D2.2.6: Specification of the delivery alignment format

Coordinator: Jérôme Euzenat (INRIA Rhône-Alpes)
François Scharffe (Innsbruck Universität),
Luciano Serafini (Universitá Trento)

Abstract.
This deliverable focusses on the definition of a delivery alignment format for tools producing alignments (mapping tools). It considers the many formats that are currently available for expressing alignments and evaluate them with regard to criteria that such formats would satisfy. It then proposes some improvements in order to produce a format satisfying more needs.
Keyword list: ontology matching, ontology alignment, ontology mapping, mediation, format, interoperability, OWL, SWRL, SBO, Alignment API, C-OWL, SEKT-ML, SKOS.

Document Identifier	KWEB/2005/D2.2.6/v1.0
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	February 2, 2006
State	final
Distribution	public

Knowledge Web Consortium

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Work package participants

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

Centre for Research and Technology Hellas
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	03.08.2005	Jérôme Euzenat	creation
0.2	01.11.2005	François Scharffe	input on SEKT Mapping language
0.3	12.11.2005	Jérôme Euzenat	description of the alignment API
0.4	27.11.2005	Jérôme Euzenat	various contributions
0.5	10.12.2005	Jérôme Euzenat	mapping integration
0.6	14.12.2005	Jérôme Euzenat	refined introduction and chapter 3
0.6a	15.12.2005	François Scharffe, Luciano Serafini	improved chapter 2 and 4
0.7	16.12.2005	Jérôme Euzenat	refined conclusion and various improvements
0.8	22.12.2005	Jérôme Euzenat	general harmonisation
0.9	07.01.2006	Jérôme Euzenat	minor fixes
1.0	30.01.2006	Jérôme Euzenat	added SKOS renderer + quality control corrections
1.1	02.02.2006	Jérôme Euzenat	corrected small mistake in the SEKTMappingRenderer

Executive Summary

Ontology heterogeneity on the semantic web is a problem that prevents applications from interoperating. In order to solve this problem a lot of effort has been devoted to design algorithms for matching ontologies. These algorithms can directly deliver the programs (transformations, mediators, translators, bridging axioms) that help solving these problems.

However, all these programs are generated from a roughly similar basis expressing the relation between ontology entities, that we call an alignment. Because alignments can be used by all these applications as well as other tools for generating or editing them, it would be useful that the applications share a common alignment format.

This deliverable investigates the specification of a format for ontology alignment. It starts by defining what is expected from such a format in the context of the semantic web. Then we review the existing formats that have been developed (or not) for expressing alignments: OWL, SBO, C-OWL, SWRL, Alignment format, SEKT Mapping language and SKOS. We observe that there is a continuum from lightweight formats that do not commit to any ontology language but have limited expressiveness to heavyweight formats that can express complex alignments with the helps of an expressive language.

Committing to some ontology language is a problem if we want the alignment format to be used by many tools. In particular, we would like that this format be used by lightweight applications such as those considering folkosomies or directories. Another argument in favour of these simple formats is that most of the ontology matching tools are not able to deliver complex alignments. However, when alignments are provided by users, the definition of complex alignments is possible and should be preserved by the format.

Therefore, it is necessary to attempt at reconciling these different formats.

In a first approach, it is necessary to ensure some interoperability between these formats. We present some limited attempts to bring interoperability. In particular, we show how the Alignment format can generate most of the other formats (starting, of course, from a lightweight alignment) and how SEKT Mapping language allows to generate more expressive alignments in other formats.

Moreover, the Alignment format, made from the beginning the provision to extension and, in particular, language extensions. We describe in the last chapter, the specification of an embedding of the SEKT Mapping language within the Alignment format as a Level 2 alignment. The resulting format remains language independent but brings the expressiveness of the SEKT Mapping language. This result is further extended by more standard metadata as well as libraries of type comparators.

Contents

1 Introduction: Requirements for an alignment format **2**
- 1.1 Context . . . 2
- 1.2 Motivations . . . 3
- 1.3 Requirements . . . 4
- 1.4 Document outline . . . 4

2 Existing formats **5**
- 2.1 OWL . . . 5
- 2.2 MAFRA Semantic bridging ontology (SBO) . . . 6
- 2.3 Contextualized OWL (C-OWL) . . . 8
- 2.4 SWRL . . . 10
- 2.5 Alignment format . . . 11
- 2.6 SEKT Mapping language (OML) . . . 16
- 2.7 SKOS . . . 20

3 Comparison of existing formats **23**
- 3.1 Criteria . . . 23
- 3.2 Comparison . . . 25
- 3.3 Synthesis . . . 25

4 Interoperability **27**
- 4.1 Generating SEKT-ML/C-OWL/SKOS from Alignment API . . . 27
- 4.2 Generating RDF, OWL and WSML from the SEKT Mapping Language API . . . 31
- 4.3 Conclusions . . . 34

5 Towards a unified format **36**
- 5.1 Mapped entities . . . 36
- 5.2 Library of comparators and operators . . . 38
- 5.3 Extended library of relations . . . 40
- 5.4 Metadata normalization . . . 40
- 5.5 Example . . . 43
- 5.6 Limitations . . . 45

6 Conclusions **46**

Chapter 1

Introduction: Requirements for an alignment format

1.1 Context

Like the web, the semantic web will have to be distributed and heterogeneous. Its main problem is the integration of the resources that compose it. For contributing solving this problem, data will be expressed in the framework of ontologies. However, ontologies themselves can be heterogeneous and some work has to be done to achieve interoperability.

Semantic interoperability can be grounded on ontology reconciliation: finding relationships between concepts belonging to different ontologies. We call this process "ontology matching". The ontology matching problem can be described in one sentence: given two ontologies each describing a set of discrete entities (which can be classes, properties, rules, predicates, etc.), find the relationships (e.g., equivalence or subsumption) holding between these entities. This set of relations is called an alignment.

[Bouquet *et al.*, 2004] provided a definition of the alignment structure so as to be able to use and reuse it in various situations. Given two ontologies O and O', alignments are made of a set of correspondences (called mappings when the relation is oriented) between pairs of (simple or complex) entities $\langle e, e' \rangle$ belonging to O and O' respectively.

A correspondence is described as a quadruple:

$$\langle e, e', R, n \rangle$$

where:

- e and e' are the entities (e.g., formulas, terms, classes, individuals) between which a relation is asserted by the correspondence;
- R is the relation, between e and e', asserted by the correspondence. For instance, this relation can be a simple set-theoretic relation (applied to entities seen as sets or their interpretation seen as sets), a fuzzy relation, a probabilistic distribution over a complete set of relations, a similarity measure, etc.
- n is a degree of confidence in that correspondence (this degree does not refer to the relation R, it is rather a measure of the trust in the fact that the correspondence is appropriate – "I trust 70% the fact that the correspondence is correct/reliable/..." – and can be compared

with the certainty measures provided by meteorological agencies). The trust degree can be computed in many ways, including users' feedback or log analysis.

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

$$URI1 = URI2$$

while a more elaborate one could be:

$$URI1(x, y, z) \Leftarrow_{.85} URI2(x, w) \wedge URI3(z, concat(y, w))$$

The first one express the equivalence (=) of what is denoted by two URIs (with full confidence), while the second one is a Horn-clause expressing that if there exists a w such that $URI2(x, w)$ and $URI3(w, concat(y, z))$ is true in one ontology then $URI1(x, y, z)$ must be true in the other one (and the confidence in this clause is here quantified with a .85 degree).

1.2 Motivations

This definition of an alignment is rather abstract and does not provide a concrete format that can be used for expressing these alignments. "Reifying" alignments in a standardised format can be very useful in various contexts:

- for collecting hand-made or automatically created alignments in libraries that can be used for linking two particular ontologies;
- for modularising matching algorithms, e.g., by first using terminological alignment methods for labels, having this alignment agreed or amended by a user and using it as input for a structural alignment method;
- for comparing the results with each others or with possible "standard" results;
- for generating from the output of different algorithms, various forms of interoperability enablers. For instance, one might generate transformations from one source to another, bridge axioms for merging two ontologies, query wrappers (or mediators) which rewrite queries for reaching a particular source, inference rules for transferring knowledge from one context to another.

The goal of this deliverable is to propose some language to express these reified alignments so that they can be exchanged between applications.

We claim that alignments are more intelligible than transformations: they only express correspondences between ontology entities, not the way they must be used (as a transformation of ontologies, a query transformation or a data translation?). This can be the basis for studying their properties (moreover, these properties can also be inferred from the methods used for generating alignments).

The problem is thus to design an alignment format which is general enough for covering most of the needs (in terms of language and alignment output) and developed enough for offering the above functions.

1.3 Requirements

The requirements that can be put on the format to be defined are the following:

- being Web ready: in particular using URIs and semantic web languages (XML, RDF);
- being language independent: this allows alignments between ontologies written in different languages;
- being simple so that current ontology matching tools can manipulate it without having to implement a full-fledged knowledge representation language;
- being expressive so that it can cover an important part of the usable relations between ontologies;
- supporting many different uses: in particular not being committed to one particular usage (being used as axioms in one language, or being used for some particular transformation);
- supporting as many different kinds of manipulation (trimming, composing, etc.) as possible.

[Euzenat, 2004] proposed an alignment format and an application programming interface (API) for manipulating alignments, illustrated through a first implementation. This first implementation offered a relatively limited expression language but provided many support function. It has been used in the Ontology Alignment Evaluation Initiative campaigns as well as in other tools.

A number of other formats have been proposed for expressing alignments in the context of numerous applications. The purpose of this deliverable is to define an alignment format that can satisfy most of the needs of existing formats.

1.4 Document outline

In the next chapter, the various formats that can be used for expressing alignments in a declarative manner are presented. Chapter 3, then evaluates the similarity and differences between these formats in order to better understand the need of such a format. We demonstrate how some of these formats are able to achieve limited interoperability with others (mostly through export/import functions). In the last chapter we specify a common format which is based on both the Alignment format and SEKT Mapping language and which can be the basis for an expressive and independent alignment format.

Chapter 2

Existing formats

At the beginning of Knowledge web there were very few formats (in fact only the SBO was documented). Independently, many different efforts proposed their own format or at least what could be considered an alignment format. We briefly present here the various formats that have been proposed so far for expressing relations between ontologies on the world wide web. We mostly compare these formats based on their syntax. Each section ends with a conclusion that summarises the benefits and drawbacks of the formats from that standpoint.

A deeper analysis of some of these in terms if semantics and expressiveness are provided in deliverable 2.2.5 [Hitzler *et al.*, 2005].

2.1 OWL

OWL in itself can be considered as a language for expressing correspondences between ontologies. As a matter of fact, the `equivalentClass` and `equivalentProperty` primitives have been introduced for relating elements in ontologies describing the same domain. This use is even documented by the W3C best practices working group [Uschold, 2005]. However, these primitives are only shorthands for other primitives (i.e., `subClassOf`, `subPropertyOf`) that already allow to relate entities. So the following OWL ontology:

```
<owl:Class rdf:about="http://ian.ac.uk#Nappy">
  <owl:equivalentClass rdf:resource="http://jim.us#Diaper"/>
</owl:Class>

<owl:Class rdf:about="http://ian.ac.uk#City">
  <owl:subClass ="http://jim.us#City"/>
</owl:Class>

<owl:Property rdf:about="http://ian.ac.uk#cv">
  <owl:subProperty rdf:resource="http://jim.us#Résumé"/>
</owl:Property>
```

can already be seen as an alignment expressing the equivalence of classes `Nappy` and `Diaper`, the coverage of class `City` in ontology identified by `ian` by `City` in that identified by `jim`, and that of the `cv` property by the `Résumé` property in the same ontologies. Moreover, any ontology, as soon as it involves entities from different ontologies, expresses alignments. For instance:

```
<owl:Class rdf:ID="Woman">
  <owl:equivalentClass>
    <owl:Class>
      <owl:subClassOf rdf:resource="http://www.example.org/ontology2#Person"/>
      <owl:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://www.example.org/ontology2#gender"/>
          <owl:hasValue rdf:resource="http://www.example.org/ontology2#W"/>
        </owl:Restriction>
      </owl:subClassOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

expresses that a `Woman` in ontology `http://www.example.org/ontology1` is equivalent to a `Person` in ontology `http://www.example.org/ontology2` whose `gender` is `W`.

2.1.1 Conclusion

Not surprisingly, the OWL language can be used as an alignment expression language. However, using it this way has some drawbacks:

1. It forces to use one ontology language: OWL. It is still possible to relate this way ontologies that are expressed in other languages (without benefiting from the construction of complex terms). However, the alignment will not benefit from the content of the ontologies themselves.
2. It mixes alignment and definitions. This is a problem for the clarity of alignments as well as for lightweight applications which do not want to interpret the OWL language.
3. It is interpreted only in the framework of a global interpretation of one OWL theory. It is difficult to use this expression for only importing data expressed under one ontology into another one (because this application requires sorting out definitions from correspondences).

Other languages have been set up for overcoming these problems: SKOS solves the first problem (but introduces its own language), SWRL solves problem (2) and C-OWL attempts to solve problem (3). These languages will be presented hereafter.

2.2 MAFRA Semantic bridging ontology (SBO)

MAFRA [da Silva, 2004; Mädche *et al.*, 2002] means "Mapping framework"[1]. It is a whole system for extracting mappings from ontologies and executing them as data transformation from one ontology to another. The system was first designed to work with the DAML+OIL language.

MAFRA does not define a real exchange format for ontology alignment. Rather, it provides an ontology, called Semantic Bridge Ontology. The instantiation of this ontology constitutes an ontology mapping document which is finally such a format. The serialisation of this format has not been described in detail in documents so we freely use our own transcription[2].

[1] It is also the name of a city in Portugal featuring a rich palace.

[2] [Mädche *et al.*, 2002] presents SBO as a DAML+OIL ontology, but it seems to have evolved a lot since then and we have not been able to find a serialisation that could stand as a proper format.

The main concepts in this ontology are SemanticBridges and Services. A SemanticBridge is tied to the Services that are able to implement the bridge (as a data transformation). A Service can be thought of as a function: $f : Arg^n \longrightarrow Arg^m$ which maps tuples of arguments into tuples of arguments. It is identified (one can imagine by a URI). The arguments are typed and can be ontology concepts, property paths, litterals or arrays of such. For instance, the service CopyAttribute (which copies an attribute from an ontology to another) is defined by:

```
pt.ipp.isep.gecad.mafra.services.ttransformations.CopyAttribute: path -> path
pt.ipp.isep.gecad.mafra.services.ttransformations.CountProperties: path -> integer
```

SemanticBridges (which can be ConceptBridges or PropertyBridges) express a relation between two sets of entities by composing elementary services that are applied to them. For instance, a SemanticBridge between two ontologies such as this one defines Individual and the target ontology defines both Man and Woman will be expressed in the following way:

```
ConceptBridge: Indiv2Woman
    x: <o1#Individual>; genre == 'W' -> <o2#Woman>
ConceptBridge: Indiv2Man
    x: <o1#Individual>; genre == 'M' -> <o2#Man>
exclusive: Indiv2Woman, Indiv2Man
```

Entities to be mapped are identified within the ontology (instances) through a path. Paths serve dual purposes of navigating within the ontology structure and providing the context further characterising the concerned entities. In this context paths play exactly the same role as in Xpath. They are further enriched with conditions (in the example above, <o1#Individual>; genre == 'W' is a path with condition that the final step genre has value W (the complete example involves regular expressions on the string value). More complex bridges can then be expressed, as an example given in [da Silva, 2004]:

```
PropertyBridge: spouseIn2noMarriages
    y: <o1#Individual:spouseIn>
->  <o2#Individual:noMarriages> = countProperties( y )
```

It means that the occurence of property spouseIn in ontology o1 will be translated in the count of these properties to be recorded in the property noMarriages in ontology o2. Here <o1#Individual:spouseIn> is a path and countProperties is a service.

An ontology mapping document satisfying SBO is a collection of such bridges plus information on the concerned ontologies as well as constraints (such as the "exclusive" of the first example expressing that only one of the two bridges can be triggered).

2.2.1 Conclusion

SBO provided a framework for expressing alignments. This format is used as output of ontology matchers and input of data transformations.

The format provided by SBO is not very clear since all the language is described in UML. This is a minor problem that could be solved by exposing some RDF/XML format (a previous version of the framework had been described as a DAML ontology [Mädche *et al.*, 2002]). Moreover, this format is a relatively complex language that is tied to the MAFRA architecture.

It does not separate the declarative aspect of relations from the more operational one of service: the relations are described in functions of the services able to implement them. The services can

be arbitrary small (like string concatenation) or large (like implementing a complete alignment by a program). On the one hand, this guarantees that these alignments can be used: SBO-documents can readily be used as data transformations. On the other hand, this does not help using these alignments in other ways (for merging ontologies or mediating queries for instance).

2.3 Contextualized OWL (C-OWL)

C-OWL is an extension of the OWL language to express mappings between heterogeneous ontologies. The constructs in C-OWL are called *bridge rules*, and they allow to express a family of semantic relations between concepts/roles and individuals interpreted in heterogeneous domains. Given two ontologies O_1 and O_2 a bridge rule from O_1 to O_2 expresses a semantic relation between a concept/role/individual of O_1 and a concept/role/individual of O_2. Bridge rules are directional in the sense that bridge rules from O_1 to O_2 are not the inverse of the bridge rules from O_2 to O_1.

Abstract syntax There are five types of bridge rules for concepts, five for roles and five for individuals. We report the abstract syntax with their intuitive in the following table.

Abstract syntax	**Intuitive meaning**
$i : A \xrightarrow{\sqsubseteq} j : B$	the concept A in ontology i is *more specific* that the concept B in the ontology j
$i : A \xrightarrow{\sqsupseteq} j : B$	the concept A in ontology i is *more general* that the concept B in the ontology j
$i : A \xrightarrow{\equiv} j : B$	the concept A in ontology i is *equivalent* to the concept B in the ontology j
$i : A \xrightarrow{\perp} j : B$	the concept A in ontology i is *disjoint* from the concept B in the ontology j
$i : A \xrightarrow{*} j : B$	the concept A in ontology i *overlaps* with the concept B in the ontology j

The above table reports only the bridge rules on concepts, but analogous expressions are possible in C-OWL when A and B are either roles or individuals.

Bridge rules form O_1 to O_2 must be read from the target ontology (O_2) viewpoint. Namely they express how the target ontology (ontology 2) sees or translates the source ontology (ontology 1) in itself. Furthermore, the relations "more general", "less general", etc., are not restricted to be mere set theoretical relations, as the domains of interpretation of the two ontologies may be completely different (a mediating relation between the elements of the domain of interpretation is required).

Concrete syntax C-OWL concrete syntax is given in XML form, and it looks like the one given in the following example of a C-OWL file

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!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 owl   "http://www.w3.org/2002/07/owl#" >
     <!ENTITY cowl  "http://www.itc.it/cowl#" >
```

```
   ]>
<rdf:RDF
  xmlns      ="&cowl;"
  xmlns:cowl ="&cowl;"
  xmlns:owl  ="&owl;"
  xmlns:rdf  ="&rdf;"
  xmlns:rdfs ="&rdfs;"
  xml:base   ="http://www.itc.it/cowl/example#"
  >
<cowl:Mapping>
    <cowl:sourceOntology>
      <owl:Ontology rdf:about="http://www.itc.it/ontologies/source.owl"/>
    </cowl:sourceOntology>
    <cowl:targetOntology>
      <owl:Ontology rdf:about="http://www.itc.it/ontologies/target.owl"/>
    </cowl:targetOntology>
    <cowl:bridgeRule>
      <cowl:Into>
        <cowl:source>
          <owl:Class rdf:about="http://www.itc.it/ontologies/source.owl#Article"/>
        </cowl:source>
        <cowl:target>
          <owl:Class rdf:about="http://www.itc.it/ontologies/target.owl#Publication"/>
        </cowl:target>
      </cowl:Into>
    </cowl:bridgeRule>
    <cowl:bridgeRule>
      <cowl:Onto>
        <cowl:source>
          <owl:Class rdf:about="http://www.itc.it/ontologies/source.owl#Person"/>
        </cowl:source>
        <cowl:target>
          <owl:Class rdf:about="http://www.itc.it/ontologies/target.owl#GraduateStudent"/>
        </cowl:target>
      </cowl:Onto>
    </cowl:bridgeRule>
    <cowl:bridgeRule>
      <cowl:Equivalent>
        <cowl:source>
          <owl:Class rdf:about="http://www.itc.it/ontologies/source.owl#Student"/>
        </cowl:source>
        <cowl:target>
          <owl:Class rdf:about="http://www.itc.it/ontologies/target.owl#Student"/>
        </cowl:target>
      </cowl:Equivalent>
    </cowl:bridgeRule>
    <cowl:bridgeRule>
</cowl:Mapping>
</rdf:RDF>
```

Semantics The full details of the formal semantic for C-OWL bridge rules is provided in [Bouquet *et al.*, 2003]. The semantics of bridge rules from an ontology O_1 to ontology O_2 is given w.r.t. a distributed interpretation. A distributed interpretation for O_1 and O_2 is a 3-tuple $\mathfrak{I} = \langle \mathcal{I}_1, \mathcal{I}_2, r_{12} \rangle$ where $\mathcal{I}_1$ and $\mathcal{I}_2$ are models of O_1 and O_2 respectively, and r_{12}, called the *domain relation* is a subset of $\Delta^{\mathcal{I}_1} \times \Delta^{\mathcal{I}_2}$. The domain relation models a translation function from the domain of interpretation of O_1 to the domain of interpretation of O_2. Intuitively $\langle d, d' \rangle \in r_{12}$ means that d' is one among the possible translation of d into $\Delta^{\mathcal{I}_2}$.

1. $\mathfrak{I} \models 1 : A \xrightarrow{\sqsubseteq} 2 : B$ if $r_{12}(A^{\mathcal{I}_1}) \subseteq B^{\mathcal{I}_2}$;
2. $\mathfrak{I} \models 1 : A \xrightarrow{\sqsupseteq} 2 : B$ if $r_{12}(A^{\mathcal{I}_1}) \supseteq B^{\mathcal{I}_2}$;

3. $\mathfrak{I} \models 1 : A \xrightarrow{\equiv} 2 : B$ if $r_{12}(A^{\mathcal{I}_1}) = B^{\mathcal{I}_2}$;
4. $\mathfrak{I} \models 1 : A \xrightarrow{\bot} 2 : B$ if $r_{12}(A^{\mathcal{I}_1}) \cap B^{\mathcal{I}_2} = \emptyset$;
5. $\mathfrak{I} \models 1 : A \xrightarrow{*} 2 : B$ if $r_{12}(A^{\mathcal{I}_1}) \cap B^{\mathcal{I}_2} \neq \emptyset$;

Decision procedure A tableaux based decision procedure for ontology spaces, i.e., a set of ontologies connected via bridge rules, is described in [Serafini *et al.*, 2005] and has been implemented in a peer-to-peer distributed reasoning system called DRAGO [3] and described in [Serafini and Tamilin, 2005]. The decision procedure supports only bridge rules between concepts, with the exception of the $\xrightarrow{*}$ bridge rules.

2.3.1 Conclusion

The C-OWL proposal can express relatively simple alignments (no constructed classes are expressed, only named classes are used). The more expressive part lays in the relations used by the mapping. These alignment have a clear semantics, however it is given from a particular standpoint: that of the target ontology. C-OWL is based on the OWL language but relatively independent from this language which is confined at expressing the entities (the alignment part being specific).

2.4 SWRL

Some people want to be able to express rules and not only concept definitions. SWRL [Horrocks *et al.*, 2004] (Semantic Web Rule Language) is a rule language for the semantic web. It extends OWL with an explicit notion of rule (from RuleML) that is interpreted as first order Horn-clauses. These rules can be understood as correspondences between ontologies (especially when elements from the head and the body are from different ontologies).

SWRL mixes the vocabulary from RuleML for exchanging rules with the OWL vocabulary for expressing knowledge. It defines a rule (`ruleml:imp`) with a body (`ruleml:body`) and head (`ruleml:head`) parts.

```
<ruleml:imp>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="http://www.example.org/ontology2#Person" />
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="http://www.example.org/ontology2#gender">
      <ruleml:var>p</ruleml:var>
      <owlx:Individual owlx:name="W" />
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom  swrlx:property="http://www.example.org/ontology1#Woman">
      <ruleml:var>p</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>
```

[3] http://trinity.dit.unitn.it/drago

This last rule expresses that a `Person` in ontology `http://www.example.org/ontology2` with `W` as the value of its `gender` attribute is a `Woman` in ontology `http://www.example.org/ontology1`.

The introduction of variables within constructs of the OWL language provides more expressiveness to the language (in particular it allows to express what was called role-value maps in description logics or feature path equations in feature algebras). SWRL also provide a set of built-in predicates on the various datatypes provided by XML Schema as well as operators on collections (like count).

2.4.1 Conclusion

SWRL rules can be used for expressing the correspondences between ontologies. These correspondences are expressed between formulas and interpreted as Horn-clauses. They have the advantage over genuine OWL to be well identified as rules and are easier to manipulate as an alignment format than OWL which is also used to express ontologies.

Like the OWL example, these rules have the drawback of forcing the use of OWL and are interpreted as merging ontologies. Again, the expression of a rule like the one above fixes the use that can be made: the rule will help considering some people of the first ontology as women in the second ontology. However, the rules work as a set of logical rules, not rewrite rules, so this really provides the use for merging, not transforming[4].

2.5 Alignment format

As briefly sketched above and before [Euzenat, 2003], in first approximation, an alignment is a set of pairs of elements from each ontology. However, as already pointed out in [Noy and Musen, 2002], this first definition does not cover all needs and all alignments generated. So [Euzenat, 2004] provided an Alignment format on several levels, which depends on more elaborate alignment definitions.

2.5.1 Alignment

The alignment description can be stated as follows:

a level used for characterising the type of correspondence (see below);

a set of correspondences which express the relation holding between entities of the first ontology and entities of the second ontology. This is considered in the following subsections;

an arity (default 1:1) Usual notations are 1:1, 1:m, n:1 or n:m. We prefer to note if the mapping is injective, surjective and total or partial on both side. We then end up with more alignment arities (noted with, 1 for injective and total, ? for injective, + for total and * for none and each sign concerning one mapping and its converse): ?:?, ?:1, 1:?, 1:1, ?:+, +:?, 1:+, +:1, +:+, ?:*, *:?, 1:*, *:1, +:*, *:+, *:*. These assertions could be provided as input (or constraint) for the alignment algorithm or as a result by the same algorithm.

This format is simpler than most of the alignment representations presented here, but is supposed producible by most matching tools.

To this strict definition can be added much more information (in particular when the format is expressed in RDF) such as:

[4] http://co4.inrialpes.fr/align

- the generating algorithm;
- date of creation;
- is the alignment homogeneous (in language or entity).

2.5.2 Level 0

The very basic definition of a correspondence is the one of a pair of discrete entities in the language (identified by URIs). This first level of alignment has the advantage not to depend on a particular language. Its definition is roughly the following:

entity1 the first aligned entity. It is identified by an URI and corresponds to some discrete entity of the representation language.

entity2 the second aligned entity with the same constraint as entity1.

relation (default "=") the relation holding between the two entities. It is not restricted to the equivalence relation, but can be more sophisticated (e.g., subsumption, incompatibility [Giunchiglia and Shvaiko, 2003], or even some fuzzy relation).

strength (default $\top$) denotes the confidence held in this correspondence. Since many alignment methods compute a strength of the relation between entities, this strength can be provided as a normalised measure. The measure should belong to an ordered set M including a maximum element $\top$ and a minimum element $\bot$. Currently, we restrict this value to be a float value between $0.$ and $1.$. If found useful, this could be generalised into any lattice domain.

id an identifier for the correspondence.

A simple pair can be characterised by the default relation "=" and the default strength $\top$. These default values lead to consider the alignment as a simple set of pairs.

On this level, the aligned entities may be classes, properties or individuals. But they also can be any kind of complex term that is used by the target language. For instance, it can use the concatenation of firstname and lastname considered in [Rahm and Bernstein, 2001] if this is an entity, or it can use a path algebra like in:

```
hasSoftCopy.softCopyURI = hasURL
```

However, in the format described above and for the purpose of storing it in some RDF format, it is required that these entities (here, the paths) are discrete and identifiable by a URI.

A full example of the Level 0 Alignment format in RDF is the following:

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF SYSTEM "align.dtd">

<rdf:RDF xmlns='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#'>
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>**</type>
  <onto1>http://www.example.org/ontology1</onto1>
  <onto2>http://www.example.org/ontology2</onto2>
  <map>
```

```
    <Cell>
      <entity1 rdf:resource='http://www.example.org/ontology1#reviewedarticle'/>
      <entity2 rdf:resource='http://www.example.org/ontology2#article'/>
      <measure rdf:datatype='&xsd;float'>0.6363636363636364</measure>
      <relation>=</relation>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource='http://www.example.org/ontology1#journalarticle'/>
      <entity2 rdf:resource='http://www.example.org/ontology2#journalarticle'/>
      <measure rdf:datatype='&xsd;float'>1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

It describes a many-to-many Level 0 alignment between two bibliographic ontologies. It contains two correspondences that identify `reviewedarticle` to `article` and `journalarticle` to `journalarticle respectively`. These correspondence use the equivalence relation and a confidence measure (.64 in the former case and 1. in the latter).

Level 0 alignments are basic but found everywhere: there are no algorithm that cannot account for such alignments. It is, however, somewhat limited: there are other aspects of alignments that can be added to this first approximation.

2.5.3 Level 1

Level 1 replaces pairs of entities by pairs of sets (or lists) of entities. A level 1 correspondence is thus a slight refinement of level 0, which fills the gap between level 0 and level 2. However, it can be easily parsed and is still language independent.

2.5.4 Level 2 (L)

More general correspondence expressions can be useful. For instance, [Masolo *et al.*, 2003] provides a number of bridges from their ontology of services to the currently existing semantic web service description language in first order logic. These kind of correspondences can be expressed with level 2 alignments.

Level 2 considers sets of expressions of a particular language (L) with variables in these expressions. Correspondences are thus directional and correspond to a clause:

$$\forall \overline{x_f}(f \implies \exists \overline{x_g} g)$$

in which the variables of the left hand side are universally quantified over the whole formula and those of the right hand side (which do not occur in the left hand side) are existentially quantified. This level can express correspondences like:

$$\forall x, z\ grandparent(x, z) \implies \exists y; parent(x, y) \wedge parent(y, z)$$

This kind of rules (or restrictions) is commonly used in logic-based languages or in the database world for defining the views in "global-as-view" of "local-as-view" approaches [Calvanese *et al.*, 2002]. It also resembles the SWRL rule language [Horrocks *et al.*, 2003] when used

with OWL (see §4.1.3 for a simple example of such rules). These rules can also be generalised to any relation and drop the orientation constraint.

Level 2 can be applied to other languages than OWL (SQL, regular expressions, F-Logic, etc.). For instance, the expression can apply to character strings and the alignment can denote concatenation like in:

```
name = firstname+" "+lastname
```

The Alignment format has been given an OWL ontology and a DTD for validating it in RDF/XML. It can be manipulated through the Alignment API which is presented below.

2.5.5 Alignment API

A JAVA API can be used for implementing this format and linking to alignment algorithms and evaluation procedures. It is briefly sketched here.

Classes

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

It is essentially made of three interfaces. We present here, under the term "features", the information that the API implementation must provide. For each feature, there are the usual reader and writer accessors:

`Alignment` The Alignment interface describes a particular alignment. It contains a specification of the alignment and a list of cells. Its features are the following:

- **`xml`** (value: "yes"/"no") indicates if the alignment can be read as an XML file compliant with the DTD;
- **`level`** (values "0", "1", "2*") indicates the level of alignment format used;
- **`type`** (values: "11", "1?", "1+", "1*", "?1", "??", "?+", "?*", "+1", "+?", "++", "+*", "*1", "*?", "?+", "**") the type of alignment;
- **`onto1`** (value: URL) the first ontology to be aligned;
- **`onto2`** (value: URL) the second ontology to be aligned;
- **`map`** (value: `Cell`*) the set of correspondences between entities of the ontologies.

`Cell` The Cell interface describes a particular correspondence between entities. It provides the following features:

- **`rdf:resource`** (value: URI) the URI identifying the current correspondence;
- **`entity1`** (value: URI) the URI of some entity of the first ontology;
- **`entity2`** (value: URI) the URI of some entity of the second ontology;
- **`measure`** (value: float between 0. and 1.) the confidence in the assertion that the relation holds between the first and the second entity (the higher the value, the higher the confidence);
- **`relation`** (value: `Relation`) the relation holding between the first and second entity.

`Relation` The Relation interface does not mandate any particular feature.

To these interfaces, implementing the format, are added a couple of other interfaces:

`AlignmentProcess` The AlignmentProcess interface extends the Alignment interface by providing an `align` method. This interface must be implemented for each alignment algorithm.

`Evaluator` The Evaluator interface describes the comparison of two alignments (the first one could serve as a reference). Its features are the following:

`align1` (value: URI) a first alignment, sometimes the reference alignment;
`align2` (value: URI) a second alignment which will be compared with the first one.

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

Functions

Of course, this API does provide support for manipulating alignments. It offers a number of services for manipulating the API. As in [Bechhofer *et al.*, 2003], these functions are separated in their implementation. The following primitives are available:

parsing/serialising an alignment from a file in RDF/XML (`AlignmentParser.read()`, `Alignment.write()`);
computing the alignment, with input alignment (`Alignment.align(Alignment, Parameters)`);
thresholding an alignment with threshold as argument (`Alignment.cut(double)`);
hardening an alignment by considering that all correspondences whose strength is strictly greater than the argument is converted to $\top$, the others being $\bot$ (`Alignment.harden(double)`);
comparing one alignment with another (`Evaluator.eval(Parameters)`) and serialising them (`Evaluator.write()`);
outputting alignment in a particular format (SWRL, OWL, XSLT, RDF, etc.) (`Alignment.render(visitor)`);

In addition, alignment and evaluation algorithms accept parameters. These are put in a structure that allows storing and retrieving them. The parameter name is a string and its value is any Java object. It is advised to have them serialised as a string because external interface (such as the Procalign command-line interface will provide them as such). The parameters can be the various weights used by some algorithms, some intermediate thresholds or the tolerance of some iterative algorithms.

This Alignment API has been implemented in Java on top of the OWL API. This implementation has been used for various purposes: online alignment at Innsbruck, Evaluation tool in the Ontology Alignment Evaluation Initiative, many extensions of it use it for implementing matching algorithms (oMAP from CNR, OLA form INRIA/University of Montréal) or support it (FOAM from Karlsruhe, CMS from Southampton).

2.5.6 Conclusion

The Alignment format used with the Alignment API allows to express alignments without committing to some language. It is not targeted towards a particular use of the alignments and offer generators for a number of other formats. But, by opposition to the languages presented so far, this genuine format does not offer much expressiveness.

However, one good feature of this format is its openness which allows to introduce new relations and if necessary new type of expressions while keeping the compatibility with poorly expressive languages.

2.6 SEKT Mapping language (OML)

2.6.1 Introduction

The Ontology Management Working Group[5], jointly with the SEKT[6] and DIP[7] European projects, aims at developping a complete ontology management suite for the semantic web. As part of this effort the SEKT project has developed an ontology mapping language [de Bruijn *et al.*, 2004]. This language provides a complete format as a basis to represent ontology mappings. This format serves to express mappings resulting from a matching algorithm or from a graphical mapping tool. It has the advantage of being independant from the ontology language, thus giving a common basis to research on schema matching techniques.

The Ontology Mapping Language is also used to specify semantic web services data mediators in the WSMO (Web Services Modeling Ontology)[8] [Roman *et al.*, 2004]. The mediation between semantic web services is a crucial part in this field. When different services have to communicate together but are described using different ontologies, a data mediator comes into play to align the different vocabularies. The core part of a mediator definition is a mapping between the two ontologies plus some non functional properties. At run-time, the mediator is used to rewrite queries and transform instances, thus allowing communication between the services.

This section presents the language in its human-readable abstract syntax form. An RDF vocabulary is also available [Scharffe, 2005].

2.6.2 Presentation of the mapping language

This language provides a set of constructs to express mappings between classes, attributes, relations and instances of an ontology.

A set of operators associated to each type of entities gives the possibility to combine them. Table 2.2 presents the different operators for each type of entity.

Each operator has a cardinality, an effect and some related semantics. The semantics are related to the logical formalism used to represent the ontologies. For example the semantics of the 'and' operator between two classes is linked to the semantic of 'and' in OWL if the mappings are grounded to OWL.

The conditions under which the mappings are valid are specified by introducing a conditional field in the mapping rules. These conditions may be a class condition or an attribute condition. The class conditions are based on their nested attribute values, occurrences or types while the attribute condition are based on their own values or types. Table 2.3 displays the different conditions the mapping language can express.

To get a clear but expressive language, limited constructs for the most common cases of mappings are defined, allowing the user to define arbitrary logical expressions to represent those which

[5] http://www.omwg.org
[6] http://www.sekt-project.org
[7] http://dip.semanticweb.org
[8] http://www.wsmo.org

Language Construct	Description
ClassMapping	Mapping between two classes
AttributeMapping	Mapping between two attributes
RelationMapping	Mapping between two relations
ClassAttributeMapping	Mapping between a class and an attribute
ClassRelationMapping	Mapping between a class and a relation
ClassInstanceMapping	Mapping between a class and an instance
IndividualMapping	Mapping between two instances

Table 2.1: SEKT-ML mapping types.

Entity	Operator
Class	and, or, not, join
Attribute	and, or, not, inverse, symetric, reflexive, transitive closure, join
Relation	and, or, not, join

Table 2.2: SEKT-ML logical constructions.

Range	Name
Class conditions	attributeValueCondition
	attributeTypeCondition
	attributeOccurenceCondition
Attribute Conditions	valueCondition
	typeCondition

Table 2.3: SEKT-ML attribute comparators

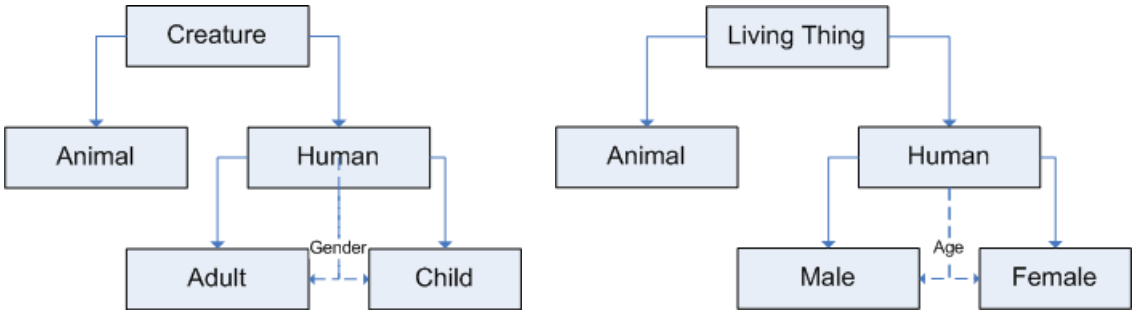

Figure 2.1: Two ontologies.

do not have constructs. These logical expressions must be written according to the two ontology modelling languages.

The syntax of this language has been designed to be intuitive and human readable. This results in a verbose syntax far from the often used XML syntaxes. However XML and RDF syntaxes are available. The language comes with a Java API that provides parsing and serialising methods to and from an object model of the mapping document. Figure 2.1 presents two simple ontologies representing the same domain but with a different modelling perspective. We will give the mapping having the 'Living Thing' ontology as source and the 'Creature' ontology as target.

The top concepts `Living_Thing` and `Creature` presents a terminological mismatch of synonymy. On both ontologies the concepts `human` and `animal` are modelled using the same label. These three cases are simple class to class mappings expressed in the mapping language. Here are the statements representing these mappings.

```
classMapping(
  annotation(<"rdfs:label">  'Creature to LivingThing')
  annotation(<"http://purl.org/dc/elements/1.1/description">
    'Map the person concept to the livingThing concept')
  bidirectional
  <"http://ontologies.omwg.org/creature#Creature">
  <"http://ontologies.omwg.org/livingThing#Living Thing">)

classMapping(
  annotation(<"rdfs:label"> 'Animal to Animal')
  bidirectional
  <"http://ontologies.omwg.org/creature#Animal">
  <"http://ontologies.omwg.org/livingThing#Animal">)

classMapping(
  annotation(<"rdfs:label"> 'human to human')
  bidirectional
  <"http://ontologies.omwg.org/creature#Human">
  <"http://ontologies.omwg.org/livingThing#Human">)
```

The annotation fields allow the input of annotations, for instance, title or description. This field is also used when the mappings are resulting from an algorithm, whereby information like the confidence degree of the mapping and the algorithm used are stated. The RDFS and Dublin Core namespaces are used to indicate the nature of the descriptions. Another field express the

directionality of the mappings. By default a mapping is bidirectional, meaning that the source and target entities are equivalent. It may also be unidirectional, meaning that the target entity somehow subsumes the source one.

The complexities come when mapping the `Male` and `Female` concepts in the `living Thing` ontology to the subconcepts of `Human`, namely `Adult` and `Child` in the `creature` ontology. A `Human Male` or `Female` is an `Adult`/`Child` if his or her `age` is greater than or equal to/lower than 18. This kind of mapping is represented using a condition. The concepts are considered mapped only if the condition specified in the mapping rule is valid. Following is the representation of such a condition for this example.

```
classMapping(
  annotation(<"rdfs:label"> 'conditional female to adult')
  unidirectional
  <"http://ontologies.omwg.org/creature#Female">
  <"http://ontologies.omwg.org/livingThing#Adult">
  attributeValuecondition(
     <"http://ontologies.omwg.org/Creature#age '>=18'))

classMapping(
  annotation(<"rdfs:label"> 'conditional female to child')
  unidirectional
  <"http://ontologies.omwg.org/creature#Female">
  <"http://ontologies.omwg.org/livingThing#Child">
  attributeValuecondition(
       <"http://ontologies.omwg.org/creature#age '<18'))
```

The same rules must then be written for the `male` concept in order to realise a complete mapping. A mapping between the `female`/`male` concepts in the source ontology and the `female`/`male gender` attribute in the target one may also be created. This kind of mapping is saying: "The instances of the female concept in the source ontology are equivalent to the instances having a gender attribute with the value 'female' in the target ontology". Here is the representation in terms of the mapping language.

```
classAttributeMapping(
  annotation(<"rdfs:label"> 'map female to gender:female')
  unidirectional
  <"http://ontologies.omwg.org/creature#Female">
  <"http://ontologies.omwg.org/livingThing#gender:female">)

classAttributeMapping(
  annotation(<"rdfs:label"> 'map the male to the gender:male')
  unidirectional
  <"http://ontologies.omwg.org/creature#Male">
  <"http://ontologies.omwg.org/livingThing#gender:male">)
```

2.6.3 Conclusion

The SEKT Mapping language is an expressive alignment format offering many kinds of relations and entity constructor to the users. One of its main advantages is its ontology language independence, giving a common format for expressing mappings.

So this proposal has a middle man position: it is independent from any particular language but expressive enough for covering a large part of the other languages.

2.7 SKOS

SKOS [Miles and Brickley, 2005b; 2005a] means "Simple Knowledge Organisation System". The SKOS core vocabulary is an RDF Schema aiming at expressing relationships between lightweight ontologies (as known as folkosomies) or thesauri. It is currently under development and thus quite unstable at the time of writing.

The goal of SKOS is to be a layer on top of other formalisms able to express the links between entities in these formalisms.

Concept and relation descriptions

SKOS allows to identify the concepts that are present in the other ontologies. The concept description part of SKOS, seems quite redundant with other languages of that family; it seems that it is designed for being able to take advantage of these concepts (e.g., in a GUI) rather to only express the alignments.

Here are such concept descriptions dedicated to the description of the SKOS and OWL concepts. It defines various ways of presenting the concept (with labels in several languages and alternate labels and symbols). It also provides the opportunity to add various notes and informal definitions to the concept.

```
<skos:Concept rdf:about="http://www.w3c.org#skos">
  <skos:prefLabel>Simple Knowledge Organisation System</skos:prefLabel>
  <skos:altLabel>SKOS</skos:altLabel>
  <skos:altLabel xml:lang="fr">Système simple d'organisation de la
                  connaissance</skos:altLabel>
  <skos:hiddenLabel xml:lang="fr">SSOC</skos:hiddenLabel>
  <skos:definition>The SKOS core vocabulary is a RDFS schema which aims at
                  expressing relationships between lightweight ontologies
                  (as known as folkosomies) or thesauri.</skos:definition>
  <skos:editorialNote>This is not an official definition</skos:editorialNote>
</skos:Concept>

<skos:Concept rdf:about="http://www.w3c.org#owl">
  <skos:prefLabel>Web ontology language</skos:prefLabel>
  <skos:altLabel>OWL</skos:altLabel>
  <skos:altLabel xml:lang="de">EULE</skos:altLabel>
  <skos:prefSymbol rdf:resource="http://www.cs.umd.edu/users/hendler/2003/OWL2.gif" />
  <skos:definition>OWL is an ontology language for the web recommended by W3C.</skos:definition
  <skos:historyNote>OWL is not an acronym</skos:historyNote>
  <skos:hasTopConcept rdf:resource="&owl;#Thing"/>
</skos:Concept>
```

SKOS also alows to describe collections of concepts given by their enumeration:

```
<skos:Collection rdf:about="http://www.example.org#alignmentFormats">
  <rdfs:label>Alignment formats</rdfs:label>
  <skos:member rdf:resource="http://www.w3c.org#skos"/>
  <skos:member rdf:resource="http://www.w3c.org#owl"/>
  <skos:member rdf:resource="http://www.w3c.org#swrl"/>
  <skos:member rdf:resource="http://www.wsmg.org#ml"/>
  <skos:member rdf:resource="http://knowledgeweb.semanticweb.org/heterogeneity#alignapi"/>
  <skos:member rdf:resource="http://www.example.org#mafra"/>
</skos:Collection>
```

property	domain	range	inverse	property
broader	concept	concept	narrower	transitive
related	concept	concept	related	symmetric

Table 2.4: SKOS relation properties.

property	domain	range	inverse
subject	resources	concept	isSubjectOf
primarySubject	resource	concept	isPrimarySubjectOf

Table 2.5: SKOS annotation properties.

Concept relations

SKOS defines so-called "semantic relationships" that express relations between SKOS concepts. For instance, that a term used in a thesauri is broader than another. There are three such relations as defined in Table 2.4.

The relations between concepts enable the assertion of the relative inclusion of concepts as broader or narrower terms as well as another, informal, relation. The following displays the `RDFSchema` concept that is narrower than `ontologyLanguage` but broader than `SKOS`. It is also related to `folkosomyLanguages`.

```
<skos:Concept rdf:about="http://www.w3c.org#rdfschema">
  <skos:prefLabel>RDF Schema</skos:prefLabel>
  <skos:altLabel>RDFS</skos:altLabel>
  <skos:broader rdf:resource="http://www.example.org#ontologyLanguage"/>
  <skos:narrower rdf:resource="http://www.w3c.org#skos"/>
  <skos:related rdf:resource="http://www.example.org#folkosomyLanguage"/>
  <skos:editorialNote>This is not an official definition</skos:editorialNote>
<skos:Concept/>
```

Broader and narrower are transitive properties while related is symmetric.

Annotations

In addition, but of least interest here, SKOS defines annotation properties that enable users to use SKOS concepts for describing resources (annotate them directly with SKOS). It defines the vocabulary displayed in Table 2.5.

This is used below to express that `SKOS` is the `primarySubjectOf` its specification and a `subjectOf` this document.

```
<skos:Concept rdf:about="http://www.w3c.org#skos">
  <skos:isPrimarySubjectOf rdf:resource="http://www.w3.org/TR/2005/swbp-skos-core-spec" />
  <skos:isSubjectOf rdf:resource="http://www.inrialpes.fr/exmo/cooperation/kweb/heterogeneity/d
</skod:Concept>
```

and the SKOS guide [Miles and Brickley, 2005a] could have the following annotations:

```
<foaf:Document>
  <skos:primarySubject rdf:resource="http://www.w3c.org#skos" />
  <skos:subject rdf:resource="http://www.w3c.org#rdfs" />
</foaf:Document>
```

meaning that its `primarySubject` is `SKOS` and a `secondarySubject` is `RDFSchema`.

2.7.1 Conclusion

SKOS has the advantage of being a lightweight vocabulary defining from the ground a rich collection of relations between entities. Since it uses URIs for referring to objects it is fully integrated in the semantic web architecture and is not committed to a particular language. In fact one of the main advantage of SKOS is that it lifts any kind of organised description into an easily usable set of classes. The relation part has the advantage of being very general but the drawback of lacking formal semantics (more semantics on these terms can be brought by using the OWL vocabulary).

One of the main interest of SKOS, would be to define relations and concepts as instantiating the SKOS vocabulary like in:

```
<rdf:Class rdf:about="http://www.w3c.org/owl#Class">
  <rdfs:subClassOf rdf:resource="http://www.w3c.org/skos#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.w3c.org/owl#subClassOf">
  <rdfs:subPropertyOf rdf:resource="http://www.w3c.org/skos#narrower"/>
</rdf:Property>
```

However, this mixes three vocabularies (RDFS, OWL and SKOS). This is one of the main weakness of SKOS. Like other formats which do not separate the ontologies from the correspondences, SKOS, in its most convenient form mixes the highest power of RDF Schema and the expression of the alignments. This form of extensibility (through RDFS) prevents any non RDFS understanding application to fully grasp SKOS content.

Chapter 3

Comparison of existing formats

We have drawn local conclusions concerning each format presented in the previous chapter. We will here compare these formats globally in order to see emerging patterns. For that purpose, we define a set of evaluation criteria (§3.1), we apply them to the different formats (§3.2) and we finally analyse these results (§3.3).

3.1 Criteria

In order to meet the requirements stated in Chapter 1, we explicit them here in a number of criteria to be applied to the existing systems. Several kinds of criterion can be presented with regard to each of the requirements.

3.1.1 Web compatibility

The capacity of the format to be manipulated on the web. This involves, its possible expression in XML, RDF and/or RDF/XML, as well as, the possibility to identify entities by URIs. This should, in principle, enable the extensibility of the format by introducing new properties as well as the referencing of particular correspondences individually.

This aspect is covered by the RDF/XML and URI criteria of Table 3.1.

3.1.2 Languages independence

The ability to express alignments between entities described in different languages. This is often related to the use of URIs. In fact, language independence is mostly related to simplicity.

This aspect is covered by the Language and Model criteria of Table 3.1.

3.1.3 Simplicity

The capacity to be dealt with in a simple form by simple tools. In particular, requiring inference for correctly manipulating the alignment (or requiring that the alignment format covers an important part of some ontology representation language), is not a sign of simplicity. A well structured format will help achieving this goal.

This aspect is covered by the Relations, Terms, Type rest, Cardinality, Variables and Built-in criteria of Table 3.1.

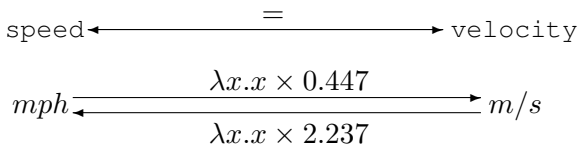

Figure 3.1: Two conceptually equivalent properties, with different associated mappings.

3.1.4 Expressiveness

The capacity of the format to express complex alignments. This means that alignments are not restricted to matching entities identifies by URIs but can assemble them. The constructions for expressing alignments can be arbitrary complex (in fact, it can be more complex than the knowledge expressed in the ontologies).

This aspect is covered by the Relations, Terms, Type rest, Cardinality, Variables and Built-in criteria of Table 3.1.

This expressiveness can be considered from the standpoint of the richness of constructors available for expressing the terms in correspondence and the relations that can be expressed between these terms. In particular, we can distinguish the kind of terms, typing constraints, cardinality constraints, interdependence constraints expressed through variables and paths.

3.1.5 Extendibility

Extendibility is the capacity to extend the format with specific purpose information in such a way that the tools which use this format are not disturbed by the extensions. Most of the system presented here exhibit one kind of extendibility tied to the use of RDF which allows any new relation and object to be added. More particular kind of extendibility are exhibited by SBO and SEKT Mapping language in the plug-in architechture enabling the addition of new transformations, and the Alignment format in which the language itself can be extended.

This aspect is covered by the "+" signs in Table 3.1.

3.1.6 Purpose independence

Purpose independence is rather a consequence of various factors expressed below. We mention the intended use of each of the formats. It is clear for instance that a format designed for data integration, with very precise selection constraints, will rather be difficult to use in transforming ontologies. As an example, consider an ontology pair with a couple of equivalent attributes `speed` and `velocity` as in Table 3.1. A schema-level purpose independent alignment would record that these two properties are equivalent. However, when using this alignment, it may be necessary to use more information, namely that the first one is expressed in miles-per-hour and the second one is expressed in meter-per-second, so equivalent values require conversions.

This aspect is covered by the Target application criterion of Table 3.1.

3.1.7 Executability

Executability is the capacity to be directly usable in mediators. This means there are tools available for directly interpreting the format as a program processing knowledge. Executability is rather

Format	OWL	SBO	C-OWL	SWRL	Alignment	SEKT-ML	SKOS
Target app.	Merging	Data transf.	Data int.	Data int.	Generic	Data transf.	Merging
Language	OWL	UML	OWL	OWL	+		RDFS
Model	OWL		OWL+	OWL			
Execution	Logical	Transf	Logical	Logical		Logical	Alg.
RDF/XML	√		√	√	√	√	√
URI	√		√	√	√	√	√
Measures						√	
Relations	sc/sp		sc/sp	imp	sc/sp+	sc/sp/...	sc/sp
Terms	C/P/I	C/P/I	C/P/I	C/P/I	URI	C/P/I	C/P
Type rest	√	√	√	√		√	
Cardinality	√		√	√			
Variables				√			
Built-in		√+		√	+		

Table 3.1: Features of the presented systems

opposed to language independence.

This aspect is covered by the Execution criterion of Table 3.1.

3.2 Comparison

Table 3.1 provides the value of all the formats presented in the previous section for each of the criterion.

3.3 Synthesis

In fact, the real difference between these formats lies in the continuum between:

- very general languages easy to understand but unable to express the complexity of complex alignments (SKOS, Level 0 Alignment format), and
- very expressive languages which semantics dictates the use and which requires deep understanding of the language (OWL, SWRL, C-OWL, MAFRA).

The SEKT Mapping language stands in the middle of this continuum.

While most of the matching algorithms are only able to express the first kind of alignments, both kinds of languages are useful.

Most of the expressive formats have a surface heterogeneity due to the languages on which they are based (UML, OWL, WSML), however, they have very similar features for refering to ontology constructs (classes, properties), using logical formula constructions (conjunction, implications, quantifiers), as well as datatype and collection buit-in operators. It is even surprising that there is not much heterogeneity in these expressive languages given that complexity is a factor of: the language used for expressing the ontologies, the language used for expressing the related entities, the semantics given to alignments and the language used for expressing relations.

It would thus be useful to avoid that these expressing languages to commit to one kind of language so that their results cannot be used by others. We investigate hereafter, how this can be achieved.

Chapter 4

Interoperability

This chapter considers solutions that can help reaching short term convergence. It should help potential users who hesitate to commit to a particular format to be able to switch as much as possible from one format to another.

To that extent, there are different possible solutions. The first one involves using some transformation language (like XSLT) for translating from one format to another. This requires (and shows the benefit) of having the format in XML syntax.

Another solution consists of benefiting from API to generate the adequate format. This opportunity is mostly provided by language independent formats (the Alignment format and SEKT Mapping language).

We present below a number of these bridges between languages, first from the Alignment API (Section 4.1 and second form the SEKT Mapping language (Section 4.2.

4.1 Generating SEKT-ML/C-OWL/SKOS from Alignment API

The obtained alignment can, of course, be generated in the RDF serialisation form of the Alignment format. However, there are other formats available.

The API provides the notion of a visitor of the alignment cells. These visitors are used in the implementation for rendering the alignments. So far, the implementation is provided with four such visitors:

RDFRendererVisitor displays the alignment in the RDF format described in §2.5. An XSLT stylesheet is available for displaying the alignments in HTML from the RDF/XML format.

OWLAxiomsRendererVisitor generates an ontology merging both aligned ontologies and comprising OWL axioms for expressing the subsumption, equivalence and exclusivity relations.

XSLTRendererVisitor generates an XSLT stylesheet for transforming data expressed in the first ontology in data expressed in the second ontology;

COWLMappingRendererVisitor generates a C-OWL mapping [Bouquet and Serafini, 2003], i.e., a set of relations expressed between elements (in fact classes) of two ontologies.

SWRLRendererVisitor generates a set of SWRL [Horrocks *et al.*, 2003] rules for inferring from data expressed in the first ontology the corresponding data with regard of the second ontology.

SEKTMappingRendererVisitor generates a mapping document as was defined in the SEKT document [de Bruijn *et al.*, 2004].

SKOSRendererVisitor generates a SKOS mapping document.

Some of these methods, like XSLT or SWRL, take the first ontology in the alignment as the source ontology and the second one as the target ontology.

4.1.1 Generating axioms

OWL itself provides tools for expressing axioms corresponding to some relations that we are able to generate such as subsumption (`subClassOf`) or equivalence (`equivalentClass`). From an alignment, the `OWLAxiomsRendererVisitor` visitor generates an ontology that merges the previous ontologies and adds the bridging axioms corresponding to the cells of the alignment.

They can be generated from the following command-line invocation:

```
$ java -jar lib/procalign.jar -i fr.inrialpes.exmo.align.impl.SubsDistNameAlignment
   file://localhost$CWD/rdf/onto1.owl file://localhost$CWD/rdf/onto2.owl -t .6
   -r fr.inrialpes.exmo.align.impl.OWLAxiomsRendererVisitor
```

which returns:

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <owl:Ontology rdf:about="">
    <rdfs:comment>Aligned ontollogies</rdfs:comment>
    <owl:imports rdf:resource="http://www.example.org/ontology1"/>
    <owl:imports rdf:resource="http://www.example.org/ontology2"/>
  </owl:Ontology>

  <owl:Class rdf:about="http://www.example.org/ontology1#reviewedarticle">
    <owl:equivalentClass rdf:resource="http://www.example.org/ontology2#article"/>
  </owl:Class>

  <owl:Class rdf:about="http://www.example.org/ontology1#journalarticle">
    <owl:equivalentClass rdf:resource="http://www.example.org/ontology2#journalarticle"/>
  </owl:Class>

</rdf:RDF>
```

4.1.2 Generating translations

Alignments can be used for translation as well as for merging. Such a transformation can be made on a very syntactic level. The most neutral solution seems to generate translators in XSLT. However, because it lacks deductive capabilities, this solution is only suited for transforming data (i.e., individual descriptions) appearing in a regular form.

The `XSLTRendererVisitor` generates transformations that recursively replace the names of classes and properties in individuals. The renderer produces stylesheets like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
```

```
  <xsl:template match="http://www.example.org/ontology1#reviewedarticle">
    <xsl:element name="http://www.example.org/ontology2#article">
      <xsl:apply-templates select="*|@*|text()"/>
    </xsl:element>
  </xsl:template>

  <xsl:template match="http://www.example.org/ontology1#journalarticle">
    <xsl:element name="http://www.example.org/ontology2#journalarticle">
      <xsl:apply-templates select="*|@*|text()"/>
    </xsl:element>
  </xsl:template>

  <!-- Copying the root -->
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <!-- Copying all elements and attributes -->
  <xsl:template match="*|@*|text()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

4.1.3 Generating SWRL Rules

Finally, this transformation can be implemented as a set of rules which will "interpret" the correspondence. This is more adapted than XSLT stylesheets because, we can assume that a rule engine will work semantically (i.e., it achieves some degree of completeness with regard to the semantics) rather than purely syntactically.

The `SWRLRendererVisitor` transforms the alignment into a set of SWRL rules which have been defined in [Horrocks *et al.*, 2003]. The result on the same example will be the following:

```
<?xml version="1.0" encoding="UTF-8"?>

<swrlx:Ontology swrlx:name="generatedAl"
                xmlns:swrlx="http://www.w3.org/2003/11/swrlx#"
                xmlns:owlx="http://www.w3.org/2003/05/owl-xml"
                xmlns:ruleml="http://www.w3.org/2003/11/ruleml#">
  <owlx:Imports rdf:resource="http://www.example.org/ontology1"/>

  <ruleml:imp>
    <ruleml:_body>
      <swrlx:classAtom>
        <owlx:Class owlx:name="http://www.example.org/ontology1#reviewedarticle"/>
        <ruleml:var>x</ruleml:var>
      </swrlx:classAtom>
    </ruleml:_body>
    <ruleml:_head>
      <swrlx:classAtom>
        <owlx:Class owlx:name="http://www.example.org/ontology2#journalarticle"/>
        <ruleml:var>x</ruleml:var>
```

```
      </swrlx:classAtom>
    </ruleml:_head>
  </ruleml:imp>

...
</swrlx:Ontology>
```

Of course, level 2 alignments would require specific renderers targeted at their particular languages.

4.1.4 Generating C-OWL mappings

The `COWLMappingRendererVisitor` transforms the alignment into a set of C-OWL mapping which have been defined in [Bouquet and Serafini, 2003]. The result on the same example will be the following:

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:cowl="http://www.itc.it/cowl#"
    xml:base="http://www.itc.it/cowl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

  <cowl:Mapping rdf:ID="">
    <cowl:sourceOntology>
      <owl:Ontology rdf:about="http://www.example.org/ontology1"/>
    </cowl:sourceOntology>
    <cowl:targetOntology>
      <owl:Ontology rdf:about="http://www.example.org/ontology2"/>
    </cowl:targetOntology>
    <cowl:bridgeRule>
     <cowl:Equivalent>
       <cowl:source>
         <owl:Class rdf:about="http://www.example.org/ontology1#reviewedarticle"/>
       </cowl:source>
       <cowl:target>
         <owl:Class rdf:about="http://www.example.org/ontology2#journalarticle"/>
       </cowl:target>
     </cowl:Equivalent>
    </cowl:bridgeRule>
  ...
  </cowl:Mapping>
</rdf:RDF>
```

4.1.5 Generating SEKT-ML mappings

The `SEKTMappingRendererVisitor` transforms the alignment into a SEKT mapping document which have been defined in [de Bruijn *et al.*, 2004]. The result on the same example is the following:

```
MappingDocument(<"">
  source(<"http://www.example.org/ontology1">)
  target(<"http://www.example.org/ontology2">)
```

```
  classMapping( <"#s44261">
    bidirectional
    <"http://www.example.org/ontology1#reviewedarticle">
    <"http://www.example.org/ontology2#article">
  )

  classMapping( <"#s4201">
    bidirectional
    <"http://www.example.org/ontology1#journalarticle">
    <"http://www.example.org/ontology2#journalarticle">
  )

)
```

Of course, level 2 alignments would require specific renderers targeted at their particular languages.

4.1.6 Generating SKOS

The `SKOSRendererVisitor` transforms the alignment into a SKOS document which have been defined in [de Bruijn *et al.*, 2004]. The result on the same example is the following:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">

  <skos:Concept rdf:about="http://www.example.org/ontology1#journalarticle">
    <skos:related rdf:resource="http://www.example.org/ontology2#journalarticle"/>
  </skos:Concept>

  <skos:Concept rdf:about="http://www.example.org/ontology1#reviewedarticle">
    <skos:related rdf:resource="http://www.example.org/ontology2#article"/>
  </skos:Concept>

</rdf:RDF>
```

4.2 Generating RDF, OWL and WSML from the SEKT Mapping Language API

The Mapping language presented in Section 2.6 is used via a Java API. It provides classes and methods for parsing mapping documents, manipulating the mappings objects in memory and serialising them in various formats. Apart from the original abstract syntax format the API proposes exporting in RDF, OWL-DL and WSML. An example of mapping exported in these various formats is given in this section.

4.2.1 RDF syntax of the mapping language

We first give the RDF triples corresponding to the mapping document header:

```
_"http://sw.deri.org/~francois/mappings/creature2livingThing"
rdf#type
```

```
map#mappingDocument

_"http://sw.deri.org/~francois/mappings/creature2livingThing"
map#onto1
_"http://sw.deri.org/~francois/ontologies/o1"

_"http://sw.deri.org/~francois/mappings/creature2livingThing"
map#onto2
_"http://sw.deri.org/~francois/ontologies/o2"

_"http://sw.deri.org/~francois/mappings/creature2livingThing"
dc#creator
_"http://sw.deri.org/~francois/foaf.rdf"
```

A simple and a more complex mapping rule examples are following. We use for this example $o1$ and $o2$ as shortened namespaces.

```
_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule1"
rdf:type
map#ClassMapping

_"http://sw.deri.org/~francois/mappings/creature2livingThing"
map#ClassMapping
_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule1"

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule1"
map#directionality
xsd:string^^"bidirectional"

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule1"
map#hasSource
o1#creature

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule1"
map#hasTarget
o2#livingThing

_"http://sw.deri.org/~francois/mappings/creature2livingThing"
map#ClassMapping
_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"
map#directionality
xsd:string^^"unidirectional"

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"
map#hasSource
o1#Female

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"
map#hasTarget
o2#Adult

_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"
map#condition
map#attributeValueCondition
```

```
_"http://sw.deri.org/~francois/mappings/creature2livingThing#rule2"
map#conditionId
xsd:string^^"cond"

xsd:string^^"cond" map#onAttribute o1#age

xsd:string^^"cond" map#conditionOperator map#greaterOrEqual

xsd:string^^"cond" map#conditionValue xsd:integer18
```

4.2.2 Generating OWL

Simple mappings might be expressed in OWL-DL. This gives some restriction since OWL cannot express rules. It is however planned to extend this export using the SWRL extension of OWL.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY base "http://sw.deri.org/~francois/mappings/Creature2livingThing/">
]>
<rdf:RDF
  xmlns:rdf="&rdf;"
  xmlns:owl="&owl;"
  xmlns:xsd="&xsd;"
  xml:base="&base;">
  <owl:Ontology rdf:about="&base;">
    <owl:imports rdf:resource="http://sw.deri.org/~francois/ontologies/creature/"/>
    <owl:imports rdf:resource="http://sw.deri.org/~francois/ontologies/livingthing/"/>
  </owl:Ontology>
  <owl:Class rdf:about="http://ontologies.omwg.org/creature#Creature">
    <rdfs:comment>Map the creature concept to the livingThing concept</rdfs:comment>
    <owl:equivalentClass>
      <owl:Class rdf:about="http://sw.deri.org/~francois/ontologies/livingthing#Living Thing/">
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:about="http://ontologies.omwg.org/livingThing#Adult"/>
    <owl:subClassOf rdf:resource="http://ontologies.omwg.org/creature#Female"/>
  </owl:Class>
</rdf:RDF>
</xml>
```

4.2.3 Generating WSML

WSML (Web Services Modelling Language) is also generated using the export method of a mapping document. It uses a variant of WSML (WSML-DL) which cannot express rules.

```
ontology  _"http://sw.deri.org/~francois/mappings/Creature2livingThing/"
  nfp
  endnfp
  axiom mappingRuleNumber1
```

```
    nfp
rdfs#label hasValue "Creature to LivingThing"
_"http://purl.org/dc/elements/1.1/description" hasValue "Map the creature concept to
             the livingThing concept"
MappingConfidenceMeasure hasValue "1.0"
    endnfp
  definedBy
?x memberOf _"http://ontologies.omwg.org/livingThing#LivingThing" impliedBy
?x memberOf _"http://ontologies.omwg.org/creature#Creature".

  axiom mappingRuleNumber2
    nfp
     rdfs#label hasValue "Creature to LivingThing"
_"http://purl.org/dc/elements/1.1/description" hasValue "Map the creature concept to
             the livingThing concept"
MappingConfidenceMeasure hasValue "1.0"
    endnfp
  definedBy
    ?x memberOf _"http://ontologies.omwg.org/livingThing#LivingThing" implies
    ?x memberOf _"http://ontologies.omwg.org/creature#Creature".

  axiom mappingRuleNumber3
    nfp
rdfs#label hasValue "conditional female to adult"
MappingConfidenceMeasure hasValue "1.0"
    endnfp
  definedBy
    ?x memberOf _"http://ontologies.omwg.org/livingThing#Adult" impliedBy
    ?x memberOf _"http://ontologies.omwg.org/creature#Female".
```

4.3 Conclusions

Figure 4.1 summarises the transformations presented in this chapter. At first sight, it could seem that the alignment format is the ideal exchange format because it allows to generate, directly or indirectly, so many different formats. However, this is not the case because the SEKT Mapping language is more expressive than the alignment format. The alignment format can only generate simple alignments. So the best option is to enhance the expressiveness of the Alignment format so that it more fully covers the expressive formats. To that extent, the best option is certainly embed the SEKT Mapping language within the Alignment format.

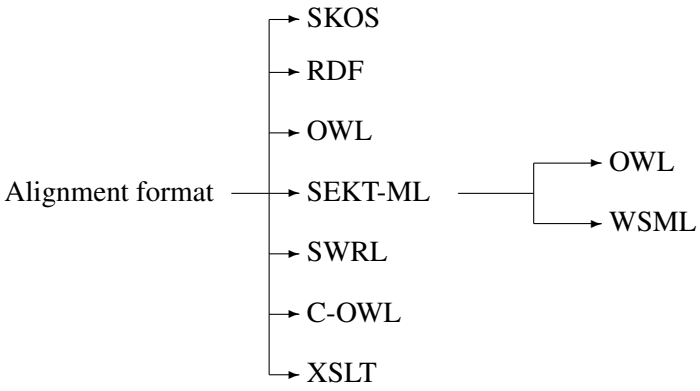

Figure 4.1: Possible export function among alignment formats.

Chapter 5

Towards a unified format

In this chapter we attempt at going deeper into the format integration satisfying our initial requirements. As mentioned above, the Alignment format made provision for embedding expressive languages in order to suit particular needs. So, instead of confining the users to simple match between ontologies, it can be extended to express more complex alignments. The benefit from the standpoint of applications using this format is that they can still use the simple alignment that may exist.

We will specify this embedding here in order to demonstrate the applicability and to provide a format which can support expressive languages while preserving its applicability in less expressive contexts (in particular it can be passed to algorithms that do not read the correspondence content but only manipulate its form, like applying thresholds).

The extension is based on the SEKT Mapping language because it was designed from the beginning to be independent from the target language and it offers a dedicated RDF expression. The format specified here can be considered as a Level 2 format of the Alignment format. We will further identify it as "2OML".

This first section presents the new kind of entities that can be mapped at this more expressive level. This entity construction language is completed by adding comparators and operators on datatypes that appear in entity construction (Section 5.2). Then, the specific kinds of relations that are used in the SEKT Mapping language are introduced (Section 5.3). In Section 5.4, some enhancements to the standard metadata provided by the Alignment format are presented in order to achieve compatibility with the SEKT Mapping language, as well as satisfy the needs expressed during experiments (Section 5.4). Finally, section 5.5 provides an example of the new format.

5.1 Mapped entities

The main change in considering Level 2 extensions of the Alignment format lays in the definition of the entities that are mapped. In the Alignment format, these entities were only identified by their URI. In the SEKT Mapping language specification these entities are typed and compound entities. We describe below the proposed syntax for the mapped entities to be embedded within the alignment format. It is expressed as a grammar because, the compactness of the grammar form helps better understanding it (especially as a format). However, the resulting documents are proper RDF/XML so it would be useful when the language is stable to provide a RDF-Schema (or other formalisms).

This syntax roughly follows that of the SEKT Mapping language, but makes explicit the type of entities that were left implicit in [Scharffe, 2005].

There are basically four kinds of entities: classes, attributes, relations and instances. They can be simply identified by their URIs like in the genuine Alignment format. However, they can as well be composed within the format: this is what provides this format more expressiveness. The SEKT Mapping language features boolean constructors (`And`, `Or` and `Not`) and relation constructors (`Inverse`, `Transitive`, `Symmetric` and `Reflexive` closures as well as `Join`).

Contrary to the SEKT Mapping language, we have introduced the constraints on these objects within the objects themselves because it is where they apply and are useful. In addition we found necessary to add a notion of "path" in the attributes and relations in order to access some of them.

The language, though RDF, is given below under the form of a grammar (the language namespace is http://knowledgeweb.semanticweb.org/heterogeneity/alignment/2oml/):

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

⟨*classconst*⟩ ::= `<and rdf:parseType="Collection">` ⟨*classexpr*⟩ ⟨*classexpr*⟩+ `</and>`
| `<or rdf:parseType="Collection">` ⟨*classexpr*⟩ ⟨*classexpr*⟩+ `</or>`
| `<not>` ⟨*classexpr*⟩ `</not>`

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

⟨*restriction*⟩ ::= `<Restriction>`
`<property>` ⟨*path*⟩ `</property>`
`<comparator rdf:resource="` ⟨*uriref*⟩ `"/>`
`<value>` ⟨*pov*⟩ `</value>`
`</Restriction>`

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

⟨*value*⟩ ::= ⟨*simplevalue*⟩
| `<Apply rdf:resource="` ⟨*uriref*⟩ `" rdf:parseType="Collection">` ⟨*pov*⟩*
`</Apply>`

⟨*path*⟩ ::= `<Attribute rdf:about=` ⟨*uriref*⟩ `/>`
| `<Path rdf:resource="http//URI#empty">`
| `<Path><first><Attribute rdf:about=` ⟨*uriref*⟩ `/></first> <next>` ⟨*path*⟩
`</next></Path>`

⟨*attexpr*⟩ ::= `<Attribute rdf:about=` ⟨*uriref*⟩ `>` ⟨*attcond*⟩* `</Attribute>`
| `<Attribute>` ⟨*attconst*⟩ ⟨*attcond*⟩* ⟨*atttranf*⟩ `</Attribute>`

⟨*attconst*⟩ ::= `<and rdf:parseType="Collection">` ⟨*attexpr*⟩ ⟨*attexpr*⟩+ `</and>`
| `<or rdf:parseType="Collection">` ⟨*attexpr*⟩ ⟨*attexpr*⟩+ `</or>`
| `<not>` ⟨*attexpr*⟩ `</not>`

⟨*attcond*⟩ ::= `<valueCondition>` ⟨*restriction*⟩ `</valueCondition>`
| `<typeCondition>` ⟨*restriction*⟩ `</typeCondition>`

⟨*atttransf*⟩ ::= `<transf rdf:resource="` ⟨*uriref*⟩ `">` ⟨*pov*⟩* `</transf>`
| `<service rdf:resource="` ⟨*uriref*⟩ `id="` ⟨*uriref*⟩ `">` ⟨*pov*⟩* `</service>`

⟨*relexpr*⟩ ::= `<Relation rdf:about=` ⟨*uriref*⟩ `>` ⟨*relcond*⟩* `</Relation>`
| `<Relation>` ⟨*relconst*⟩ ⟨*relcond*⟩* `</Relation>`

⟨*relconst*⟩ ::= `<and rdf:parseType="Collection">` ⟨*relexpr*⟩ ⟨*relexpr*⟩+ `</and>`
| `<or rdf:parseType="Collection">` ⟨*relexpr*⟩ ⟨*relexpr*⟩+ `</or>`
| `<not>` ⟨*relexpr*⟩ `</not>`
| `<inverse>` ⟨*relexpr*⟩ `</inverse>`
| `<symetric>` ⟨*relexpr*⟩ `</symetric>`
| `<transitive>` ⟨*relexpr*⟩ `</transitive>`
| `<reflexive>` ⟨*relexpr*⟩ `</reflexive>`
| `<join>` ⟨*relexpr*⟩ ⟨*relexpr*⟩+ ⟨*joinexpr*⟩ `</join>` // todo

⟨*instexpr*⟩ ::= `<Instance rdf:about"` ⟨*uriref*⟩ `"/>`

The roots of this grammar (*classexpr*, *attexpr*, *relexpr* and *instexpr*) are the entities that will now appear within the Alignment `Cell` elements.

5.2 Library of comparators and operators

Since mapped entities can feature conditions applying to data values, it is necessary to define the available predicates on these values and sometimes available constructors. The SEKT Mapping language specification does not cover this aspect. In the grammar above, these are identified in the `comparator` and the `Apply` constructors by URIs.

For that purpose, we draw on the Web standards and propose the use of the XQuery and XPath functions [Malhotra *et al.*, 2005] relying in XML Schema datatypes just like SWRL [Horrocks *et al.*, 2004] does. Figure 5.1 reproduces the hierarchy of these datatypes (these are limited to a subset of atomic types).

The comparators and constructors are presented in Table 5.1 and 5.2. For most of them, they are taken from XQuery and XPath function. However, their namespace is the same as the one of the language.

Type	Id	Origin	explaination
numeric	add	XQuery	Returns the arithmetic sum of the first argument through the last 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 last argument.

Table 5.1: XML Schema datatypes and constructors (built-in for dates have been omitted).

Type	Id	Origin	explaination
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 second argument over the third argument.
numeric	mod	XQuery	Returns the modulo of the arithmetic quotient of the second argument over the third 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 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.
string	replace	XQuery	Returns its first string argument with every substring matched by the regular expression 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.

Table 5.1: XML Schema datatypes and constructors (built-in for dates have been omitted).

Type	Id	Origin	explaination
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 5.1: XML Schema datatypes and constructors (built-in for dates have been omitted).

One possibility that is left out in this preliminary version is that of introducing new datatypes. For instance, introducing a nucleic acid datatype with efficient operations could be useful in some applications (not in all of course).

5.3 Extended library of relations

The Alignment format requires the implementation of relations that are expressed by the correspondences. Those that are present in the level 0 implementation of the Alignment API are relatively imprecise (they are =, <, >, <> for equivalence, subsumption and disjointness). In order to take into account the variety of relations from the SEKT Mapping language, it is necessary to introduce new relations.

Moreover, the SEKT Mapping language distinguishes the orientation of the relations (by the bidirectional or unidirectional indicator). The Alignment format does not offer the possibility to express this for the reason that the correspondences are considered as relations instead of functions (as the word mapping indicates). Hence, there is no directionality: their goal is the expression of the relation. The directionality indicates the possibility to exploit the relation in one way or another: this is a matter of implementation of the relation (as is the choice to identify a source and a target in the SEKT Mapping language).

To overcome this problem, the directionality information is introduced within the relation for expressing if it is an "equivalence" relation or not.

Table 5.3 provides the various relations introduced in the extension. Obviously it is still possible to introduce more such relations if necessary (e.g., AttributePropertyEquivalence for instance). Each relation has the same semantics as the corresponding one in the Mapping language [Scharffe, 2005].

Some relations, like InstanceClassEquivalence and ClassInstanceEquivalence, may seem redundant. In fact they are here because the first element always belongs to the first identified ontology while the second one belongs to the second ontology.

5.4 Metadata normalization

The Alignment format defines a number of standard metadata which are or can be embedded within the format. It also allows to add non standard annotations just like new RDF relations

Type	Id	Origin	explaination
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 5.2: XML Schema datatypes and comparators.

```
anyAtomicType
 |- xs:anyURI
 |- ordered
 | |- xs:string
 | |- xs:date
 | |- xs:time
 | |- xs:duration
 | |- xs:boolean
 | |- numeric
 |     |- xs:float
 |     |- xs:decimal
 |     |   |- xs:integer
 |     |- xs:double
 |- collection
       |- list
```

Figure 5.1: Hierarchy of simple types and collections

Alignment format Level 2	SEKT-ML Relation	SEKT Direction
ClassEquivalence	ClassMapping	bidirectional
ClassMapping	ClassMapping	unidirectional
AttributeEquivalence	AttributeMapping	bidirectional
AttributeMapping	AttributeMapping	unidirectional
RelationEquivalence	RelationMapping	bidirectional
RelationMapping	RelationMapping	unidirectional
InstanceEquivalence	InstanceMapping	bidirectional
InstanceMapping	InstanceMapping	unidirectional
ClassAttributeEquivalence	ClassAttributeMapping	bidirectional
ClassAttributeMapping	ClassAttributeMapping	unidirectional
ClassRelationEquivalence	ClassRelationMapping	bidirectional
ClassRelationMapping	ClassRelationMapping	unidirectional
ClassInstanceEquivalence	ClassInstanceMapping	bidirectional
ClassInstanceMapping	ClassInstanceMapping	unidirectional
AttributeClassEquivalence	AttributeClassMapping	bidirectional
AttributeClassMapping	AttributeClassMapping	unidirectional
RelationClassEquivalence	RelationClassMapping	bidirectional
RelationClassMapping	RelationClassMapping	unidirectional
InstanceClassEquivalence	InstanceClassMapping	bidirectional
InstanceClassMapping	InstanceClassMapping	unidirectional

Table 5.3: SEKT Mapping language relations adapted for the Alignment format.

Annotation	Type	Content
type	charchar	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 2OML for the language defined before)
method	classname	the algorithm that provided it (or if it has been provided by hand)
measure	xsd:double	the confidence in each correspondence

Table 5.4: Standard annotations for the Alignment format.

Annotation	Type	Content
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 a issuing source
arguments	Arguments	the arguments in favour or against a correspondence [Euzenat *et al.*, 2005]

Table 5.5: New annotations for the Alignment format.

can be added to any RDF nodes (and the current implementation of the Alignment API allows to preserve this).

However, while using the Alignment API we have felt the need to use annotations which purpose is general enough so they could be included in a standard format. Moreover, the kind of annotations put on alignments is also extensible. So far, alignments contain the annotations featured in Table 5.4.

Other valuable information that may be added to the alignment format are presented in Table 5.5.

5.5 Example

Here is an example of the alignment format embedding the SEKT Mapping language. It corresponds to the three kinds of mappings that have been presented in Section 2.6.

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF SYSTEM "align.dtd">

<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/SEKT-ML'
       xmlns:xsd='http://www.w3.org/2001/XMLSchema#'>
<Alignment>
  <xml>yes</xml>
  <dc:creator rdf:resource="http://www.inrialpes.fr/exmo/people/euzenat"/>
  <dc:date>2005/12/12</dc:date>
  <method>manual</method>
  <purpose>example</purpose>
  <level>2oml</level>
  <type>**</type>
  <onto1>http://ontologies.omwg.org/creature</onto1>
  <onto2>http://ontologies.omwg.org/livingThing</onto2>
  <map>
    <Cell>
      <dc:description>Map the person concept to the livingThing concept</dc:description>
      <entity1 rdf:resource='http://ontologies.omwg.org/creature#creature'/>
      <entity2 rdf:resource='http://ontologies.omwg.org/livingThing#livingThing'/>
      <measure rdf:datatype='&xsd;float'>1.</measure>
      <relation>ClassEquivalence</relation>
    </Cell>
    <Cell>
      <rdfs:label>conditional female to child</rdfs:label>
      <entity1>
        <omwg:Class rdf:about="http://ontologies.omwg.org/creature#female">
          <omwg:attributeValueCondition>
            <omwg:Restriction>
              <omwg:property>
                <omwg:Attribute rdf:resource="http://ontologies.omwg.org/creature#age"/>
              </omwg:property>
              <omwg:comparator rdf:resource="http://www.owmg.org/SEKT-ML#SuperiorBoundConditi
              <omwg:value rdf:datatype='&xsd:int'>18</omwg:value>
            </omwg:Restriction>
          </omwg:attributeValueCondition>
         </omwg:Class>
      </entity1>
      <entity2 rdf:resource='http://ontologies.omwg.org/livingThing#child'/>
      <measure rdf:datatype='&xsd;float'>1.</measure>
      <relation>ClassMapping</relation>
    </Cell>
    <Cell>
      <rdfs:label>map female to gender:female</rdfs:label>
      <entity1 rdf:resource='http://ontologies.omwg.org/creature#female'/>
      <entity2>
        <omwg:Class rdf:about="http://ontologies.omwg.org/livingThing#Animal">
          <omwg:attributeValueCondition>
            <omwg:Restriction>
              <omwg:property>
                <omwg:Attribute rdf:resource="http://ontologies.omwg.org/livingThing#gender"/
              </omwg:property>
              <omwg:comparator rdf:resource="&xsd;equal"/>
              <omwg:value rdf:datatype='&xsd:string'>female</omwg:value>
            </omwg:Restriction>
          </omwg:attributeValueCondition>
        </omwg:Class>
      </entity2>
      <measure rdf:datatype='&xsd;float'>1.</measure>
```

```
        <relation>ClassAttributeMapping</relation>
      </Cell>
    </map>
  </Alignment>
</rdf:RDF>
```

5.6 Limitations

We have attempted here at casting most of the SEKT Mapping language within the Alignment format. This latter format has the benefit of being simple when simple things must be expressed. However, it already shows some limits of this simplicity that may be useful to overcome in the future. We briefly consider them here:

Cardinality conditions Cardinality conditions can be expressed in languages like OWL and could easily be added to the set of conditions on classes and properties.

Global conditions it may be useful to express global conditions over the considered entities (i.e. conditions that apply across the two main entities related by the correspondence). These conditions should be expressed within the entities.

Variables In exactly the same vein, if rules like SWRL Rules have to be expressed, it will be necessary to introduce variables that allows to unify terms from both side of the rule. These variables are better expressed at the correspondence level and

It may be necessary to improve this level 2 format in the future if these features become necessary.

Chapter 6

Conclusions

There are now many different languages that can express alignments for different purposes. We have reviewed a number of these languages from the standpoint of their syntax, i.e., rather as an exchange format between applications which need alignments.

There are two main groups of formats: very expressive ones that can be compared with ontology languages and simple formats that do not require heavy inference means for being used. One of these languages (SEKT Mapping language) stands in the middle.

In order to take advantage of the numerous alignment algorithms in a variety of contexts, we have demonstrated how, from some independent formats, it is possible to generate other formats.

But this step is not enough for sharing. So we have specified, in the last chapter, the use of the SEKT Mapping language as an expressive language embedded within the Alignment format. This provides we a format that is powerful enough for being comparable with expressive languages and yet independent from the ontology languages. It remains compatible with the initial goal of the Alignment format: being independent from ontology languages.

The implementation of this proposal is planned for next year. We will do our best for preserving the compatibility of tools that already run with both the SEKT Mapping language and the Alignment format. So this implementation will progress carefully. Simultaneously we consider providing the semantics of this format (with regard to the semantics of the ontology language used), in the same way as it is proposed in deliverable 2.2.5 [Hitzler *et al.*, 2005].

Another topic that will be considered next year by the work package 2.2 is the generation of effective programs such as query mediators or ontology transformations from this alignment format.

Bibliography

[Bechhofer *et al.*, 2003] Sean Bechhofer, Rapahel Voltz, and Phillip Lord. Cooking the semantic web with the OWL API. In *Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island (FL US)*, 2003.

[Bouquet and Serafini, 2003] Paolo Bouquet and Luciano Serafini. On the difference between bridge rules and lifting axioms. Technical Report DIT-03-004, University of Trento (IT), January 2003.

[Bouquet *et al.*, 2003] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-owl – contextualizing ontologies. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science (LNCS)*, pages 164–179, Sanibel Island (FL, USA), October 2003. Springer Verlag.

[Bouquet *et al.*, 2004] Paolo Bouquet, Jérôme Euzenat, Enrico Franconi, Luciano Serafini, Giorgos Stamou, and Sergio Tessaris. Specification of a common framework for characterizing alignment. deliverable D2.2.1, Knowledge web NoE, 2004.

[Calvanese *et al.*, 2002] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A framework for ontology integration. In Isabel Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuinness, editors, *The emerging semantic web*, pages 201–214. IOS Press, Amsterdam (NL), 2002.

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

[de Bruijn *et al.*, 2004] Jos de Bruijn, Douglas Foxvog, and Kerstin Zimmerman. Ontology mediation patterns library. Deliverable D4.3.1, SEKT, 2004.

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

[Euzenat, 2003] Jérôme Euzenat. Towards composing and benchmarking ontology alignments. In *Proc. ISWC-2003 workshop on semantic information integration, Sanibel Island (FL US)*, pages 165–166, 2003.

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

[Giunchiglia and Shvaiko, 2003] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. In *Proc. IJCAI 2003 Workshop on ontologies and distributed systems, Acapulco (MX)*, pages 139–146, 2003.

[Hitzler *et al.*, 2005] Pascal Hitzler, Jérôme Euzenat, Markus Krötzsch, Luciano Serafini, Heiner Stuckenschmidt, Holger Wache, and Antoine Zimmermann. Integrated view and comparison of alignment semantics. deliverable 2.2.5, Knowledge web NoE, 2005.

[Horrocks *et al.*, 2003] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: a semantic web rule language combining OWL and RuleML, 2003. www.daml.org/2003/11/swrl/.

[Horrocks *et al.*, 2004] Ian Horrocks, Peter Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: a semantic web rule language combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL/.

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

[Malhotra *et al.*, 2005] Ashok Malhotra, Jim Melton, and Norman Walsh. XQuery 1.0 and XPath 2.0 functions and operators. Technical report, World Wide Web Consortium (W3C), 2005.

[Masolo *et al.*, 2003] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, and Alessandro Oltramari. Ontology library. Deliverable D18, Wonderweb, 2003.

[Miles and Brickley, 2005a] Alistair Miles and Ban Brickley. Skos core guide. Technical report, World Wide Web Consortium (W3C), http://www.w3.org/TR/2005/swbp-skos-core-guide, 2005.

[Miles and Brickley, 2005b] Alistair Miles and Ban Brickley. Skos core vocabulary. Technical report, World Wide Web Consortium (W3C), http://www.w3.org/TR/2005/swbp-skos-core-spec, 2005.

[Noy and Musen, 2002] Natasha Noy and Mark Musen. Evaluating ontology-mapping tools: requirements and experience. In *Proc. 1st workshop on Evaluation of Ontology Tools (EON2002), EKAW'02*, 2002.

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

[Roman *et al.*, 2004] Dumitru Roman, Holger Lausen, and Uwe Keller. Web service modeling ontology standard (WSMO-standard). Working Draft D2v0.2, WSMO, 2004.

[Scharffe, 2005] François Scharffe. Mapping and merging tool design. deliverable D7.2, Ontology Management Working Group, 2005.

[Serafini and Tamilin, 2005] L. Serafini and A. Tamilin. DRAGO: Distributed reasoning architecture for the semantic web. In A. Gomez-Perez and J. Euzenat, editors, *Proc. of the Second European Semantic Web Conference (ESWC'05)*, volume 3532 of *Lecture Notes in Computer Science*, pages 361–376. Springer-Verlag, May 2005.

[Serafini *et al.*, 2005] Luciano Serafini, Alex Borgida, and Andrei Tamilin. Aspects of distributed and modular ontology reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI-05*, Edinburgh, Scotland, 2005.

[Uschold, 2005] Mike Uschold. Achieving semantic interoperability using RDF and OWL - v4, 2005.

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	D2.3.7	**Negotiation/argumentation techniques among agents complying to different ontologies** defines in particular a service and protocol for exchanging and negotiating alignments that take advantage of the features required to alignment formats.
KW	D2.5.1	**Specification of coordination of rules and ontology languages** described SWRL and C-OWL in detail.
SEKT	D4.4.1	**Ontology Mediation Management V1** defines the Mapping language at the source of the SEKT-ML described here.