An API for Ontology Alignment

Jérôme Euzenat

INRIA Rhône-Alpes, Montbonnot, France,
Jerome.Euzenat@inrialpes.fr

Abstract. Ontologies are seen as the solution to data heterogeneity on the web. However, the available ontologies are themselves source of heterogeneity. This can be overcome by aligning ontologies, or finding the correspondence between their components. These alignments deserve to be treated as objects: they can be referenced on the web as such, be completed by an algorithm that improves a particular alignment, be compared with other alignments and be transformed into a set of axioms or a translation program. We present here a format for expressing alignments in RDF, so that they can be published on the web. Then we propose an implementation of this format as an Alignment API, which can be seen as an extension of the OWL API and shares some design goals with it. We show how this API can be used for effectively aligning ontologies and completing partial alignments, thresholding alignments or generating axioms and transformations.

1 Introduction

Like the web, the semantic web will have to be distributed and heterogeneous. Its main problem will be 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 will have 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 alignment". The ontology alignment 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.

Imagine that one agent wants to query another for bibliographic data but they do not use the same ontology (see Table 1). The receiver of the query can produce some alignment for merging both ontologies and understanding the message or for translating the query in terms of its favorite ontology. Additionally, it could store the alignment for future use or choose to communicate it to its partner for it to translate messages. The ability to align ontologies can be seen as a service provided to the agents. As such it will benefit for some explicit representation of the alignment.

"Reifying" alignment results in a standardized 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;

© Springer-Verlag Berlin Heidelberg 2004

- for modularizing alignments 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 problem is thus to design an alignment format which is general enough for covering most of the needs (in terms of languages and alignment output) and developed enough for offering the above functions. We propose an alignment format and an application programming interface (API) for manipulating alignments, illustrated through a first implementation. Providing an alignment format is not solely tied to triggering alignment algorithms but can help achieving other goals such as:

- allowing a user to select parts of alignments to be used,
- transforming the alignment into some translation programme or articulation axioms,
- thresholding correspondence in an alignment on some criterion,
- improving a partial alignment with a new algorithm,
- comparing alignment results,
- publishing ontology alignments on the web.

The design of this API follows that of the OWL API [1] in separating the various concerns that are involved in the manipulation and implementation of the API.

In the remainder, we shall define the current alignment format (§2), present it as an alignment API (§3) and its implementation on top of the OWL-API (§4). Then we will demonstrate the extension of the system by adding new algorithms (§5), composing alignments (§6), generating various output formats (§7) and comparing the alignments (§8).

2 Alignment Format

As briefly sketched above and before ([4]), in first approximation, an alignment is a set of pairs of elements from each ontology. However, as already pointed out in [9], this first definition does not cover all needs and all alignments generated. So the alignment format is provided on several levels, which depend on more elaborate alignment definitions.

2.1 Alignment

The alignment description can be stated as follows:

a level used for characterizing 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 the alignment representation of [7], but is supposed producible by most alignment tools.

2.2 Level 0

The very basic definition of a correspondence is the one of a pair of discrete entities in the language. 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 operators (e.g., subsumption, incompatibility [5], or even some fuzzy relation).
strength: (default 1.) 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 normalized measure. This measure is by no mean characterizing the relationship (e.g., as a fuzzy relation which should be expressed in the relation attribute), but reflects the confidence of the alignment provider in the relation holding between the entities. 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 "1.". 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 [11] 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.

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.

2.3 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.

2.4 Level 2 (L)

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:

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

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 [2]. It also resembles the SWRL rule language [6] when used with OWL (see §7.3 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
```

This alignment format has been given an OWL ontology and a DTD for validating it in RDF/XML. Given such a format, we will briefly describe how is an API designed around it (§3) before explaining its implementation (§4) and functions (§5-§8).

3 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.

3.1 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:
- **`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;

3.2 Functions

Of course, this API does provide support for manipulating alignments. It offers a number of services for manipulating the API. As in [1], these functions are separated in their implementation. The following primitives are available:

parsing/serializing an alignment from a file in RDF/XML (`AlignmentParser.read()`, `Alignment.write()`);
computing the alignment, with input alignment (`Alignment.align(Alignment, Parameters)`);
thresholding alignment with threshold as argument (`Alignment.cut(double)`);
hardening an alignment by considering that all correspondences are absolute (`Alignment.harden(double)`);
comparing one alignment with another (`Evaluator.eval(Parameters)`) and serialising them (`Evaluator.write()`);

outputting alignment in a particular format (rule, axioms, transformations) (`Alignment.render(stream,visitor)`);

These functions are more precisely described below.

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. The parameters can be the various weights used by some algorithms, some intermediate thresholds or the tolerance of some iterative algorithms.

4 Implementation and Example

For validating the API, we carried out a first implementation, on top of the OWL API [1], which is presented here and illustrated in the following sections.

4.1 Default Implementation

A (default) implementation of this API can be found in the `fr.inrialpes.exmo.-align.impl` package. It implements the API by providing the simple basic classes: `BasicAlignment`, `BasicCell`, `BasicRelation`, `BasicParameters` and `Basic-Evaluator`. These classes provide all the necessary implementation for the API but the algorithm specific methods (`Alignment.align()` and `Evaluator.eval()`). It also provides an RDF/XML parser that can parse the format into an `Alignment` object.

Along with these basic classes the default implementation provides a library of other classes, mentioned below.

4.2 Command-Line Interface

There is a stand-alone program (`fr.inrialpes.exmo.align.util.Procalign`) which:

– Reads two OWL/RDF ontologies;
– Creates an alignment object;
– Computes the alignment between these ontologies;
– Displays the result.

This programme implements the standard structure for using the API.

Additionally, a number of options are available:

– displaying debug information (-d);
– controlling the way of rendering the output (-r);
– deciding the implementation of the alignment method (-i);
– providing an input alignment (-a).

4.3 Processing an Example

Using this alignment processor works in the following way (assuming that `$CWD` corresponds to the current directory):

```
$ java -jar lib/procalign.jar -i fr.inrialpes.exmo.align.impl.SubsDistNameAlignment
    file://localhost$CWD/rdf/onto1.owl file://localhost$CWD/rdf/onto2.owl
    -o aligns/sample.owl
```

This asks for aligning the ontology `onto1.rdf` and `onto2.rdf` with the `SubsDist NameAlignment` class which implements a substring based distance on the labels of classes and properties, and output the result to the `sample.owl` file. Since the results are not very satisfying, the program can be called again by asking to threshold under .6:

```
$ 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
```

which returns:

```
<?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>
    <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>
```

4.4 Example Data

In order to provide a simple example, we picked up, on the web, two bibliographic ontologies described in OWL:

- eBiquity Publication Ontology Resource[1],
- BibTeX Definition in Web Ontology Language[2].

These ontologies are obviously based on BibTeX (see Table 1). This explains the relative ease of alignment with very simple algorithms. The only purpose of this example is to

[1] http://ebiquity.umbc.edu/v2.1/ontology/publication.owl
[2] http://visus.mit.edu/bibtex/0.1/

Table 1. Classes and properties in the three ontologies (From BibTeX, this table avoids technical terms like key and crossref). Standard BibTeX classes have been separated from non-standard ones (some of the non-standard-but-common are mentioned in parenthesis).

BibTeX	UMBC	MIT	BibTeX	UMBC	MIT
	Resource			SoftCopy	
	Publication	Entry			
		Unpublished			Conference
article	Article	Article	book	Book	Book
inbook	InBook	Inbook	incollection	InCollection	Incollection
inproceedings	InProceedings	Inproceedings	mastersthesis	MastersThesis	Mastersthesis
misc	Misc	Misc	phdthesis	PhdThesis	Phdthesis
proceedings	Proceedings	Proceedings	techreport	TechReport	Techreport
			booklet		Booklet
title	title	hasTitle	author	author	hasAuthor
editor	editor	hasEditor	publisher	publisher	hasPublisher
edition	edition	hasEdition	chapter	chapter	hasChapter
series	series	hasSeries	pages	pages	hasPages
volume	volume	hasVolume	number	number	hasNumber
note	note	hasNote	address	address	hasAddress
organization	organization	hasOrganization	journal	journal	hasJournal
booktitle	booktitle	hasBooktitle	shool	school	hasSchool
howpublished		howPublished	institution	institution	hasInstitution
type	type	hasType	year		hasYear
annotation		hasAnnotation			
(affiliation)		hasAffiliation	(abstract)	abstract	hasAbstract
(copyright)		hasCopyright	(content)	hasContent	
(keywords)	keywords	hasKeywords			
(URL)	softCopyURI	hasURL	(ISBN)		hasISBN
(size)	softCopySize	hasSize	(ISSN)		hasISSN
(location)		hasLocation	(LCCN)		hasLCCN
(language)		hasLanguage	(price)		hasPrice
	softCopy			relatedProject	
	softCopyFormat			counter	
	version			description	
	publishedOn				pageChapterData
	firstAuthor				humanCreator

demonstrate the use of the API and its implementation. However, both ontologies display a number of differences, especially in the name of properties.

The expected alignments of both ontologies is the one given in Table 1. It as been described in the alignment format for the purpose of evaluating the results of the various algorithms (see §8).

5 Adding an Algorithm

There are many different methods for computing alignments. However, they always need at least two ontologies as input and provide an alignment as output (or as an intermediate step because some algorithms are more focussed on merging the ontologies for instance). Sometimes they can take an alignment or various other parameters as input.

5.1 Extending the Implemented Algorithms

The API enables the integration of the algorithms based on that minimal interface. It is used by creating an alignment object, providing the two ontologies, calling the `align` method which takes parameters and initial alignment as arguments. The alignment object then bears the result of the alignment procedure.

Adding new alignments methods amounts to create a new `AlignmentProcess` class implementing the interface. Generally, this class can extend the proposed `BasicAlignment` class. The `BasicAlignment` class defines the storage structures for ontologies and alignment specification as well as the methods for dealing with alignment display. All methods can be refined (no one is final). The only method it does not implement is `align` itself.

5.2 Collection of Predefined Algorithms

Several algorithms have been run to provide the alignments between these two ontologies:

NameEqAlignment: Simply compares the equality of class and property names (once downcased) and align those objects with the same name;

EditDistNameAlignment: Uses an editing (or Levenstein) distance between (downcased) entity names. It thus has to build a distance matrix and to choose the alignment from the distance;

SubsDistNameAlignment: Computes a substring distance on the (downcased) entity name;

StrucSubsDistNameAlignment: Computes a substring distance on the (downcased) entity names and uses and aggregates this distance with the symmetric difference of properties in classes.

These simple algorithms should increase the accuracy of the alignment results. We will see what happens in §8.

6 Composing Alignments

One of the claimed advantages of providing a format for alignments is the ability to improve alignments by composing alignment algorithms. This would allow iterative alignment: starting with a first alignment, followed by user feedback, subsequent alignment rectification, and so on. A previous alignment can, indeed, be passed to the `align` method as an argument. The correspondences of this alignment can be incorporated in those of the alignment to be processed.

For instance, it is possible to implement the `StrucSubsDistNameAlignment`, by first providing a simple substring distance on the property names and then applying a structural distance on classes. The new modular implementation of the algorithm yields the same results.

Moreover, modularizing these alignment algorithms offers the opportunity to manipulate the alignment in the middle, for instance, by thresholding. As an example, the algorithm used above can be obtained by:

– aligning the properties;
– thresholding those under `threshold`;
– aligning the classes with regard to this partial alignment;
– generating axioms (see below);

Selecting by other criterion (only retaining the alignments on classes or in some specific area of the ontology) is also possible this way.

7 Generating Output

The obtained alignment can, of course, be generated in some RDF serialisation form like demonstrated by the example of §4.3. 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.

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;

SWRLRendererVisitor generates a set of SWRL [6] rules for inferring from data expressed in the first ontology the corresponding data with regard of the second ontology.

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.

7.1 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>
```

7.2 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.

We have implemented an `XSLTRendererVisitor`, which 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>
```

7.3 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 [6]. 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.

8 Comparing Alignments

Last, but not least, one of the reasons for having a separate alignment format is to be able to compare the alignments provided by various alignment algorithms. They can be compared with each other or against a "correct" alignment. For that purpose, the API proposes the `Evaluator` interface.

8.1 Implementing Comparison

The implementation of the API provides a `BasicEvaluator` which implements a container for the evaluation (it has no `eval` method).

Implementing a particular comparator thus consists in creating a new subclass of `BasicEvaluator` and implementing its `eval` method that will compare two alignments (the first one can be considered as the reference alignment). Currently available subclasses are:

PRecEvaluator, which implements a classical precision/recall/fallout evaluation as well as the derived measures introduced in [3]. Precision is the ratio between true positive and all aligned objects; Recall is the ratio between the true positive and all the correspondences that should have been found.

SymMeanEvaluator, which implements a weighted symmetric difference between the entities that are in one alignment and those common to both alignments (missing alignments count for 0., others weight 1. complemented by the difference between their strengths). This is thus a measure for the similarity of two alignments. The result is here split between the kinds of entity considered (Class/Property/Individual).

8.2 Examples

We provided, by hand, an alignment file that corresponds to the alignment of the classes and properties in Table 1. Each cell, is ranked with strength 1. and relation "=". The result of applying a particular evaluator (here `PRecEvaluator`) is obtained by calling a simple command-line evaluation on the files representing the standard alignment (`bibref.owl`) and the result of the application of a particular alignment method (here `EditDistName.owl`):

```
$ java -cp lib/procalign.jar fr.inrialpes.exmo.align.util.EvalAlign
   -i fr.inrialpes.exmo.align.impl.PRecEvaluator
   file://localhost$CWD/aligns/bibref.owl file://localhost$CWD/aligns/EditDistName.owl
```

The result is:

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:map='http://www.atl.external.lmco.com/projects/ontology/ResultsOntology.n3#'>
  <map:output rdf:about=''>
    <map:precision>0.6976744186046512</map:precision>
    <map:recall>0.9375</map:recall>
   <fallout>0.3023255813953488</fallout>
    <map:fMeasure>0.8000000000000002</map:fMeasure>
    <map:oMeasure>0.53125</map:oMeasure>
    <result>1.34375</result>
  </map:output>
</rdf:RDF>
```

As can be noted, we use the format developed at Lockheed[3] extended with a few attributes, but any other format could have been generated. We have used these comparisons for providing the figures of Table 2.

Table 2. Performance measure for the algorithms presented in §5 with the ontologies presented in §4 (the reference alignment has 32 correspondences).

	eqstring	editdist	substring	substring property	substring property threshold=.4	substring property threshold=.7
Precision	1.	.70	.72	.72	.82	.97
Recall	.31	.94	.97	.97	.97	.94
Weighted	.48	.63	.71	.70	.75	.82

[3] http://www.atl.external.lmco.com/projects/ontology/

9 Conclusion

In order to exchange and evaluate results of alignment algorithms, we have provided an alignment format. A Java API has been proposed for this format and a default implementation described. We have shown how to integrate new alignment algorithms, compose algorithms, generate transformations and axioms and compare alignments.

This API and its implementation could fairly easily be adapted to other representation languages than OWL. In particular, if there were some ontology API, that would constitute a better basis for the implementation than the OWL API. It is unlikely that this format and API will satisfy all needs, but there is no reason why it could not be improved.

To our knowledge, there is no similar API. There are some attempts at defining ontology alignment or ontology mapping either very close to a particular use [5,11] or instead a very abstract definition which is neither immediately implementable nor sharable [7]. This was the analysis of [9], which led to the conclusion that alignment algorithms are not comparable and that their results are different. So, there would be no alignment format. On the contrary, our position is that it is better to define an alignment challenge as general as possible and that we use it, not for identifying the best tool, but for characterising strengths and weaknesses of these algorithms. Moreover, the advantage of having an alignment format that can generate – nearly for free – transformations, axioms and merged ontologies, should be an incentive for algorithm developers to generate that format.

The closest works we know are the RDFT and MAFRA systems. The main goal of RDFT is to represent mappings that can be executed and imported in a transformation process [10]. These mappings correspond to sets of pairs between simple entities (RDFS classes and properties) with a qualification of the relation holding. The ontology is expressed in DAML+OIL. Surprisingly, if the correspondences are generated by Bayes techniques no strength is retained. This format is aimed at using the mapping but no hints are given for adding alignment algorithms or extending the format. MAFRA provides an explicit notion of semantic bridges modelled in a DAML+OIL ontology [8]. The MAFRA Semantic bridges share a lot with the mapping format presented here: they can be produced, serialised, manipulated and communicated through the web. Moreover, the semantic bridges are relatively independent from the mapped languages (though they can map only classes, attributes and relations). They have, however, been built for being used with the MAFRA system, not to be open to external uses, so the classes of the ontology are rather fixed and cannot easily be extended towards new relations or new kinds of mappings. This format is also tailored to the processing architecture used (with non declarative primitives in the transformations).

All the software and the examples presented here are available to everyone [4]. The sources of the API and its implementation are available through an anonymous CVS server. Readers are invited to use it and report any need that is not covered in the current state of the API.

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

Acknowledgements. This work has been partly supported by the European Knowledge Web network of excellence (IST-2004-507482). It had benefited from discussions with Petko Valtchev, Paolo Bouquet and Sergio Tessaris. The author thanks the OWL API developers for their great work.

References

1. 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.
2. 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.
3. Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Proc. GI-Workshop "Web and Databases", Erfurt (DE)*, 2002. http://dol.uni-leipzig.de/pub/2002-28.
4. 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.
5. Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. In *Proc. IJCAI 2003 Workshop on ontologies and distributed systems, Acapulco (MX)*, pages 139–146, 2003.
6. 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/.
7. Jayant Madhavan, Philip Bernstein, Pedro Domingos, and Alon Halevy. Representing and reasoning about mappings between domain models. In *Proc. 18th National Conference on Artificial Intelligence (AAAI 2002), Edmonton (CA)*, pages 122–133, 1998. http://citeseer.nj.nec.com/milo98using.html.
8. Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proc. ECAI workshop on Knowledge Transformation for the Semantic web, Lyon (FR)*, pages 60–68, 2002.
9. 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.
10. Borys Omelayenko. Integrating vocabularies: discovering and representing vocabulary maps. In *Proc. 1st International Semantic Web Conference (ISWC-2002), Chia Laguna (IT)*, pages 206–220, 2002.
11. Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.