Expressive alignment language and implementation

Jérôme Euzenat, François Scharffe, Antoine Zimmermann

HAL Id: hal-00822892
https://hal.inria.fr/hal-00822892

Submitted on 15 May 2013

D2.2.10: Expressive alignment language and implementation

Coordinator: Jérôme Euzenat (INRIA)
François Scharffe (U. Innsbruck), Antoine Zimmermann (INRIA)

Abstract.
This deliverable provides the description of an alignment language which is both expressive and independent from ontology languages. It defines the language through its abstract syntax and semantics depending on ontology language semantics. It then describes two concrete syntax: an exchange syntax in RDF/XML and a surface syntax for human consumption. Finally, it presents the current implementation of this expressive language within the Alignment API taking advantage of the OMWG implementation.
Keyword list: ontology matching, ontology alignment, alignment language, expressive alignments, alignment semantics.

Document Identifier	KWEB/2004/D2.2.10/1.0
Project	KWEB EU-IST-2004-507482
Version	1.0
Date	August 31, 2007
State	final
Distribution	public

Knowledge Web Consortium

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Work package participants

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

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

Changes

Version	Date	Author	Changes
0.1	13.02.2006	Jérôme Euzenat	creation
0.2	21.09.2006	Antoine Zimmer-mann	fill in the work on syntax and semantics
0.3	17.03.2007	Jérôme Euzenat	add work on the OWL ontology
0.4	30.04.2007	Jérôme Euzenat	redraw prictures checked grammar
0.5	09.05.2007	François Scharffe	aligned grammar with implementation
0.6	10.06.2007	Jérôme Euzenat	improved language descriptions
0.7	15.06.2007	Jérôme Euzenat	improved semantics and quality assessment ready
0.8	10.07.2007	Jérôme Euzenat	further improvements
0.9	16.07.2007	Jérôme Euzenat	last checks
1.0	31.08.2007	Jérôme Euzenat	taken quality control comments into account

Executive Summary

Interoperability between heterogeneous systems can be achieved by matching the ontologies used by these systems. When one wants to exchange or manipulate matching results, called alignments, this requires an alignment language. There are some alignment languages available (discussed in Deliverable 2.2.6), but they usually lack at least one of the following features: being independent from the ontology languages (so that the ontologies can be written in different languages) and being expressive enough for dealing with complex cases. We provide in introduction a non exhaustive but compelling set of examples that illustrates the need for an expressive language.

In this deliverable we address these issues by designing a new alignment language (sketched in D2.2.6). This language puts together the expressiveness of a language designed by the Ontology Matching Working Group and the openness of the Alignment API and format.

We first define an abstract syntax and semantics for such a language based on our work on alignment semantics. This semantics covers all constructors of the proposed language in a description logic style. Moreover, it is defined "in function of" the semantics of ontology languages without any prior knowledge of them. Indeed, the semantics only requires that ontology languages have a model theoretic semantics. It defines the semantics of the alignment language constructs as a function of the semantics of the aligned ontologies.

Then we provide two concrete syntax for this language. The offered syntax serve different purposes: the abstract syntax is a minimal syntax for supporting the definition of the language semantics; the exchange syntax is an XML syntax facilitating storage and exchange of alignments among programs; the surface syntax aims at providing a more readable view of the alignments to human users. The three syntaxes are equivalent in the sense that they allow to express the same alignments.

Finally, we present the implementation support for this language: it builds on the two code bases of the OMWG working group and the Alignment API and is embedded in both systems. It provides parsing and serialising of the two concrete syntax. Loaded alignments are expressed within the Alignment API and thus benefit from all the API services: serialising the alignments in various operational languages, trimming the alignments under a threshold, evaluating an alignment against some known reference alignment, storing alignments on persistent storage, etc.

The future of this work is first to be thoroughly tested and widely adopted. Providing this language together with the WSMT software and the Alignment API is a promise of wide dissemination already. If there is enough interest, this language will be the first candidate for standardising because of its main features: expressiveness, clear semantics, implementation available and wide use.

Contents

Chapter 1

Introduction

In a general sense, an ontology alignment can express any kind of semantic relationship existing between the entities of two ontologies[1]. The difficulty of representing semantic relations is increased in heterogeneous knowledge-based systems like the semantic web because of the diversity of ontology representation languages and differences in conceptualisation, scope, scale, or granularity. So we motivate the need for an expressive alignment language with several ontology mediation scenarios in § 1.1, and a concrete example that focus on two wine-related ontologies, in § 1.2. This example illustrates the expressiveness requirements for alignment languages. Based on these, Chapter 2 defines the abstract syntax and model theoretic ontology semantics of such a language. Chapter 3 offers two operational syntax for the language that are used in tools implemented it and presented in Chapter 4.

1.1 Ontology mediation scenarios

Ontology mediation as a generic term gathers a set of techniques needed to achieve interoperability in semantically-enabled systems. In [Scharffe *et al.*, 2005], a set of use cases of ontology mediation is identified as follows:

Query rewriting: An application uses data described according to a source ontology o and has to answer a query q written in terms of o. It may however need to evaluate the query against data described using another ontology o', e.g., by querying another source. The query q needs to be rewritten in a query q' expressed in terms of o'. In order to achieve this, a query rewriting system needs the *correspondences* existing between the concepts and properties defined in o and o'. They are obtained as an alignment A through a matching process (Matcher) as illustrated in Figure 1.1. A generator provides a query $mediator$ from A which is able to transform q into a query q' expressed with regard to o'. Once the query rewritten and addressed to the target database, the resulting instances may have to be processed using instance transformation and instance mediation techniques described in the two following paragraphs.

---

[1]Ontology is here used in a broad sense: they can be databases, terminologies, as well as ontologies. Concerning the language syntax presented here, the only condition is that the entities be accessible by URI. With regard to semantics, their ontology languages must have a model-theoretic semantics.
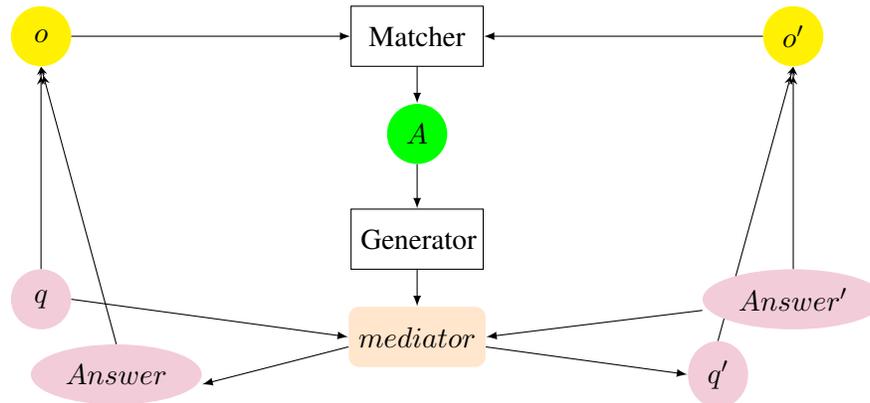
Figure 1.1: Query rewriting illustration (inspired from [Euzenat and Shvaiko, 2007]).

Instance translation: Instance descriptions, i.e., assertions about instances involving concepts and properties of one ontology o, may have to be translated for being considered in the context of an application using another ontology o'. This is typically the case in semantic web service composition. The instance translation process depicted in Figure 1.2 uses the alignment A between o and o'. This supposes that the instance translation process can be derived from this alignment. We refer the reader to § 1.2.3 for a concrete example on instance transformation. Once the instances translated, an operation must be performed to verify if there are some duplicates between the local and transformed instances, i.e., if one of the transformed instances from o is not equivalent to an already existing instance in o'. This operation is called instance mediation.
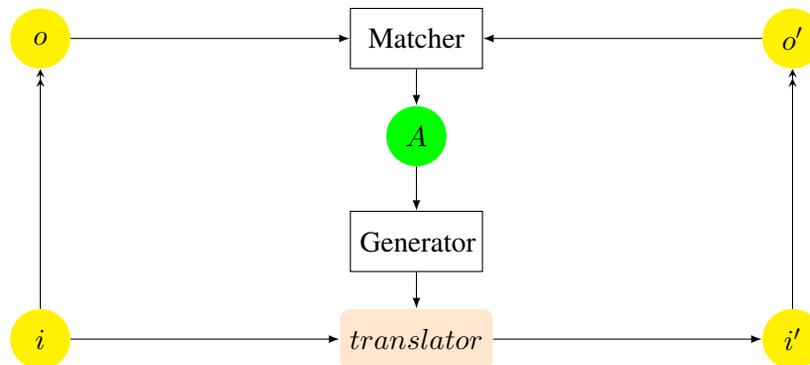
Figure 1.2: Instance translation illustration (inspired from [Euzenat and Shvaiko, 2007]).

Instance mediation: Comparing two instances in order to find out if they must be interpreted as the same individual, and eventually unifying them, is the goal of instance mediation (also called instance identification or record linkage in databases). In that perspective we can distinguish between two phases in the instance mediation process: first the instances must be recognised to be referring to the same individual; once two instances are identified as being the same, they need to be merged in one new instance combining the properties of both. In-

stance mediation is necessary in a query rewriting scenario once the target ontology queried and the set of returned instances transformed in terms of the source ontology. Figure 1.3 illustrates the instance mediation process. It takes advantage of the alignment A between ontology o and o' to help identifying instances. This can happen when the alignments provides correspondences between classes (or tables) and properties involved as keys.

Figure 1.3: Instance mediation illustration.

This set of use cases shows the centrality of alignments in ontology mediation. Further such examples are provided in [Euzenat and Shvaiko, 2007]. The Web Ontology Lang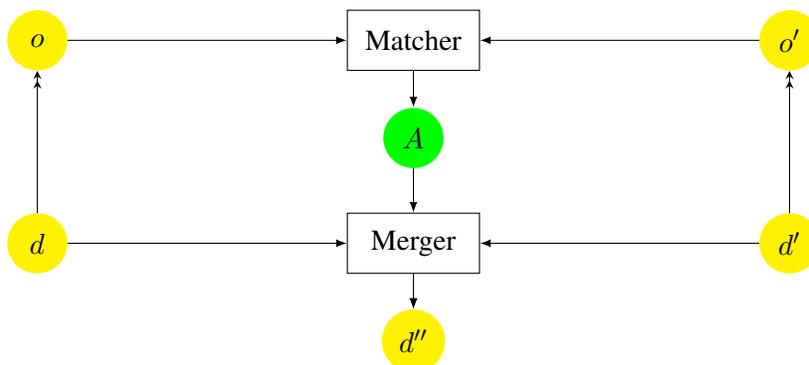uage OWL [Dean and Schreiber (eds.), 2004] includes a few constructs to specify equivalence or subsumption between classes and relations but as we will see in § 1.2 these are not sufficient to specify the complex correspondences that can arise when aligning two ontologies. The next section provides examples of such correspondences.

1.2   Motivating examples

This section shows, from an example, what kind of correspondences must be expressible between ontological entities. We align the popular "Wine ontology"[2], with the "Ontologie du vin"[3]. We will refer to these two ontologies respectively by `Wine` and `Vin` in the following. `Wine` is written in OWL, while `Vin` is written in WSML, the Web Services Modeling Language [Lausen *et al.*, 2005]. The ontologies both represent important properties related to wine like the geographic origin, colour, taste, and the type of grapes used to make it, and so forth. As trying to align the two ontologies, we show that simple one-to-one correspondences between entities are not sufficient to represent the full alignment between these ontologies.

This alignment could be necessary in a scenario of a wine trading service for example, gathering data from multiple semantically described information sources.

1.2.1   Are you more Red or White wine? (Subsumption Relations)

There is an obvious equivalence correspondence between each main concept of the two ontologies both representing a wine. This correspondence would have easily been detected by an algorithm

---

[2]Accessible at `http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#`
[3]Accessible at `http://sw.deri.org/~francois/ontologies/OntologyDuVin.wsml`

using a multilingual component by looking at the concepts labels: "`Wine`" in `Wine` and "`Vin`" in `Vin`. However, the multiple properties of each concept are requiring a deeper investigation and sometimes more complex relations than simple one to one equivalent correspondences.

The colour of a wine is in `Wine` represented through the class "`WineColor`" itself defined by three instances: "`White`", "`Rose`" and "`Red`". In `Vin` the class "`Type`" has nine instances describing the type of a wine. Red, Rose and White are among these instances together with other types of less common wines. In `Wine` the property "`hasColor`" relates a wine to its colour as defined above, while in `Vin` the property "`type`" has this role. There is a subsumption correspondence between these two properties as there is a subsumption correspondence between "`WineColor`" in `Wine` being subsumed by "`Type`" in `Vin`. For the alignment to be complete the corresponding instances need to be related. The instances left over in "`Type`" are part of a domain which is not represented in `Wine`. Figure 1.4 shows part of the set of correspondences we have just defined.
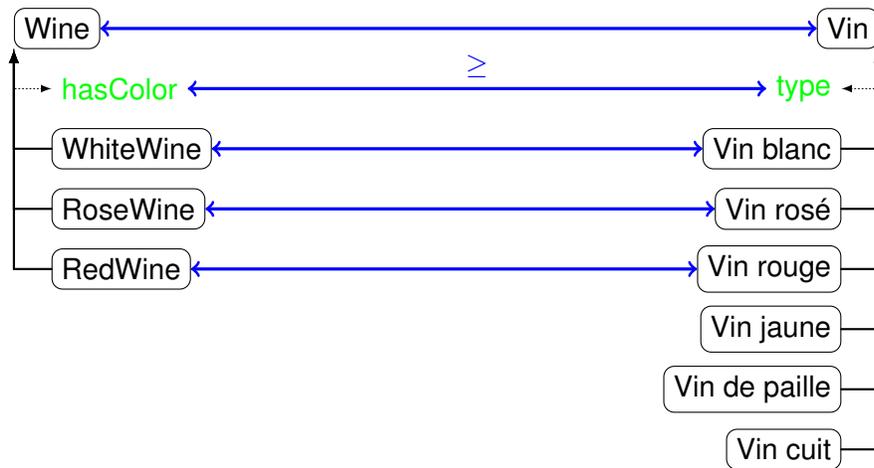
Figure 1.4: Correspondences between wines and their type or colour.

We provide the translation of these correspondences in first-order logic to help readers understanding the meaning of these pictures. Like in alignments, only the correspondences, i.e., arrows and frames, are provided in this format. The ontologies themselves, e.g., subsumption assertions, are not part of the alignment. These first-order logic expressions quatify over a set of individuals whose interpretation is assumed common, e.g., $\forall x, Wine(x) \Leftrightarrow Vin(x)$ means that any x classified as $Wine$ in the first ontology, is to be classified as Vin in the second one and vice versa.

In the present case, these correspondences can be trivially transcribed as:

$$\forall x, [Wine(x) \Leftrightarrow Vin(x)]$$
$$\forall x, [WhiteWine(x) \Leftrightarrow Vin\,blanc(x)]$$
$$\forall x, [RoseWine(x) \Leftrightarrow Vin\,rosé(x)]$$
$$\forall x, [RedWine(x) \Leftrightarrow Vin\,rouge(x)]$$
$$\forall x, y, [hasColor(x, y) \Leftarrow type(x, y)]$$

1.2.2   Bordeaux wine is the best (Value Restriction)

We illustrate, in the following, a need to restrict the set of entities entering the scope of a particular correspondence. A Bordeaux wine is modelled as an instance of the class "`BordeauxWine`" in `Wine`, while in `Vin` the property "`terroir`"[4] relates a wine to an instance of the "`Terroir`" concept. To specify a correspondence between wines elaborated in the Bordeaux region we need to restrict the scope of the `Vin` class to those instances which are in the relation "`terroir`" with values `Bordeaux`.
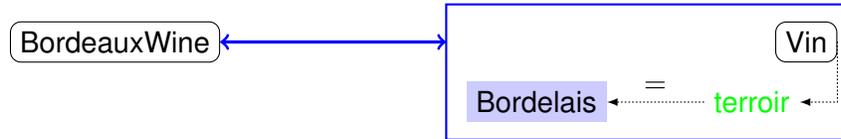
Figure 1.5: Correspondence between wines from Bordeaux.

In predicate calculus, this can be expressed by:

$$\forall x, [BordeauxWine(x) \Leftrightarrow Vin(x) \wedge terroir(x, < \#Bordelais >)]$$

with $< \#Bordelais >$ a particular individual of ontology o'.

Note that there is another kind of compound object that we call context. There are subtle differences between the two expressions of Figure 1.6:
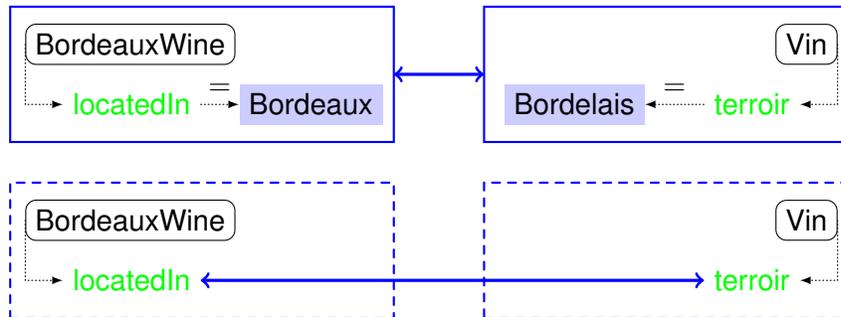
Figure 1.6: Two kinds of compound object put in correspondence: constraint on matched objects and context of matched objects.

The first part expresses that `Wines` located in `Bordeaux` are equivalent to `Vin` whose `terroir` is the `Bordelais` area. By contrast the second part expresses that in the context of `Wines` and `Vin`, `locatedIn` is equivalent to `terroir`.

The expression of the first part corresponds to the expression of Figure 1.5, while the second part can be expressed as:

$$\forall x, [(BordeauxWine(x) \wedge Vin(x))$$
$$\Rightarrow (\forall y, locatedIn(x, y) \Leftarrow terroir(x, y)) \wedge (\forall y, locatedIn(x, y) \Rightarrow terroir(x, y))]$$

Of course, these two kinds of objects can be mixed in the same correspondence.

---

[4]*Terroir* is a French word for a particular agricultural region.

1.2.3 But quality also depends on the vintage year (Type Conversion)

An important aspect of a wine is the year it has been worked out. Both ontologies represent this data using a class named "`VintageYear`" in `Wine` and "`Millésime`" in `Vin`. Both classes have a property pointing to the value of the year. However in `Wine` the year is modelled using an integer while in `Vin` a date type is used. The correspondence should indicate in this case what is the relations between the instance values, i.e., which transformations are required to correctly map equivalent values.
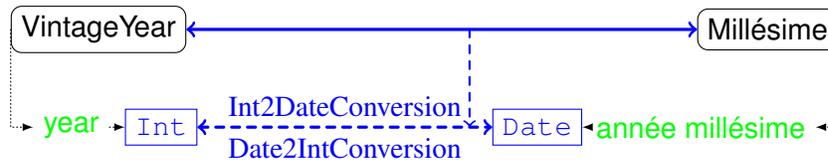
Figure 1.7: Correspondence between wine vintage years.

This can be expressed as the following:

$$\forall x, [(VintageYear(x) \Leftrightarrow Millésime(x))$$
$$\wedge(VintageYear(x) \wedge \exists y : Int; year(x,y) \Rightarrow annéemillésime(x, Int2DateConversion(y))$$
$$\wedge(Millésime(x) \wedge \exists y : Date; annéemillésime(x,y) \Rightarrow year(x, Date2IntConversion(y))]$$

1.2.4 Some may prefer locally grown wines (Path equations)

Some customers prefer wine that is bottled and sold directly from the producer ("`LocallyGrownWine`"). Our `Vin` ontology does not feature this category. However, it could be possible to match this `LocallyGrownWine` class with a restriction of the `Vin` class, namely that its `propriétaire` (owner) and `négociant` (first seller) are the same.
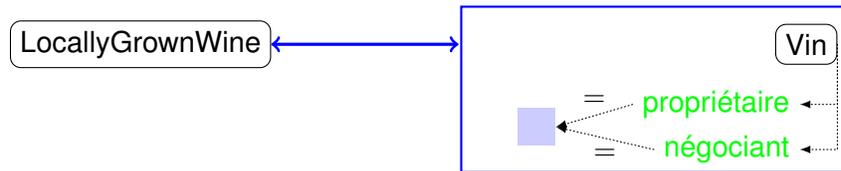
Figure 1.8: Correspondence with constraint relations.

This can easily be rendered as:

$$\forall x, [LocallyGrownWine(x) \Leftrightarrow (Vin(x) \wedge \exists y; [propriétaire(x,y)) \wedge négociant(x,y)])]$$

The correspondences presented in this section are natural and can be expected to occur frequently when mediating between ontologies. Even if they are easy to understand, they require the use of an expressive formalism in order to be expressed correctly.

1.3　Synthesis

Uncoupling alignments from ontologies brings a solution to the integration of the heterogeneous ontologies described using different formalisms. But it also requires expressiveness: all examples above are calling for an expressive alignment language. In deliverable 2.2.6, we surveyed existing such languages. We do not reproduce this survey here. It suggested, in particular, to merge the Ontology Mapping Language [Scharffe and de Bruijn, 2005] developed by the Ontology Management Working Group, and the Alignment format [Euzenat, 2004] developed by INRIA. This would benefit from the expressiveness of the OMWG Mapping language and the openness of the Alignement API.

Both languages are already independent from concrete ontology representation languages but provided different facilities: the Alignment format has been designed as an extensible framework for expressing alignments offering an operational implementation for manipulating the alignments while the OMWG format is an expressive alignment format requiring more complex parsing and rendering procedures. The Alignment format is used in several systems and in particular in the OAEI initiative while the OMWG format is used in the WSMT tool for web service manipulation. Merging both formats allows expressive OMWG alignments to be considered as Alignments in the Alignment format and benefits from all the tools built around the Alignment API. On the opposite, the Alignment API can take advantage of an expressive format.

This deliverable presents a language, result of the integration of both formalisms, for expressing all kinds of alignments presented in this chapter independently from any ontology representation language. This language is presented through its syntax and semantics. It is illustrated through the examples given before and the implementation of this language is finally described.

Chapter 2

Abstract syntax and semantics

This chapter describes the language we designed to express ontology alignments. It offers an expressive language following the mapping language of [Scharffe and de Bruijn, 2005] as considered in [Euzenat *et al.*, 2005a] and its semantics is defined independently of any ontology representation language as sketched in [Zimmermann and Euzenat, 2006].

We first introduce an abstract syntax for this language allowing the expression of the examples of the previous chapter. We then explain the principles used for defining a semantics independent from the ontology languages and provide the semantics of this expressive alignment language according to the abstract syntax.

2.1 Abstract syntax

We will define a simple language with maximal expressivity: our goal is to provide the constructors for expressing these correspondences and defining their meaning. Of course, for complexity reasons, it would be natural to reduce the extent of the language. For describing this syntax, we use as much as possible the conventions used for description logics [Baader *et al.*, 2003], however we sometimes divert from them for openness (when a comparator is provided by description logics while in our case it is external to the language) or aesthetic reasons (like avoiding double superscript).

We will consider hereafter that alignments are sets of correspondences satisfying the following grammar. Correspondences (X) themselves are relations between entities (E). They are here restricted to equivalence and subsumption relations as well as membership of an individual (i) to a class (C):

$$(2.1) \qquad X := E \equiv E \mid E \sqsubseteq E \mid E \sqsupseteq E$$

$$(2.2) \qquad \mid i \in C \mid C \ni i$$

We restricted here the set of relations in order to be more precise in the semantics. However the Alignment format is extensible and new relations can be added, so the definition would rather be:

$$(2.3) \qquad X ::= E \; rel \; E$$

For instance enabling to use relations as $fatherOf$ between individuals.

The entities that will be found in correspondences are classes (C), relations (R), properties (P), attributes (A) and instances or individuals (i):

(2.4) $$E ::= C \mid R \mid P \mid A \mid i$$

Separation of ontological entities in five types comes from the ontology mapping language described in [Scharffe and Kiryakov, 2005]. This separation is justified as expression definitions slightly vary depending on this type and their semantics will differ as well. *Class expressions* represent classes or sets of classes linked together via operators. *Property expressions* represent relations whose codomain (or range) is a datatype. *Relations* are standing between two classes. So, the difference between Relation and Properties corresponds to the difference between ObjectProperty and DatatypeProperty in OWL. *Attributes* are relations and properties put in a particular context (see § 1.2.2). Finally, instances of classes can be put in correspondence via *Instance expressions*. To make the distinction between Relations and Properties, we will sometimes use "class relation" and "data property". The structure of expressions varies depending on the expression type. For example, Property expressions may have value restrictions whereas Class expression restrictions are more related to instances or particular attributes of instances. The main construct allows URI of an entity to be directly given to build an expression using constructors. Constructors tell how to group the set of entities given in the expression and are interpreted in model-theoretic terms in § 2.3. Expression constructors can be composed of sub-expressions. Conditions on class, property and relation expressions restrict the scope of the set of entities constructed in the expression.

Alignments relate entities of ontology languages such as OWL, F-logics or others. We consider that these entities (identified as entities in the Alignment format) are typed. The different types that will be considered are the following:

– Classes: c;
– Relations: r;
– Properties: p;
– Instances: i.

Data values and types must also be part of the entities to be considered:

– Data type: d;
– Data value: v;

They are however considered as external to the Alignment language.

From these entities, the Alignment language has constructors for creating more complex expressions. These constructors are the classical boolean algebra expressions (*and*, *or* and *not*) as well as the existence of constraints on class expressions (these constraints being expressed with external operators are not defined like in description logics):

(2.5) $$C ::= c$$
(2.6) $$\mid C \sqcup C \mid C \sqcap C \mid \neg C$$
(2.7) $$\mid \exists K$$

Relation expressions will also be created out of the relations, their boolean combination, constraints on their domain and range, and either their converse relation or their symmetric, transitive

or reflexive closures:

(2.8) $\qquad\qquad\qquad R ::= r$

(2.9) $\qquad\qquad\qquad\quad |\ R \sqcup R\ |\ R \sqcap R | \neg R$

(2.10) $\qquad\qquad\qquad\quad |\ \mathrm{dom}(C)\ |\ \mathrm{range}(C)$

(2.11) $\qquad\qquad\qquad\quad |\ \mathrm{inv}(R)$

(2.12) $\qquad\qquad\qquad\quad |\ \mathrm{sym}(R)\ |\ \mathrm{trans}(R)\ |\ \mathrm{refl}(R)$

Property expressions are similar but less complex since they cannot involve properties that only hold for relations (symmetry, transitivity, converse and reflexivity):

(2.13) $\qquad\qquad\qquad P ::= p$

(2.14) $\qquad\qquad\qquad\quad |\ P \sqcup P\ |\ P \sqcap P\ |\ \neg P$

(2.15) $\qquad\qquad\qquad\quad |\ \mathrm{dom}(C)\ |\ \mathrm{range}(d)$

(2.16)

To these are added *attributes*, i.e., properties or relations with a restricted domain. However, these are strictly equivalent to the following expression:

$$A \equiv R \sqcap dom(C) \text{ or } A \equiv P \sqcap dom(C)$$

In order to draw constraints on property and attribute values we introduce the notion of a path which is a sequence of relations possibly empty and possibly ending by a property:

(2.17) $\qquad\qquad\qquad Q ::= Q'\ |\ p\ |\ Q'.p$

(2.18) $\qquad\qquad\qquad Q' ::= \epsilon\ |\ r\ |\ Q'.r$

Note that we have restricted these paths to atomic steps, i.e., each element of the sequence is identified by a relation or property in the ontology but not a relation expression (R). These paths corresponds to role-value-maps in description logics. They are useful for expressing complex constraints (see below).

The values that can be found verbatim in these constraints can be either data values, individual instances, paths or the evaluation of some operation transf to a set of values:

(2.19) $\qquad\qquad\qquad V ::= v$

(2.20) $\qquad\qquad\qquad\quad |\ i$

(2.21) $\qquad\qquad\qquad\quad |\ Q$

(2.22) $\qquad\qquad\qquad\quad |\ \mathrm{transf}(V*)$

The constraints and operations that can be applied to values are useful to express concrete domain constraints. This was for instance in the definition of § 1.2.3 in which integers are converted into date. The language does consider that these functions (which return a data value) and comparators (which returns a boolean) are external operation with a well-known semantics. As a first set of constraints on datatypes, we consider using datatypes from XQuery (on numeric, string, collections and uri) as well as their comparators. The full set of these comparators is given in Appendix B. This provides:

- 23 functions (fn);
- 14 predicates (cp).

This allows the expression of expressions like:

$$length(collection)\ less\text{-}than\ multiply(integer, integer)$$

Once values are available, the predicates can be evaluated to compare these values. This is the basis of constraints that can be raised against paths for comparing their values, their datatypes or their multiplicity:

(2.23) $K ::= Q\ cp\ V$ (Value restriction)

(2.24) $\mid Q\ cp\ d$ (Type restriction)

(2.25) $\mid\ \mid Q \mid\ cp\ i$ (Multiplicity restriction)

This is used for expressing, e.g., $surname\ subStringOf\ fullname$, $age \in [12\ 16]$ or that $|child| \geq 3$. All these restrictions allow expressing more closely the relation between concepts or properties of different ontologies.

Example 1. *We want to restrict to the wines for which an adjacent region produces a wine made by the same producer. So we define the abstract first path Q as* `locatedIn.adjacentRegion` *and the second one V as* `hasMaker.producesWine.locatedIn`*. The comparison between the values pointed by the first path, and the values pointed by the second one gives the expected result. Abstractly, this is represented like this:*

$$\exists locatedIn.adjacentRegion = hasMaker.producesWine.locatedIn$$

2.2 Ensuring autonomous semantics

In this section, we provide a semantics for the formerly described alignment language. The semantics is based on the satisfaction of correspondences.

Informally, a correspondence $e\ rel_m\ e'$ means: there is a relation, denoted by symbol rel, between the entity described by e and the entity described by e', and our confidence in this relation being valid is equal to m. In fact, to the assertion $e\ rel\ e'$ is assigned a degree of trust m. The formal semantics given here defines the meaning of the former assertion independently of the confidence value. The way this measure is dealt with is left to an external formalism that we do not discuss here.

So, we henceforth consider a correspondence as a triple $\langle e, e', rel \rangle$, composed of:

- two entity expressions e and e';
- a relation symbol rel.

The difficulty in defining the interpretations of entity expressions is due to the fact that these expressions are built upon ontological entities which have there own interpretation in the ontological language [Borgida and Serafini, 2003]. The semantics of the language of two aligned ontologies can differ widely from each other, and also from the semantics of this alignment language. Therefore, we have proposed a semantics of aligned ontologies that has two levels of interpretation [Zimmermann and Euzenat, 2006], as visualised in Figure 2.1.
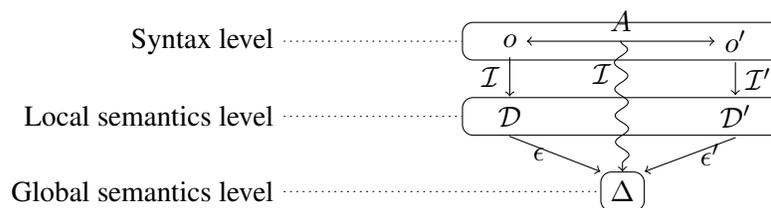
Figure 2.1: Two levels of semantics.

In Figure 2.1, o and o' are two ontologies related with alignment A. $\mathcal{I}$ (resp. $\mathcal{I}'$) denotes an interpretation of o (resp. o') with interpretation domain $\mathcal{D}$ (resp. $\mathcal{D}'$). Since these interpretations and domains may be very different in their structure, we use functions ϵ and ϵ', called equalising functions, to correlate these local domains into a commensurate global domain (Δ), in which alignments are interpreted.

We first give the interpretation of entity expressions independently of the ontological languages semantics (§ 2.3). Then we give the formal semantics of correspondences and define the fundamental notion of a model of aligned ontologies, which encompasses both local and global levels of semantics (§ 2.4).

2.3 Interpreting expressions

We define the full interpretation of entity expressions in the next subsections. The first item is not, properly speaking, part of the semantics. Datatypes, operators, comparators and data transformations are managed by the application, and we will assume that all that is defined in this part can be used in all following definitions. The semantics is given with regard to the abstract syntax.

2.3.1 Datatypes, operators, comparators and transformations

Datatypes are interpreted as in RDF and OWL. Different applications may bring their own datatypes, and their management is quite independent of the semantics. These datatypes are identified in the syntax by:

- Data type: d;
- Data value: v;

A datatype describes the set of values belonging to the type, the set of character strings representing these values and the way to convert these strings to values.

Definition 1 (Datatype). *A datatype d is characterised by:*

- *its lexical space $L(d)$ (a set of character strings);*
- *its value space $V(d)$;*
- *a mapping $L2V(d) : L(d) \rightarrow V(d)$ from the lexical space to the value space.*

Definition 2 (Datatype map). *A datatype map D is a partial mapping from URI references to datatypes.*

The datatype map allows the retrieval of datatypes from the URI identifying them.

This alignment language uses operators and comparators that are tied to specific datatypes. There must be support for these datatypes when implementing this alignment language.

Interpreting operators: Operators are used to define values that are not explicitly provided by a URI reference or a literal, but that can be calculated with existing values. A URI reference appearing in a `transf` element denotes the functions to be used for this calculation.

Definition 3 (Data operator). *A* data operator *is a triple $\langle k, (t_i)_{1 \leq i \leq k}, t_r, f \rangle$ such that:*

- k *is the* arity *of the operator;*
- $\forall 1 \leq i \leq k$, t_i *is the* datatype *of the ith operand;*
- t_r *is the datatype of the result of the operation;*
- $f : t_1 \times \cdots \times t_k \to t_r$ *is a function.*

Definition 4 (Operator map). *An* operator map *is a partial map from URI references to data operators.*

Interpreting comparators: Comparators serve to compare two values or elements. In fact, a comparator is a particular case of operator, namely a binary operator with a resulting domain being boolean.

Definition 5 (Comparator). *A* comparator *is a data operator $cp = \langle 2, (t, t'), \text{Bool}, f \rangle$.*

Operator map is also generalised from comparator map. Table B.2 in Appendix gives a library of common comparators.

Interpreting data transformation: Data transformations are more complicated functions that are used to convert data into other data, in order to render possible the mediation between two conceptually similar system with divergent concrete representations. A data transformation appears in the semantics just as an operator, where the operands are composed of the input data and the parameters of the function, and the result of the operation is the output of the function.

The implementation of a data transformation can exist outside the application, for example as a web service. So, in the semantics, it is interpreted as a function.

In the remainder, it will be assumed that datatypes, operators and comparators are provided by the application, and a datatype map D, a comparator map $g_{.} : cp \mapsto g_{cp}$ and a transformation/operator map $h_{.} : \text{transf} \mapsto h_{\text{transf}}$ are supposed to exist when we define the semantics.

Similarly to datatypes, different applications can implement different operators so that the list of interpretable operators can be extended. As an example, we provide a library of data operators in Table B.1 in Appendix B.

2.3.2 Interpreting literals

Literal values v are interpreted according to a datatype map D.

Definition 6 (Literals interpretation). *Let D be a datatype map. Let $\mathcal{L}$ be a set of literals. An interpretation of $\mathcal{L}$ is a pair $\mathcal{I}_L = \langle \mathbb{D}, \cdot^{\mathcal{I}_L} \rangle$ with $\mathbb{D}$ the domain of interpretation of literals and $\cdot^{\mathcal{I}_L} : \mathcal{L} \to \mathbb{D}$ such that:*

- $(\texttt{"}v\texttt{"})^{\mathcal{I}_L} = \texttt{"}v\texttt{".}$[1], *i.e., plain literals are interpreted as themselves,*
- $(\texttt{"}v\texttt{"}\texttt{\^{}\^{}}d)^{\mathcal{I}_L} = L2V(D(d))(v)$ *if* $v \in L(D(d))$

If a literal is ill-formed (like, for instance, `true^^&xsd;integer`) then no interpretation can exist.

2.3.3   Interpreting URI references

The basic notion of interpretation of URI references is very liberal. Indeed, given a set $\mathcal{U}$ of URI references, an interpretation of $\mathcal{U}$ is just a mapping from the elements of $\mathcal{U}$ to a domain of interpretation. More formally:

Definition 7 (URIref interpretation). *A URIref interpretation of a set $\mathcal{U}$ of URI references is a pair $\mathcal{I}_U = \langle \Delta, \cdot^{\mathcal{I}_U} \rangle$ such that Δ is a set called the global domain of interpretation and $\cdot^{\mathcal{I}_U} : \mathcal{U} \to \Delta$.*

Note that the set Δ (the global domain of interpretation) may contain elements that are themselves sets of elements, or sets of pairs or even sets of sets.

2.3.4   Interpreting path expressions

A path is essentially the same as a relation, so it must be interpreted as a set of pairs of elements.
   It is defined by a (potentially empty) sequence of relations possibly ended by a property:

$$Q ::= Q' \mid p \mid Q'.p$$
$$Q' ::= \epsilon \mid r \mid Q'.r$$

In the abstract syntax, ϵ denotes the empty path, r denotes a relation and p a property represented as a URI reference.

Definition 8 (Path interpretation). *A path interpretation is a triple $\mathcal{I}_P = \langle \mathcal{D}, \mathcal{I}_L, \cdot^{\mathcal{I}_P} \rangle$ with $\mathcal{D}$ a set called the* object domain *and $\mathcal{I}_L$ a literal interpretation such that:*

- $\epsilon^{\mathcal{I}_P} = \mathcal{D} \times \mathcal{D}$;
- $r^{\mathcal{I}_P} \subseteq \mathcal{D} \times \mathcal{D}$;
- $(Q'.r)^{\mathcal{I}_P} = \{ \langle x, z \rangle \in \mathcal{D} \times \mathcal{D} / \exists y \in \mathcal{D}, \langle x, y \rangle \in Q'^{\mathcal{I}_P} \wedge \langle y, z \rangle \in r^{\mathcal{I}_P} \}$;
- $p^{\mathcal{I}_P} \subseteq \mathcal{D} \times \mathbb{D}$;
- $(Q.p)^{\mathcal{I}_P} = \{ \langle x, z \rangle \in \mathcal{D} \times \mathbb{D} / \exists y \in \mathcal{D}, \langle x, y \rangle \in Q^{\mathcal{I}_P} \wedge \langle y, z \rangle \in p^{\mathcal{I}_P} \}$.

As we will see, this definition introduces a necessary constraint on the URIref interpretation: URI references representing a relation should be interpreted as a set of pairs.

2.3.5   Interpreting paths or values

In the abstract syntax, v is a literal interpreted as a value, i is a URI reference interpreted as an individual, Q is a path expression and $\texttt{transf}$ denotes an operator, given by a URI reference. It is assumed that the number of operands in the parenthesis is equal to the arity of the associated

---

[1]We consistently use the same notations for each kind of interpretation, so that we do not redefine existentially all their components in further definition. For instance, when a definition defines a literal interpretation $\mathcal{I}_L$, it will be assumed that the set $\mathbb{D}$ and function $\cdot^{\mathcal{I}_L}$ are also defined.

operator. Since a path or value can take several different forms (simple value, URI reference, path or operation), it embeds the necessary interpretations for all these variants, according to the definitions given above.

$$V ::= v$$
$$| \, i$$
$$| \, Q$$
$$| \, \mathrm{transf}(V*)$$

The interpretation of paths and values is more difficult because we authorise paths in the expression of values. Paths are interpreted as relations. We need to keep these relations intact, for instance for comparing the value at the end of two paths from a single instance. So, even though it may seem unnatural, we have chosen to interpret all values as a set of pairs.

Definition 9 (Path or value interpretation). *A path or value interpretation is a tuple $\mathcal{I}_V = \langle \mathcal{I}_U, \mathcal{I}_P, \cdot^{\mathcal{I}_V} \rangle$ with $\mathcal{I}_P$ a path interpretation built upon a literal interpretation $\mathcal{I}_L$ such that:*

- $v^{\mathcal{I}_V} = \mathcal{D} \times \{v^{\mathcal{I}_L}\}$;
- $i^{\mathcal{I}_V} = \mathcal{D} \times \{i^{\mathcal{I}_U}\}$;
- $Q^{\mathcal{I}_V} = Q^{\mathcal{I}_P}$;
- $\mathrm{transf}(V_1, \ldots, V_n)^{\mathcal{I}_V} = \{\langle x, h_{\mathrm{transf}}(y_1, \ldots, y_n)\rangle / \langle x, y_1 \rangle \in V_1^{\mathcal{I}_V} \wedge \cdots \wedge \langle x, y_n \rangle \in V_n^{\mathcal{I}_V}\}$.

By consistently interpreting paths and values as pairs of elements, we do not need to separate cases in the definitions below.

2.3.6 Interpreting restrictions

Restrictions are of three types: value restrictions, type restrictions and multiplicity restrictions. Whatever the restriction, it corresponds to a class definition. Each type of restriction is interpreted differently from the others, but they share the same abstract syntax:

$$K ::= Q \; cp \; V \qquad\qquad \text{(Value restriction)}$$
$$| \, Q \; cp \; d \qquad\qquad \text{(Datatype restriction)}$$
$$| \, Q \; cp \; n \qquad\qquad \text{(Multiplicity restriction)}$$

The comparator is denoted by cp and is identified by a URI reference. Q is a path expression and V is a path or value expression.

Interpreting value restrictions: A value restriction gives the class of individuals for which the comparison cp holds between a value at the end of the path Q and a value denoted by V.

Definition 10 (Value restriction interpretation). *A value restriction interpretation is a tuple $\mathcal{I}_{VR} = \langle \mathcal{I}_V, \cdot^{\mathcal{I}_{VR}} \rangle$ with $\mathcal{I}_V$ a path or value interpretation with an object domain $\mathcal{D}$ such that:*

- $(Q \; \mathrm{cp} \; V)^{\mathcal{I}_{VR}} = \{x \in \mathcal{D} / \exists y, y' \in \mathcal{D} \cup \mathbb{D}, \langle x, y \rangle \in Q^{\mathcal{I}_V} \wedge \langle x, y' \rangle \in V^{\mathcal{I}_V} \wedge g_{\mathrm{cp}}(y, y')\}$.

with g_{cp} the data operator associated with cp

Interpreting type restrictions: In this case, d denotes one or several datatypes. The type restriction imposes that the values pointed by the path Q belongs to the datatypes identified by d.

Definition 11 (Type restriction interpretation). *A type restriction interpretation is a tuple $\mathcal{I}_{TR} = \langle \mathcal{I}_V, \cdot^{\mathcal{I}_{TR}} \rangle$ with $\mathcal{I}_V$ a path or value interpretation with an ovbject domain $\mathcal{D}$ such that:*

$$- \quad (Q \text{ cp } d)^{\mathcal{I}_{TR}} = \{x \in \mathcal{D} / \forall y \in \mathcal{D} \cup \mathbb{D}, \langle x, y \rangle \in Q^{\mathcal{I}_V} \Rightarrow g_{\text{cp}}(y, V(D(d)))\}.$$

with g_{cp} the data operator associated with cp

Interpreting occurrence restrictions: In this case, V denotes an integer (or a set of integers), which is meant to denote a cardinality. The comparator compares this number to the cardinality of the attribute denoted by the path Q.

Definition 12 (Occurrence restriction interpretation). *An occurrence restriction interpretation is a tuple $\mathcal{I}_{OR} = \langle \mathcal{I}_V, \cdot^{\mathcal{I}_{OR}} \rangle$ with $\mathcal{I}_V$ a path or value interpretation with an object domain $\mathcal{D}$ such that:*

$$- \quad (Q \text{ cp } n)^{\mathcal{I}_{OR}} = \{x \in \mathcal{D} / g_{\text{cp}}(|\{y \in \mathcal{D} \cup \mathbb{D}; \langle x, y \rangle \in Q^{\mathcal{I}_V}\}|, n^{\mathcal{I}_L})\}.$$

with g_{cp} the data operator associated with cp

Comparators for occurrence restrictions are maxCardinality ($\leq$), minCardinality ($\geq$) or cardinality ($=$). We define a *restriction interpretation* as follows:

Definition 13 (Restriction interpretation). *A restriction interpretation $\mathcal{I}_K$ is the combination of a value restriction interpretation, a type restriction interpretation and an occurrence restriction interpretation having the same path or value interpretation.*

2.3.7 Interpreting class expressions

Classes are interpreted as sets of elements. Class constructors have a very common semantics which is exactly the basic description logic interpretation.

$$
\begin{aligned}
C ::=\ & c \\
& |\ C \sqcup C \mid C \sqcap C \mid \neg C \\
& |\ \exists K
\end{aligned}
$$

The single c denotes a class represented by a URI reference. C denotes a class expression. The constructors are represented in a typical description logics fashion ($\sqcup$ for union/or, $\sqcap$ for intersection/and, $\neg$ for negation/not).

Definition 14 (Class interpretation). *A class interpretation is a tuple $\mathcal{I}_C = \langle \mathcal{I}_K, \cdot^{\mathcal{I}_C} \rangle$, with $\mathcal{I}_K$ a restriction interpretation with object domain $\mathcal{D}$, such that:*

- $c^{\mathcal{I}_C} \subseteq \mathcal{D}$;
- $(C_1 \sqcup C_2)^{\mathcal{I}_C} = C_1^{\mathcal{I}_C} \cup C_2^{\mathcal{I}_C}$;
- $(C_1 \sqcap C_2)^{\mathcal{I}_C} = C_1^{\mathcal{I}_C} \cap C_2^{\mathcal{I}_C}$;
- $(\neg C)^{\mathcal{I}_C} = \mathcal{D} \setminus C^{\mathcal{I}_C}$;
- $(\exists K)^{\mathcal{I}_C} = K^{\mathcal{I}_K}$.

2.3.8   Interpreting relation expressions

Class properties are interpreted as set-theoretic relations, i.e., sets of pairs of elements.

$$R ::= r$$
$$| \ R \sqcup R \ | \ R \sqcap R | \neg R$$
$$| \ \mathrm{dom}(C) \ | \ \mathrm{range}(C)$$
$$| \ \mathrm{inv}(R)$$
$$| \ \mathrm{sym}(R) \ | \ \mathrm{trans}(R) \ | \ \mathrm{refl}(R)$$

In this abstract syntax, r is a single URI reference, that denotes a relation between instances. Symbol R denotes a (class) relation. Constructors are given in a Description-Logic-like syntax: union ($\sqcup$), intersection ($\sqcap$), complement ($\neg$), range restriction (range), domain restriction (dom). C represents a class. Finally, inv, sym, trans and refl denotes the inverse, the symmetric closure, the transitive closure and reflexive closure respectively.

Definition 15 (Relation interpretation). *A Relation interpretation is a tuple* $\mathcal{I}_{CP} = \langle \mathcal{I}_C, \cdot^{\mathcal{I}_{CP}} \rangle$, *with* $\mathcal{I}_C$ *a class interpretation, built on path interpretation* $\mathcal{I}_P$ *with object domain* $\mathcal{D}$, *such that:*

- $r^{\mathcal{I}_{CP}} = r^{\mathcal{I}_P}$;
- $(R_1 \sqcup R_2)^{\mathcal{I}_{CP}} = R_1^{\mathcal{I}_{CP}} \cup R_2^{\mathcal{I}_{CP}}$;
- $(R_1 \sqcap R_2)^{\mathcal{I}_{CP}} = R_1^{\mathcal{I}_{CP}} \cap R_2^{\mathcal{I}_{CP}}$;
- $(\neg R)^{\mathcal{I}_{CP}} = \mathcal{D} \times \mathcal{D} \setminus R^{\mathcal{I}_{CP}}$;
- $\mathrm{range}(C)^{\mathcal{I}_{CP}} = \mathcal{D} \times C^{\mathcal{I}_C}$;
- $\mathrm{dom}(C)^{\mathcal{I}_{CP}} = C^{\mathcal{I}_C} \times \mathcal{D}$;
- $\mathrm{inv}(R)^{\mathcal{I}_{CP}} = \{\langle x, y \rangle / \langle y, x \rangle \in R^{\mathcal{I}_{CP}}\}$;
- $\mathrm{sym}(R)^{\mathcal{I}_{CP}} = R^{\mathcal{I}_{CP}} \cup \mathrm{Inv}(R)^{\mathcal{I}_{CP}}$;
- $\mathrm{refl}(R)^{\mathcal{I}_{CP}} = R^{\mathcal{I}_{CP}} \cup \{\langle x, x \rangle / x \in \mathcal{D}\}$;
- $\mathrm{trans}(R)^{\mathcal{I}_{CP}} = R^{\mathcal{I}_{CP}} \cup \{\langle x, z \rangle / \exists y \in \mathcal{D}, \langle x, y \rangle \in R^{\mathcal{I}_{CP}} \wedge \langle y, z \rangle \in \mathrm{trans}(R)^{\mathcal{I}_{CP}}\}$;

2.3.9   Interpreting data property expressions

Data properties are also interpreted as set-theoretic relations, but since the codomain has to be a datatype, values are restricted to datatype values.

$$P ::= p$$
$$| \ P \sqcup P \ | \ P \sqcap P \ | \ \neg P$$
$$| \ \mathrm{dom}(C) \ | \ \mathrm{range}(d)$$

This syntax and semantics are similar to those of relations.

Definition 16 (Data property interpretation). *A data property interpretation is a tuple* $\mathcal{I}_{DP} = \langle \mathcal{I}_C, \cdot^{\mathcal{I}_{DP}} \rangle$, *with* $\mathcal{I}_C$ *a class interpretation, built on path or value interpretation* $\mathcal{I}_V$ *with object domain* $\mathcal{D}$, *such that:*

- $p^{\mathcal{I}_{DP}} = p^{\mathcal{I}_V}$;
- $(P_1 \sqcup P_2)^{\mathcal{I}_{DP}} = P_1^{\mathcal{I}_{DP}} \cup P_2^{\mathcal{I}_{DP}}$;
- $(P_1 \sqcap P_2)^{\mathcal{I}_{DP}} = P_1^{\mathcal{I}_{DP}} \cap P_2^{\mathcal{I}_{DP}}$;
- $(\neg P)^{\mathcal{I}_{DP}} = \mathcal{D} \times \mathbb{D} \setminus P^{\mathcal{I}_{DP}}$;
- $(\mathrm{dom}(C))^{\mathcal{I}_{DP}} = C^{\mathcal{I}_C} \times \mathcal{D}$;
- $(\mathrm{range}(d))^{\mathcal{I}_{DP}} = \{\langle x, y \rangle \in P^{\mathcal{I}_{DP}} / y \in L(D(d))\}$;

Attribute definitions that have been introduced in §1.2.2 can be expressed in first order logic as:

$$C(x) \wedge P(x,y)$$

which corresponds to:

$$dom(C) \sqcap P$$

2.3.10 Interpreting instance expressions

Instances are always identified by a URI reference. They are interpreted by a URIref interpretation (see §2.3.3).

2.3.11 Interpreting entity expressions

This is the general interpretation of all expressions of the language. An entity expression is of one of the five following types (instance, class, relation, property and attribute):

$$E ::= C \mid R \mid P \mid A \mid i$$

The abstract syntax shall be understood as: C for class expression, R for class relation expression, P for data property expression, A for attribute expression and i for instance expression.

Definition 17 (Expression interpretation). *An* expression interpretation *is a tuple*

$$\mathcal{I} = \langle \Delta, \mathcal{D}, \mathbb{D}, \mathcal{I}_L, \mathcal{I}_U, \mathcal{I}_P, \mathcal{I}_V, \mathcal{I}_K, \mathcal{I}_C, \mathcal{I}_{CP}, \mathcal{I}_{DP}, \cdot^{\mathcal{I}} \rangle$$

with:

- $\mathcal{I}_L = \langle \mathbb{D}, \cdot^{\mathcal{I}_L} \rangle$ *a literal interpretation;*
- $\mathcal{I}_U = \langle \Delta, \cdot^{\mathcal{I}_U} \rangle$ *a URIRef interpretation;*
- $\mathcal{I}_P = \langle \mathcal{D}, \mathcal{I}_L, \cdot^{\mathcal{I}_P} \rangle$ *a path interpretation;*
- $\mathcal{I}_V = \langle \mathcal{I}_U, \mathcal{I}_P, \cdot^{\mathcal{I}_V} \rangle$ *a path or value interpretation;*
- $\mathcal{I}_K = \langle \mathcal{I}_V, \cdot^{\mathcal{I}_K} \rangle$ *a restriction interpretation;*
- $\mathcal{I}_C = \langle \mathcal{I}_K, \cdot^{\mathcal{I}_C} \rangle$ *a class interpretation;*
- $\mathcal{I}_{CP} = \langle \mathcal{I}_C, \cdot^{\mathcal{I}_{CP}} \rangle$ *a relation interpretation;*
- $\mathcal{I}_{DP} = \langle \mathcal{I}_C, \cdot^{\mathcal{I}_{DP}} \rangle$ *a data property interpretation;*

such that:

- $E^{\mathcal{I}} \in \Delta$;
- $i^{\mathcal{I}} = i^{\mathcal{I}_U}$;

- $c^{\mathcal{I}} = c^{\mathcal{I}_U}$;
- $p^{\mathcal{I}} = p^{\mathcal{I}_U}$;
- $r^{\mathcal{I}} = r^{\mathcal{I}_U}$.
- $C^{\mathcal{I}} = C^{\mathcal{I}_C}$;
- $R^{\mathcal{I}} = R^{\mathcal{I}_{CP}}$;
- $P^{\mathcal{I}} = P^{\mathcal{I}_{DP}}$.

The two entities appearing in a correspondence are interpreted with this definition. Once these two interpretations are known, they have to be compared according to the relation given in the correspondence. This comparison will determine the validity of the interpretation with regard to the correspondence.

2.4 Interpreting correspondences

In order to relate the semantics of aligned ontologies to the semantics of our defined language, we first give general notions of model-theoretic semantics that should encompass most of the language semantics that are used to represent ontologies. So, we first have to define an interpretation of an ontology, in model-theoretic terms.

Definition 18 (Interpretation of an ontology). *Given an ontology o, an interpretation m of o is a pair $\langle \mathcal{I}, \mathcal{D} \rangle$ such that $\mathcal{D}$ is a set called the domain of interpretation and $\mathcal{I}$ is a function from elements of o to elements of a domain of interpretation $\mathcal{D}$.*

Among interpretations, there are particular ones that are said to *satisfy* the ontology. The local semantics of ontologies determine the satisfaction relation $\models$ that relates interpretations to satisfied ontologies, *i.e.*, $m \models o$ if and only if m satisfies o. The interpretations that satisfy o are called the models of o and denoted by $\mathrm{Mod}(o)$.

In this language, ontological entities appear in the form of URI references denoting instances, classes or properties. These entities have an interpretation in both the ontology language and the alignment language semantics. In order to connect the alignment interpretation to the local (ontological) interpretations, the simplest solution would be to define the expression interpretation of these entities as being equal to their local interpretation. Unfortunately, the diversity of formalism and conceptualisation is such that it is often impossible to interpret two ontologies in the same global domain.

Therefore, in order to assess the validity of a relation between heterogeneous ontologies, we have introduced the notion of *equalising function* [Zimmermann and Euzenat, 2006], which serves to make the domains commensurate.

Definition 19 (Equalising function). *Given a family of interpretations $(m_i)_{i \in I} = \langle \mathcal{D}_i, \mathcal{I}_i \rangle_{i \in I}$ of local ontologies, an* equalising function *for $(m_i)_{i \in I}$ is a family of functions $(\epsilon_i)_{i \in I}$ from the local domains of interpretation $\mathcal{D}_i$ to a global domain Δ (i.e., for all $i \in I$, $\epsilon_i : \mathcal{D}_i \to \Delta$).*

So equalising functions not only define a global domain for the interpretation of a set of ontologies, but also define how local domains are correlated in the global interpretation.

Example 2. *Many semantics impose instances to be interpreted as atomic elements of a domain, while classes are sets of atomic elements. In such cases, the equivalence of an instance and a class would be unsatisfiable without equalising function.*

The ultimate goal of the semantics is to define the satisfaction of aligned ontologies. We give the following preliminary definition.

Definition 20 (Aligned ontologies). *The structure made of two ontologies o and o' and an alignment A between these ontologies is called aligned ontologies and denoted by $A(o, o')$.*

In order to interpret aligned ontologies, we not only need a global interpretation of the ontologies, but also an interpretation function that complies with the constructors and operators of the mapping language.

Definition 21 (Interpretation of aligned ontologies). *Let o and o' be two ontologies aligned with alignment A. An interpretation of $A(o, o')$ is a triple $\langle \mathcal{I}, \mathcal{M}, \epsilon \rangle$ composed of:*

- *an expression interpretation $\mathcal{I}$ having URI interpretation $\mathcal{I}_U$;*
- *a pair $\mathcal{M} = \langle m, m' \rangle$ of local models where $m \in \mathrm{Mod}(o)$ and $m' \in \mathrm{Mod}(o')$;*
- *an equalising function $\epsilon = \langle \epsilon, \epsilon' \rangle$ for $\mathcal{M}$ to a global domain Δ.*

Moreover, for all URI reference u appearing in the first (respectively second) entity expressions of A, $u^{\mathcal{I}_U} = \epsilon(u^m)$ (respectively $u^{\mathcal{I}_U} = \epsilon'(u^{m'})$.)

As noted in § 2.2, a correspondence is denoted by a triple $\langle E_1, E_2, rel \rangle$. Its syntax is:

$$X := E \equiv E \mid E \sqsubseteq E \mid E \sqsupseteq E$$
$$\mid i \in C \mid C \ni i$$

or more generally:

$$X ::= E \ rel \ E$$

Interpretation of correspondences depends on the interpretation of relations. The interpretation of the symbol that denotes a correspondence relation does not vary from one alignment interpretation to the other. A relation symbol rel is interpreted as a binary relation $\widetilde{rel}$.

The satisfaction of a correspondence is an important notion of the semantics. Indeed, a correspondence in an alignment acts as an axiom in an ontology: it allows discriminating valid and invalid interpretations. Consequently, it serves to define a model of aligned ontologies.

Definition 22 (Satisfied correspondence). *Let $c = \langle e, e', rel \rangle$ be a correspondence of an alignment A between ontologies o and o'. c is satisfied by an interpretation $\langle \mathcal{I}, \mathcal{M}, \epsilon \rangle$ of $A(o, o')$ if and only if $\langle e^{\mathcal{I}}, e'^{\mathcal{I}} \rangle \in \widetilde{rel}$. This is written $\mathcal{I} \models c$.*

For instance, for the relations used in the above X grammar, the interpretation can be the following:

$$\mathcal{I} \models e \equiv e' \text{ if and only if } e^{\mathcal{I}} = e'I$$
$$\mathcal{I} \models e \sqsubseteq e' \text{ if and only if } e^{\mathcal{I}} \subseteq e'I$$
$$\mathcal{I} \models e \sqsupseteq e' \text{ if and only if } e^{\mathcal{I}} \supseteq e'I$$
$$\mathcal{I} \models i \in C \text{ if and only if } i^{\mathcal{I}} \in C^{\mathcal{I}}$$
$$\mathcal{I} \models C \ni i \text{ if and only if } i^{\mathcal{I}} \in C^{\mathcal{I}}$$

In which $\widetilde{\equiv}$ is $=$, $\widetilde{\sqsubseteq}$ is $\subseteq$, $\widetilde{\sqsupseteq}$ is $\supseteq$, $\widetilde{\in}$ is $\in$, and $\widetilde{\ni}$ is $\ni$,

If all correspondences of an alignment are satisfied, then the interpretation is called a model of the alignment.

Definition 23 (Model of aligned ontologies)**.** *A model of aligned ontologies $A(o, o')$ is an interpretation of $A(o, o')$ that satisfies all the correspondences of A. If a model is written $\mathcal{M}$, then it is noted $\mathcal{M} \models A$.*

From these notions, it is possible to define the set of consequences of aligned ontologies [Zimmermann and Euzenat, 2006].

2.5   Synthesis

We have provided an abstract syntax and semantics for an expressive alignment language. They are summarised in Appendix A in a single table.

The specificity of this semantics is its capacity to interpret systems relating heterogeneous formalisms. So it is a technical mean to offer an expressive language that can bridge between heterogeneous ontology languages and yet take this semantics into account.

This semantics covers the rather rich set of constructors and operators that gives it an elaborate expressiveness. Examples gave a hint of the possibilities offered by the language. However, this expressiveness is achieved at the cost of a hard reasoning procedure. Indeed, since an instance can be interpreted as a class or a relation, and class complement and relation complement exist, this language is most likely to have undecidable satisfiability.

Chapter 3

Operational syntaxes

The abstract syntax is sufficient for defining the semantics of the language. It role is to support inductive semantic definitions. However it is not sufficient for using in operational applications. First, it is ambiguous: when a URI is used in this syntax, it is not always possible to decide if it is a property or a relation, a class or an individual. This propagates through a number of operators. So, it is preferable to design a syntax which resolves this ambiguity. Then, the abstract syntax only focuses on the expression of correspondences and does not cover many other useful information about alignments such as the identification of the alignment and the matched ontologies or metadata such as the method used for establishing the alignment.

Hence, in order to be used by applications, we provide below two concrete syntax:

The exchange syntax (§3.1) is used for exchange of alignments between different systems. It is an extension of the Alignment format [Euzenat, 2004]. This syntax is expressed in RDF/XML so that it can be parsed easily and have standard serialisation methods for being interchanged through networks.

The surface syntax (§3.2) is meant to be more easily read by human users and extends the "human readable" syntax of [Scharffe and de Bruijn, 2005]. It addresses the problem of displaying, and expressing, alignments in a less verbose format than RDF/XML.

Both syntax are designed to be equivalent and to correspond to the abstract syntax presented in Chapter 2.

3.1   Exchange syntax

In order to be exchanged in tools, this language requires a concrete syntax. We provide below a first concrete syntax dedicated to the exchange of information between systems. As is most practical in modern computing this syntax is an XML syntax. However, in order to be as compliant as possible with semantic web technology, this XML syntax follows the RDF/XML rules and an ontology describing the concepts used in this RDF/XML syntax has been defined (and is available in Appendix C).

This syntax combines the alignment format of [Euzenat, 2004] and the OMWG mapping language of [Scharffe and de Bruijn, 2005] as considered in [Euzenat *et al.*, 2005a]. It takes from the first the XML syntax and structure, and embeds the complex expression constructs expressible in the second.

3.1.1   General structure

This syntax has been defined from the Alignment format [Euzenat, 2004], the OMWG Ontology mapping language[1] and our own deliverable 2.2.6 [Euzenat *et al.*, 2005b]. The language follows (and extends) the Alignment format structure until the entity level in which the specific entity constructs provided in this deliverable have been defined.

The syntax is here presented under a Backus-Naur form, though it is in reality order independent, i.e., the order of appearance of attributes or elements does not matter.

The default namespace applying to elements and attributes in the following grammar descriptions is `omwg` standing for `http://www.owmg.org/TR/d7/d7.2/`, `align` is equivalent to `http://knowledgeweb.semanticweb.org/heterogeneity/alignment/`.

The structure of the alignment is the same as that of the Alignment format:

⟨*alignment*⟩ ::= `<align:Alignment rdf:about="`⟨*uri*⟩`">`
              ⟨*annotation*⟩`*`
              `<align:level>2OMWG</align:level>`
              `<align:onto1>`⟨*onto*⟩`</align:onto1>`
              `<align:onto2>`⟨*onto*⟩`</align:onto2>`
              `(<align:map>`⟨*cell*⟩`</align:map>)*`
            `</align:Alignment>`

This structure contains information which is not featured in the abstract syntax (because it has no impact on the definiton of the semantics). Among the annotations used by this format, a very important one is the definition of the alignment level. This is a string which in the case of the expressive language should be "2OMWG". This tells tools that the alignment is on level 2, i.e., correspondences are across constructed entities, and that the corresponding entities are specified according to this document. This ensures the compatibility with other extensions of the format.

As this format allows defining correspondences between ontologies written in different languages the declaration of the two aligned ontologies includes the identification of the formalism used to represent them. This has been added to the Alignment format. The `Ontology` element gather into one place the information about the ontologies:

⟨*onto*⟩   ::= `<align:Ontology rdf:about="`⟨*uri*⟩`">`
             `<align:location>` ⟨*url*⟩ `</align:location>`
             `<align:formalism>` ⟨*formalism*⟩ `</align:formalism>`
              ⟨*description*⟩
             `</align:Ontology>`

⟨*formalism*⟩ ::= `<align:Formalism align:uri="`⟨*uri*⟩`" align:name="`⟨*string*⟩`" />`

The alignment itself is structured as a set of cells, each representing a correspondence between two entity expressions:

⟨*cell*⟩   ::= `<align:Cell rdf:about="`⟨*uri*⟩`">`
             ⟨*annotation*⟩`*`
             `<align:entity1>`⟨*entity*⟩`</align:entity1>`

---

[1]http://www.omwg.org/TR/d7/

```
            <align:entity2>⟨entity⟩</align:entity2>
            <align:measure>⟨value⟩</align:measure>
            <align:relation>⟨relation⟩</align:relation>
        </align:Cell>
```

When the alignment results from an algorithm, each correspondence may be discovered with a certain degree of certainty. It might be the case that the algorithm uses a set of rules giving an evidence that two entities must be related to a certain extend. The ⟨*measure*⟩ reflects the confidence given to each correspondence. It takes values between 0 and 1.

The type of entities being part of the correspondence and the relation standing between them (equivalence or subsumption) are represented by the ⟨*relation*⟩.

The list of relations defined in our language is currently the following:

⟨*relation*⟩ ::= Equivalence | Subsumes | SubsumedBy
            | InstanceOf | HasInstance.

Here is an example of an alignment using only simple entities as in the previous Alignment format:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE rdf:RDF SYSTEM "align.dtd" [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY wine "http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#">
  <!ENTITY vin "http://ontology.deri.org/vin#">
]>

<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owmg="http://www.owmg.org/TR/d7/d7.2/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
 <Alignment>
    <xml>yes</xml>
    <dc:creator rdf:resource="http://sw.deri.org/~francois/"/>
    <dc:date>2006/06/07</dc:date>
    <method>manual</method>
    <purpose>example</purpose>
    <level>0</level>
    <type>**</type>
    <onto1>
      <Ontology rdf:about="&wine;">
        <location>http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine</location>
        <formalism>
          <Formalism name="OWL 1.0" uri="http://www.w3.org/2002/07/owl#"/>
        </formalism>
      </Ontology>
    </onto1>
    <onto2>
      <Ontology rdf:about="&vin;">
        <location>http://sw.deri.org/~francois/ontologies/OntologyDuVin.wsml</location>
        <formalism>
          <Formalism name="WSML"
                  uri="http://www.wsmo.org/wsml/wsml-syntax/wsml-dl"/>
```

```
        </formalism>
      </Ontology>
    </onto2>
    <map>
      <Cell>
        <entity1 rdf:resource="&wine;VintageYear"/>
        <entity2 rdf:resource="&vin;Millesime"/>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>equivalence</relation>
      </Cell>
      <Cell>
        <entity1 rdf:resource="&wine;yearValue"/>
        <entity2 rdf:resource="&vin;annee_millesime"/>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>equivalence</relation>
      </Cell>
      <Cell>
        <entity1 rdf:resource="&wine;locatedIn"/>
        <entity2 rdf:resource="&vin;hasTerroir"/>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>subsumed</relation>
      </Cell>
      <Cell>
        <entity1 rdf:resource="&wine;White"/>
        <entity2 rdf:resource="&vin;Blanc"/>
        <measure rdf:datatype="&xsd;float">1.0</measure>
        <relation>equivalence</relation>
      </Cell>
    </map>
  </Alignment>
```

Simple expressions are used in the following example. More complex ones will be studied later in this section. In `Vin`, `Blanc` (French word for white) is a class that contains the variety of white-yellow colours like golden, pale yellow, and so forth.

This example, is only a level 0 example because it does not consider complex entities as defined by our level 2 language.

A benefit of this language is its ability of modelling complex correspondences. Expressions nested in cells express such correspondences and will be further described.

There are basically five types of entities: Classes, Properties, Relations, Attributes and Instances.

$\langle entity \rangle$ ::= $\langle classexpr \rangle$
            | $\langle relexpr \rangle$
            | $\langle propexpr \rangle$
            | $\langle attexpr \rangle$
            | $\langle instance \rangle$

They can be simply identified by their URIs like in the genuine Alignment format or be composed within the format: this is the additional expressiveness that this format provides. We use boolean constructors (`And`, `Or` and `Not`) and relation constructors (`Inverse`, `Transitive`, `Symmetric` and `Reflexive` closures). Relations and properties allows one to distinguish whether an attribute has class instances in its codomain (Relations) or datatype values (Properties). This

distinction is needed because several relation constructors are not applicable to (datatype) properties.

We describe the different expressions in the remaining subsections.

3.1.2 Class expression

A class expression allows the specification of complex classes by grouping them using sets operators and by applying restrictions to those sets. We distinguish between three operators for class expressions: $\langle not \rangle$, $\langle and \rangle$, $\langle or \rangle$. Their semantics is comparable to the semantic of difference, union, intersection in set theory.

```
⟨classexpr⟩ ::= <Class rdf:about="⟨uri⟩"/>
            |  <Class>
                  ⟨classconst⟩?
                  ⟨classcond⟩ *
               </Class>
```

So a $\langle classexpr \rangle$ is either a class identified by its URI or a complex class expression made of restrictions and/or constructions. Here, the conditons of the abstract syntax are directly associated with the other connectors. Construction for classes are the usual boolean connectives:

```
⟨classconst⟩ ::= <and>
                   <Collection> (<item>⟨classexpr⟩</item>)+ </Collection>
                 </and>
             |  <or>
                   <Collection> (<item>⟨classexpr⟩</item>)+ </Collection>
                 </or>
             |  <not>⟨classexpr⟩</not>
```

The following example shows a correspondence featuring a class expression using the "or" constructor.

```
<align:Cell>
  <align:entity1>
    <Class rdf:about="&wine;WineFlavor"/>
  </align:entity1>
  <align:entity2>
    <Class>
      <or><Collection>
        <item><Class rdf:about="&vin;Acidite"/></item>
        <item><Class rdf:about="&vin;Astreingence"/></item>
        <item><Class rdf:about="&vin;Amertume"/></item>
      </Collection></or>
    </Class>
  </align:entity2>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
  <align:relation>subsumes</align:relation>
</align:Cell>
```

Conditions are constraints applying to classes:

⟨*classcond*⟩ ::= `<attributeValueCondition>` ⟨*constraint*⟩ `</attributeValueCondition>`
    `|` `<attributeTypeCondition>` ⟨*constraint*⟩ `</attributeTypeCondition>`
    `|` `<attributeOccurenceCondition>` ⟨*constraint*⟩ `</attributeOccurenceCondition>`

These three types of conditions restrict the scope of a class expression by applying a restriction to the value, type or cardinality of a particular property or relation. They enforce the presence of a particular attribute, the class of the object instantiating the attribute or the value of this attribute in the case of a datatype.

⟨*constraint*⟩ ::= `<Restriction>`
     `<onProperty>`⟨*path*⟩`</onProperty>`
     `<comparator>`⟨*uri*⟩`</comparator>`
     `<value>`⟨*pov*⟩`</value>`
    `</Restriction>`

  The attribute on which the condition is based is indicated using a path. A path is used to target a specific entity in the graph made by classes, properties and relations. It allows, for exemple, that a correspondence only applies to instances which are involved in a particular relation: "Wines whose producing area is located in a particular region". The following example[2] expresses this path:

```
<Path>
  <first><Relation rdf:about="&vin;hasTerroir"/></first>
  <next><Relation rdf:about="&proton;LocatedIn"/></next>
</Path>
```

  Here is the syntax for a path:

⟨*path*⟩   ::= `<Property rdf:about="`⟨*uri*⟩`"/>`
     `|` `<Relation rdf:about="`⟨*uri*⟩`"/>`
     `|` `<Self />`
     `|` `<Path>`
       `<first>`⟨*step*⟩`</first>`
       `<next>`⟨*path*⟩`</next>`
      `</Path>`

⟨*step*⟩   ::= `<Relation rdf:about="`⟨*uri*⟩`"/>`

This syntax does not prohibits through the grammar to have several attribute steps in a path. As we have seen before, paths can be used in place of values, the ⟨*pov*⟩ production (for path-or-value expresses this):

⟨*value*⟩   ::= ⟨*string*⟩
     `|` ⟨*instance*⟩
     `|` `<Apply operation="` ⟨*uri*⟩ `">` ⟨*pov*⟩`*` `</Apply>`

⟨*pov*⟩   ::= ⟨*path*⟩
     `|` ⟨*value*⟩

---

[2]This example makes use of the Proton ontology: http://proton.semanticweb.org/.

Thanks to paths and restrictions, it is possible to restrict a correspondence to "Wines whose producing region is located in Aquitaine" using the following restriction. The comparator is issued from the comparators list given table B.2 in Appendix.

```
<Restriction>
  <onProperty>
    <Path>
      <first><Relation rdf:about="&vin;hasTerroir"/></first>
      <next><Relation rdf:about="&proton;LocatedIn"/></next>
    </Path>
  </onProperty>
  <comparator rdf:resource="equal"/>
  <value rdf:datatype="&xsd;string">Acquitaine</value>
</Restriction>
```

We can with this restriction build the correspondence from § 1.1 between "BordeauxWine" and "Vin" whose "terroir" is located in "Aquitaine"[3].

```
<align:Cell>
  <align:entity1>
    <Class rdf:about="&wine;BordeauxWine">
  </align:entity1>
  <align:entity2>
    <Class rdf:about="&vin;Vin">
      <attributeValueCondition>
        <Restriction>
          <onProperty>
            <Path>
              <first><Relation rdf:about="&vin;hasTerroir"/></first>
              <next><Relation rdf:about="&proton;LocatedIn"/></next>
            </Path>
          </onProperty>
          <comparator rdf:resource="equal"/>
          <value rdf:datatype="&xsd;string">Acquitaine</value>
        </Restriction>
      </attributeValueCondition>
    </Class>
  </align:entity2>
</align:Cell>
```

An example of occurrence restriction would be the wines produced in a region with no adjacent region, such as an island. For instance Moscatel Madeira wine is produced on the island of Madeira.

```
<align:Cell>
  <align:entity1>
    <Class rdf:about="&wine;Wine">
      <attributeOccurrenceCondition>
        <Restriction>
          <onProperty>
            <Path>
              <first><Relation rdf:about="&wine;locatedIn"/></first>
              <next>< Relation rdf:about="&wine;adjacentRegion"/></next>
```

_____

[3]Aquitaine is the administrative region to which belongs the city of Bordeaux

```
          </Path>
        </onProperty>
        <comparator rdf:resource="cardinality"/>
        <value rdf:datatype="&xsd;int">0</value>
      </Restriction>
    </attributeOccurrenceCondition>
  </Class>
</align:entity1>
<align:entity2>
  <Class rdf:about="&vin;Madere" />
</align:entity2>
<align:measure rdf:datatype="&xsd;float">1.0</align:measure>
<align:relation>subsumes</align:relation>
</align:Cell>
```

Another example modeling the correspondence presented in the scenario § 1.1. This correspondence uses a path to state that a locally grown wine is a wine whose owner (propriétaire) is the same person as the first seller (négociant) for this wine.

```
<align:Cell>
  <align:entity1>
    <Class rdf:about="&wine;LocallyGrownWine"/>
  </align:entity1>
  <align:entity2>
    <Class rdf:about="&vin;Vin">
      <attributeValueCondition>
        <Restriction>
          <onProperty>
            <Relation rdf:about="&vin;propriétaire"/></first>
          </onProperty>
          <comparator rdf:resource="equals"/>
          <value><Relation rdf:about="&vin;négotiant"/></value>
        </Restriction>
      </attributeValueCondition>
    </Class>
  </align:entity2>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
  <align:relation>subsumes</align:relation>
</align:Cell>
```

3.1.3   Relation expression

Relation expressions allow constructing relations using a set of operators. Complementary to the operators described in class expressions, specific relation operators are here introduced. The ⟨*inverse*⟩ operator take the inverse relation of a given one. The ⟨*symmetric*⟩, ⟨*transitive*⟩ and ⟨*reflexive*⟩ operators construct the symmetric, transitive and reflexive closure of a given relation. Semantics for these operators are given in § 2.3.8.

Relation expressions can be either a relation from an ontology, a construction from such relations or constraints on these relations:

⟨*relexpr*⟩   ::= <Relation rdf:about="⟨*uri*⟩" />
        | <Relation>
              ⟨*relconst*⟩?

     ⟨*relcond*⟩ *
     ⟨*transformation*⟩?
     </Relation>

The constructors for relations are those of relation algebra. These are the boolean operators (and, or and not) and the binary relation operators (converse, symmetric closure, transitive closure and reflexive closure).

⟨*relconst*⟩ ::= <and>
     <Collection> (<item>⟨*classexpr*⟩</item>)+ </Collection>
     </and>
   | <or>
     <Collection> (<item>⟨*classexpr*⟩</item>)+ </Collection>
     </or>
   | <not>⟨*relexpr*⟩</not>
   | <inverse>⟨*relexpr*⟩</inverse>
   | <symmetric>⟨*relexpr*⟩</symmetric>
   | <transitive>⟨*relexpr*⟩</transitive>
   | <reflexive>⟨*relexpr*⟩</reflexive>

The restrictions are the usual domain and range restrictions:

⟨*relcond*⟩ ::= <domainRestriction>⟨*classexpr*⟩</domainRestriction>
   | <rangeRestriction>⟨*classexpr*⟩</rangeRestriction>

Domain restriction is exemplified in the following relation expression:

```
<align:Cell>
  <align:entity1>
    <Relation rdf:about="&wine;locatedIn">
      <domainRestriction>
        <Class rdf:about="&wine;Wine" />
      </domainRestriction>
    </Relation>
  </align:entity1>
  <align:entity2>
    <Relation rdf:about="&vin;has_terroir" />
  </align:entity2>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
  <align:relation>subsumedBy</align:relation>
</align:Cell>
```

3.1.4 Property expression

As class expressions, property expressions give the possibility to group properties using constructors. We have divided them into Properties and Relations according to the abstract syntax.

  Properties can either be directly addressed through their URI, constructed, restricted and transformed.

⟨*propexpr*⟩ ::= <Property rdf:about="⟨*uri*⟩" />
   | <Property>
     ⟨*attrconst*⟩?
     ⟨*attrcond*⟩*
     </Property>

The Property constructors are the usual boolean constructors:

⟨*attrconst*⟩ ::= `<and>`
     `<Collection>` (`<item>`⟨*propexpr*⟩`</item>`)+ `</Collection>`
    `</and>`
  | `<or>`
     `<Collection>` (`<item>`⟨*propexpr*⟩`</item>`)+ `</Collection>`
    `</or>`
  | `<not>`⟨*propexpr*⟩`</not>`

The Property constraints are the domain and range restriction. The range restriction is specified through constraints on the types and values of the range of the property:

⟨*attrcond*⟩ ::= `<domainRestriction>`⟨*classexpr*⟩`</domainRestriction>`
   | `<typeRestriction>`⟨*datatype*⟩`</typeRestriction>`

3.1.5 Attribute expression

Attribute expressions are a restriction of a property in the context of a particular class:

⟨*attexpr*⟩ ::= `<Attribute>`
    `<context>` ⟨*classexpr*⟩ `</context>`
    `<onProperty>` ⟨*relexpr*⟩`</onProperty>`
   `</Attribute>`

3.1.6 Instances

Instance expressions are simply specified using the instance URIs:

⟨*instance*⟩ ::= `<Instance rdf:about="`⟨*uri*⟩`"/>`

3.1.7 Metadata

It is often useful to annotate alignments with additional information. Metadata gives the opportunity to programs to exchange information which is not part of the format. This was not part of the abstract syntax.

This is achieved by introducing annotations in Alignment and Cell expressions. These annotations contain string values.

⟨*annotation*⟩ ::= < ⟨*uri*⟩ > ⟨*annotationValue*⟩ </ ⟨*uri*⟩ >

⟨*annotationValue*⟩ ::= ⟨*string*⟩

There are a number of common annotations that can be used already. They are summarised in Table 3.1[4].

The set of annotations can be extended by the users and it is a good practice for tools to be able to preserve all these annotations.

---

[4]Other types of declared annotations can be found at `http://alignapi.gforge.inria.fr/labels.html`

Annotation	Type	Content
type	char	the kind of alignment it is (1:1 or n:m for instance)
level	xsd:string	the language level used in the alignment (level 0 for the initial alignment API, level 2OMWG for the language defined here)
method	classname	the algorithm that provided it (or if it has been provided by hand)
dc:creator	xsd:string/URI	the person who produced the alignment
dc:date	xsd:date	the date of creation or modification for the alignment
purpose	xsd:string	the purpose for which the alignment has been produced
parameters	Parameters	the parameters passed to the generating algorithm
time	xsd:duration	the time spent for generating the alignment
limitations	xsd:string	the limitations of the use of the alignment
properties	undef	the properties satisfied by the correspondences (and their proof if necessary)
certificate	undef	the certificate from an issuing source
arguments	Arguments	the arguments in favour or against a correspondence

Table 3.1: Metadata labels for annotating alignments.

3.2 Surface syntax

The surface syntax is meant to be a human readable syntax. It is more compact than the XML/RDF exchange syntax but more explicit than the abstract syntax. Its compactness helps providing a short description of it through a grammar. This language is an evolution of the surface language described in [Scharffe and Kiryakov, 2005].

An alignment is expressed as a mapping document which has an identifier, source and target ontology references, possibly annotations and a list of correspondences ($\langle expression \rangle$):

$\langle mappingdocument \rangle$ ::= MappingDocument ($\langle documentid \rangle$
                    $\langle headers \rangle$
                    $\langle sourceexp \rangle$
                    $\langle targetexp \rangle$
                    $\langle annotation \rangle$*
                    $\langle expression \rangle$*)

$\langle header \rangle$ ::= ($\langle entity \rangle$ | $\langle namespace \rangle$)*

$\langle entity \rangle$ ::= XMLentity($\langle string \rangle$ $\langle irid \rangle$)

$\langle namespace \rangle$ ::= namespace($\langle string \rangle$ $\langle irid \rangle$)

$\langle sourceexp \rangle$ ::= onto1($\langle formalism \rangle$ $\langle ontologyid \rangle$)

$\langle targetexp \rangle$ ::= onto2($\langle formalism \rangle$ $\langle ontologyid \rangle$)

$\langle annotation \rangle$ ::= annotation($\langle irid \rangle$ $\langle propertyvalue \rangle$)

$\langle formalism \rangle$ ::= formalism($\langle string \rangle$ $\langle irid \rangle$)

Annotations are property-values pairs; ontologies are simply described by their identifier and language description.

⟨*expression*⟩ ::= `MappingRule(` ⟨*mappingid*⟩ ⟨*annotation*⟩`*` ⟨*measure*⟩`?` ⟨*relation*⟩`?`
                   `entity1(` ⟨*entity*⟩ `)` `entity2(` ⟨*entity*⟩ `)`

⟨*mappingid*⟩ ::= `id(` ⟨*irid*⟩ `)`

⟨*measure*⟩ ::= `measure(` ⟨*float*⟩ `)`

⟨*relation*⟩ ::= `relation(` ⟨*relname*⟩ `)`

⟨*relname*⟩ ::= `equivalent` | `subsume` | `subsumed` | `isa` | `asi`

The entities which are put in correspondences are either classes, relations, properties, attributes or instance expressions. They can be associated with transformations which are either internal functions or service call:

⟨*entity*⟩   ::= `Class(` ⟨*classexpr*⟩ `)`
               | `Relation(` ⟨*relationexpr*⟩ `)`
               | `Property(` ⟨*propertyexpr*⟩ `)`
               | `Attribute(` ⟨*classexpr*⟩ ⟨*relationexpr*⟩ `)`
               | `Instance(` ⟨*instanceid*⟩ `)`

Class, relation and attribute expressions closely corresponds to those presented in the previous sections:

⟨*classexpr*⟩ ::= ⟨*classid*⟩
               | `and(` [first]: ⟨*classexpr*⟩ [second]: ⟨*classexpr*⟩`+` `)`
               | `or(` [first]: ⟨*classexpr*⟩ [second]: ⟨*classexpr*⟩`+` `)`
               | `not(` ⟨*classexpr*⟩ `)`
               | ⟨*condition*⟩

⟨*relationexpr*⟩ ::= ⟨*relationid*⟩
               | `and(` [first]: ⟨*relationexpr*⟩ [second]: ⟨*relationexpr*⟩`+` `)`
               | `or(` [first]: ⟨*relationexpr*⟩ [second]: ⟨*relationexpr*⟩`+` `)`
               | `not(` ⟨*relationexpr*⟩ `)`
               | `domain(` ⟨*classexpr*⟩ `)`
               | `range(` ⟨*classexpr*⟩ `)`
               | `inverse(` ⟨*relationexpr*⟩ `)`
               | `symetric(` ⟨*relationexpr*⟩ `)`
               | `transitive(` ⟨*relationexpr*⟩ `)`
               | `reflexive(` ⟨*relationexpr*⟩ `)`

⟨*propertyexpr*⟩ ::= ⟨*propertyid*⟩
               | `and(` [first]: ⟨*propertyexpr*⟩ [second]: ⟨*propertyexpr*⟩ ⟨*propertyexpr*⟩`*` `)`
               | `or(` [first]: ⟨*propertyexpr*⟩ [second]: ⟨*propertyexpr*⟩ ⟨*propertyexpr*⟩`*` `)`
               | `not(` ⟨*propertyexpr*⟩ `)`
               | `domain(` ⟨*classexpr*⟩ `)`
               | `range(` ⟨*typeexpr*⟩ `)`

Conditions can be specified on paths so these are introduced and defined precisely as in the previous sections:

⟨*condition*⟩ ::= valuerestriction(⟨*path*⟩ ⟨*comparator*⟩ ⟨*pathorvalue*⟩)
      | domainrestriction(⟨*path*⟩ ⟨*comparator*⟩ ⟨*typeid*⟩)
      | cardinality(⟨*path*⟩ ⟨*comparator*⟩ ⟨*number*⟩)

⟨*comparator*⟩ ::= > | >= | < | <= | = | != | ⟨*irid*⟩

⟨*pathorvalue*⟩ ::= ⟨*path*⟩
      | ⟨*literal*⟩
      | instance(⟨*instanceid*⟩)
      | ⟨*transformation*⟩

⟨*path*⟩   ::= ⟨*relationid*⟩
      | ⟨*propertyid*⟩
      | path(⟨*relationid*⟩* ⟨*propertyid*⟩?)

⟨*transformation*⟩ ::= transformation(⟨*functionid*⟩ ⟨*pathorvalue*⟩*)
      | transformation(⟨*service*⟩ ⟨*irid*⟩ ⟨*pathorvalue*⟩*)

⟨*functionid*⟩ ::= ⟨*string*⟩ | ⟨*irid*⟩

There are two different ways to express transformations depending on their reliance on embedded functions or web services.

Finally litterals are identified like in RDF

⟨*literal*⟩   ::= ⟨*typedliteral*⟩
      | ⟨*plainliteral*⟩
      | ⟨*number*⟩
      | ⟨*string*⟩

⟨*typedliteral*⟩ ::= ⟨*plainliteral*⟩ ^^ ⟨*typeid*⟩

⟨*plainliteral*⟩ ::= " ⟨*literalcontent*⟩* " ⟨*languagetag*⟩?

Most items are identified by IRIs (the internationalised version of URIs, i.e., allowing other characters than ASCII).

⟨*documentid*⟩ ::= ⟨*irid*⟩

⟨*ontologyid*⟩ ::= ⟨*irid*⟩

⟨*classid*⟩   ::= ⟨*irid*⟩

⟨*propertyid*⟩ ::= ⟨*irid*⟩

⟨*attributeid*⟩ ::= ⟨*irid*⟩

⟨*relationid*⟩ ::= ⟨*irid*⟩

⟨*instanceid*⟩ ::= ⟨*irid*⟩

⟨*irid*⟩   ::= <" ⟨*iri*⟩ ">

Here is an example of this surface syntax:

```
MappingDocument(
  id(<"http://oms.omwg.org/example/">)
  XMLEntity("wine" <"http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#">)
  XMLEntity("vin" <"http://ontology.deri.org/vin#">)
  namespace("xmlns" <"http://knowledgeweb.semanticweb.org/heterogeneity/alignment">)
  namespace("xmlns:dc" <"http://purl.org/dc/elements/1.1/">)
  namespace("xmlns:align" <"http://www.w3.org/2001/XMLSchema#">)
  annotation(<"dc:creator"> "http://sw.deri.org/~francois/")
  annotation(<"dc:date"> "2006/06/07")
  annotation(<"method">, "manual")
  annotation(<"level">, "2OMWG")
  annotation(<"type">, "**")
  onto1( formalism('owl',<"http://www.w3.org/2002/07/owl">) <"&wine;"> )
  onto2( formalism('wsml',<"http://www.wsmo.org/wsml/wsml-syntax/wsml-dl">) <"&vin;"> )
  MappingRule(
       id("MappingRule_0")
       measure(1.0)
       relation(equivalence)
       entity1( Class(<"&wine;VintageYear">))
       entity2( Class(<"&vin;Millesime">)))
  MappingRule(
       id("MappingRule_1")
       measure(1.0)
       relation(equivalence)
       entity1( Attribute(<"&wine;yearValue">))
       entity2( Attribute(<"&vin;anneeMillesime">)))
  MappingRule(
       id("MappingRule_2")
       measure(.9)
       relation(subsumption)
       entity1( Relation(<"&wine;locatedIn">))
       entity2( Relation(<"&vin;hasTerroir">)))
  MappingRule(
       id("MappingRule_3")
       measure(1.0)
       relation(subsumption)
       entity1( Class(<"&wine;Bordeaux">))
       entity2( Class(and(<"&vin;Vin">,
                        valuerestriction(
                             path(<"vin;hasTerroir"> <"&proton;LocatedIn">)
                             equal "Bordeaux")))))
  MappingRule(
       id("MappingRule_5")
       measure(1.0)
       relation(equivalence)
       entity1( Property(<"&wine;hasVintageYear">))
       entity2(Property(and(<"&vin;annee_millesime">
                        transformation(Transformation:Date2Int))))))
  MappingRule(
       id("MappingRule_4")
```

```
measure(1.0)
relation(subsumes)
entity1( Class(<"&vin;WineFlavor">))
entity2(Class(or(<"&vin;Acidite">
                 <"&vin;Astreingence">
                 <"&vin;Amertume">)))))
```

3.3 Synthesis

We have provided two different syntax for expressive alignments.   They are expressive enough for covering the examples provided in Chapter 1. These syntaxes are designed for being strictly equivalent, though (for space reasons) we did not provide transformation rules. The exchange syntax and the surface syntax are designed for being realised and handled by systems. They are indeed handled in input and output by the OMWG parsers. The abstract syntax is only meant for specifying the semantics of such alignments and is not available in any tool.

Going one step further in this direction would be the design of a graphical expressive syntax for alignments. However, it is difficult to separate such a syntax from ontology syntax. The figures of Chapter 1 could be considered as the sketch of such a syntax and they display a syntax for ontology languages as well.

Next chapter consider the implementation of this expressive language.

Chapter 4

Expressive language implementation

We have implemented tools for supporting the new alignment format by extending and using together two ontology alignment tools. These two tools where developed relatively independently but in compatible ways so we have adapted them in order to achieve the best implementation of the proposed language. We first present both tools independently and explain the integration that has been achieved.

4.1 OMWG Mapping API

The OMWG Mapping API[1] has been developed for providing an expressive language to mediate between semantic web services in the WSMX framework. However, it is independent from that framework. This section introduce the Java API developed and implemented to support the mapping language. The different component constituting the API, namely the parsers, the object model, the export module and the adapter interface are presented.

4.1.1 Parsers

The mapping language has two syntaxes: a surface syntax and a RDF/XML syntax.

The surface syntax parser is based on an EBNF grammar and corresponds to the language presented in §3.2. We used Sablecc[2] to generate a parser from the grammar specification. A tree-walker goes through the abstract-syntax tree and populate the object model of the API. The EBNF grammar is available in [Scharffe and Kiryakov, 2005].

The RDF/XML syntax parser is based on the javax.xml.parser[3] Java package and works similarly.

4.1.2 Object Model

The parsers instanciate an object model providing means to manipulate alignments. Classes and properties of the model follow the structure of the language. The top-level class MappingDocument contains information about the document: the source ontology, the target ontology, the different correspondences and annotations. It is possible to access and modify annotations, id of

---

[1]http://sourceforge.net/projects/mediation/
[2]http://www.sablecc.org
[3]http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/package-summary.html

the document, the source and target ontologies. It is also possible to add or suppress a correspondence. Once the mapping document edited, it can be exported via the export module.

4.1.3 Export Module

The export module offers the possibility to export mapping documents in various gounding formats. It allows to export in the surface syntax of the mapping language, in OWL and in WSML. This implementation gives the possibility to use alignments at run-time by loading them into a mediator.

4.1.4 Known uses

The mapping API is used by the mapping editor from the Web Services Modelling Toolkit[4] (WSMT) in order to represent ontology alignments and execute them in a runtime mediator. In a similar way, Ontomap, the mapping tool from OntoStudio[5] allows to export alignments as mapping documents using the Mapping API. An online store for mapping documents[6] makes use of the Mapping API to check the validity of submitted alignments.

4.2 Alignment API implementation

The Alignment API[7] is an API and implementation for expressing and sharing ontology alignments [Euzenat, 2004]. It is a Java description of tools for accessing the common format. It defines four main interfaces (Alignment, Cell, Relation and Evaluator) and proposes the following services:

- – Storing, finding, and sharing alignments;
- – Piping alignment algorithms (improving an existing alignment);
- – Manipulating (thresholding and hardening);
- – Generating processing output (transformations, axioms, rules);
- – Comparing alignments.

4.2.1 Parsers

The Alignment API uses a general format for expressing alignments in a uniform way. The goal of this format is to be able to share on the web the available alignments. It helps systems using alignments, e.g., mergers, translators, to take advantage of any alignment algorithm and it will help alignment algorithms to be used in many different tasks. The format is expressed in RDF, so it is freely extensible, and has been defined by a DTD (for RDF/XML), an OWL ontology and an RDF Schema. Aligned entities are identified by their URIs.

However, the parser is designed to work only with the XML format.

---

[4]http://wsmt.sourceforge.net
[5]http://www.ontoprise.com
[6]http://oms.omwg.org
[7]http://alignapi.gforge.inria.fr

4.2.2 Object model and manipulation

The object model implements the Alignment API and provides the following features:

- a base implementation of the interfaces with all useful facilities;
- a library of sample matchers;
- a library of renderers;
- a library of evaluators (precision/recall, generalized precision/recall, precision/recall graphs and weighted Hamming distance);
- a parser for the format.

This implementation is now made available as an Alignment server which offers all these functions through HTTP/HTML, HTTP/SOAP and FIPA ACL interfaces.

Instanciating this API is achieved by refining the base implementation by implementing the align() method. Doing so, the new implementation will benefit from all the services already implemented in the base implementation.

4.2.3 Renderers

Renderers corresponding to the output formats are available for XSLT, SWRL, OWL, C-OWL, OMWG Mapping Language, and SKOS.

4.2.4 Known uses

The alignment API has been used for the processing of the EON Ontology Alignment Contest and the Ontology Alignment Evaluation Initiative 2005. It is used in the people's portal alignment tool at DERI Innsbruck and used or output by a number of alignment tools (among which OLA that we develop in common with the University of Montréal, CMS from University of Southampton or oMap from CNR/Pisa).

4.3 Combination of both API

Because these two tools are currently in use under different context we aimed at integrating them by preserving their independent behaviour. The goal of integrating these tools was to have the Alignment API benefit from an already expressive alignment language and maybe the WSMT alignment editor. From the standpoint of the OMWG Mapping API, being sited on top of the Alignment API provides a number of facilities: being used in all the contexts in which the Alignment API is used (in particular various matching algorithms and the Alignment server) and providing low level facilities (more renderers, sophisticated extraction algorithms, etc.).

Because the two implementations are compatible, we implemented them by making the OMWG mapping API object model an extension of the Alignment API implementation object language. Then, we "reimplemented" the base accessors of the OMWG mapping API to the accessors of the Alignment API so that any tool using the Alignment API could manipulate the MappingDocuments and the correspondences they contain. This is presented in Figure 4.1.

Parsing the Mapping Documents is something that was fully implemented in the OMWG mapping API, so we made the Alignement API to take advantage of them. Concerning the surface language, the corresponding OMWG parser, when invoked, creates MappingDocuments which
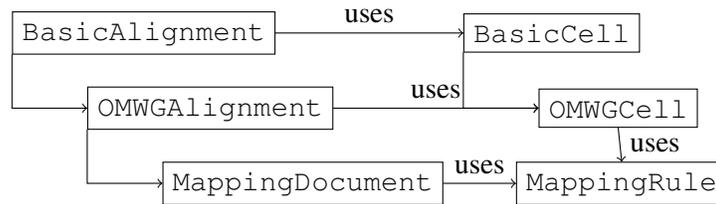
Figure 4.1: Integration of the object model of the OMWG Mapping API on top of the Alignment API.
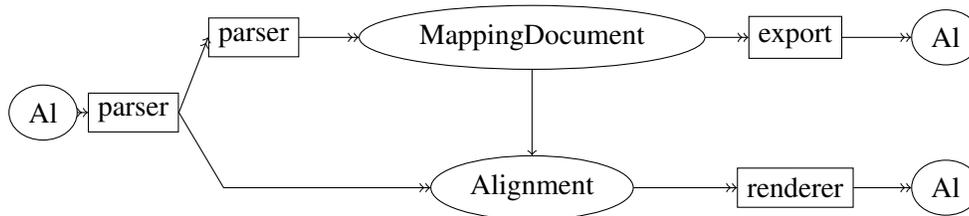
Figure 4.2: Invocation of the functions of the OMWG Mapping API and the Alignment API on the resulting objects.

are now also Alignment in the sense of the Alignment API, so they can be manipulated by this API. Concerning the RDF/XML format, we make the Alignment API parser switch to the OMWG mapping API parser whenever it encounters an alignment written in this expressive language. This is achieved when the parser recognises the "2OMWG" value in the level attribute of the alignment. The result is also a MappingDocument.

The objects created by the two parsers are now objects of the Alignment API and can be manipulated as such. Since they are still objects of the OMWG mapping API, then they can also be used as such.

Last, the Alignment API has a particular way of outputing alignments through "renderers". These had to be extended to be used with MappingDocuments. The Mapping API export module is still usable with the MappingDocuments (and the RDF/XML renderer of the Alignment API reuses parts of the OMWG implementation).

These dispositions are presented in Figure 4.2.

4.4   Synthesis

We have presented the integration of two existing tools in order to implement support for the expressive alignment language that has been presented in this deliverable. The result is fully integrated within both the Alignment API implementation and the OMWG Mapping API. It can be found at the Alignment API URL for the moment (starting with version 3.1).

This integration allows to retain the advantage of both tools, including the expressive language described here, while benefiting of the advantage of the other.

Chapter 5

Conclusions

The ontology alignment syntax and semantics presented in this report form a complete language. It is possible to use this language for modelling complex correspondences between ontologies. As shown on simple examples between two ontologies related to the wine domain, such complex correspondences are necessary to correctly represent the domain overlap between two heterogeneous ontological representations.

We provided support for this language in terms of expressing correspondences and manipulating them in a practical already usable setting. By combining the simplicity of the Alignment Format and the expressivity of the Mapping Language, we propose a language fulfilling the need for ontology mediation on the semantic web. The presented language takes into account the different types of entities used to represent structured data and allows as well to specify transformation of the data itself. Moreover, this language enables mediation between heterogeneous ontologies or schemata described using different languages. Mediating between different formalisms is made possible by the use of model theoretic semantics using local interpretations lifted to a global domain via equalising functions.

We provided an alignment language as expressive as possible without any regard to its complexity. We aimed at providing a very expressive language that people will taylor if they do not need the expressiveness or if they want to trade it for decidable or efficient reasoning. By not restricting the expressiveness of the language a priori, we have allowed ourselves to provide the syntax and the semantics for the whole language at once. In fact, reasoning with such languages requires not uniquely reasoners in this language but reasoners in the two connected ontology languages. Of course, providing inference support for such a language could be an interesting challenge.

The language as described will not look any new for someone moderately accointed with description logics [Baader *et al.*, 2003]. Indeed, we did not attempt to be original in this matter. But the reader should note that this language allows to use any kind of language as ontology languages. It is not meant to be used only with ontology languages based on description logics, though we assume a model theoretic semantics to these languages.

This language can be used for many different applications. From graphical user interfaces assisting in aligning ontologies to web service mediator rewriting queries or transforming instances a wide range of application can be foreseen. The expressiveness of this language might lead research in ontology matching to develop algorithms able to find complex correspondences such as those presented here.

Bibliography

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The description logic handbook: theory, implementations and applications*. Cambridge University Press, 2003.

[Borgida and Serafini, 2003] Alex Borgida and Luciano Serafini. Distributed Description Logics: Assimilating information from peer sources. *J. Data Semantics*, pages 153–184, 2003.

[Dean and Schreiber (eds.), 2004] Mike Dean and Guus Schreiber (eds.). OWL web ontology language: reference. Recommendation, W3C, 2004. http://www.w3.org/TR/owl-ref/.

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

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

[Euzenat *et al.*, 2005b] Jérôme Euzenat, François Scharffe, and Luciano Serafini. Specification of the delivery alignment format. Deliverable 2.2.6, Knowledge web NoE, 2005.

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.

[Lausen *et al.*, 2005] Holger Lausen, Jos de Bruijn, Axel Polleres, , and Dieter Fensel. WSML – a language framework for semantic web services. In *Proceedings of the W3C Workshop on Rule Languages for Interoperability*, Washington (DC US), 2005.

[Scharffe and de Bruijn, 2005] François Scharffe and Jos de Bruijn. A language to specify mappings between ontologies. In *Proc. of the SITIS05 IEEE Conference*, 2005.

[Scharffe and Kiryakov, 2005] François Scharffe and Atanas Kiryakov. Omwg d7.2: Mapping and merging tool design. Working Draft D7.2v0.2, OMWG, 2005.

[Scharffe *et al.*, 2005] François Scharffe, Jos de Bruijn, and Douglas Foxvog. Ontology mediation patterns library v2. Deliverable D4.3.2, SEKT, 2005.

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

Appendix A

Semantic chart

This appendix summarises the abstract syntax and semantics of the expressive alignment language. It is dependent upon:

- an external type system providing functions $D(\cdot)$, $L(\cdot)$ and $L2V(\cdot)$ mapping values to a set $\mathbb{D}$ (§2.3.1);
- an external set of operators providing comparators g_{cp} and transformations h_{transf} (§2.3.1);
- external ontology semantics providing $\mathcal{I}_U$ mapping URI to a set Δ and instances to a set $\mathcal{D}$ (§2.4).

In case of syntactically incorrect expressions, this semantics will provide undefined results.

Abstract syntax	Interpretation	Domain
Literals (v)	$\mathcal{I}_L$	$\mathbb{D}$
$\texttt{"}v\texttt{"}\char`\^\char`\^d$		$L2V(D(d))(v)$
$\texttt{"}v\texttt{"}$		$\texttt{"}v\texttt{"}$
URI References (u)	$\mathcal{I}_U$	$\Delta \supseteq \mathcal{D} \cup 2^{\mathcal{D}} \cup 2^{\mathcal{D} \times (\mathcal{D} \cup \mathbb{D})}$
u		$u^{\mathcal{I}_U}$
Paths (Q)	$\mathcal{I}_P$	$2^{\mathcal{D} \times (\mathcal{D} \cup \mathbb{D})}$
p		$p^{\mathcal{I}_U}$
r		$r^{\mathcal{I}_U}$
$Q.p$		$\{\langle x,y \rangle \in \mathcal{D} \times \mathbb{D} / \exists z \in \mathcal{D} / \langle x,z \rangle \in Q^{\mathcal{I}_P} \wedge \langle z,y \rangle \in p^{\mathcal{I}_P}\}$
$Q.r$		$\{\langle x,y \rangle \in \mathcal{D} \times \mathcal{D} / \exists z \in \mathcal{D} / \langle x,z \rangle \in Q^{\mathcal{I}_P} \wedge \langle z,y \rangle \in r^{\mathcal{I}_P}\}$
ϵ		$\{\langle x,x \rangle / x \in \mathcal{D}\}$
Paths or values (V)	$\mathcal{I}_V$	$2^{\mathcal{D} \times (\mathcal{D} \cup \mathbb{D})}$
v		$\mathcal{D} \times \{v^{\mathcal{I}_L}\}$
i		$\mathcal{D} \times \{i^{\mathcal{I}_U}\}$
Q		$Q^{\mathcal{I}_P}$
$\mathrm{transf}(V_1 \ldots V_k)$		$\{\langle x, h_{\mathrm{transf}}(y_1, \ldots, y_n) \rangle / \langle x, y_1 \rangle \in V_1^{\mathcal{I}_V} \wedge \cdots \wedge \langle x, y_n \rangle \in V_k^{\mathcal{I}_V}\}$
Restrictions (K)	$\mathcal{I}_K$	$2^{\mathcal{D}}$
$Q\ cp\ V$		$\{x \in \mathcal{D} / \exists y, y' \in \mathcal{D} \cup \mathbb{D} / \langle x,y \rangle \in Q^{\mathcal{I}_P} \wedge \langle x,y' \rangle \in V^{\mathcal{I}_V} \wedge g_{\mathrm{cp}}(y,y')\}$
$Q\ cp\ d$		$\{x \in \mathcal{D} / \forall y \in \mathcal{D} \cup \mathbb{D}, (\langle x,y \rangle \in Q^{\mathcal{I}_P}) \Rightarrow g_{\mathrm{cp}}(y, L(D(d)))\}$
$|Q|\ cp\ n$		$\{x \in \mathcal{D} / g_{\mathrm{cp}}(|\{y \in \mathcal{D} \cup \mathbb{D} / \langle x,y \rangle \in Q^{\mathcal{I}_P}\}|, n^{\mathcal{I}_L})\}$

Table A.1: Abstract syntax and semantics.

Abstract syntax	Interpretation	Domain
Classes (C)	$\mathcal{I}_C$	$2^{\mathcal{D}}$
c		$c^{\mathcal{I}_U}$
$C \sqcup C'$		$C^{\mathcal{I}_C} \cup C'^{\mathcal{I}_C}$
$C \sqcap C'$		$C^{\mathcal{I}_C} \cap C'^{\mathcal{I}_C}$
$\neg C$		$\mathcal{D} \setminus C^{\mathcal{I}_C}$
$\exists K$		$K^{\mathcal{I}_K}$
Relations (R)	$\mathcal{I}_{CP}$	$2^{\mathcal{D} \times \mathcal{D}}$
r		$r^{\mathcal{I}_U}$
$R \sqcup R'$		$R^{\mathcal{I}_{CP}} \cup R'^{\mathcal{I}_{CP}}$
$R \sqcap R'$		$R^{\mathcal{I}_{CP}} \cap R'^{\mathcal{I}_{CP}}$
$\neg R$		$\mathcal{D} \times \mathcal{D} \setminus R^{\mathcal{I}_{CP}}$
$\mathrm{dom}(C)$		$\{\langle x,y\rangle \in \mathcal{D} \times \mathcal{D} / x \in C^{\mathcal{I}_C}\}$
$\mathrm{range}(C)$		$\{\langle x,y\rangle \in \mathcal{D} \times \mathcal{D} / y \in C^{\mathcal{I}_C}\}$
$\mathrm{inv}(R)$		$\{\langle x,y\rangle / \langle y,x\rangle \in R^{\mathcal{I}_{CP}}\}$
$\mathrm{sym}(R)$		$R^{\mathcal{I}_{CP}} \cup \mathrm{inv}(R)^{\mathcal{I}_{CP}}$
$\mathrm{trans}(R)$		$R^{\mathcal{I}_{CP}} \cup \{\langle x,z\rangle / \exists y \in \mathcal{D}, \langle x,y\rangle \in R^{\mathcal{I}_{CP}} \wedge \langle y,z\rangle \in trans(R)^{\mathcal{I}_{CP}}\}$
$\mathrm{refl}(R)$		$R^{\mathcal{I}_{CP}} \cup \{\langle x,x\rangle / x \in \mathcal{D}\}$
Properties (P)	$\mathcal{I}_{DP}$	$2^{\mathcal{D} \times \mathbb{D}}$
p		$p^{\mathcal{I}_U}$
$P \sqcup P'$		$P^{\mathcal{I}_{DP}} \cup P'^{\mathcal{I}_{DP}}$
$P \sqcap P'$		$P^{\mathcal{I}_{DP}} \cap P'^{\mathcal{I}_{DP}}$
$\neg P$		$\mathcal{D} \times \mathbb{D} \setminus P^{\mathcal{I}_{DP}}$
$\mathrm{dom}(C)$		$\{\langle x,y\rangle \in \mathcal{D} \times \mathbb{D} / x \in C^{\mathcal{I}_C}\}$
$\mathrm{range}(d)$		$\{\langle x,y\rangle \in \mathcal{D} \times \mathbb{D} / y \in L(D(d))\}$
Instances (i)	$\mathcal{I}_U$	$\mathcal{D}$
i		$i^{\mathcal{I}_U}$
Expressions (E)	$\mathcal{I}$	$\Delta \supseteq \mathcal{D} \cup 2^{\mathcal{D}} \cup 2^{\mathcal{D} \times (\mathcal{D} \cup \mathbb{D})}$
C		$C^{\mathcal{I}_C}$
R		$R^{\mathcal{I}_{CP}}$
P		$P^{\mathcal{I}_{DP}}$
i		$i^{\mathcal{I}_U}$

Table A.1: Abstract syntax and semantics.

Appendix B

Data manipulation

B.1   Data operator table

Type	Id	Origin	explanation
numeric	add	XQuery	Returns the arithmetic sum of the first argument through the second argument.
numeric	subtract	XQuery	Returns the arithmetic difference of the first argument minus the second argument.
numeric	multiply	XQuery	Returns the arithmetic product of the first argument by the second argument.
numeric	divide	XQuery	Returns the arithmetic quotient of the first argument over the second argument.
numeric	integer-divide	XQuery	Returns the integer part of the arithmetic quotient of the first argument over the second argument.
numeric	mod	XQuery	Returns the modulo of the arithmetic quotient of the first argument over the second argument.
numeric	pow		Returns the first argument raised to the second argument power.
numeric	unary-minus	XQuery	Returns its argument with the sign changed.
string	concat	XQuery	Concatenates two strings.
string	substring	XQuery	Returns the substring of its first argument starting at the position denoted by its second argument and ending at the one denoted by the third argument.
string	length	XQuery	Returns the integer corresponding to the number of characters of the string in argument.
string	normalize-space	XQuery	Returns the whitespace-normalised value of the string in argument.
string	upper-case	XQuery	Returns the upper-cased value of the string in argument.
string	lower-case	XQuery	Returns the lower-cased value of the string in argument.
string	translate	XPath/XQuery	Returns its first string argument with occurrences of characters contained in the second argument replaced by the character at the corresponding position in the string of the third argument.

Table B.1: Operators.

Type	Id	Origin	explanation
string	replace	XQuery	Returns its first string argument with every substring matched by the regular expression in the second argument replaced by the replacement string of the third argument.
string	tokenize	XQuery	Returns a sequence of strings whose values are ordered substrings of the first argument separated by substrings that match the regular expression the second argument.
uri	resolveURI	XQuery	Returns the URI reference value of its argument resolved.
collection	concatenate	XQuery	Returns the concatenation of its list arguments.
collection	intersection		Returns a list containing elements found in both the first list argument and the second list argument.
collection	union		Returns a list containing the elements found in any of its list arguments.
collection	difference		Returns a list containing the elements of the first list argument that are not members of the second list argument.
integer	length	Lisp	Returns the number of elements in its list argument.

Table B.1: Operators.

B.2   Comparator table

Type	Id	Origin	explanation
all	equal	XQuery	Satisfied iff the first argument and the second argument are the same.
all	not-equal	SWRL	The negation of equal.
ordered	less-than	XQuery	Satisfied iff the first argument and the second argument are both in some implemented type and the first argument is less than the second argument according to a type-specific ordering (partial or total), if there is one defined for the type. The ordering function for the type of untyped literals is the partial order defined as string ordering when the language tags are the same (or both missing) and incomparable otherwise.
ordered	less-than-or-equal	SWRL	Either less than, as above, or equal, as above.
ordered	greater-than	XQuery	Similarto less-than.
ordered	greater-than-or-equal	SWRL	Similar to less-than-or-equal.
string	contains	XQuery	Satisfied iff the first argument contains the second argument (case sensitive)
string	starts-with	XQuery	Satisfied iff the first argument starts with the second argument.
string	ends-with	XQuery	Satisfied iff the first argument ends with the second argument.
string	matches	XQuery	Satisfied iff the first argument matches the regular expression in the second argument.
collection	contains	XQuery	Satisfied iff the first argument contains the second argument
collection	includes	XQuery	Satisfied iff the first argument contains all the elements the second argument.
collection	includes-strictly	XQuery	Satisfied iff the first argument contains more elements than the the second argument.
collection	empty		Satisfied iff its first list argument is an empty list.

Table B.2: Comparators.

Appendix C

OWL Ontology

This appendix exhibits the alignment exchange language as a OWL ontology (version of 17/03/2007) that can be found at `http://www.omwg.org/TR/d7/ontology/alignment/`.

```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:align="http://knowledgeweb.semanticweb.org/heterogeneity/alignment#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns="http://www.omwg.org/TR/d7/d7.2/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.omwg.org/TR/d7/d7.2/">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/PoV">
    <owl:equivalentClass>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Path"/>
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Value"/>
        </owl:unionOf>

      </owl:Class>
    </owl:equivalentClass>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A path or a value</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Cell">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>

          <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/entity2"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
```

```xml
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>

        <owl:onProperty>
          <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/entity1"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>

        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/relation"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>

          <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/measure"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Cells are the objects representing mapping rules</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/AttributeCondition">

    <rdfs:subClassOf>
      <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Condition"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Set of conditions that can be applied on Attribute entities</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Instance">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Instance entities represent instance data of the ontology.</rdfs:comment>
    <rdfs:subClassOf>

      <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Entity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/AttributeOccurenceCondition">
    <rdfs:subClassOf>
      <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/ClassCondition"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This set a condition on the existence (in term of instanciation) of one of the class attri
```

```
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Class">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/condition"/>
          </owl:onProperty>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >0</owl:minCardinality>

        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Entity"/>
      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Class entities represent the objects of the ontology.</rdfs:comment>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Formalism">

      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/name"/>
          </owl:onProperty>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:minCardinality>
        </owl:Restriction>

      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:cardinality>
          <owl:onProperty>
            <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/uri"/>
          </owl:onProperty>
        </owl:Restriction>

      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Entity">
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:maxCardinality>
          <owl:onProperty>

            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/operator"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Entities are ontological entities part of the mapping rules (cells)</rdfs:comment>
```

```
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Ontology">
      <rdfs:subClassOf>

        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/formalism"/>
          </owl:onProperty>
          <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:cardinality>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>

      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Ontologies are of this type</rdfs:comment>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Condition">
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >The class Condition groups the various conditions that can be applied on entities. It is p
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Value">
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>

        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:cardinality>
              <owl:onProperty>
                <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/stringValue"/>
              </owl:onProperty>
            </owl:Restriction>

            <owl:Restriction>
              <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
              >1</owl:cardinality>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/instanceValue"/>
              </owl:onProperty>
            </owl:Restriction>
          </owl:unionOf>
        </owl:Class>

      </rdfs:subClassOf>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >A Value object can be instanciated either with an instanceValue or a stringValue</rdfs:com
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/ClassCondition">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/restriction"/>

          </owl:onProperty>
```

```
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/Condition"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Set of conditions that can be applied on Class entities</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Restriction">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onProperty"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/value"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/comparator"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Restrictions objects are of this type</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Alignment">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/type"/>

        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
```

```
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:maxCardinality>

    <owl:onProperty>
      <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/xml"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:maxCardinality>

    <owl:onProperty>
      <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/method"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:minCardinality>

    <owl:onProperty>
      <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/map"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Alignment objects represent an alignment between two ontologies. Each alignment is alos co
<rdfs:subClassOf>
  <owl:Restriction>

    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:maxCardinality>
    <owl:onProperty>
      <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/level"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>

    <owl:onProperty>
      <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onto1"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>

    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
```

```
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onto2"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>

      <owl:onProperty>
        <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/purpose"/>
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Attribute">

  <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >0</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/condition"/>
      </owl:onProperty>
    </owl:Restriction>

  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Attribute enties represent properties of the ontology which range is a datatype.</rdfs:com
</owl:Class>
<owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/TypeCondition">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This set a condition on the atribute type.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/AttributeCondition"/>
</owl:Class>
<owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/AttributeValueCondition">

  <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/ClassCondition"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This set a condition on the value of one of the class attributes</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/ValueCondition">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>

        <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/comparator"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
      >This set a condition on the attribute value.</rdfs:comment>
      <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/AttributeCondition"/>
      <rdfs:subClassOf>
        <owl:Restriction>

          <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:cardinality>
          <owl:onProperty>
            <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/value"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/RelationCondition">

      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Set of conditions that can be applied on Relation entities</rdfs:comment>
      <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/Condition"/>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Path">
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Pathes objects represent pathes through the RDF graph in order to reach a particular set o
Example: The set of persons who have a female sibling having exactly two children.</rdfs:comme
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>

          <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:cardinality>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/first"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>

          <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</owl:cardinality>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/next"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
    <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Collection">

      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >2</owl:minCardinality>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/item"/>
          </owl:onProperty>
        </owl:Restriction>
```

```
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/AttributeTypeCondition">
    <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/ClassCondition"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This set a condition on the type of one of the class attributes.</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Relation">
    <rdfs:subClassOf>

      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >0</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/condition"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Relation entities represent properties of the ontology which range is a Class entity. Only

    <rdfs:subClassOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
  </owl:Class>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/operator">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >operators to combine entities</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onto1">
    <rdfs:subPropertyOf>

      <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/ontology"/>
    </rdfs:subPropertyOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >First ontology of the alignment</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/instanceValue">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Value"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A value given as an instance entity</rdfs:comment>
    <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Instance"/>

  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/entity">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Cell"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Entities of a Cell</rdfs:comment>
    <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/next">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Path"/>

    <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Path"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Indicate the following path of a path, or nil to terminate a path</rdfs:comment>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/transitive">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >transitive closure of a relation</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onProperty">
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Restriction"/>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Path"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >properties applied to pathes</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/formalism">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Indicate the formalism (ontology language) of the aligned ontologies</rdfs:comment>

  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Formalism"/>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Ontology"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/inverse">
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Inverse of a relation</rdfs:comment>
  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/onto2">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/ontology"/>
  </rdfs:subPropertyOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Second ontology of the alignment</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/rangeRestriction">

  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Class"/>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Restrict the range of the Relation to a given Class expression</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/entity1">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >First entity of a cell</rdfs:comment>
  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/entity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/item">
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Collection"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/condition">
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Entity"/>
```

```
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Define a condition on an expression in order to restrict its scope.</rdfs:comment>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Condition"/>

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/restriction">
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Condition"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A restriction specifies a the set of entities that will fulfill the related condition.</rd
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Restriction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/or">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >cunjunction of the entities</rdfs:comment>

  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Collection"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/reflexive">
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Reflexive of a relation</rdfs:comment>
  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/map">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Relate an alignment to each of its cells</rdfs:comment>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Cell"/>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Alignment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/domainRestriction">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Restrict the domain of an Attribute or Relation to the given Class expression</rdfs:commen

  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/RelationCondition"/>
        <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/AttributeCondition"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Class"/>

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/and">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >intersection of the entities</rdfs:comment>
  <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Collection"/>
  <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/symmetric">
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
```

```
    <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Relation"/>
    <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Symmetric of a relation</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/ontology">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Alignment"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Represent the source and target ontologies of an alignment</rdfs:comment>
    <rdfs:range rdf:resource="http://www.omwg.org/TR/d7/d7.2/Ontology"/>

  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/not">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Negation of an entity</rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/operator"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/first">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Path"/>
    <rdfs:range>

      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Attribute"/>
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Relation"/>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <Entity rdf:about="http://www.omwg.org/TR/d7/d7.2/null-entity">
                <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >This entity is used to terminate a path.</rdfs:comment>

              </Entity>
            </owl:oneOf>
          </owl:Class>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Indicates the first element in a path</rdfs:comment>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/entity2">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Second entity of a cell</rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="http://www.omwg.org/TR/d7/d7.2/entity"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/xml">
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">

          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >no</rdf:first>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          </rdf:rest>
```

```
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >yes</rdf:first>
          </owl:oneOf>
        </owl:DataRange>
      </rdfs:range>

      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Indicate if the alignment is expressed in XML format. (deprecated?)</rdfs:comment>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/relation">
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Cell"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >relation between the aligned entities in a mapping rule.</rdfs:comment>
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">

          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >equivalence</rdf:first>
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >subsumption</rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:parseType="Resource">
                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >disjoint</rdf:first>

              </rdf:rest>
              <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >instanceOf</rdf:first>
            </rdf:rest>
          </rdf:rest>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/uri">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >uri of a formalism (ontology language)</rdfs:comment>
    <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Formalism"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/value">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>

      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/ValueCondition"/>
          <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Restriction"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
   >A value indicate the value compared in a condition. It can take either the value of an Att

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/measure">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Measure of confidence in a mapping rule.</rdfs:comment>
  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Cell"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/purpose">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Describe the purpose of the alignment</rdfs:comment>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/method">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Indicates the method, the name of the tool or algoriithm that outputted the alignment</rdf
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/stringValue">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A value given as a string</rdfs:comment>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/comparator">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Comparators are used in conditions in order to make comparisons of attribute values. The v
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/Restriction"/>
        <owl:Class rdf:about="http://www.omwg.org/TR/d7/d7.2/ValueCondition"/>

      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/level">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Indicates the level of the alignment (deprecated?, alway 2)</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>

</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/type">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Indicate the arity of the alignment (Deprecated, always **)</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.omwg.org/TR/d7/d7.2/name">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Name of a formalism or ontology langage</rdfs:comment>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

  <rdfs:domain rdf:resource="http://www.omwg.org/TR/d7/d7.2/Formalism"/>
```

```
    </owl:DatatypeProperty>
    <rdf:Description rdf:about="http://knowledgeweb.semanticweb.org/heterogeneity/alignment">
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >RDF vocabulary for the alignment format</rdfs:comment>
    </rdf:Description>
    <Path rdf:about="http://www.omwg.org/TR/d7/d7.2/nil">
      <first rdf:resource="http://www.omwg.org/TR/d7/d7.2/null-entity"/>
      <next rdf:resource="http://www.omwg.org/TR/d7/d7.2/nil"/>

      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Represent a null path. Used to end a path</rdfs:comment>
    </Path>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.2, Build 311)  http://protege.stanford.edu -->
```

Related deliverables

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

Project	Number	**Title** and relationship
KW	D2.2.1	**Specification of a common framework for characterising alignment** provided the framework for alignments.
KW	D2.2.2	**Specification of a benchmarking methodology for alignment techniques** describes the use of the Alignment format in evaluation.
KW	D2.2.3	**State of the art on ontology alignment** provides use cases and motivation for using ontology alignment.
KW	D2.2.5	**Integrated view and comparison of alignment semantics** compares several alignment formalisms on the basis of their expressiveness.
KW	D2.2.6	**Specification of the delivery alignment format** compares several alignment formats and proposes a join between the SEKT mapping language and the Alignment format. This deliverable updates 2.2.6 by creating this joint format, providing it with a semantics and implementing it.
SEKT	D4.4.1	**Ontology Mediation Management V1** defines the Mapping language at the source of the OMWG-ML described here.
OMWG	D7.2	**Ontology Mapping Language RDF/XML Syntax** defines the RDF/XML syntax for the language presented here after the work we have done for this deliverable.