Journal of Universal Computer Science, vol. 18, no. 2 (2012), 194-217
submitted: 29/9/10, accepted: 16/12/11, appeared: 28/1/12 © *J.UCS*

MaF: An Ontology Matching Framework

Jorge Martinez-Gil, Ismael Navas-Delgado, and José F. Aldana-Montes
(Department of Computer Science, University of Málaga
Boulevard Louis Pasteur 35, PC 29071, Málaga, Spain
jorgemar@acm.org, {ismael, jfam}@lcc.uma.es)

Abstract: In this work, we present our experience when developing the Matching Framework (MaF), a framework for matching ontologies that allows users to configure their own ontology matching algorithms and it allows developers to perform research on new complex algorithms. MaF provides numerical results instead of logic results provided by other kinds of algorithms. The framework can be configured by selecting the simple algorithms which will be used from a set of 136 basic algorithms, indicating exactly how many and how these algorithms will be composed and selecting the thresholds for retrieving the most promising mappings. Output results are provided in a standard format so that they can be used in many existing tools (evaluators, mediators, viewers, and so on) which follow this standard. The main goal of our work is not to better the existing solutions for ontology matching, but to help research new ways of combining algorithms in order to meet specific needs. In fact, the system can test more than $6 \cdot 136!$ possible combinations of algorithms, but the graphical interface is designed to simplify the matching process.
Key Words: Ontology Matching; Knowledge Integration; Software Tools
Category: M.1, M.3

1 Introduction

The notion of ontology is important as a form of representing real world facts. Ontology matching[1] is a key aspect of knowledge management [Wen, 2009]; it allows organizations to model their own knowledge without having to stick to a specific standard. In fact, there are two good reasons why most organizations are not interested in working with a standard for modeling their own knowledge: (a) it is very difficult or expensive for many organizations to reach an agreement about a common standard, and (b) these standards do not often fit to the specific needs of the all participants in the standardization process.

Ontology matching is perhaps the best way to solve the problems of heterogeneity. There are a lot of techniques for aligning ontologies very accurately [Noy, 2004], but the complex nature of the problem to be solved makes it difficult for these techniques to operate satisfactorily for all kinds of data, in all domains, and as all users expect [Kiefer et al., 2003]. As a result, techniques that combine existing methods have been proposed as a feasible solution. The goal

[1] We call ontology matching to the task of finding correspondences between ontologies and ontology alignment to the result of this task

of these techniques is to obtain more accurate matching algorithms. The way to combine these matching algorithms is currently being exhaustively researched. The reason is that obtaining satisfactory ontology alignments is a key aspect for such fields as:

- Semantic integration [Bernstein and Melnik, 2004]. This is the process of combining metadata residing in different sources and providing the user with a unified view of these data. This kind of integration should be done automatically, because manual integration is not viable, at least not for large volumes of information.
- Ontology mapping [Ehrig and Sure, 2004]. This is used for querying different ontologies. An ontology mapping is a function between ontologies. The original ontologies are not changed, but the additional mapping axioms describe how to express concepts, relations, or instances in terms of the second ontology. They are stored separately from the ontologies themselves. A typical use case for mapping is a query in one ontology representation, which is then rewritten and handed on to another ontology. The answers are then mapped back again. Whereas alignment merely identifies the relationship between ontologies, mappings focus on the representation and use of the relations.
- The Web Services industry, where Semantic Web Services (SWS) are discovered and composed [Cabral et al., 2004] in a completely unsupervised manner. Originally SWS alignment was based on exact string matching of parameters, but nowadays researchers deal with issues of heterogeneous and constrained data matching.
- Data Warehouse applications [Fong et al., 2009]. These kinds of applications are characterized by heterogeneous structural models that are analyzed and matched either manually or semi-automatically at design time. In such applications matching is a prerequisite of running the actual system.
- Similarity-based retrieval [Sistla et al., 1997]. Semantic similarity measures play an important role in the information retrieval field by providing the means to improve process recall and precision. These kinds of measures are used in various application domains, ranging from product comparison to job recruitment.
- Agent communication [Kun et al., 2010]. Current software agents need to share a common terminology in order to facilitate the data interchange between them. Using ontologies is a promising technique to facilitate this process, but there are several problems related to the heterogeneity of the ontologies used by the agents which make the understanding at semantic level difficult. Ontology matching can solve this kind of problem.

All this means that the business and scientific communities are seeking to develop automatic or semiautomatic techniques to reduce the tedious task of creating and maintaining the ontology alignments manually. However, the nature of the problem is complex because finding good similarity functions is, data, context, and sometimes even user-dependent, and needs to be reconsidered every time new data or a new task is inspected [Kiefer et al., 2003]. Figure 1 shows an example of this fact; it is an example of alignment between ontologies representing players from two football teams. This alignment between ontologies could be true for some cases and for some people, but probably not for all. Therefore, we need mechanisms to make ontology matching as independent as possible of data, context and users.

The main contribution of this work is the presentation of a new ontology matching tool which we have called Matching Framework (MaF), therefore, we describe *detailed accounts of completed software-system projects which can serve as 'how-to-do-it' models for future work in the same field*. Our approach is based on distance and similarity methods instead of frameworks based on the definition of theoretical aspects of matching. These systems are generally accomplished by considering the underlying Description Logics (DL) on which the ontologies are founded [Kalfoglou and Schorlemmer, 2003].

The rest of this work is organized as follows: Section 2 describes the State-of-the-Art related to ontology matching and its associated problems. Section 3 describes the general characteristics for MaF framework. Section 4 shows two practical examples for MaF, the first example is focused on end users and the second is focused on algorithm developers. Section 5 describes a case study in which we solve common cases using the framework. Finally, we discuss the conclusions and the possible future improvements for the framework.

2 Problem Statement

An ontology is "a specification of a conceptualization" [Gruber, 1993], i.e. an abstract representation of the world like a set of objects. In this work, we are going to use the intuitive notion of ontology as a set of terms with relationships among them. On the other hand, as stated in [Euzenat and Shvaiko, 2007], the process of aligning ontologies can be expressed as a function where given a pair of ontologies, an (optional) input alignment, a set of parameters and a set of resources, returns an alignment.

Definition 1 (Ontology matching task). *Let O be the set of ontologies and A the set of alignments. An ontology matching task $omt : O \times O \mapsto A$ maps two input ontologies $o_1, o_2 \in O$ to an alignment $a \in A$ using a matching function.*

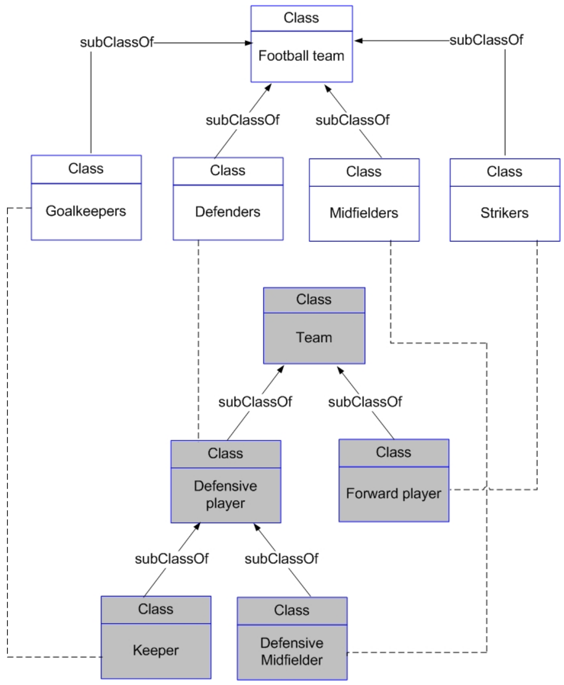

Figure 1: Example of ontology alignment. In this example we have found semantic correspondences between two ontologies from two soccer teams. The dotted lines between classes mean that a kind of semantic correspondence between them exists

Definition 2 (Ontology matching function). *An ontology matching function omf is a function* $omf : E \times E \mapsto R$ *that associates the correspondence of two entities* e_1, $e_2 \in E$ *to a score* $sc \in \Re$ *in the range [0, 1] stating the degree of confidence for the correspondence between* e_1 *and* e_2.

A score of 0 stands for complete inequality and 1 for equality of e_1 and e_2. The set of mappings are part of an alignment that can be defined formally in the following form.

Definition 3 (Ontology alignment). *An ontology alignment is a set* $\{T, MD\}$. *T is a set of tuples in the form* $\{(id, \mu_1, \mu_2, n, R)\}$. *id is an identifier,* μ_1 *and* μ_2 *are entities belonging to two different ontologies, R is the semantic correspondence between these entities and n is a real number between 0 and 1 representing the mathematical probability that R may be true* [Euzenat and Shvaiko, 2007].

The entities that can be related are the concepts, properties, individuals and, even axioms of the ontologies. On the other hand, MD is metadata (date, time consumption and so on) related to the matching process for information purposes and it is only relevant for collecting statistical data like the computational efficiency of the process. We have focused on concepts, properties and individuals.

On the other hand, n can represent equivalence, generalization, subsumption, disjointness and, so on. Without explanation here, it is going to state equivalence only.

There are a lot of matching functions. Most of them are used to obtain an ontology alignment. These functions exploit a wide range of information, such as ontology characteristics (i.e. metadata, element names, data types, and structural properties), characteristics of individuals, as well as background knowledge from dictionaries, thesauri, even web resources. Most authors tend to categorize simple matchers in three groups defined by [Rahm and Bernstein, 2001]: Element-Based matchers, Structure-Based matchers, and Hybrid matchers. These kinds of matchers are described and their representative implementations are discussed in the next subsection.

Definition 4 (Alignment evaluation). *An alignment evaluation ae is a function ae* : $A \times A_R \mapsto precision \times recall$*, where precision and recall are real numbers ranging over the unit interval* $[0, 1]$.

Precision states the fraction of retrieved correspondences that are relevant for a matching task. Recall is the fraction of the relevant mappings that are obtained successfully in a matching task. In this way, precision is a measure of exactness and recall a measure of completeness. The problem here is that techniques can be optimized either to obtain high precision at the cost of the recall or, alternatively, recall can be optimized at the cost of the precision. For this reason a measure, called f-measure, is defined as a weighting factor between precision and recall. In this work, we use the most common configuration which consists of weighting precision and recall equally.

2.1 Element-Based Matching Algorithms

Element-Based Matching Algorithms are methods that take into account only textual information about the entities (instead of their relations to other entities). This textual information can be exploited in many ways: comparing the identifiers of the entities, their associated comments, the identifiers of their children, their associated individuals, and so on. In general, there is a common agreement for grouping these methods. These are the most notable categories:

– Text similarity methods. Text similarity methods are string based techniques for identifying similar elements [Euzenat and Shvaiko, 2007]. Such a method may be used to identify identical classes of two ontologies based on their similar label or description [Stoilos et al., 2005]. This includes global namespaces, too. In [Navarro, 2001] a survey can be seen.

– Keyword extraction algorithms. This kind of algorithm consists of identifying keywords that can be used to detect some kind of meaning and therefore to identify the semantics of a class and its relation to other classes. This is performed by looking for proper nouns [Vazquez and Swoboda, 2007] or by analyzing the frequency of common terms [Cilibrasi and Vitanyi, 2007].

– Language-based algorithms. Language-based methods such as removing unnecessary words (stop-words) or performing text stemming can be used to handle class or attribute names [Ji et al., 2006]. For example, they can be used in order to detect that the class "lift" and "elevator" represents the same object in the real world. This also means considering typical language based prefixes or suffixes as well as other text pattern matching tools [Ierusalimschy, 2009].

– Identification of word relations. This involves the inclusion of linguistic resources such as lexicons and thesauri in order to identify synonyms. It is also common to use a lexical database such as WordNet [Wordnet, 2009] to identify relationships between concepts. In recent times, web knowledge extraction techniques are being used in this field too.

2.2 Structure-Based Matching Algorithms

Structured-Based Matching Algorithms are computational methods that take into account the structure of the ontology (instead of textual information about them). These are the most outstanding categories:

– Class inheritance analysis (is-a). This method considers the inheritance between classes in order to identify "is-a"-relationships. For example, we might consider two ontologies A and B. Ontology A might contain a "car", while B contains "vehicle". The class inheritance analysis tries to find inheritance relationships between the concepts of A and B (a "car" -is-a-"vehicle").

– Structural analysis / Taxonomic structure. The structural analysis identifies identical classes by looking at their attributes and related classes. The main idea behind algorithms of this kind is that two classes of two ontologies are similar or identical if they have the same attributes and the same neighbor classes.

– Data interpretation and analysis of key properties. Since instances are often included, it is possible to identify similar classes by looking at their instances. If two classes have the same instances, then they will most likely match each other. In some cases, it might be possible to identify the meaning of an attribute by looking at an instance. For example, if a string contains "years old" then it represents an age related attribute.

– Graph-Mapping. This kind of algorithm can be used to identify similar structures in two ontologies by looking for identical parts [Ziegler, 2006]. Unlike the structural analysis, the graph based mapping method interprets only the graphical representation of the ontology structure and looks at paths, children and leaves.

2.3 Semantic Matching Algorithms

According to Euzenat and Shvaiko [Euzenat and Shvaiko, 2007], semantic matching algorithms handle the input based on its semantic interpretation. It is assumed that if two entities are the same, then they share the same interpretations. Thus, they are well grounded deductive methods. Most outstanding approaches are propositional satisfiability and description logics reasoning techniques. The problem of these techniques is that "pure deductive methods do not perform very well alone for an essentially inductive task like ontology matching". Hence they need a preprocessing phase which provides entities which are declared, for example, to be equivalent [Ehrig, 2007].

2.4 Contribution to the State-of-the-art

Despite the large number of existing techniques, experience tells us that finding an appropriate solution is far from being trivial. As we commented earlier, the heterogeneity and ambiguity of data description makes it unavoidable that optimal mappings for many pairs of entities will be considered as best results by none of the existing ontology matching algorithms. For this reason, researchers have introduced the notion of composite matchers which are aggregations of simple matching algorithms.

The main idea of this kind of matchers is to combine similarity values predicted by multiple matchers to determine correspondences between ontology elements. In this way, it is possible to benefit from both the high degree of precision of some algorithms and at the same time the broader coverage of others [Eckert et al., 2009]. Some of the most outstanding proposals following this paradigm are COMA [Do and Rahm, 2002], COMA++ [Aumueller et al., 2005], FOAM [Ehrig and Sure, 2005], OntoBuilder [Roitman and Gal, 2006] and RiMOM [Li et al., 2009], to name a few. COMA was the first proposal for combining strategies when matching ontologies and, COMA++ improved on COMA with a nicer graphical user interface and an extension of the set of matchers. Despite these tools are the pioneers, even today, they are still considered the point of reference in the field because they showed practitioners and researchers the possibilities of matcher combination. FOAM approach began to give importance to the heuristic selection of the weights for basic matchers; they proposed using a sigmoid function to appropriately weight matchers emphasizing good matchers and de-emphasizing the worst ones. The OntoBuilder introduced the notion of top-k mappings in order to provide an alternative for a single best matching. And finally, the RiMOM framework has shown a really good performance in the past OAEI contests [Caracciolo et al., 2008]. A detailed description of these approaches is out of the scope of this work. However, these and other approaches have been reviewed in depth by Euzenat & Shvaiko[Euzenat and Shvaiko, 2007]. To the best of our knowledge, MaF is the system with the largest number of basic matchers, with the largest number of possible matcher combinations, and along with COMA [Do and Rahm, 2002] and DIKE [Papoli et al., 2003], one of the first to be described from a software experience perspective, which is one of the main challenges addressed by Shvaiko and Euzenat [Shvaiko and Euzenat, 2008].

On the other hand, several tools provide the user with semi-automatic mechanisms in order to obtain perfect mappings. Nevertheless we agree with Euzenat and Shvaiko [Euzenat and Shvaiko, 2007] when they say that "Many applications require submitting matching results to user scrutiny and control before using them, but the better the automated part of the task, the easier the control". In this way, our approach considers that the output results will be the input to such tools as correctors, evaluators, mediators, viewers and, so on, as we show in Section 4. The main goal of MaF is to provide reasonable results to the users and third party applications, and the objective of this work is to describe how to do so.

3 MaF Description

The Matching Framework (MaF) is an ontology matching framework that includes the common features that users may need. MaF uses classic algorithms, instead of those based on logics algorithms. MaF allows users to aggregate algorithms and combine them in order to operate with the input ontologies and generate the output alignment. The framework has been designed to accept OWL (DL, Lite or Full) ontologies as input, while the output will be basically represented as lists of pairs with a similarity value between 0 and 1 associated, indicating no probability to represent the same real object for the value 0 and

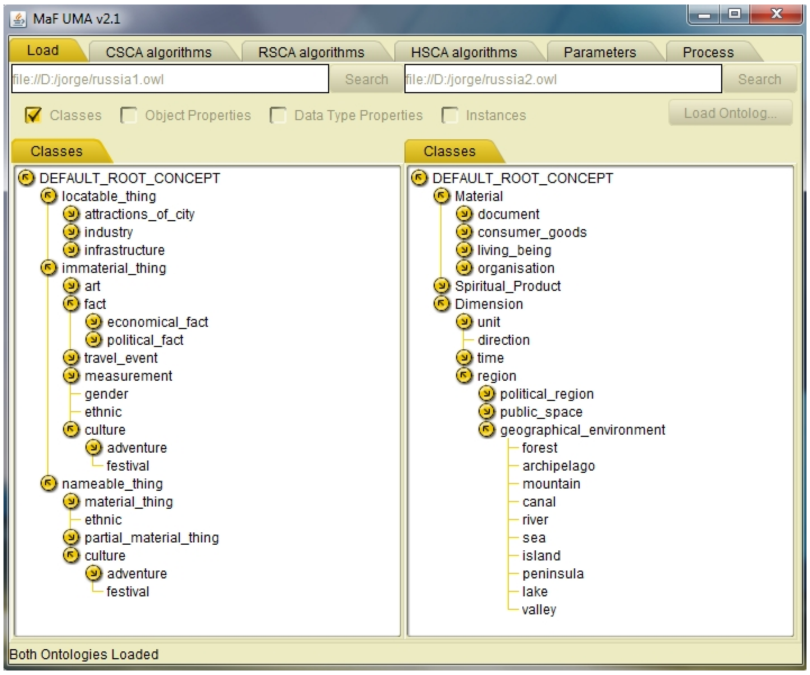

Figure 2: Screenshot from the main form. We can see the loaded ontologies in a taxonomic way in order for users to locate what they are interested in

total probability for the value 1. This output is compliant with the standard format from the Ontology Alignment Evaluation Initiative (OAEI) [OAEI, 2008]. Figure 2 shows a screenshot from the initial form of MaF where two ontologies have been renderized in a taxonomic way in order to be presented to the user. On the other hand, major characteristics for MaF are:

– MaF allows both schema and instance matching. All of the matching algorithms provided can work with concepts, object properties, datatype properties and individuals. Do not confuse instance matching with matching based on instances. MaF provides both kinds but the first one consists of the use of element-based methods to compare ontology instances, and the second consists of the comparison of concepts using their associated instances.

– MaF allows both element and structure matching. The matching algorithms can be used not only for aligning the individual elements of the ontology, but also ontology structures, too. It is possible to combine the two kinds of algorithms in order to obtain a third kind that is even more powerful and that we have called a hybrid algorithm.

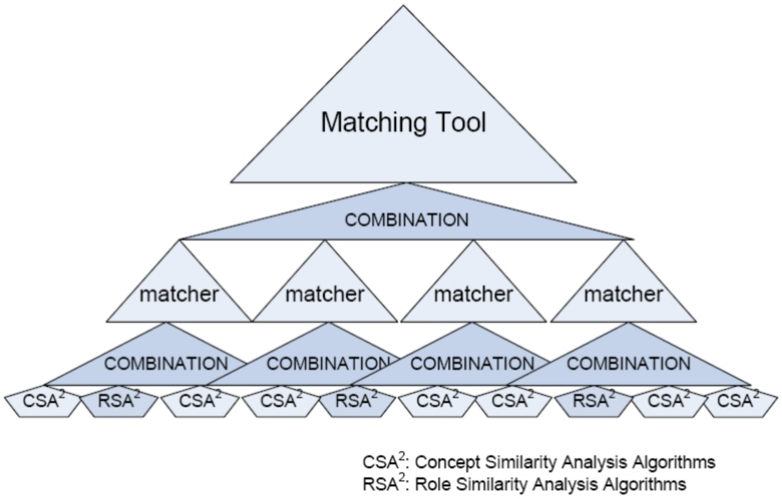

Figure 3: Three-Layer Software Architecture for MaF. In the first layer users can select basic matchers. Then in the second, users can combine the basic matchers in order to obtain a complex algorithm. Finally, it is possible to choose the combination formula and the threshold

– MaF allows both language and restriction matching. The matching algorithms can use an approach based on the language, but they can use an approach based on constraints such as relationships, too.

– MaF uses background knowledge. The framework can use knowledge from an external dictionary called WordNet [Wordnet, 2009] in order to find more complex correspondences.

– Results provided by the framework present a cardinality of 1:1. Thus, each element of the first input ontology may be aligned with a single element of the second input ontology. Moreover, output alignments are directional, thus, the techniques used to make the comparison between items return the same results regardless of the input order.

3.1 MaF Architecture

MaF has been designed as a framework. A framework is a software structure defined to support other projects and which represents a software architecture that models the overall relationship between the software components and provides a working methodology which extends or uses domain applications.

Figure 3 shows the architecture which consists of a three-layer pyramid that allows users to develop algorithms for different levels of complexity. By means of combination techniques, it is possible to climb from the first to a second logic level, where algorithms are selected according to some predefined criteria. This selection generates a matcher. The users decide on the criteria used because we suppose that they will have a good understanding of the problem to be solved and the ability to choose an appropriate strategy to address it. The next subsections will discuss in more detail the architecture of our framework. As part of the MaF kernel, we have designed the following components:

- An *Ontology Management System*, which is the responsible for reading the input ontologies and transforming them into an internal model of data representation. This management system has been developed by means of the Jena API [McBride, 2002] which allows the parsing, creation and searching of OWL models. The model is stored in main memory, so the size of the ontologies that can be processed using MaF depends largely on the main memory available in the computer which executes it.
- A *Combination Management System*, which is a component that manipulates the alignment processes so that they are easily combinable. Combinations here are made on the basis of numerical combinations of output values from the algorithms. This module belongs to the MaF kernel so it is transparent for the users.
- A *Filtering System* which is responsible for filtering values. This module belongs to the kernel of the framework and its goal is to define the value of the threshold for the output of MaF. It can be easily modified.

3.2 First Layer of the Architecture

MaF implements two kinds of algorithms in the first layer of the architecture:

Definition 5 (Concept Similarity Analysis Algorithm). *A Concept Similarity Analysis Algorithm is a kind of element-based matching algorithm that tries to find semantic correspondences between the concepts of the input ontologies only.*

Definition 6 (Role Similarity Analysis Algorithm). *A Role Similarity Analysis Algorithm is a kind of structure-based matching algorithm that tries to find semantic correspondences between the roles (properties) of the input ontologies only.*

In the rest of this work the acronyms CSA^2 and RSA^2 will be used to name to the two kinds of algorithms respectively. Both kinds of algorithms are now explained in more detail.

3.2.1 Concept Similarity Analysis Algorithms (CSA^2)

The CSA^2 that we have included are:

- **Distance Based Methods:** Block Distance, Levenshtein Distance, 3-grams Distance, Euclidean Distance, Monge Elkan Distance.
- **Name Based Methods:** Char Frequ. Similarity, Name Similarity, Soundex Similarity and Substring Similarity.
- **WordNet Based Methods:** Absolute Distance, Normal Depth, Gloss Overlap, Cosynonymy Similarity, Synonymy Similarity, Optimistic Depth, and Pessimistic Depth.

Distance Based Methods (DBM) and Name Based Methods (NBM) are based on [Cohen et al., 2003]. WordNet Based Methods (WBM) are new implementations for algorithms described in [Pedersen et al., 2004]. It should be taken into account that Optimistic Depth and Pessimistic Depth are not clearly described algorithms but they are appropriate when you are comparing a concept with the same notation but different means. For example, when comparing the terms “bucks” and “dollars”, Optimistic Depth considers that you are comparing the same term, and Pessimistic Depth considers that you are comparing money with animals.

3.2.2 Role Similarity Analysis Algorithms (RSA^2)

The RSA^2 that we have included are:

- **Class Methods:** Class Depth, Class NumChildren, Class NumLeaves, Class NumParents, Class Type.
- **Object Property Methods:** ObjectProperty Depth, ObjectProperty NumChildren, ObjectProperty NumParents.
- **Datatype Property Methods:** DatatypeProperty Depth, DatatypeProperty NumChildren, DatatypeProperty NumParents, DatatypeProperty Type.

All of these methods are new implementations from trivial algorithms that compute the number of entities related to a given one. All the algorithms have to be used with caution because different entities can share exactly the same structural configuration.

3.3 Second Layer of the Architecture

Using algorithms from the first layer does not allow to obtain good results. The advantage of MaF is that allow to combine algorithms in order to get hybrid matchers.

Definition 7 (Hybrid Similarity Analysis Algorithm). *A Hybrid Similarity Analysis Algorithm is a kind of matching algorithm that combines, at least, one CSA^2 algorithm with one RSA^2 algorithm in order to obtain a more complex technique to find correspondences between ontologies.*

For this reason, several times, combinations of this kind are called second level matchers in the literature. From now on, we are going to use the acronym HSA^2 to name this kind of matcher.

Example 2. An example of HSA^2 is an algorithm for determining the equivalence of two concepts by comparing the name of their individuals. For example, let us to imagine that with have the concept *Car* in an ontology with the following individuals: _Volvo, _Renault, _Ford, _Toyota, and _Opel. We have another ontology with the concept *Automobile* and the following individuals: _Ford, _Audi, _Fiat, _Volvo, and _Toyota. We have 10 individuals and 6 of them overlap, so we have a probability of 0.6 for the correspondence may be true.

The HSA^2 algorithms that we have included in MaF are:

- *Hybrid Name Children.* This technique is based on the detection of overlapping children's names from the entities to be compared.
- *Hybrid Name Parents.* This technique is based on the comparison of the parent's names from the entities to be compared.
- *Hybrid Name Leaves.* This technique is based on the comparison of the names of the leaves in the branch from the entities to be compared.
- *Hybrid Name Instances.* This technique is based on the on the detection of overlapped instance identifiers from the entities to be compared.
- *Hybrid Name Average Path.* This technique is based on the comparison of the paths from the entities to be compared.
- *Hybrid Name Rank Path.* This technique is based on the comparison of the paths names from the entities to be compared.
- *Hybrid Name Datatype Property.* This technique is based on the comparison of the datatype properties from the entities to be compared.

– *Hybrid Name Class Range*. This technique is based on the comparison of the ranges (TextString, NonNegativeInteger, and so on) from the entities to be compared.

Each hybrid matcher is a combination of a structural method and element-based one. For this reason, all textual comparisons can be made using the CSA^2 algorithms of the first layer. On the other hand, it should be taken into account that all of them are new implementations for algorithms proposed in the past. For example, algorithms equivalent to the Hybrid Name Children and Hybrid Name Leaves were described by Do and Rahm [Do and Rahm, 2002]. It is possible to find a detailed description for all of them in Euzenat and Shvaiko [Euzenat and Shvaiko, 2007].

3.4 Third Layer of the Architecture

The third layer of the architecture designed can be considered as a complete ontology alignment tool, which is also the recipient of the results of the matching algorithms from the lower layers, it must be able to accept instructions leading to manually adjust optimally the weights associated to the algorithms. To do that, we have provided two groups of available operations at this level: Algorithm Combination and Mapping Filtering using thresholds.

3.5 Algorithm Combination

The algorithm combination module allows users to define the way in which the score for the mappings will be computed. Let m be a mapping, let $a_1, a_2, a_3, ..., a_n$ the set of results from algorithms to be combined, then we provide the following ways to combine algorithms:

– *Average mean*. This option computes the average from the results obtained from the selected matching algorithms.

$$score_m = \frac{1}{n} \cdot \sum_{i=1}^{n} a_i$$

– *Maximum score*. This option computes the maximum value from the results obtained from the selected matching algorithms.

$$score_m = \max a_n$$

– *Minimum score*. This option computes the minimum value from the results obtained from the selected matching algorithms.

$$score_m = \min a_n$$

– *Minkowski distance.* This option allows computing the Minkowski distance value between the results obtained from algorithms.

$$score_m = \sqrt[n]{\sum_{i=1}^{n} a_i^n}$$

– *Weighted product.* This option allows composing a formula to calculate a weighted product using partial results from matchers.

$$score_m = \prod_{i=1}^{n} a_i \cdot w_i, where \sum_{i=1}^{n} w_i = 1$$

– *Weighted sum.* This option allows calculating a weighted sum using partial results from matchers.

$$score_m = \sum_{i=1}^{n} a_i \cdot w_i, where \sum_{i=1}^{n} w_i = 1$$

In this way, we have 136 basic matchers than can be aggregated in 136! different ways in the first and second layer. When the third level is reached, these 136! combinations can be combined in the 6 different ways that we have shown above. In fact, there are more possible combination because there are infinite numerable ways to configure a Minkowski distance, and even to configure a weighted sum or product. So we have that MaF allows, at least, $6 \cdot 136!$ different matchers.

3.6 Mapping Filtering

The mapping filtering module allows users to select only the most promising mappings from the set of all possible mappings. In this way, the final alignment A will be the set of mappings m filtered by a threshold value T:

– *Hard threshold.* This kind of threshold returns mappings above a specific value.

$$A = \{m, \forall m \in A \rightarrow m.score \geq T\}$$

– *Delta threshold.* This approach uses as threshold the highest similarity value out of which a particular constant value d is subtracted.

$$A = \{m, \forall m \in A \rightarrow m.score - d \geq T\}$$

– *Proportional threshold.* This uses the percentage of the highest similarity value as the threshold.

$$A = \{m, \forall m \in A \rightarrow m.score \in \{\max_T m.score\}\}$$

4 Tool Use

In this section, we are going to show a practical example for aligning ontologies using MaF. One of the main advantages of MaF is that it provides a built-in front end so the task of selecting and combining algorithms is not very difficult. MaF can be used by two types of users; end-users and algorithm developers.

4.1 End-users

We consider that the end-users are those people who only use the front-end to align ontologies. The way to proceed for this kind of users could be summarized as follows:

- *Load two OWL ontologies to align.* These ontologies have to be OWL in any of its versions (DL, Lite or Full)
- *Select the entities* (classes, object properties, datatype properties and instances) that they wish to align, as we explained in Section 3.
- *See the ontologies they have loaded in a taxonomic form.* Figure 2 allows users to do this.
- *Choose the algorithms and combine them.* At this point, the users should choose the basic matching algorithms and the formula to combine them.
- *Choose a threshold to show results.* After the ontology alignment is done, the user may choose a threshold that will filter the results achieved in the ontology alignment, to show only the results that meet this threshold. In this way, the user can filter the results, rejecting those mappings that are not of a high enough quality.
- *See/Save the output results for the matching process.* Once the results have been obtained, it may be possible to repeat the process of choosing other ontology matching algorithms, another threshold, or by changing the combination technique. In this way, the best ontology alignment can be modeled for the user.

4.2 Algorithm Developers

Algorithm developers are those who use the whole functionality of the framework to develop new matching algorithms. The functional specification for an algorithm developer is different from the role of an end-user. We have developed MaF using Eclipse[2], so it would not be difficult to extend it. Moreover, in Table 1, we show a summary for the initially provided features included in MaF.

[2] http://www.eclipse.org

All provided algorithms, the different kind of combinations and the different techniques for thresholding are presented.

4.3 Usability of MaF

We have borrowed a methodology from Brooke [Brooke, 1996] in order to test the usability of our framework, we have asked several undergraduate and graduate students in the field of Computer Science for working with several ontology matching frameworks. They have to value with a number several key points concerning to these frameworks.

In addition to MaF, we have considered three additional ontology matching frameworks in order to compare them. We have chosen COMA++ (Web Edition) [Aumueller et al., 2005], Ontobuilder [Roitman and Gal, 2006], and FOAM [Ehrig and Sure, 2005] for the reasons already advanced previously, and we have asked to our students for solving the OAEI benchmark using them. We did not tell the students that MaF is our software tool. Moreover, it should be taken into account that the students had a good knowledge of databases and ontologies, but most of them are not experts in the ontology matching field.

From the results of this experiment, we have obtained that COMA++ and MaF are the tools with the highest degree of usability. For example, it can be extracted that COMA++ is the system that students would like to use more frequently, the system which needs less technical support, the most consistent software tool and the system which needs least previous knowledge to get started and use it.

However, according to our experiment, MaF is the least complex system, the easiest system to use, the system with the best integration of the functions, the system that can be learned the most quickly and the tool which is the less cumbersome to use. The tests give us evidence of the benefits of using MaF in matching scenarios and validate the design of our user interface. The results can not be taken as statistically conclusive, so we will keep working in this regard in future work.

5 Case Study: Solving a benchmark

We have solved a case study that consists of solving several tests from the OAEI Benchmark [OAEI, 2008]. This benchmark dataset offers several test cases which try to measure the quality of proposed methods and tools when solving several use cases which are common in ontology matching scenarios. It should be taken into account that we can solve the cases of the benchmark using our understanding of the problem and appropriately selecting the matchers to address it. Our purpose is not to compete with optimized algorithms, but to show that it is possible to use our tool for solving common scenarios. Table 2 shows several of

Nr	Feature	Nr	Feature
	Distance Based Methods		**Hybrid Comparison Methods**
1	Block Distance (a)	33-45	Hybrid Name Children (a-m)
2	Levenshtein Distance (b)	46-58	Hybrid Name Parents (a-m)
3	3-grams Distance (c)	59-71	Hybrid Name Leaves (a-m)
4	Euclidean Distance (d)	72-84	Hybrid Name Instances (a-m)
5	Monge Elkan Distance (e)	85-97	Hybrid Name Av. Path (a-m)
6	Smith Waterman Distance (f)	98-110	Hybrid Name Rank Path (a-m)
7	Jaro Distance (g)	111-123	Hybrid Name Data.Prop. (a-m)
8	Needleman Wunch Distance (h)	124-136	Hybrid Name Class Range (a-m)
9	SWG Distance (i)		
	Name Based Methods		
10	Char Frequency Similarity (j)		
11	Soundex Similarity (k)		
12	Name Similarity (l)		
13	Substring Similarity (m)		
	WordNet Based Methods		**Combinations**
14	Absolute Distance	1	Average Combination
15	Normal Depth	2	Maximum Combination
16	Gloss Overlap	3	Minimum Combination
17	Cosynonymy Similarity	4	Minkowski Combination
18	Optimistic Depth	5	Weighted Product Combination
19	Synonymy Similarity	6	Weighted Sum Combination
20	Pessimistic Depth		
	Class Methods		
21	Class Depth		
22	Class NumChildren		
23	Class NumLeaves		
24	Class NumParents		
25	Class Type		
	Property Methods		**Thresholds**
26	OProperty Depth	1	Hard Threshold
27	OProperty NumChildren	2	Delta Threshold
28	OProperty NumParents	3	Proportional Threshold
29	DProperty Depth		
30	DProperty NumChildren		
31	DProperty NumParents		
32	DProperty Type		

Table 1: List of features (matching algorithms, combinations, and kind of the thresholds) included in MaF

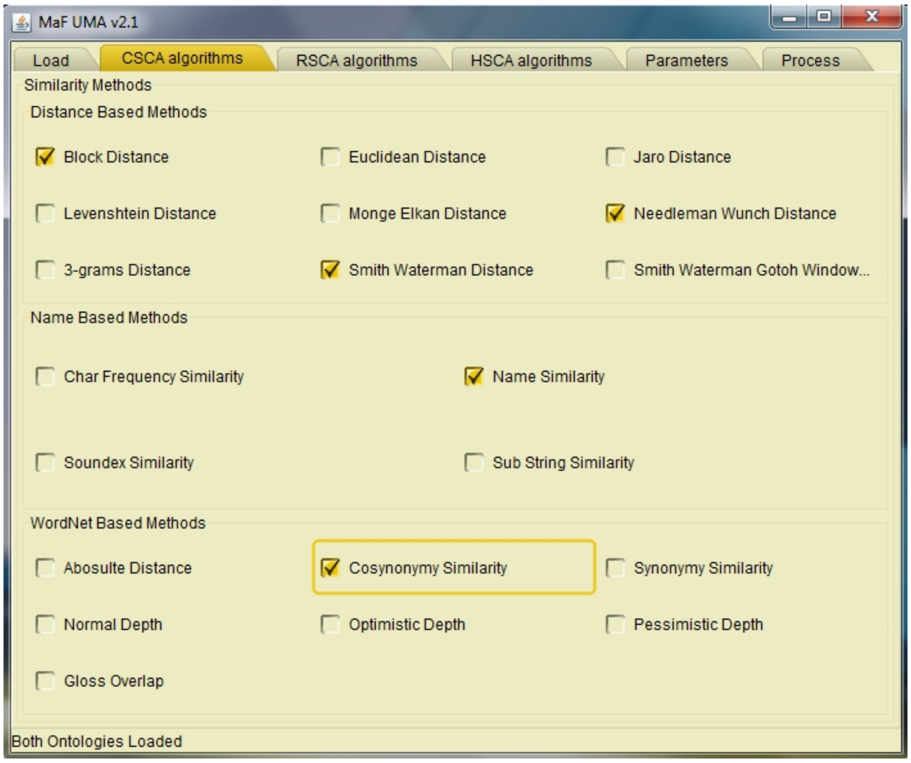

Figure 4: Screenshot from the first layer of MaF. CSA^2, thus, Distance Based Methods, Name Based Methods, WordNet Based Methods can be chosen using this form

the most representative cases of the benchmark dataset [OAEI, 2008], the configuration that we propose and the results that we have obtained. The test has been performed using the concepts only.

The working mode is as follows: The process begins when the user selects the two ontologies to be processed. After that, the user has the option of defining matching algorithms to be used from the first and second layer. In Figure 4, it is possible to see the screenshot for the selection of algorithms of this kind. The second layer allows user to choose the hybrid algorithms. In the third layer, the composition formula and the threshold for filtering the values in the output results are defined. Finally, the tool performs the matching between the two ontologies according to these criteria. Figure 5 shows an example of the output for this step. The format follows a standard format so that it could be useful as an input for other applications.

Test:	101
Description:	Comparing an ontology to itself
Solution:	Whichever CSA^2 is appropriate to do that
Results:	Precision 1.00 Recall 1.00 F-Measure 1.00
Explanation:	Comparing a object to itself must return 1 (by definition)
Test:	201
Description:	Labels are modified and moved arbitrarily
Solution:	Very artificial case. We choose Hybrid Name Instances
Results:	Precision 0.82 Recall 0.53 F-Measure 0.64
Explanation:	It is difficult for MaF to work in no-meaning scenarios
Test:	202
Description:	Labels are modified and comments deleted
Solution:	We didn't look for comments in 201.
Results:	Same as in 201
Explanation:	Recall is low, only classes with individuals can be compared
Test:	203
Description:	We have generated some misspellings in the target ontology
Solution:	Char Frequency Similarity to detect typos and misspellings
Results:	Precision 1.00 Recall 1.00 F-Measure 1.00
Explanation:	Typos in short words are difficult to detect: low threshold
Test:	204
Description:	Different naming conventions for labels
Solution:	Choose the Name Similarity Method.
Results:	Precision 1.00 Recall 0.89 F-Measure 0.94
Explanation:	Procedure can be changed in `csca.normalizationMethods`
Test:	205
Description:	Labels are replaced by synonyms
Solution:	Maximum of 3-Gram, Synonym and Hybrid N. Instances
Results:	Precision 0.85 Recall 0.87 F-Measure 0.86
Explanation:	WordNet is not perfect. It is a good idea to complement it
Test:	206
Description:	Target is translated to another language other than English
Solution:	Hybrid N. Inst.(3-Gram) and Soundex
Results:	Precision 0.72 Recall 0.72 F-Measure 0.72
Explanation:	Very difficult case. We hope to find similar sounds
Test:	301
Description:	Bibliography ontologies from BibTeX and MIT
Solution:	Maximum of 3-Gram and Hybrid N. Average Path (3-Gram)
Results:	Precision 0.65 Recall 0.97 F-Measure 0.78
Explanation:	Real problems needs a good heuristic to be solved

Table 2: Case Study for solving part of the OAEI Benchmark using MaF

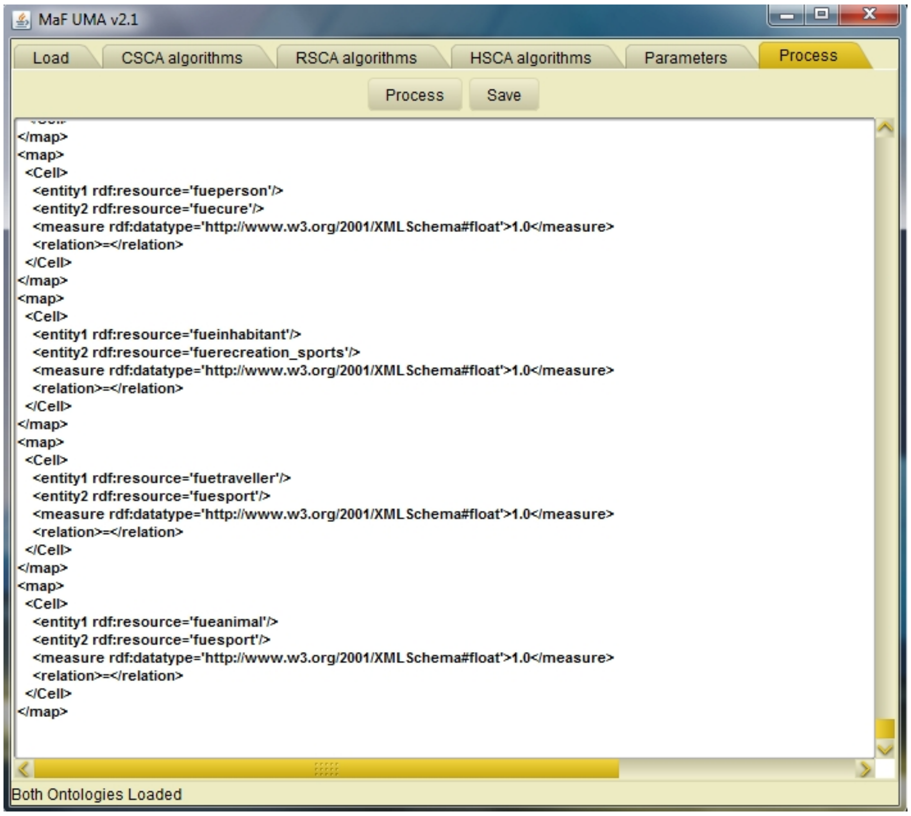

Figure 5: Screenshot from the result form of MaF. Results are generated following a standard format proposed by the OAEI in order to be used by other software tools and applications

6 Conclusions

In this work, we have presented our experience when designing, developing and, using MaF, an ontology matching framework that has been designed using a pyramidal three-layer software architecture for combining basic ontology matching algorithms. The purpose of facilitating the combination of algorithms is to obtain more accurate and user-dependent ontology alignments. To the best of our knowledge, the MaF tool provides the largest number of basic ontology matching algorithms and the biggest number of possible matcher combinations (up to $6 \cdot 136!$ different combinations). The software tool has been described from a user experience perspective, i.e., we are more interested on the user experience rather than on optimizing the final results, as most of the other proposals have done in the past.

On the other hand, MaF is a software framework and, therefore, its main goal is not provide a *golden matcher* (i.e. an optimal instance of this framework), but to provide users with the necessary help to create algorithms that meet their needs. Moreover, programmers can extend the functionality easily. As we have shown using a case study, most common situations in ontology matching can be solved, although the quality of the output depends largely on the expertise of the user and their ability to choose an appropriate combination of matchers to solve the problem which is being addressed.

As future work, we plan to work mainly on two improvements. The first of them consists of extending MaF so that we can be sure that does not need human intervention to perform the ontology matching tasks, thus, MaF will be able to automatically choose the appropriate matching algorithms and thresholds for each situation. This is commonly referred to as ontology matching self-tunning in the literature [Lee et al., 2007]. It is possible to read a preliminary version for this proposal in [Martinez-Gil et al., 2008]. On the other hand, we aim to extend MaF with a suitable engineering solution so that it can process very large ontologies i.e., with thousands of entities, both accurately and efficiently.

Acknowledgements

We wish to thank to the anonymous reviewers for the comments and suggestions which have helped to improve this work. We thank to Lisa Huckfield for proofreading this manuscript. This work has been funded by Spanish Ministry of Innovation and Science through: *REALIDAD: Efficient Analysis, Management and Exploitation of Linked Data.*, Project Code: TIN2011-25840 and by the Department of Innovation, Enterprise and Science from the Regional Government of Andalucia through: *Towards a platform for exploiting and analyzing biological linked data*, Project Code: P11-TIC-7529.

References

[Aumueller et al., 2005] Aumueller D, Do HH, Massmann S, Rahm E. Schema and ontology matching with COMA++. *Proceedings of the SIGMOD Conference*, 2005; 906-908.

[Bernstein and Melnik, 2004] Bernstein PA, Melnik S. Meta Data Management. *Proceedings of the International Conference on Data Engineering*, 2004; 875.

[Brooke, 1996] Brooke J. Sus: A quick and dirty usability scale. *Usability evaluation in industry* 1996; Taylor and Francis.

[Cabral et al., 2004] Cabral L, Domingue J, Motta E, Payne TR, Hakimpour F. Approaches to Semantic Web Services: an Overview and Comparisons. *Proceedings of the European Semantic Web Symposium*, 2004; 225-239.

[Caracciolo et al., 2008] Caracciolo C, Euzenat J, Hollink L, Ichise R, Isaac A, Malais V, Meilicke C, Pane J, Shvaiko P, Stuckenschmidt H, Svab-Zamazal O, Svatek V. Results of the Ontology Alignment Evaluation Initiative. *Proceedings of the Ontology Matching Workshop at ISWC*, Karlsruhe, Germany, 2008.

[Cilibrasi and Vitanyi, 2007] Cilibrasi R, Vitanyi PMB. The Google Similarity Distance. *IEEE Trans. Knowl. Data Eng.* 2007; 19(3):370-383.

[Cohen et al., 2003] Cohen WW, Ravikumar P, Fienberg SE. A Comparison of String Distance Metrics for Name-Matching Tasks. *Proceedings of IIWeb*, 2003; 73-78.

[Do and Rahm, 2002] Do HH, Rahm E. COMA - A System for Flexible Combination of Schema Matching Approaches. *Proceedings of the VLDB Conference*, 2002; 610-621.

[Do et al., 2002] Do HH, Melnik S, Rahm E. Comparison of Schema Matching Evaluations. Proceedings of Web, Web- Services, and Database Systems, Erfurt, Germany, 2002; 221-237.

[Eckert et al., 2009] Eckert K., Meilicke C, Stuckenschmidt H. (2009) Improving Ontology Matching Using Meta-level Learning. *Proceedings of the European Semantic Web Conference*, 2009; 158-172.

[Ehrig and Sure, 2004] Ehrig M, Sure Y. Ontology mapping - an integrated approach. *Proceedings of the European Semantic Web Conference*, 2004; 7691.

[Ehrig and Staab, 2004] Ehrig M, Staab S. QOM - Quick Ontology Mapping. *Proceedings of the International Semantic Web Conference*, 2004; 683-697.

[Ehrig and Sure, 2005] Ehrig M, Sure Y. FOAM - Framework for Ontology Alignment and Mapping - Results of the Ontology Alignment Evaluation Initiative. *Proceedings of Integrating Ontologies*, 2005.

[Ehrig, 2007] Ehrig M. *Ontology Alignment: Bridging the Semantic Gap.* Springer, 2007.

[Euzenat and Shvaiko, 2007] Euzenat J, Shvaiko P. *Ontology Alignment.* Springer, 2007.

[Fong et al., 2009] Fong J, Shiu H, Cheung D. A relational-XML data warehouse for data aggregation with SQL and XQuery. *Softw., Pract. Exper.* 2009; 38(11):1183-1213.

[Giunchiglia et al., 2004] Giunchiglia F, Shvaiko P, Yatskevich M. S-Match: an Algorithm and an Implementation of Semantic Matching. *Proceedings of the European Semantic Web Symposium,* 2004: 61-75.

[Gruber, 1993] Gruber T. A translation approach to portable ontology specifications. *Knowledge Adquisition* 1993; 5(2):199-220.

[Ierusalimschy, 2009] Ierusalimschy R. A text pattern-matching tool based on Parsing Expression Grammars. *Softw., Pract. Exper.* 2009; 39(3):221-258.

[Kalfoglou and Schorlemmer, 2003] Kalfoglou Y, Schorlemmer WM. IF-Map: An Ontology-Mapping Method Based on Information-Flow Theory. *J. Data Semantics*, 2003; **1**:98-127.

[Kiefer et al., 2003] Kiefer C, Bernstein A, Stocker M. (2007) The Fundamentals of iSPARQL: A Virtual Triple Approach for Similarity-Based Semantic Web Tasks. *Proceedings of the International Semantic Web Conference*, 2007; 295-309.

[Kun et al., 2010] Kun Z, Manwu X, Hong Z, Jian X. Agent service matchmaking algorithm for autonomic element with semantic and QoS constraints. Knowl.-Based Syst. 2010; 23(2):132-143.

[Ji et al., 2006] Ji Q, Liu W, Qi G, Bell DA. LCS: A Linguistic Combination System for Ontology Matching. *Proceedings of the KSEM*, 2006; 176-189.

[Lee et al., 2007] Lee Y, Sayyadian M, Doan A, Rosenthal AS. eTuner: tuning schema matching software using synthetic scenarios. *VLDB J.* 2007; 16(1):97-122.

[Levenshtein et al., 1966] Levenshtein V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics-Doklady*, 1966; 10: 707-710.

[Li et al., 2009] Li J, Tang J, Li Y, Luo Q. (2009) RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Trans. Knowl. Data Eng.* 21(8); 1218-1232.

[Martinez-Gil et al., 2008] Martinez-Gil J, Alba E, Aldana-Montes JF. Optimizing Ontology Alignments by Using Genetic Algorithms. *Proceedings of NatuReS, 2008.*

[McBride, 2002] McBride B. Jena: A Semantic Web Toolkit. *IEEE Internet Computing* 2002; 6(6): 55-59.

[Navarro, 2001] Navarro G. A guided tour to approximate string matching. *ACM Comput. Surv.* 2001; 33(1):31-88.

[Noy, 2004] Noy N. Semantic Integration: A Survey Of Ontology-Based Approaches. *ACM Sigmod Record*, 2004; 33(4):65-70.

[OAEI, 2008] Ontology Evaluation Initiative. http://oaei.ontologymatching.org. Visit date: 30-oct-2008.

[Papoli et al., 2003] Palopoli L, Terracina G, Ursino D. DIKE: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Softw., Pract. Exper.* 2003; 33(9):847-884.

[Pedersen et al., 2004] Pedersen T, Patwardhan D, Michelizzi J. WordNet::Similarity - Measuring the Relatedness of Concepts. *Proceedings of the AAAI conference*, 2004; 1024-1025.

[Rahm and Bernstein, 2001] Rahm E, Bernstein P. A survey of approaches to automatic schema matching. VLDB J., 2001; 10(4):334-350.

[Roitman and Gal, 2006] Roitman H, Gal A. OntoBuilder. Fully Automatic Extraction and Consolidation of Ontologies from Web Sources Using Sequence Semantics. *Proceedings of the EDBT Workshops*, 2006; 573-576.

[Shvaiko and Euzenat, 2008] Shvaiko P, Euzenat J. Ten Challenges for Ontology Matching. *Procedings of the OTM Conferences (2)*, 2008; 1164-1182.

[Sistla et al., 1997] Sistla AP, Yu CT, Venkatasubrahmanian R. Similarity Based Retrieval of Videos. *Proceedings of the International Conference on Data Engineering* 1997; 181-190.

[Stoilos et al., 2005] Stoilos G, Stamou GB, Kollias SD. A String Metric for Ontology Alignment. *Proceedings of the International Semantic Web Conference*, 2005; 624-637

[Vazquez and Swoboda, 2007] Vazquez R, Swoboda N. Combining the Semantic Web with the Web as Background Knowledge for Ontology Mapping. *Proceedings of the OTM Conferences (1)*, 2007; 814-831.

[Wen, 2009] Wen YF. An effectiveness measurement model for knowledge management. *Knowl.-Based Syst.*, 2009; 22(5):363-367 .

[Wordnet, 2009] WordNet. `http://wordnet.princeton.edu`. Visit date: 11-march-2009.

[Ziegler, 2006] Ziegler P, Kiefer C, Sturm C, Dittrich KR, Bernstein A. Detecting Similarities in Ontologies with the SOQA-SimPack Toolkit. *Proceedings of the Extending Databases Technologies Conference* 2006; 59-76.