2009 Fifth International Conference on Next Generation Web Services Practices

COMP: Comparing Ontology Matching Plug-in

Pavel Tyl
Institute of Computer Science
Academy of Sciences of the Czech Republic
Prague, Czech Republic
Email: pavel.tyl@tul.cz

Jan Loufek
Faculty of Mechatronics
Technical University of Liberec
Liberec, Czech Republic
Email: jan.loufek@tul.cz

Abstract

Improving the ontology integration seems to be an important thing for maintenance of cooperation among various directions of ontology engineering (semantic web services, social networks, etc.). While particular ontologies cover a view at one specific domain of interest, many applications require much more general approach to describe their data. On this account it approaches to a promising solution to the semantic heterogeneity, ontology matching.
Matching can be improved in various ways, such as improving matching strategies, tools and systems, basic techniques and methods or by explaining, representing and further processing and evaluating of matching results. This paper describe a matching plug-in into the open-source ontology editor, Protégé. The plug-in is called COMP and it is a tool for comparing and evaluating various matching techniques and strategies.

1. Introduction

The first part of this paper describes the motivation for ontology matching (and the importance of its improvement) and then matching process itself – theory, definitions and some selected matching tools. The second part describes the plug-in itself – its principles, structure, graphical interface, implemented methods and testing, used Java and external libraries and interfaces. The third part concludes the work and discusses possible future work.

1.1. Motivation

Ontology integration is an important activity for ontology engineering (semantic web services, social networks, etc.). Particular ontologies usually cover just one specific domain, but many applications require data from several domains, in general overlapping. In this case ontolology matching can be done.

Ontology matching can be improved in various ways: by improving matching strategies, tools and systems, basic techniques and methods or by explaining, representing and further processing and evaluating of matching results. Therefore we are developing a matching plug-in into the well-known and widespread open-source ontology editor and knowledge acquisition system, Protégé. The plug-in is called COMP and it is a tool for comparing and evaluating various matching techniques and strategies to help to find the appropriate ones for the concrete ontologies.

1.2. Ontology matching process

Ontology matching is the process of finding relationships or correspondences between entities of different ontologies which have to be semantically compared and, eventually, joined. The output of a matching process is a set of such correspondences between two or more ontologies called *ontology alignment*. The oriented version of an ontology alignment is *ontology mapping*[1].

Let's define the previous more formally:
Definition 1 (Correspondence):
Given two ontologies o and o', a **correspondence** between o and o' is a 5-tuple: $\langle id, e, e', r, n \rangle$ such that:

- *id* is a unique *identifier* of the correspondence,
- e and e' are *entities* of o and o' (e. g., classes),
- r is a *relation* (e. g., equivalence ($=$), inclusion ($\sqsupseteq$), disjointness ($\perp$), etc.),
- μ is a *confidence measure* (typically in the [0,1] range) [4].

Definition 2 (Alignment):
Given two ontologies o and o', an **alignment** (*A*) between o and o':

- is a set of correspondences between o and o',
- with some additional metadata (multiplicity: 1–1, 1–*, method, date, properties, etc.) [4].

Definition 3 (Matching process):
The **matching process** (see Fig. 1) can be described by

1. Mathematical definition of mapping requires relation of equivalence, but ontology mapping can be seen as a collection of mapping rules with same direction (from one ontology to another, *Source* → *Target*).

978-0-7695-3821-1/09 $25.00 © 2009 IEEE
DOI 10.1109/NWeSP.2009.21

IEEE computer society

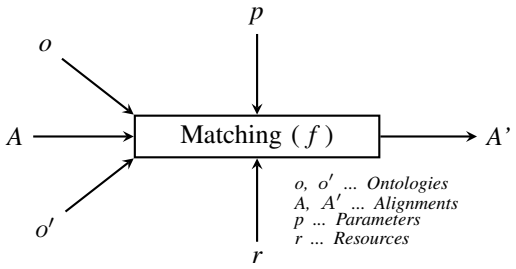

Figure 1. Schema of a matching process.

the function *f*, that from two source ontologies *o* and *o'*, input (preliminary) alignment *A*, a set of parameters (e. g., threshold) and resources (e. g., provenance metadata) returns alignment *A'* between these two ontologies:

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

1.3. Matching tools

Ontology matching is in most cases maintained manually or semi-automatically, mostly with support of some graphical user interface. Manual specification of ontology parts for matching is time consuming and moreover error prone process. Therefore there is a need for the development of faster and/or less complicated methods, which can process ontologies at least semi-automatically.

There exist several ontology editors. Let's mention only editors able to match (and sometimes more – merge, manipulate, etc.) ontologies:

Chimaera [2]
Chimaera is a software system that supports users in creating and maintaining distributed ontologies on the web. Two major functions it supports are merging multiple ontologies together and diagnosing individual or multiple ontologies. It supports users in such tasks as loading knowledge bases in differing formats, reorganizing taxonomies, resolving name conflicts, browsing ontologies, editing terms, etc.

- Browser-based environment for editing, merging and testing large ontologies,
- Matching only a small subset of features, in addition suggests candidates for taxonomy reorganization.

Protégé Prompt [7]
Prompt is an extension plug-in to the Protégé editor [12]. Among other operations with pairs of ontologies (merging, extraction, versioning, ...) Prompt offers also an interface for transformation of one ontology into another one. Therefore it is using matching first.

- Ontology merging and translation support. Matches computed by algorithm are weighted according to interestingness and provided for the user to approve. Also keeps tabs on possible inconsistencies introduced,
- CogZ (Cognitive Support and Visualization for Human-Guided Mapping Systems) tries to improve the user interface of Prompt. It investigates user support during ontology mapping.

2. Plug-in COMP

COMP (Comparing Ontology Matching Plug-in) is a plug-in to Protégé-OWL 4.0 (see Fig. 2). The Protégé-OWL editor is an extension of Protégé that supports the Web Ontology Language (OWL). OWL [9] is the most recent development in standard ontology languages, endorsed by the World Wide Web Consortium (W3C) to promote the Semantic Web vision. An OWL ontology may include descriptions of classes, properties and their instances.

2.1. Principles

COMP should be (among others) a tool for comparing ontology matching. That is to be able to compare various matching algorithms. But its goal is not to be only an evaluation tool, but it should help to find appropriate algorithms, methods or their combinations for different kinds (formats, size, ...) or parts (basic root concepts, leave concepts, instances, properties, etc.) of ontologies, when every ontology has its own feature set.

If n_1 is number of concepts in *o* and n_2 is number of concepts in *o'*, then the goal is to find optimal mapping function M_1 indicating a mapping by first method from each concept in *o* to a corresponding concept(s) in *o'*. Generally, each element from *o* creates correspondence as a one-dimensional array with maximum n_2 elements from *o'*. Consequently, we have to compare it with function M_2 for the second method and its output correspondence array

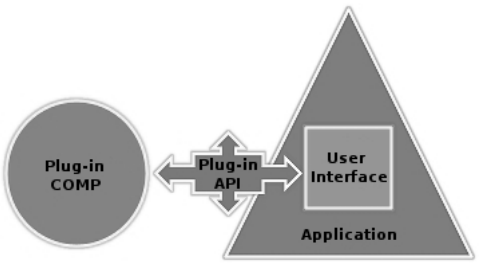

Figure 2. Connection of the plug-in to the application.

with maximum n_2 elements.

In our approach we would like to expect both *internal relationships* and *external relationships*. Therefore, we define the following:

Definition 4 (Internal relationship):
Intensional relationship of a concept *c* is defined as a tuple:

$$IR_c = (N_c, P_c, I_c),$$

which describes the basic features of the concept, where:

- N_c is a name of the concept *c*. Every concept has only one unique name.
- P_c is a set of properties related with the concept.
- I_c is a set of instances associated with the concept.

Definition 5 (External relationship):
Extensional relationship of a concept *c* profiles the structural property of the concept by its relations with other concepts (determines situation of the concept in the whole ontology) and is defined as:

$$ER_c = R_c,$$

where:

- R_c is a set of relations that related the concept with other concepts in the ontology.

From above, COMP should show both results (or subresults) of function M_1 and M_2 using IR_c and ER_c. COMP is designed for transparent depiction of all the relationships represented by splines between concepts (with their name, properties and appropriate instances), that we can select by mouseclick. All the results can be saved for later reuse.

2.2. Package structure

The package is structured into three parts. The first one is the root directory with classes providing *communication with Protégé* itself or creating an output alignment format. The second part is a directory containing classes providing *graphical representation* of the plug-in. The last part is a directory with classes containing methods for *finding correspondences* between elements in input ontologies.

2.2.1. Main class MatchOntology.java. The class *MatchOntology* is the main class of the plug-in. This class is inherited from the class *AbstractOWLViewComponent*, which do not reflect the changes of selected class. The class implements only methods *initialiseOWLView* and *disposeOWLView*. The class provides connection between the plug-in and the Protégé system, creates graphical environment (during initialization), sets a list of used ontologies ("recent files") and provides saving newly opened ontologies.

2.2.2. Output class Output.java. The class *Output* provides the output connection derived from realized test of two ontologies. In this plug-in we use C-OWL alignment [1] as an output format. C-OWL is a language whose syntax and semantics have been obtained by extending the OWL syntax and semantics to allow for the representation of contextual ontologies. It express explicit mappings between heterogeneous ontologies (rather their classes or individuals), when their contents are kept local and therefore not shared with other ontologies. This format implements so called bridging rules between ontologies, that identify found relationship between elements of source ontologies *o* and *o'*. The C-OWL format implements five bridging rules (later saved by the module in the C-OWL format):

- more general,
- more specific,
- equivalent,
- disjoint,
- overlapping.

2.2.3. Classes for ontology matching. In the part *tests* of the package are saved such classes, that implement testing of particular elements of the input ontologies. For simplification we created a class *DefaultTest* containing methods and initialized parameters, that every test on these ontologies must contain. These are especially methods for communication with graphical interface of the plug-in.

Creating of a new test is not difficult and it has two steps:
1) creating the class for testing,
2) initiate the test class into graphical interface.

All the costs needed to matching take place in the class constructor. When the selected method finds a match, it is saved into arrays *set1* and *set2*. The text representation of the relationships type for C-OWL are saved into an array *type* and as the last it is necessary to assign to an array *odds* probability of the match. Then we have to implement the method *returnString* providing only the text protocol about used methods and parameters. Implementation of new testing methods can be realized by adding parameters into *Choice* class in the root part of the package.

2.3. Graphical Interface

Graphical user interface is divided into four panes, such as many other plug-ins for Protégé. Therefore the plug-in can be easily increased for additional functions. The first pane (*Main*) is important for loading an ontology, that will be matched with an ontology loaded in an active Protégé interface. There is a list of recently used files and some settings (i. e., directories, conditionals, ...) too.

The second pane (*Overview*) is a table with general information (names, URIs, count of classes, ...) about the

ontologies we would like to match. The third pane is a text list of all classes. Their structure is signed by row offset. The fourth and the most important pane (*Matching*) contains the selection menu with methods applicable for ontologies, a menu for determining the result scheme, a button for saving the output C-OWL alignment. There is a component for result visualisation under these control elements. This component is put together from two JTree components. In this method there are functions for depiction of connection splines between ontology elements (see Fig. 5). This component is inspired by matching tool COMA++ [3] (compare Fig. 4 and Fig. 5).

2.3.1. Working with classes in graphical representation of the module. During creation of graphical representation of output from this module was necessary to rebuild the component for visualisation of classes in both ontologies. With unique name assumption in one chosen ontology we fill given trees by object hierarchy and then we can work with these object on the same level. Three lists of corresponding elements are returned after testing of similarity in ontologies (same elements for each ontology and a value of similarity for each relationship). These lists are inserted (together with selected colour) into the component. We use object CubicCurve2D for drawing the splines. In the middle of the spline we show the percent value of similarity. After selecting some class, we show only such relationships, that relate with this class and its subclasses. We can also pack or unpack some parts of ontology.

2.4. Implemented methods and testing

Within the frame of the plug-in we implemented several basic methods comparing foremost text strings in term of word similarity of element names in ontology. From this reason we conduct the experiments on OAEI [8] benchmark test suite related with our implemented methods.

Test library consist of reference ontology for making different tests according to various criterion. Reference ontology is coming out attitudinal point of view, how

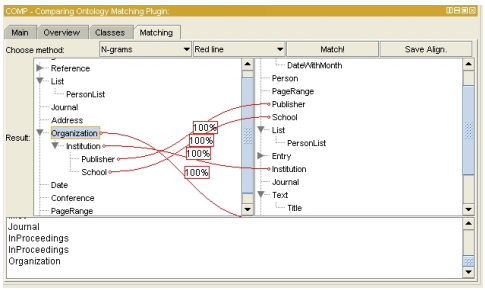

Figure 3. The fourth pane of the plug-in – *Matching*.

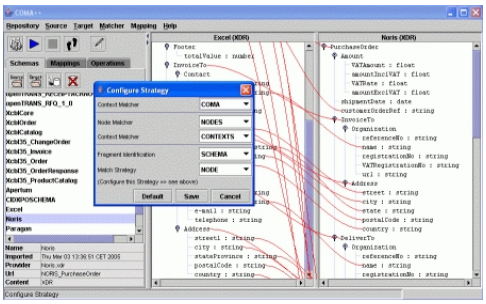

Figure 4. COMA++ GUI – configuration of matchers.

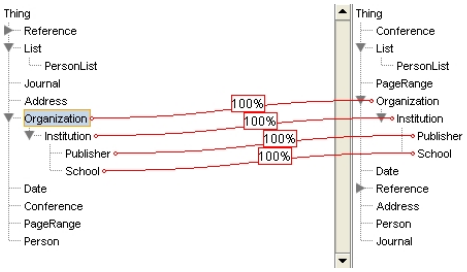

Figure 5. Mapping of correspondencies for the selected class and its subclasses.

bibliographic ontology should look. Of course, it exist a lot of different classification of publications (i. e., based on content and quality). Concrete reference ontology in test suite is based on categories of publications. Reference ontology is compared with various modifications of this ontology.

Systematic experimental testing suite is built on one reference ontology and its variants. These ontologies are described in OWL-DL and serialized into RDF/XML format. Reference ontology contains 33 named classes, 24 object properties, 40 datatype properties, 56 named instances and 20 anonymous instances. Variants are aimed especially at characterization of behaviour of different tools, not at real life situations. Variants have three groups:

1) **Easy tests** – here we compare reference ontology with itself, other irrelevant ontology (i. e., wine ontology) or the same ontology with different constraints (OWL-Lite).
2) **Systematic tests** – if we remove some elements from reference ontology, we can test algorithms in situations, when certain part is missing. Ontologies in this section are ready for testing in six categories:
   - *Names* – names of objects can be substitute by random words, synonyms, other rules or words in different language than English.
   - *Comments* – comments can be removed or translated to different language.
   - *Hierarchies* – can be removed, extended or lim-

ited.
- *Individuals* – can be removed.
- *Properties* – can be removed or limited over deleted classes.
- *Classes* – can be extended by several classes or limited.

3) **real-life tests** – four real-life ontologies containing bibliographic links found on the Web and let unchanged (excepting added attributes *xmlns* and *xml:base*)

These test suite should be lasting measure and therefore can be used in many applications. Figure 6 shows comparing reference ontology with ontology containing among others random class names and we can see the comparison of two different results for two methods – in this case concretely 3-Gram similarity (blue line) and Levenshtein distance (red line).

2.5. Used Java libraries

For graphical representation of plug-in we used *Swing* component package. Swing is a library of user elements on Java platform for controlling computer over graphical interface. Swing provides application interface for creating and maintenance of classic graphical user interface. We can make windows, dialogues, buttons, frames, combo boxes, etc.

For maintenance of actions called by user input we used *AWT* library, basic library of graphical interface in Java. Swing uses methods of classes in AWT library, that contains components and constructs to its maintenance natively included in Java. For simplification of user actions (opening ontologies, setting directories, ...) we use class *Properties* for representation of persistent set of properties. These can be loaded via data flow (i. e., file) and particular keys of the list are represented as datatype string. These information are saved into separate configuration ini file in a directory for Protégé configuration.

2.6. OWL-API – Ontology interface

OWL-API [10] is a Java interface and implementation for ontological language OWL specified by W3C. It is developed at University of Manchester, but it has important contributors from various groups and companies. This interface is a headstone of the Protégé system itself. The last version of this API is focused on OWL-2, that includes OWL-Lite, OWL-DL and some elements of OWL-Full. OWL-API is an open-source interface under LGPL license and it contains following components:

- Application interface for implementation functional OWL-2 relationships in memory.
- RDF/XML parser and writer.
- OWL/XML parser and writer.
- OWL parser and writer of functional syntax
- Support for integration with reasoners (i. e., Pellet [11], Fact++ [5], ...).
- Support for debugging and tuning.

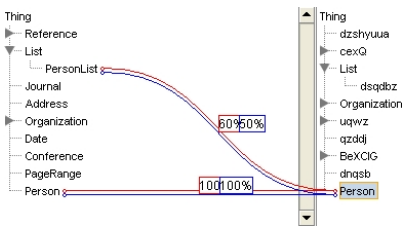

Figure 6. Comparing of basic matching methods.

The core of OWL-API provides data structures representing OWL ontologies, it contains classes for maintenance (creation, manipulation, syntax analysis, expressing and reasoning) objects in ontologies and roughly corresponds to abstract OWL syntax.

Class OWLOntology – It provides explicit contexts for classes and properties.

Class OWLClass – It serves for representation of ontological class. The class itself is quite "light" object (it contains no information about definitions to be used with it). Axioms related to the class contains object *OWLOntology*, because of relationships within the frame of whole ontology.

OWLProperty – OWL distinguishes two type of properties, first properties of object (related to instances) and the second data properties (related to concrete values of instances). These classes are strictly differentiated (*OWLObjectProperty* and *OWLDataProperty*).

Maintenance of ontologies – Structures of data model provide representation of basic elements. Maintenance and creation of ontologies is controled using class *OWLOntologyManager*[2]. Manager is responsible for monitoring ontology, working with concrete ontology formats and changes in ontology.

URI Mapping – Class *URIMapper* gives support for separation of logical and physical URIs (separation for off-line work, temporary saving of ontologies, solving problems with identification name/placing during import).

Factory and Storer – Classes *OWLFactory* and *OWLStorer* serve for loading or saving ontologies. It represents further level of abstraction over parsers and interpreters. It allow us to get ontologies from various data sources (not only files and streams, but also databases, repositories, ...).

2. It supersedes *OWLConnection* and *OWLManager* from original implementation.

Ontology formats – Class *OWLOntologyFormat* represents a format for concrete data input (OWL, RDF/XML). Format can contain information about exact serialization (e. g., definitions of namespaces, sorting, structure, ...). After parsing the manager hold the information about original ontology format.

2.7. Used external libraries

For easy extending of plug-in functionality we used some specialized libraries, that simplify an access to testing ontologies. All the packages are mapped in *manifest file*.

SecondString [13] – This open-source Java-based package contains algorithms (JaroWinkler, Levenshtein, N-Gram, etc.) for comparing text strings from different views. This package is mainly for scientists in information integration or language processing area. It contains also tools for systematic evaluation of performance on test data sets. It is not ideal for large data files.

Java Wordnet Interface (JWI) [6] – It is a package developed by MIT[3]. It is easy to use and extensible library providing an interface to WordNet [14] version 1.6–3.0. It is free to use for non-commercial purposes.

3. Conclusion and future work

Our plug-in for ontology matching was thoughtfully proposed with the view of logical separation of objects creating this plug-in. It was very important to elaborate interface of particular objects because of easier implementation of further (especially testing) classes. Plug-in development is in progress recently and no doubt it needs improvement. But it seems to be the first matching attempt for the latest "pure OWL" version of powerful system Protégé.

First of all, there is a need of increasing the plug-in by wider offer of different methods (and their settings customization), test them deeply and find the best possible combinations for different types of ontologies. First of all we are going to use more advantages and features of Wordnet [14] and its API.

We would like to take into the matching process all of the ontology elements, not only classes, but also properties, instances (there is a great progress in this less investigated area now) and other metainformation in ontologies (provenance, namespaces, versions, ...), and look for the relationships implied. Next step could be for example experiments with more than two ontologies, extending plug-in of different output formats (SWRL, WSML, Alignment format, ...) and many more.

3. Massachusetts Institute of Technology.

Acknowledgment

This project is partly realized under the state subsidy of the Czech Republic within the research and development project "Advanced Remediation Technologies and Processes Center" 1M0554 – Programme of Research Centers PP2-DP01 supported by Ministry of Education and under the financial support of the ESF and the state budget of the Czech Republic within the research project "Intelligent Multimedia E-Learning Portal", registration No. CZ.1.07/2.2.00/07.0008 – ESF OP EC.

References

[1] Bouquet, P. – Giunchiglia, F., Harmelen, F. v. – Serafini, L. – Stuckenschmidt, H. C-OWL: Contextualizing Ontologies. In *Proc. 2^{nd} Internanational Semantic Web Conference (ISWC)*, p. 164–179, Sanibel Island (FL US), 2003.

[2] Chimaera – Software system for creating and maintaining distributed ontologies on the web [online]. http://www-ksl.stanford.edu/software/chimaera.

[3] COMA++ – A System for Flexible Combination of Matching Algorithms [online]. http://dbs.uni-leipzig.de/en/Research/coma.html.

[4] Euzenat, J. – Shvaiko, P. Ontology Matching. Springer-Verlag, Berlin/Heidelberg, 2007. ISBN 978-3-540-49611-3.

[5] FACT++ – OWL-DL Reasoner [online]. http://owl.man.ac.uk/factplusplus.

[6] JWI – MIT Java Wordnet Interface [online]. http://projects.csail.mit.edu/jwi.

[7] Noy, F. N. – Musen, M. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. 17^{th} National Conference of Artificial Intelligence (AAAI)*, p. 450–455, Austin (TX US), 2000.

[8] OAEI – Ontology Alignment Evaluation Initiative [online]. http://oaei.inrialpes.fr.

[9] OWL – Web Ontology Language / W3C Semantic Web Activity [online]. http://www.w3.org/2004/OWL.

[10] OWL-API – OWL Application Interface [online]. http://owlapi.sourceforge.net.

[11] Pellet – OWL-DL Reasoner [online]. http://pellet.owldl.com.

[12] Protégé – Ontology Editor and Knowledge Acquisition System [online]. http://protege.stanford.edu.

[13] SecondString – Approximate String-matching Techniques [online]. http://secondstring.sourceforge.net.

[14] WordNet – Lexical database [online]. http://wordnet.princeton.edu.