbf{f}(att(R_M)))$	Desc. 2
Intersección vacía ($A_O \perp R_M$)	R_M no participa en $\mathcal{E}_M(A_O)$	Desc. 3

Table C.8: Heuristics associated with each possible semantic relation between ontology attributes and relation of the relational model.

1. If the ontology attribute A_O is equivalent to the relation R_M , then A_O is probably confined to such relation. The transformation function will be any \mathbf{f} that combines the values of the different attributes of R_M ($att(R_M)$)
2. If the ontology attribute A_O is more or less general than the relation R_M , then the attributes of R_M will require some transformation to obtain the values of A_O . It will probably be necessary to define also $e_{C_O}^{Cond}$ over the attributes of R_M ($att(R_M)$).

3. If the ontology attribute A_O is disjunct with the relation R_M , then the attributes R_M will not form part of the A_O specification.

C.2.8 Heuristic specification of ontology relations

The specification of ontology relations is carried out according to a heuristic procedure based on the search of graphs among concept specifications. Given the ontology concepts A and B for which the specifications $\mathcal{E}_M(A)$ and $\mathcal{E}_M(B)$ exist, the specification of the relation R (also of the ontology) whose domain is A and whose range is B will be carried out **by searching a minimum path** (implicit or explicit) among the relations implied in the specifications of A $Rel(\mathcal{E}_M(A))$ and B , $Rel(\mathcal{E}_M(B))$. Then, **the notion of a graph of restrictions for referential integrity in a relational model** and that of the extended graph will be formalized. These graphs are non-directed graphs and capture the links between relations; the graphs are produced by foreign keys and implicit relations (the extended graph), and they are built as follows:

1. Each relation R of the model will produce a graph node.
2. Each explicit foreign key $fki(R_1.A_{R_1}, R_2.A_{R_2})$ will produce an arc between the two relations R_1 and R_2 that communicates.
3. When the graph is the extended one, potentially implicit the foreign keys will also be considered.

Over a graph of the type described above, we can define a explicit path C_{Ex} between the nodes R_1 and R_2 of length n ($long(C) = n$) as an ordered sequence of n arcs A_1, \dots, A_n of such graph, which forms a non-cyclic path between the given node R_1 and R_2 .

Similarly, an implicit path C_{Im} of length n ($long(C_{Im}) = n$) is defined between the nodes R_1 and R_2 in the same way as in the previous case but now over the extended graph, that is, the graph that includes the arcs due to potentially implicit foreign keys.

Bibliography

- [Barrasa, 2007] J. Barrasa. *Modelo para la definición automática de correspondencias semánticas entre ontologías y modelos relacionales*. PhD thesis, Facultad de Informatica, Universidad Politecnica de Madrid, Madrid, Spain, March 2007.
- [Bouquet *et al.*, 2003] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In Katya Sycara and Dieter Fensel, editors, *Proceedings of the Second International Semantic Web Conference*, number LNCS 2870 in Lecture Notes in Computer Science, pages 130–145. Springer Verlag, 2003.
- [Chalupsky, 2000] Hans Chalupsky. Ontomorph: A translation system for symbolic logic. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, San Francisco, CA, 2000. Morgan Kaufmann.
- [Corcho, 2005] Oscar Corcho. *A layered declarative approach to ontology translation with knowledge preservation*. IOS Press, Amsterdam, The Netherlands, 2005.
- [Dou *et al.*, 2003] D. Dou, D. McDermott, and P. Qi. Ontology translation on the semantic web. In *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics*, pages 952–969, Catania, Sicilia, Italy, 2003.
- [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, Thanh Le Bach, Jesús Barrasa, Paolo Bouquet, Jan De Bo, Rose Dieng-Kuntz, Marc Ehrig, Manfred Hauswirth, Mustafa Jarrar, Ruben Lara, Diana Maynard, Amedeo Napoli, Giorgos Stamou, Heiner Stuckenschmidt, Pavel Shvaiko, Sergio Tessaris, Sven Van Acker, and Ilya Zaihrayeu. State of the art on ontology alignment. deliverable 2.2.3, 2004.
- [Euzenat *et al.*, 2005] Jérôme Euzenat, Philippe Guérin, and Petko Valtchev. Oia in the oaei 2005 alignment contest. In Ben Ashpole, Jérôme Euzenat, Marc Ehrig, and Heiner Stuckenschmidt, editors, *Proc. K-Cap 2005 workshop on Integrating ontology, Banff (CA)*, pages 97–102, 2005.
- [Euzenat *et al.*, 2007] Jérôme Euzenat, Antoine Isaac, Christian Meilicke, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Svab, Vojtech Svatek, Willem Robert van Hage, and Mikalai Yatskevich. Results of the ontology alignment evaluation initiative 2007. In Pavel Shvaiko, Jérôme Euzenat, Fausto Giunchiglia, and Bin He, editors, *Proc. 2nd ISWC 2007 international workshop on ontology matching (OM)*, Busan (KR), pages 96–132, 2007.
- [Euzenat, 2001] Jérôme Euzenat. Towards a principled approach to semantic interoperability. In *Proceedings of the IJCAI workshop on Ontologies and information sharing*, Seattle (WA US), 2001.
- [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.

- [Giunchiglia *et al.*, 2005] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, and M. Uschold, editors, *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.
- [Grosso *et al.*, 1998] William E. Grosso, John H. Gennari, Ray W. Ferguson, and Mark A. Musen. When knowledge models collide (how it happens and what to do). In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW 98)*, Banff, Canada, April 1998.
- [Haase *et al.*, 2006] Peter Haase, Pascal Hitzler, Sebastian Rudolph, Guilin Qi, Marko Grobelnik, Igor Mozetič, Damjan Bojadžiev, Jerome Euzenat, Mathieu d'Aquin, Aldo Gangemi, and Carola Catenacci. Context languages – state of the art. Deliverable D3.1.1, NeOn, 2006.
- [Hammer and McLeod, 1993] Joachim Hammer and Dennis McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Journal for Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.
- [Kim and Seo, 1991] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multi-database systems. *Computer*, 24(12):12–18, 1991.
- [Klein, 2001] Michel Klein. Combining and relating ontologies: an analysis of problems and solutions. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA, August 4–5, 2001.
- [Magnini *et al.*, 2003] Bernardo Magnini, Luciano Serafini, and Manuela Speranza. Making explicit the semantics hidden in schema models. In *Proceedings of the Workshop on Human Language Technology for the Semantic Web and Web Services*, Sanibel Island, Florida, USA, 2003.
- [Rahm and Bernstein, 2001] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
- [Tamma, 2001] Valentina Tamma. *An ontology model supporting multiple ontologies for knowledge sharing*. PhD thesis, University of Liverpool, 2001.
- [Visser *et al.*, 1997] Pepijn R. S. Visser, Dean M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.