

GENOMA: GENeric Ontology Matching Architecture

Roberto Enea¹, Maria Teresa Pazienza¹, Andrea Turbati¹

¹ DII, ART Group, University of Rome, Tor Vergata
roberto.enea@gmail.com
{pazienza, turbati}@info.uniroma2.it

Abstract. Even though a few architectures exist to support the difficult ontology matching task, it happens often they are not reconfigurable (or just a little) related to both ontology features and applications needs.

We introduce GENOMA, an architecture supporting development of Ontology Matching (OM) tools with the aims to reuse, possibly, existing modules each of them dealing with a specific task/subtasks of the OM process. In GENOMA flexibility and extensibility are considered mandatory features along with the ability to parallelize and distribute the processing load on different systems. Thanks to a dedicated graphical user interface, GENOMA can be used by expert users, as well as novice, that can validate the resulting architecture.

We highlight as main features of developed architecture:

- to select, combine and set different parameters
- to evaluate the matching tool applied to big size ontologies
- efficiency of the OM tool
- automatic balancing of the processing load on different systems

keywords: ontology matching, evaluation, architecture

1 Introduction

As a matter of fact ontology development became a very frequent task for either expert or novice users. Ontologies can be written using different standards: among others, the most frequently used is RDF¹ (Resource Description Framework) together with its two extensions: RDFS² (RDF Scheme), and OWL³ (Web Ontology Language). In creating a new ontology generally two possible approaches are adopted:

1. starting from a shared vocabulary (as the FOAF⁴ ontology when describing people-related terms) and then adding the new domain specific data
2. defining everything from scratch.

¹ <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>

² <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>

³ <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>

⁴ <http://www.foaf-project.org/>

Consequently, using more than one ontology in a new task can be extremely difficult, since different resources⁵ (identified by different lexicalizations) of distinct ontologies can be used to identify the same concept. While humans naturally deal with such an ambiguity and try to understand the similarity between these resources, systems cannot easily do the same. This means that even when using a formal representation for an ontology (with a given serialization), the richness of the natural language is still an issue!

In this paper, we present GENOMA, an architecture that helps in the development of Ontology Matching (O.M.) systems, which are tools that infer different types of similarities among two or more ontologies. By first, we introduce the most relevant tools for O.M., we compare them thus providing a motivation for our architecture (section 2). In the following section 3, and in its subsections, we introduce our new architecture, describing its main features. Finally, with section 4, we provide a small summary of what we achieved with our research activity. Description of Ontology Matching instances is not the objective of this paper.

2 Ontology Matching tools: an Overview

Ontology Matching is the branch of the Ontology Engineering that aims in finding similarities between resources of two or more ontologies. It tries not just to find similarities between pair of resources, but also to align ontologies and then to merge them in a new ontology, which is composed of all resources, found in the input ones. While to recognize similarities between ontologies several techniques and algorithms have been defined, the basic process [1] is always the same: the input to the framework consists in two or more ontologies from which, using a matcher (that can be composed of a single module or be built on top of several minor modules), it is produced an alignment matrix. The matcher(s) can be configured using specific parameters as well as some external linguistic resources (such as WordNet⁶, for example). The resulting alignment matrix is $N \times M$ (where N is the number of resources of the first ontology, M is the number of the resources of the second ontology) and contains the similarity values between these resources. A_{ij} is the similarity value obtained from a particular algorithm between the i -resource of the first ontology and the j -resource of the second ontology.

Main differences among existing O.M. tools are:

- the size of the ontologies they are able to manage;
- the formal language in which the ontologies should be written (mostly RDF and one of its serialization, such as RDF/XML or N-Triples);
- which resources they are able to compare (classes, instances, properties, ...);
- the natural languages in which the two ontologies should be written (or if they are able to compare ontologies written in different languages);

⁵ In this paper, we use the term resources to refer to any element inside an ontology (classes, instances or properties) unless otherwise specified

⁶ <http://wordnet.princeton.edu/>

- The cardinality of the output alignment for each resource it returns (1:1 or n:m);
- open source or proprietary (a common problem when dealing with any software tools);
- if they use external data or not (this can affect their license as well);
- adoption of just syntactic matching or also some sort of semantic matching approach;

To have more details on the possible features of the matching process, please refer to [2], in our paper we are interested in discussing just how to combine them in a unified architecture and in evaluating the similarity metrics used in the matching tool. The ideas behind the similarity value are that:

- each resource has the maximum similarity value with itself;
- each similarity value is greater than or equal to zero (since normally they are normalized values, they are between 0 and 1);
- considering the same two resources, the similarity value remains the same irrespective of which ontology is considered as the first one in the comparison.

Once these three properties are upheld, the similarity value can be computed using any kind of techniques: string-based, language-based, structure-based, scheme-based, extensional-based, relational-based, probabilistic-based and semantic-based.

Let us introduce what are considered the state of the art Ontology Matching (OM) tools in order to better understand some of the existing solutions for this task.

2.1 Cupid

Cupid [3] is a scheme-based OM tool. It was developed in the context of a collaboration of the University of Washington, the Microsoft Research and the University of Leipzig. It implements an algorithm, which uses both a linguistic matcher (using an external thesaurus) and a structure-based matcher. Input ontologies are considered as graphs. The implemented algorithm is divided into three steps. First, it uses the linguistic matcher to compute the similarity values considering the external thesaurus. Then it adopts a structure-based matcher to obtain the similarity values. Finally, it combines the values obtained in previous phases using a weighted sum and then it produces the alignment matrix.

2.2 COMA and COMA++

COMA [4] (Combination of Matching Algorithm) adopts a parallel approach using matchers. It was developed at the University of Leipzig and it is composed of an extendible library of matchers, a framework which can be used to combine the matchers in a single algorithm and a platform for the evaluation of the results. COMA is provided with six elementary matchers, five hybrid matchers and one matcher that uses the other matchers. Most of these matchers are string-based and linguistic, supporting external resources. COMA offers a more flexible architecture than Cupid and considers the user feedback.

COMA++ [5] is the newer version of COMA that introduces an optimization in re-using the alignment, a better implementation of the base matchers and a user interface.

2.3 iMAP

iMap [6] can be considered as an atypical Ontology Matching tools, since it works on scheme matching and not on what is normally defined as an ontology. This means that iMap tries to find complex matches in scheme, such as “our-price = price * (1 + tax-rate)”⁷ or it is able to discover that the field *address* should be matched with the field *street* and the field *number*. The tool iMAP deals with the problem of matching as a search problem in a, possibly, enormous space. Its modules are called *searchers*, and they work in parallel, each one searching for a specific pattern. For example, a searcher analyzes only string fields, while another focuses on fields containing integers. The operations each searcher performs, depend on the data type they are looking for.

The algorithm implemented in iMAP is divided in three distinct phases:

3. All the searchers, which are working in parallel, return their results, which are the candidate for the next step. Since the number of returned results could be extremely large, a *bean search* [7] approach is adopted.
4. For each attribute of the target scheme, all the candidates of the first scheme are considered and only the ones with specific characteristics are retained.
5. Results of the previous steps are put inside a similarity matrix using the Similarity Estimator Module. This matrix is composed of the couples [target attribute, candidate attributes]. Then the requested alignment is extracted from a filtered version of this matrix.

2.4 GLUE

GLUE [8], developed by the Washington University uses machine learning techniques to discover similarities one-to-one between two distinct taxonomies. The machine learning algorithm used, can be divided into three steps:

1. It learns the joint distribution function of the classes of the two taxonomies. It uses two modules implementing the following base techniques: *content learning* (based on Bayes Learning) and *name learning* (similar to the first one). A meta-matcher performs a linear combination of the results of the other two modules. The weights for the linear combination are decided by the user.
2. The system computes the similarity between pair of classes adopting a joint distributed function provided by the user. From this, a similarity matrix is produced.
3. The alignment between the two taxonomies is obtained by applying several filtering operations.

GLUE is the evolution of a previous tool, LSD [9].

⁷ <http://pages.cs.wisc.edu/~anhai/projects/schema-matching.html>

2.5 Falcon-AO

Falcon-AO [10] was developed by the China Southeast University. It adopt a divide-et-impera approach to perform the matching between two ontologies written in RDFS or OWL. It was designed to manage large ontologies (thousands of resources) using partitioning techniques. The algorithm identifies three phases:

1. It divides the ontologies into smaller structures to take advantages of the hierarchy, through the use of the property *rdf:subClassOf*, and the use of the agglomerative clustering algorithm ROCK [11].
2. It compares the clusters, obtained in the previous step, focusing on the anchors, which are linked resources, obtained using string-based comparison. More anchors there are between two clusters, more similar those clusters are assumed to be. Only clusters with a number of anchors over a specific threshold are considered possible candidate.
3. Two matchers, one linguistic and the other structural, are adopted to generate the similarities matrices; then these matrices are aggregated into the resulting alignment matrix. If the linguistic matcher is able to obtain good results alone, the structural one, which is more complex, is not used.

2.6 ASMOV

ASMOV [12], Automated Semantic Mapping of Ontologies with Verification, was developed during a collaboration between Infotech Software and the University of Miami. It receives two ontologies and an optional alignment and returns a many-to-many alignment among the resources of the two ontologies, mainly classes and properties. It uses an iterative process. In its main phase, after a preprocessing step, it uses lexical, structural and extensional matchers to calculate, iteratively, the similarity between pairs of resources that are then aggregated in a weighted mean. It uses also external resources, such as WordNet and UMLS⁸. Then, it verifies the consistency of the proposed alignments, using five pattern typologies, which can generate inconsistencies. Its main application is the integration of bioinformatics data.

2.7 O.M. tools comparison and motivation for an O.M. Architecture

In the previous sections, we have roughly sketched a few existing O.M. tools and architectures. **Table 1** summarizes these features (according to the ones specified in the beginning of Section 2).

Apart from these features, Falcon offers the possibility to set the values for some parameters, and only COMA++ offers the possibility to decide which matchers to use, without deciding how they can communicate each other and no one even suggests the possibility to deploy his tool on a distributed environment.

⁸ <http://www.nlm.nih.gov/research/umls/>

	Falcon	ASMOV	Cupid	iMap	GLUE	COMA++
Specific for big ontologies	Yes	No	No	No	No	No
Input	RDF/RDFS/OWL	OWL	XML	XML, DB Schemas	XML, DB Schemas	RDF/RDFS/OWL
Resources matched	Classes + Properties + Instances	Classes + Properties	XML Elements	XML, DB schema Elements	XML, DB schema Elements	Classes + Properties
Specific Natural languages	No	English	English	No	No	English
Output alignment	1:1	m:n	1:1	1:1	1:1	1:1
Open source	Yes	No	No	No	No	Yes
Linguistic resources	No	WordNet	WordNet	No	No	WordNet
Type of matching	Linguistic Structural	Linguistic Structural Extensional	Linguistic Structural	Parallel searcher + similarity estimator	Statistical Approach, Machine learning	Linguistic

Table 1. O.M. tools characteristics

After analyzing previous O.M. tools, it emerges that there is no tool that is always better than the others, it depends on several factors. Then there is no architecture that can be classified as the best one or a matching techniques which is always correct. Each time is left to the application developers the decision on what matcher/architecture to choose. And it is not an easy task! That's why we decided to develop a new generic architecture, helping experts or even beginners, to try different combinations and decide which one is the best given a specific application context. Another important feature of our novel architecture is the possibility either to reuse existing matchers, or to develop new ones, always in the context of our architecture. This feature can appear normal nowadays in a context in which extendibility is a core peculiarity of almost any tool or system. Generally, in the scientific environment, the developed tools are close systems (sometimes they are even proprietary one or open one, but their code is too complex and not sufficiently explained to be easily understood and extended).

A strong requirement for our architecture is the development of an easy user interface to help users to assemble the matcher in any way they want to. Finally, since ontologies are becoming bigger and bigger, our architecture should be able to scale out by being deployed on several machine to avoid performance issues.

3 Gen.O.M.A.

Gen.O.M.A (**Generic Ontology Matching Architecture**) or GENOMA is a sort of meta architecture which helps in the development of new specific architectures for the ontology matching task. In this section by first we introduce the architecture (section 3.1), then the provided user interface, which guide the user in the creation of each new architecture and its validation (section 3.2). Then we describe some of the existing matchers modules (section 3.3) and we conclude this section by providing examples of tested and validated architectures (section 3.4), by showing their peculiarities and differences.

3.1 Architecture

In every matching system, the main problem to overcome is improving system's computational efficiency. This is particularly important when evaluating large ontologies, which are the ones used in real case matching scenarios. GENOMA⁹ offers the possibility to be executed in a distributed environment; to achieve this feature, it uses Java RMI¹⁰ (Remote Method Invocation) which enables the execution of each matchers on a given server.

In **Fig. 1** we show the architecture adopted by GENOMA. To handle ontologies, the OWL ART API¹¹ are adopted. The main component of the ontology matching process is the *Matching Composer* (sometimes called just *Composer*). This module coordinates the whole process. It parses the configuration file, handled by the module *Configuration*, of the selected engine¹² (more information about this configuration are given in section 3.2), then launches the desired matchers, using the relative parameters (the only mandatory parameters are the 2 ontologies addresses; the parameters that are specific to each matcher are optional, since they all have default values). Each matcher returns to the Composer its own processed similarity matrix, which is then passed to the next matcher, or a notification if an error occurred during ontology processing. Once all the desired matchers have completed their task by returning the similarity matrix, the Composer produces a detail log and the alignment file, that is an xml file, containing the final similarity matrix, filtered using the threshold selected by the user before starting the matching process (the *Filtered Similarity matrix* in **Fig. 1**). A matcher can uses external linguistic resources, for example by adopting Ontoling¹³ to access DICT¹⁴ and WordNet.

⁹ It can be download from: <https://bitbucket.org/aturbati/ontology-matching-architecture>

¹⁰ <https://docs.oracle.com/javase/tutorial/rmi/>

¹¹ <http://art.uniroma2.it/owlart/>

¹² An engine is the implementation of a matching architecture produced by the user interface of GENOMA

¹³ <http://art.uniroma2.it/software/OntoLing/>

¹⁴ www.dict.org

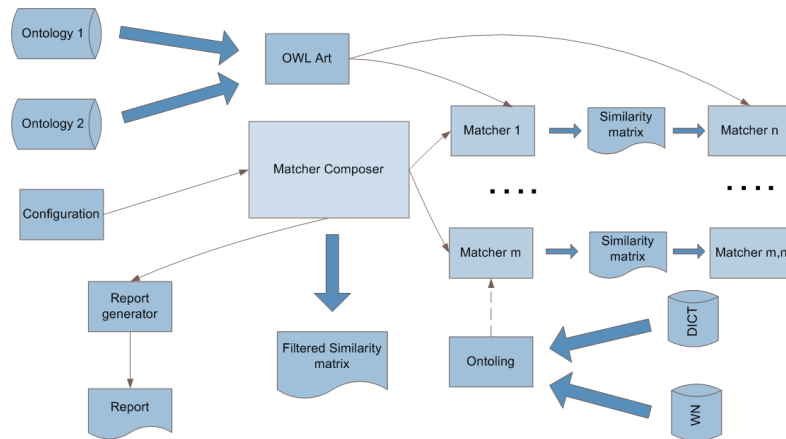


Fig. 1. GENOMA architecture

3.2 User Interface

To help the user in creating, visualizing and validating an architecture for an O.M. tool, GENOMA provides an easy to use and complete User Interface¹⁵ (see **Fig. 2** and hereafter for details). Since the beginning, the user can decide whether to use an already existing configuration or to create a new architecture, using the provided matcher (plus the ones possibly developed by himself).

When selecting an existing configuration, the user specifies the two ontologies to be compared, an optional alignment file (useful when evaluating an O.M. architecture), and the Engine (a file specifying all the information describing a given architecture). Once these information have been provided, the matching process starts. The user can monitor the entire process (its status, any error, the log file and the alignment file produced by the selected architecture): in fact all these details are stored in a MySQL¹⁶ DB, to help retrieve them and guarantee their persistency.

The user interface is even more useful when deciding to assemble a new architecture. The GUI (shown in **Fig. 2**) provides the user what he needs to create each new specific architecture. The user can either save or load a previous created architecture, as well as can validate the current architecture (see Section 3.3 to understand the validation provided by GENOMA). One important feature is the possibility to save the current work as an image, to help showing the created architecture or sharing it with colleagues to discuss it.

The creation of the new architecture is completely interactive and mainly mouse oriented, to help novices in this delicate task. By selecting the desired matcher, the user just need to specify the parameters values (or he can accept the default ones) and

¹⁵The User Interface is deployed as a web application in Tomcat.

¹⁶ www.mysql.com/

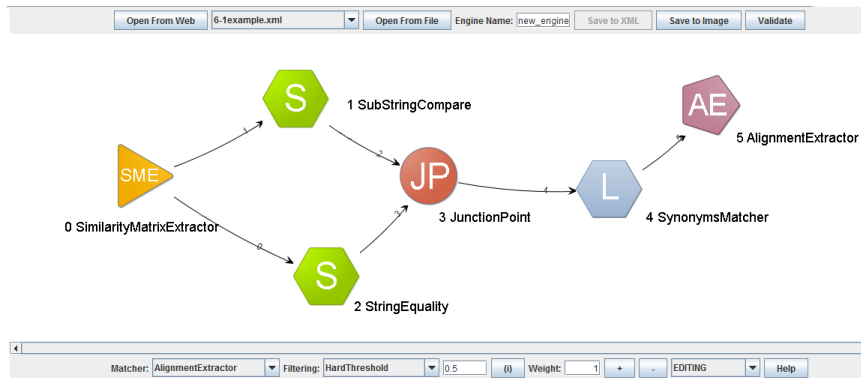


Fig. 2. User Interface when creating an architecture

then link the matchers in several possible deployment configurations (see Section 3.4). All this is achieved easily by using a Java applet and just a couple clicks of the mouse¹⁷!

3.3 Provided Base Matchers

All the elements that can be considered when assembling an architecture in the user interface, at the moment, are divided into, three main classes:

- Structural elements (known as system elements);
- String-based matchers;
- Language-based matchers.

The elements of these classes all are deployed in the same manner. The system elements are: *SimilarityMatrixExtractor* (SME), *JunctionPoint* and *AlignmentExtractor*.

The Similarity Matrix Extractor is the first element in every architecture created by using GENOMA. It can extract the similarity matrix, obtained from another architecture, so it can start the current architecture by using results obtained in the context of a previous execution.

The Junction Point is used to combine two similarity matrices, obtained from the parallel processing (see Section 3.4). Its configuration parameter consists in which aggregation function to use. The default value is the triangular norm, while other value are possible, by expanding its implementation.

The Alignment Extractor executes the inverse operation of the SME: given a similarity matrix, it generates the alignment. It also apply a filter operation before generating the alignment in case the user has provided it.

¹⁷ Just to see how easy to use it is, interested persons could access it at <http://artemide.art.uniroma2.it:8080/GenomaWeb-1.0/>










Name	Symbol	Type	Matcher Functionality	Number of edges
Similarity Matrix Extractor		System	Extraction the similarity matrix	0 input N output
Junction Point		System	Aggregation similarity matrices	2 input N output
Alignment Extractor		System	Extraction the alignment from the similarity matrix	1 input 0 output
String Equality Matcher		String-based	Comparing Strings	1 input N output
SubString Compare Matcher		String-based	Comparing Substrings	1 input N output
N-grams Compare Matcher		String-based	N-grams comparison	1 input N output
String Hamming Distance Matcher		String-based	Hamming distance in the comparison	1 input N output
Synonyms Matcher		Language-based	Comparison using synonyms	1 input N output
Cosynonyms Matcher		Language-based	Comparison using synonyms and then counting the number of synonyms they have in common	1 input N output

Table 2. Matchers list

The other classes, String-based and Language-based matchers, provide the matching part in the architecture. The former considers only the words associated to a resource (for example through the property *rdfs:label* or using the name of the resource itself) treating it as sequence of characters, while the latter performs an analysis based on the language used, for example considering synonyms obtained from a linguistic resource, such as WordNet. The provided language based matchers, in the current release, work only on the English language; the use of the framework OntoLing and Linguistic Watermark [13] with the associated resource, DICT, could be used to analyze other languages as well.

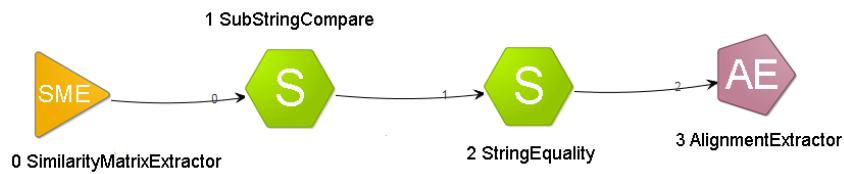


Fig. 3. Matchers in series

The system is able to automatically validate each architecture developed by the user, by checking that all the constraints are respected (for example that each architecture starts with a SME, ends with a Alignment Extractors).

In **Table 2**, the existing matchers are presented and commented.

3.4 Examples of architectures

Matchers can be deployed in series or in parallel. In this section, we present two base configurations to better understand what is possible to achieve by using GENOMA. When the user is defining an architecture, the first decision he has to make is whether to adopt a parallel or a series given two matchers.

In **Fig. 3**, we have an example of two matchers, one SubStringCompare and one StringEquality, in series, while in **Fig. 4** the same two matchers are deployed in parallel.

When two matchers are in series, the resulting similarity matrix of the first matcher is passed as input matrix to the second one. The resulting similarity values, after the second matcher, are the normalized weighted sum of the two separated values (each matcher has a weight in the interval $[0, 1]$). This means that the result is the weighted sum of all the local results of the single matchers in the series.

When the matchers are in parallel, a JunctionPoint is needed immediately after them. This component performs the aggregation of each result by using the algorithm

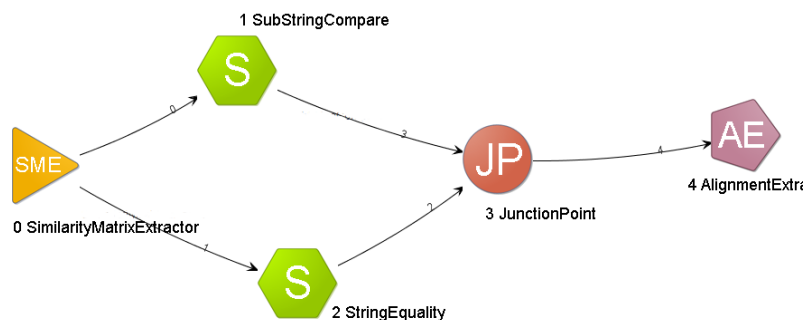


Fig. 4. Matchers in parallel

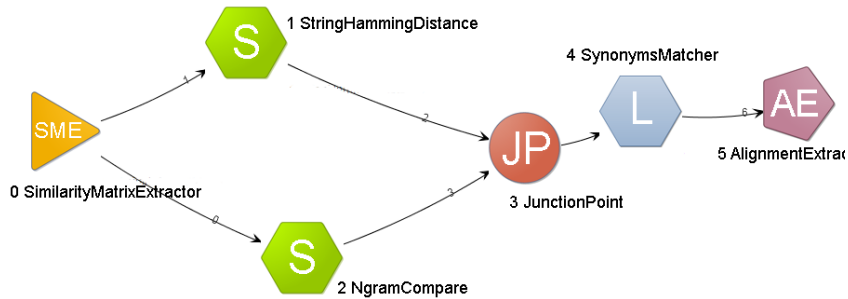


Fig. 5. Parallel and series example

selected during the configuration phase.

The easy to learn and complete user interface enables the construction of more complex architectures using the same approach: just by combining several matchers in series and in parallel. For example in **Fig. 5** we have an architecture composed of a parallel execution of two matchers (a StringHammingDistance and a NgramCompare), which are then deployed in series with another matcher (a SynonymsMatcher). Even more complex architectures, can be deployed very easily just by selecting matchers of interest.

4 Conclusion

In this article, we have described GENOMA, an architecture that helps developing, deploying and validating complex and totally customizable O.M. architecture and tools. These architectures can be changed at any given time, to find the better one given a specific domain or application needs, leaving the user the complete freedom of experimenting while defining its architecture.

Several implemented and tested matchers are provided with GENOMA, to give the possibility to immediately test it. Since it is open source and well documented, implementing new matchers, to extend its capability, is possible.

Regarding the deployment of the produced O.M. architecture, it can be on the same machine or distributed among several servers to obtain better performance by sharing the complexity required by this task.

Thanks to all these features, using GENOMA enables every user (either expert or novice) to rapidly define an O.M. architecture, to test it and possibly extends the modules functionalities by developing new matchers.

Acknowledgements. This research has been partially founded by the european project SemaGrow

References

1. Euzenat, J., Shvaiko, P.: *Ontology Matching*. Springer (2007)
2. Euzenat, J., Shvaiko, P.: *Ontology Matching*. Springer (2013)
3. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with Cupid. In : *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, CA, USA, pp.48-58 (2001)
4. Do, H.-H., Rahm, E.: COMA, a system for flexible combination of schema matching approaches. In : *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, pp.610-621 (2002)
5. Aumüller, D.a.D.H.-H.a.M.S.a.R.E.: Schema and Ontology Matching with COMA++. In : *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, pp.906-908 (2005)
6. Dhamankar, R., Lee, Y., Doan, A., Halevy, A., Domingos, P.: iMAP: discovering complex semantic matches between database schemas. *23rd International Conference on Management of Data (SIGMOD)*, 383-394 (2004)
7. Russell, S.J., Norvig, P.: *Artificial Intelligence: a Modern Approach*. Pearson Education, Englewood Cliffs (2003)
8. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to Map Between Ontologies on the Semantic Web. In : *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, USA, pp.662-673 (2002)
9. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Ontology matching: a machine learning approach. In : *Handbook on Ontologies*. Springer, Berlin (2004) 173-235
10. Hu, W., Qu, Y.: Falcon-AO: A practical ontology matching system. *Journal of Web Semantics*, 237-239 (September 2008)
11. Guha, S., Rastogi, R., Shim, K.: ROCK: a robust clustering algorithm for categorical attributes. In : *Proceedings of the 15th International Conference on Data Engineering*, Washington, DC, USA, pp.512-521 (1999)
12. Jean-Mary, Y.R., Shironoshita, E.P., Kabuka, M.R.: Ontology matching with semantic verification. *Journal of Web Semantics*, 235-251 (2009)
13. Paziienza, M.T., Stellato, A., Turbati, A.: Linguistic Watermark 3.0: an RDF framework and a software library for bridging language and ontologies in the Semantic Web. In : *Semantic Web Applications and Perspectives*, 5th Italian Semantic Web Workshop (SWAP2008), FAO-UN, Rome, Italy (2008) 15-17 December.