UFOme: A User Friendly Ontology Mapping Environment

Giuseppe Pirrò[1], Domenico Talia[1]

[1]D.E.I.S, University of Calabria
87036 Rende, Italy
{gpirro,talia}@deis.unical.it

Abstract. Recently the Ontology Mapping Problem (OMP) has been identified as a key factor towards the success of the Semantic Web and related applications. This problem arises since it is possible for different people to give, through ontologies, different *conceptualizations* of the same (or overlapping) knowledge domain. In order to tackle the OMP several algorithms have been designed. They aim at discovering correspondences (*aka* mappings) between ontology entities. However, these algorithms mostly suffer from the following shortcomings: (i) do not allow to quickly combine and/or compare different mapping strategies; (ii) do not offer support for evaluating mapping strategies in terms of quality of results and performance. In this paper we present a plugin-based system called *UFOme* along with its current implementation. We illustrate how it can be exploited to graphically design mapping tasks by connecting different types of *modules*. *UFOme* provides three categories of modules. The first one (i.e., *visualization*) allows to explore the ontologies to be mapped. The second one (i.e., *matching*) provides different types of individual matchers, exploited to discover mappings between ontologies, and a module for combining them. The third one (i.e., *evaluation*) enables to evaluate each module of the mapping task, a sub mapping task, or the mapping task in the whole w.r.t performance and quality of results.

Keywords: ontology mapping environment, ontology mapping evaluation

1 Introduction

A central factor towards the success of the Semantic Web (SW) and related applications are ontologies. Ontologies can be exploited to give *conceptualizations* of knowledge domains and to make *explicit* and *machine understandable* the meaning of the adopted terminology. The SW aims at exploiting ontologies for providing resources with semantically meaningful information. However, in distributed environments (e.g., the Web), it is not feasible having a single (and universally accepted) ontology describing a knowledge domain. There will be different ontologies each of which created w.r.t "the point of view" of its designer. That's because people see the world differently and these viewpoints inevitably get encoded into data structures. Therefore, in order to enable reciprocal understanding, such different representations (i.e., ontologies) have to be brought into "mutual agreement". This problem in literature is referred to as the ontology mapping problem (OMP). In order to overcome the OMP, several ontology mapping algorithms, aimed at discovering

correspondences (*aka* mappings) between ontology entities (e.g., classes, properties), have been proposed [1,4,11,14,15,17,18]. However, as also pointed out in [7], these algorithms are often not endowed with adequate *cognitive supports* for helping users in the various steps of a mapping task. Often they do not allow to quickly design, combine and compare different mapping strategies and do not offer support for evaluating mapping strategies in terms of quality of results and performance. Since, as pointed out by several evaluation initiatives [16], ontology mapping is not yet a fully-automated task, it is necessary to enable users to interact with the mapping system in the different phases of a mapping task, as for example: to suggest initial mapping candidates as in [15], to accept/reject mapping candidates and to evaluate results. In particular, we identified three main phases in a mapping task execution:

1. *Designing*: a user design the mapping task by choosing the different *modules*, some of which can require configuration parameters (e.g., threshold), to be included in the task. In this phase s/he can also suggest initial mapping candidates.
2. *Running*: the mapping task is executed according to the strategy defined in the *Designing* phase and values of parameters.
3. *Evaluation*: results of the mapping task are presented to the user which can validate them, perform several types of evaluation and possibly restart the running for discovering additional mapping candidates.

We argue that towards a comprehensive tool for ontology mapping, adequate supports (e.g., GUIs) in all the abovementioned phases have to be provided.

In order to cope with these requirements, we delevoped the *UFOme* (*User Friendly Ontology mapping environment*) system based on the concept of pluggable *module*. *UFOme* provides three different categories of modules each of which supports the user in one or more phases of the mapping task. The first category (i.e., *visualization*) includes a module that enables exploring the ontologies to be mapped. The second one (i.e., *matching*) provides different types of individual matchers, exploited to discover mappings between ontologies, and a module for combining individual matchers. The third one (i.e., *evaluation*) allows to evaluate each module of the mapping task, a sub mapping task, or the mapping task in the whole w.r.t performance and quality of results. *UFOme* also allows to implement both new modules and categories to be included into the system as plugins. Therefore, it paves the way towards a user-friendly, effective and extensible ontology mapping environment.

The remainder of this paper is organized as follows. Section 2 describes the *UFOme* architecture. Section 3 presents a working example. Section 4 reviews related work and compares *UFOme* with similar systems and in particular with the OLA [5] system. Section 5 concludes the paper.

2 The *UFOme* architecture

This section describes the *UFOme* architecture designed taking into account two important requirements: *extensibility* and *usability*. The first requirement is fulfilled by the concept of *module*. A *module* is a generic pluggable component designed to support one or more phases of a mapping task. The second requirement is fulfilled by exploiting several GUIs covering specific aspects of the mapping process.

2.1 The *UFOme* two-layer architecture

The *UFOme* architecture, depicted in Fig. 1, is built upon two layers: *logical layer* and *graphical layer*. The latter represents the layer of interaction with the user through the *mapping task composer*. A user can choose the set of modules to be included in the mapping task and connect them according to the mapping strategy s/he wants to implement. In this phase (i.e., *designing*) both incoming and outcoming module connections are checked in order to verify that modules receive the correct data to process (e.g., a matching module should receive two ontologies whereas an evaluation module a set of mappings). If the mapping task has been correctly composed then it can be executed.

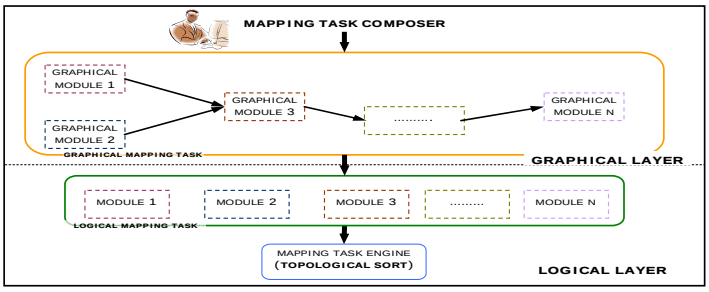

Fig. 1. The *UFOme* architecture.

A mapping task composed at the *graphical layer*, in order to be executed (*execution* phase), is converted into a mapping task at *logical layer*. Here the different modules composing the task process data they received as input. Results are both passed on to the connected modules and stored within the module for possible subsequent analysis (*UFOme* allows to individually evaluate each module). In order to guarantee the correct execution order, *UFOme* relies on the *topological sort* algorithm [2]. The topological sort of a mapping task, which can be viewed as a Directed Acyclic Graph, is a linear ordering of its modules. In particular, each node is executed before all nodes to which it has connections.

2.2 *UFOme* modules

UFOme modules are the building blocks of the system. A module can be represented by the architecture depicted in Fig. 2. It has a set of incoming connections that represent the input, and a set of outcoming connections exploited to collect results. A module also includes a set of configuration parameters.

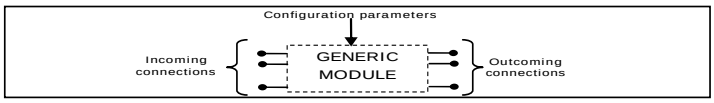

Fig. 2. A generic *UFOme* module.

Currently *UFOme* includes three categories of modules (i.e., *visualization*, *matching* and *evaluation*) that will be briefly described in the following. We want to point out that the aim of this paper is not to describe the modules but to underline the usefulness and effectiveness of the *UFOme* system.

2.2.1 Visualization

This category of modules includes the *OntoLoader* module. It, by exploiting the Jena API [9], allows to visualize an ontology and to obtain useful information such as the list of classes, properties and instances. The ontology is represented as a graph with edges representing the relationships between classes. The user can navigate the ontology and choose different types of visualizations and layouts (see Fig. 4). It is also possible to visualize in the same GUI both the ontologies to be mapped.

2.2.2 Matching

This is the most important category of modules, since through its modules the effective ontology mapping is performed. Currently, *UFOme* includes three individual matchers: *Lucene*, *String* and *Wordnet* and a module for combing them (i.e., the *Combiner* module). Here we provide an overall description of these modules.

Lucene

The Lucene [13] matcher implements the Lucene Ontology Matcher (LOM) algorithm [18]. The aim of the LOM ontology matcher is to exploit all the sources of linguistic information (e.g., local name, comments, and labels) present in the ontologies to be mapped. The LOM matcher aims at discovering mapping between entities (i.e., concepts, relationships and instances) of a *source* and *target* ontologies. In particular, each *source* ontology entity is transformed into a *virtual document* by exploiting the concept of *Lucene Document*. *Virtual documents* are stored into a Lucene index maintained in the main memory. Mappings are derived by using entities of the *target* ontology as search arguments against the index created from the *source* ontology. Similarity between *virtual documents* is computed by the scoring schema implemented in *Lucene*.

WordNet Matcher

The *WordNet* [21] matcher allows comparing ontology entities by considering their semantic meaning. In particular, for assessing similarities between entities, we adopt an adaptation of the Jiang and Conrath Metric (J&C) [10]. This metric along with several others are included in the Java WordNet Similarity Library (JWSL) [8], an ongoing project which aims at providing a Java API for accessing WordNet.

String Matcher

The string matcher implements three algorithms for comparing strings, that is, *I-Sub* [19], *Jaro Winkler* [20] and *Edit Distance* [12]. The user when choosing to include this module in a mapping task can configure the module to use one of the three implemented strategies.

The Combiner module

This module allows to combine/filter results from different matchers according to several strategies (e.g., weighting results of the matchers, introducing a threshold).

It is worthwhile pointing out that *UFOme* allows designing and implementing new matchers that can be included into the system as plugins. This way *UFOme* becomes a comprehensive mapping environment in which developers can implement and plug in new modules according to their needs.

2.2.3 Evaluation

This category of modules includes the *Evaluator*, *Comparer* and *Performance Evaluator* modules. The *Evaluator* module allows evaluating the suitability of a matching strategy in terms of quality of results. In particular, it computes measures of Precision, Recall and F-measure [3] that are classical Information Retrieval metrics. These metrics are based on the comparison of an expected result and the result returned by the system. In the context of ontology mapping, we compare a set of mappings obtained by a mapping task w.r.t a reference alignment.

The *Comparer* module allows the comparison of two matching strategies in terms of Precision, Recall and F-measure. This way the user avoids coding new programs, but just picking up graphical modules (see Fig. 3) can have an immediate background on which of these two strategies is the most appropriate.

The *Performance Evaluator* module allows to evaluate performance (in terms of time elapsed) of the different modules, of a sub mapping task (by considering a subset of modules) or of the mapping task in the whole.

3 *UFOme* : make easy ontology mapping

This section aims at showing the suitability of *UFOme* in a real ontology mapping problem. We chose two ontologies (the 101 and the 205) belonging to the OAEI 2006 [16] benchmark test suite. We examine in detail the different phases of the mapping task execution, and show how *UFOme* can be profitably exploited.

3.1 Phase 1: *Designing*

In this phase the user can choose the various modules to be included in the mapping task (see Fig. 3).

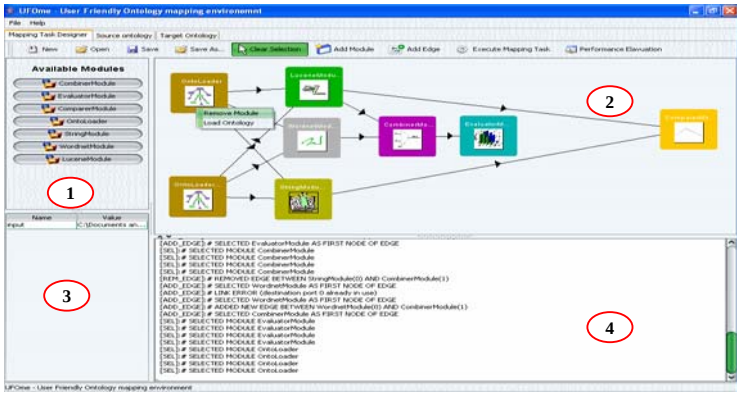

Fig. 3 The *UFOme* GUI.

An *UFOme* user can pick up the modules shown in the left hand side of the *UFOme* interface (1) and put them into the *mapping task composer* (2). Parameters of each module are assigned by exploiting the table (3). Therefore, the modules must be

connected according to the strategy that the user wants to implement. For instance in Fig. 3, the results produced by the two *OntoLoader* modules are passed on to the three matchers. Notice that the direction of the connections will be exploited by the logical layer of the *UFOme* architecture for running the topological sort algorithm (see Section 2.1). Moreover, the log area (4) provides information about mapping activities and possible errors.

After composing the mapping task, that can also be saved, the user can choose to visualize the ontologies to be mapped. That can be done by right-clicking on the *OntoLoader* modules and choosing the *Load Ontology* option (see Fig. 3). The loaded ontology appears as depicted in Fig. 4. The central part (1) shows a graph representation of the ontology while the right column (2) the ontology taxonomy. The dialog (3) allows changing the visualization layout. The toolbar (4) shows other information such as: instances, other types of relationships (i.e., not isa), domain and range of properties, and so forth. It is also possible to show the two ontologies to be mapped in the same *JTab* thus the user can discover and suggest initial mapping candidates.

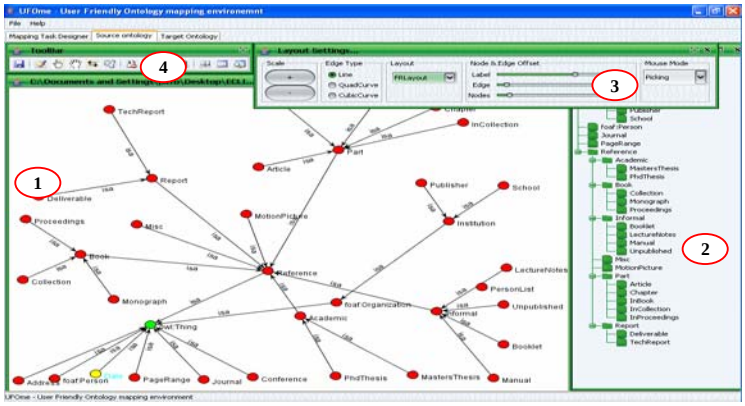

Fig. 4 The *UFOme* ontology perspective.

3.2 Phase 2: *Running*

In this phase the mapping task is executed according to the order determined by the topological sort algorithm. Results produced by each module are both stored in the module, for allowing individual analysis of the results, and passed on to the modules to which it is connected. Once executed a mapping task can be evaluated.

3.3 Phase 3: *Evaluation*

In this phase of the mapping task the user can check results of the task and improve them by choosing a different mapping strategy (i.e., a different combination of modules). In particular, while in current ontology mapping systems, designing different techniques means coding ad-hoc programs, in *UFOme* it corresponds to graphically (re)connect a set of modules.

A user, by right-clicking on a module, can find interesting information related to the execution. For instance, in Fig. 5, by right-clicking on the *Evaluator* module, the user can choose to see the correct, lost or wrong mappings discovered by the (sub) mapping task identified by the dotted area.

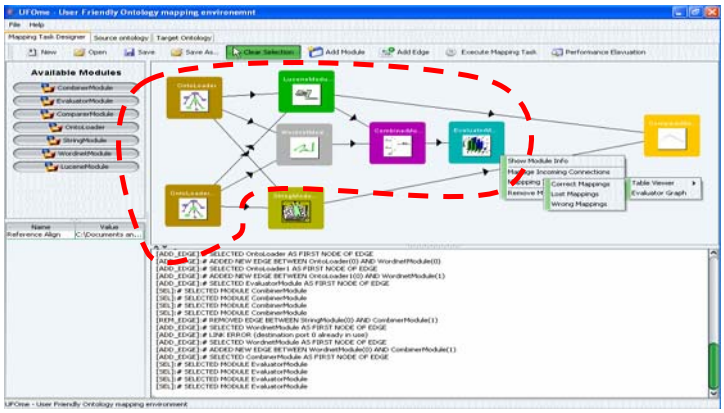

Fig. 5. *UFOme* evaluation options.

In Fig. 6 the correct mappings are compared to wrong mappings on the basis of a reference alignment.

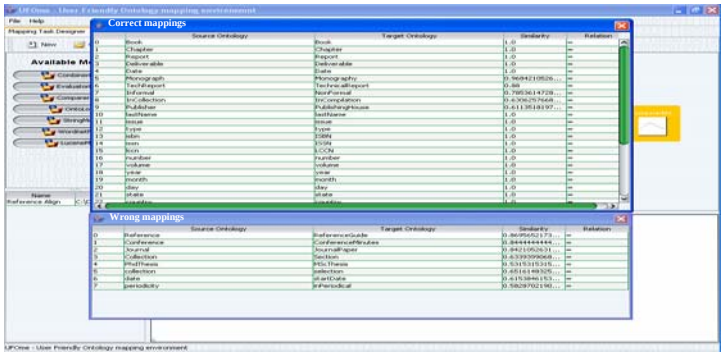

Fig. 6. Comparison between correct and wrong mappings.

Notice that the *Evaluator* module (Fig. 5) takes as input the result of the combination (obtained by the *Combiner* module) of the mappings discovered by both the *Lucene* and *WordNet* matcher. It is important pointing out that the strategies implemented by the *Combiner* module can be several (e.g., weighted sum of the mappings provided by each individual matcher, simple merging of results). The user can also choose to evaluate a mapping task in terms of Precision, Recall and F-Measure. By choosing the *Evaluator Graph* option (see Fig. 5) a new GUI will appear (see Fig. 7).

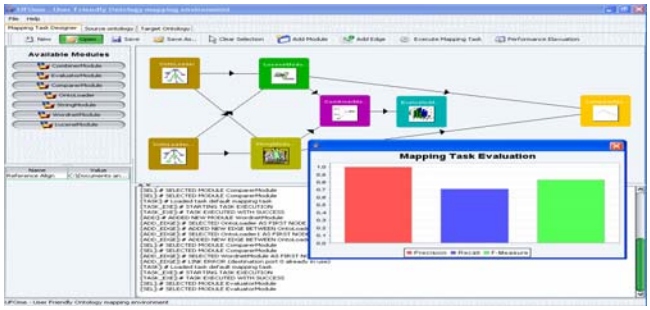

Fig. 7. Evaluation of the results of a mapping task in terms of Precision, Recall and F-Measure.

The *Comparer* module allows the comparison of two matching strategies w.r.t quality of results produced by each of them. Fig. 8 shows the comparison between the *Lucene* matcher and the *String* matcher on the considered ontologies.

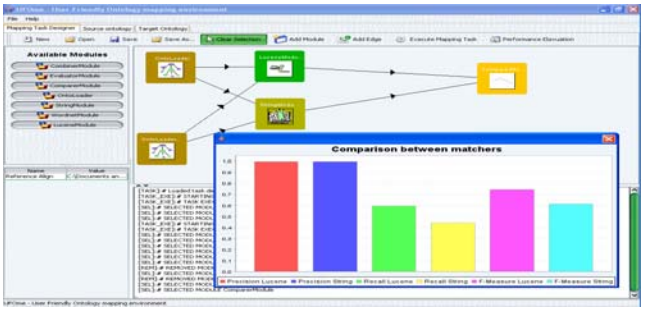

Fig. 8. Comparison between two mapping strategies.

UFOme, differently from other ontology mapping tools that underestimate the importance of performance evaluation, through the *Performance Evaluator* allows to analyze performance (in terms of time elapsed) of a mapping strategy (see Fig. 9).

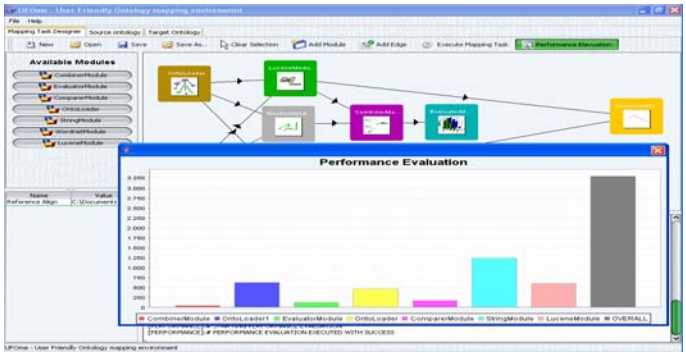

Fig. 9. Performance evaluation of the modules included in the mapping task and of the overall mapping process. Times (*y axis*) are expressed in *msec*.

Fig. 9 shows the times elapsed (on a Pentium IV 3.0 GHz with 2GB memory) by the different matchers as well as the overall mapping task execution time.

4 Related Work

To the best of our knowledge there are no system that entirely covers all the phases of a mapping task identified in the Section 1. In Table 1 we compare the main characteristics of *UFOme* with those of similar tools.

Table 1. Comparision of *UFOme* with similar tools.

	Designing				**Evaluation**	
	Ontology navigation	**Graphical Mapping Composition**	**Candidates Suggestion**	**Modular Architecture**	**Graphic Evaluation Support**	**Graphic Performance Evaluation**
UFOme	Yes	Yes	Yes	Yes	Yes	Yes
OLA [5]	Yes	No	No	No	Yes	No
Prompt [15]	Yes	No	Yes	Yes	No	No
Alignment API [6]	No	No	No	Yes	No	No

As can be noticed, some of the features of *UFOme* are supported by other tools. For instance, ontology navigation is supported by both OLA and Prompt which is implemented as a plugin of Protégé (http://protege.stanford.edu). However, *UFOme* is the only tool that provides a support for graphically composing mapping tasks. The tool closer to *UFOme* is OLA (Owl Lite Alignment). OLA [5] is a system for ontology mapping endowed with a GUI. It is built upon the API described in [6]. Both *UFOme* and OLA are endowed with a GUI. However, *UFOme* provides the mapping task *composer* that allows to: (i) quickly composing mapping tasks; (ii) combine and evaluate different alignments strategies. This latter aspect is often underestimated by mapping algorithms/tools in which designing new strategies correspond to implement new code. OLA does not support the evaluation of the *combination* of different mapping strategies, and in order to evaluate different strategies, batch programs in Java based on the API [6] need to be implemented. OLA features a tool for alignment comparison which computes different metrics (e.g. Precision, Recall). *UFOme* offers the same functionality but also features a performance evaluation module. In particular, the time elapsed for each module and the overall time of the entire mapping process are shown. Finally, *UFOme* also implements the saving of mapping tasks along with related results for future reuse.

5 Conclusions and Future Work

This paper described the *UFOme* system that features a graphical environment for supporting users in all the phases of a mapping task. To the best of our knowledge *UFOme* is the only system provided with a mapping composing interface based on graphical modules that allows a user to quickly design, combine and compare different mapping strategies. *UFOme* gives an effective support in choosing the correct mapping strategy and avoids users the burden to explicitly code new programs when changing mapping strategy. We described the architecture of the system and, through a working example, showed how it can be easily exploited by users.

Moreover, we compared it with similar systems. As future work we aim at including in the system new matching components and performing a more detailed evaluation.

References

1. Choi, N., Song, I., Han, H.: A survey on Ontology Mapping. SIGMOD Record 35(3) (2006) pp. 34--41
2. Cormen, T., Leiserson, C. E., Rivest, R. L., Stein C.: Introduction to Algorithms. MIT Press and McGraw-Hill.
3. Do, H., Melnik, S., Rahm E.: Comparison of schema matching evaluations. In Proc. of GI-Workshop Web and Databases, Erfurt, Germany, (2002)
4. Ehrig, M., Staab, S.: QOM-quick ontology mapping. In Proc. of ISWC 2004, Hiroshima, Japan, (2004) pp. 683--697
5. Euzenat, J., Loup D., Touzani D., Valtchev D.: Ontology Alignment with OLA. In Proc. of EON 2004, Hiroshima, Japan, (2004)
6. Euzenat, J.: An API for ontology alignment. In Proc. of ISWC 2004, Hiroshima, Japan , (2004) pp. 698--712
7. Falconer, S., Noy, NF, Storey, M.: Towards the need of cognitive support for ontology mapping. In Proc. of OM-2006, Athens, Georgia, USA (2006) pp. 25--37
8. Java WordNet Similarity Library (JWSL) and the Similarity Experiment. http://grid.deis.unical.it/similarity
9. Jena - The Jena Project. http://jena.sourceforce.net
10. Jiang, J., Conrath, D.: Semantic similarity based on corpus statistics and lexical taxonomy. In Proc. of ROCLING X, Taiwan (1997)
11. Kotis K, Vouros GA The HCONE Approach to Ontology Merging. In proc. of ESWS 2004, Heraklion, Greece, (2004) pp. 137-- 151
12. Levenshtein, I.V. Binary Codes Capable of Correcting Deletions, Insertion and Reversals. Soviet Physics-Doklady 10(8) (1966) pp. 707--710
13. Lucene- The Apache Lucene project. http://lucene.apache.org
14. Mitra, P., Noy, N. F., Jaiswal, A. R.: OMEN: A Probabilistic Ontology Mapping Tool. In proc. of ISWC 2004, Hiroshima, Japan (2004) pp. 71--83
15. Noy, N.F., Musen, M.A.: The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. Int. J. of Human-Computer Studies 59 (2003) 983-1024
16. Ontology Alignment Evaluation Initiative. http://oaei. ontologymatching.org
17. Pan, R., Ding Z., Yu, Y., Peng., Y.: A Bayesian Network Approach to Ontology Mapping. In Proc. of ISWC 2005, Galway, Ireland (2005) pp. 563--577
18. Pirrò, G., Talia, D.: An approach to Ontology Mapping based on the Lucene search engine library. In proc. of SWAE '07, Regensburg, Germany (2007) pp. 407--412
19. Stoilos, G., Stamou, G., and Kollias, S. A String Metric for Ontology Alignment. In proc. of ISWC 2005,Galway, Ireland, (2005) pp. 623--637
20. Winkler, W. E. The state of record linkage and current research problems. Statistics of Income Division, Internal Revenue Service Publication (4) (1999)
21. WordNet - WordNet online. http://wordnet.princeton.edu/online