YAM++ – Results for OAEI 2011[⋆]

DuyHoa Ngo, Zohra Bellahsene, Remi Coletta

University Montpellier 2, France
{firstname.lastname@lirmm.fr}

Abstract. The YAM++ system is a self configuration, flexible and extensible ontology matching system. The key idea behind YAM++ system is based on machine learning and similarity flooding approaches. In this paper, we briefly present the YAM++ and its results on Benchmark and Conference tracks on OAEI 2011 campaign.

1 Presentation of the system

Ontology matching is needed in many application domains, especially in the emergent semantic web field. It is considered as the most promising approach to solve the interoperability problems across heterogeneous data sources. Due to the various types of heterogeneity of ontologies, a matching system, generally, exploits many features of elements in ontology and combine several similarity metrics in order to improve its performance. However, finding a good combination is very difficult and time consuming even for experts. Therefore, we propose YAM++ - a (not) Yet Another Matcher approach, which firstly uses a machine learning technique to combine similarity metrics. We then apply a similarity propagation algorithm [5] to discover more semantic mappings. The advantages of using machine learning and similarity flooding techniques will make our system flexible and extensible.

1.1 State, purpose, general statement

The current implemented version YAM++ is an extension of our former schema matching system YAM [1]. However, the YAM++ aims to work with ontology matching, which is semantically richer than XML schema, so new features were added in the extension version. The main components of the new system are depicted in Figure 1.

The fully automated mapping process is performed as follows. Firstly, two to-be-matched ontologies are passed to the Parsing & Processing module. We use OWLAPI [1] and Pellet [2] to load ontology files into our specified data structure which contains all elements with their annotation and relationships in ontology. Next, the loaded ontologies are passed into Element-level matcher. The aim of this module is discovering as many as possible mappings and as high as possible the accuracy of these mappings by analyzing elements in isolation and ignoring their relations with others. This module consists of

[⋆] Supported by ANR DataRing ANR-08-VERSO-007-04.
[1] http://owlapi.sourceforge.net/
[2] http://clarkparsia.com/pellet

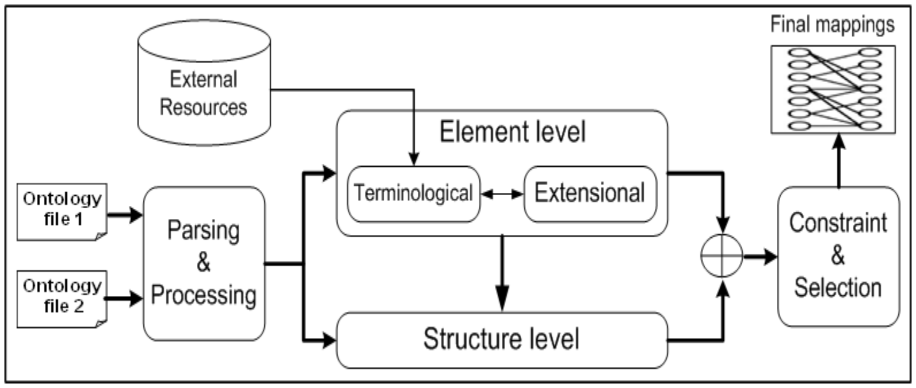

Fig. 1. YAM++ system architecture

two sub-modules: Terminological matcher and Extensional matcher. The Terminological matcher exploits annotation information of elements by various similarity metrics, and then combine them by a machine learning model. Whereas, the Extensional matcher exploits information from external data (instances) accompanying with ontologies.

Mappings discovered from Element-level matcher are passed into the Structure-level matcher module in order to discover new mappings by analyzing the positions of elements on the hierarchy structure of ontologies. The main intuition of this part is that if two elements from two ontologies are similar, their neighbors (in the same relations) might also be somehow similar [2]. In our current system, we have implemented a popular graph based technique - Similarity Flooding in the Structure-level matcher module.

Finally, the results of Element-level and Structure-level matchers are combined and passed to the Constraint & Selection module. Here, we define some semantic patterns to recognize and remove inconsistent mappings. Then, we apply assignment methods (Greedy and Hungarian algorithms) [4] to select the most appropriate mappings. Currently, our system works with 1:1 matching cardinality.

1.2 Specific techniques used

In this section, we will briefly describe four main modules: Terminological matcher, Extensional matcher, Similarity Flooding and Constraint & Selection.

Terminological matcher In this module, we exploit various features of text information used to annotate elements in ontologies. The corresponding similarity metrics have been also implemented in order to calculate the similarity score between elements from two to-be-matched ontologies. The combination and the matching process in this module are illustrated in Figure 2.

We treat the matching task as a binary classification problem. Each pair of elements from two input ontologies are considered as a machine learning object; its attributes are similarity scores calculated by a set of selected similarity metrics on these elements.

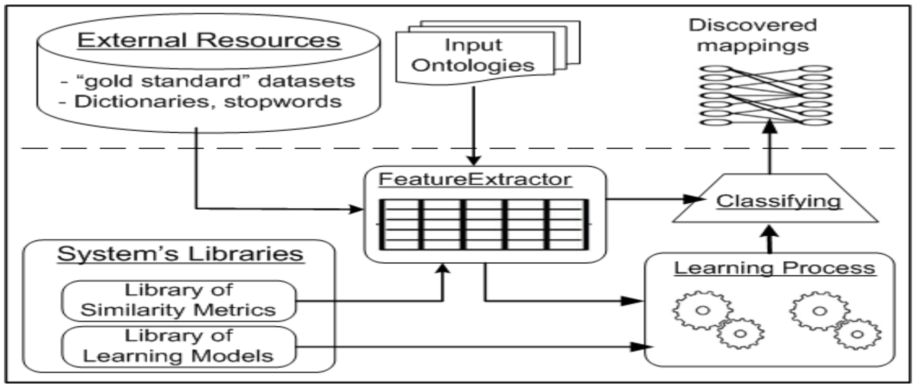

Fig. 2. Terminological matcher

In the learning phase, we use some **gold standard** datasets to generate training data. A **gold standard** is a pairs of ontologies with expert mappings between their elements provided by domain experts. In current version, the gold standard datasets are taken from Benchmark dataset published in OAEI 2009. According to our study [6] [3] , we adopt J48 - a decision tree [9] as the classification model in our system. In classifying phase, each pair of elements from two to-be-matched ontologies is predicted as matched or not according to its attributes.

In order to extract attribute values for each pair of elements, we use various similarity metrics described in [6]. They are divided into three main groups: (i) **name metrics** - which compare entities by their URI names; (ii) **label metrics** - which compare entities by their labels; and (iii) **context metrics** - which compare entities by their descriptive and context information.

Name and label metrics belong to Terminological category [2]. Most of String-based metrics (Levenstein, SmithWaterman, Jaro, JaroWinkler, MongeEklan,etc.) are taken from open source libraries SecondString [4] and SimMetrics [5]. We also have implemented popular metrics such as Prefix, Suffix, LCS [2], SubString - Stoilos [8]. In term of Language-based metrics, we developed three well-known metrics corresponding to Lin, JingConrath and WuPalmer algorithms [3]. These metrics use Wordnet [6] as an auxiliary local thesaurus. Additionally, we proposed a hybrid approach to combine string-based and language-based metrics. The detail of algorithms can be found in our paper [7].

Context metrics used in YAM++ belong to the both Structural and Extensional categories [2]. The main idea behind these metrics is described as follows. We build three types of context profile (i.e. IndividualProfile, SemanticProfile and ExternalProfile) for each element. Whereas IndividualProfile describes annotation information of individual element, SemanticProfile and ExternalProfile describe the context information of an

[3] http://www2.lirmm.fr/∼dngo/publications/A_Flexible_System_for_Ontology_Matching_full.pdf
[4] http://secondstring.sourceforge.net/
[5] http://sourceforge.net/projects/simmetrics
[6] http://wordnet.princeton.edu

element among the others. These context profiles then can be used to compare entities by some information retrieval techniques. See [7] for more detail.

Extensional matcher In many situation, to-be-matched ontologies provide data (instances), therefore, the aim of Extensional module is to discover new mappings which are complement to the result obtained from Terminological module. There are two issues we should consider here: (i) how to find similar instances from two ontologies; and (ii) given a list of matched instances, how to discover new concept/property mappings.

For the first issue, we propose two methods for discovering instance mappings as follows:

– If two instances belong to two matched classes and they have highly similar labels then they will be considered as similar. Here, the list of concept mappings is taken from the result of Terminological module. Similarity score between instance labels is computed by our metric named SentenceSimilarity.
– If two instances have high similar text in description then they will be considered as similar too. Here, a description text is taken from values of all properties described in an instance. The intuition behind is that even though a real instance were assigned by different name of IDs, properties and concepts (classes), but the values of its properties are generally not changed. Based on this idea, we use an information retrieval technique to compute the similarity of instances by their description text.

For the second issue, we apply two methods for discovering concept/property mappings as follows:

– For each pair of matched instances from two ontologies, if the text values of two properties are highly similar, then these properties are considered as similar.
– For each pair of concepts (classes) from two ontologies, if most of their instances are matched, then these classes are matched.

The discovered mappings by Element-level module are the union of discovered mappings from Terminological and Extensional modules.

SimilarityFlooding The Similarity Flooding algorithm is well-known and were described in detail in the popular schema matching system - SimilarityFlooding [5]. In this section, instead of explaining this algorithm again, we will present some main points of using it in our system. For this aim, we are going to answer two questions: (i) why do we select Similarity Flooding method; and (ii) how can we apply this method in ontology matching task.

Firstly, according to [2], there are many methods can be used to analyze the structural information of ontologies to find mappings. They are mainly based on the idea that two elements of two different ontologies are similar if all or most of their elements in the same **view** are already similar. The term **view** here maybe understood as a list of elements which can be seen from the ongoing examined element. For example, they can be direct super/sub elements, sibling elements or list of elements in the path from the root to the current element on the ontology's hierarchy, etc. However, two big problems have arisen. One as the authors commented in [2] is that these methods face problems when the viewpoint of two ontologies is highly different. The second is that if some pre-defined mappings are incorrect and these methods are run only one time to produce new mappings, then the accuracy of new results will be unconfident. Therefore, we believe that the idea of propagating a portion of similarity from one matched pair of elements to their neighbors is more flexible and applicable. Additionally, by running many iterations until the similarity scores between elements become stable, we believe that incorrect mappings (pre-defined) will be removed.

In order to apply Similarity Flooding in our system, we construct a graph from an ontology, whose nodes are concepts, properties or primitive XML Schema DataTypes and edges have types and label have been divided into 5 groups corresponding to relationships between elements in ontology, such as **subClass**, **subProperty**, **domain**, **range** and **onRestrictedProperty**. Notice that we use Pellet library as a reasoner to infer the relations between elements in ontology. It will help us to overcome the problem of different viewpoint of ontologies. Then, a pairwise connectivity graph is built in the same way that were described in the SimilarityFlooding system. We use **inverse product** formula to computing the propagation coefficient and **fixpoint formula C** [5] in the similarity propagation process.

Constraint and Selection The mappings discovered from Element-level and Structure-level are joined and passed to this module in order to remove inconsistent mappings. We combine results of Element and Structure level by a **dynamic weighted aggregation** method. The idea is as follows:

– After Similarity Flooding stop, we run Double-Hungarian assignment method in order to find the two most similar elements from the target ontology for each element in the source ontology. We call the result of this step by SMap - a list of possible mappings.
– From the list mappings obtained from Element-level matcher (we call it EMap), we find a mapping which also exists in the SMap but with the lowest score. We call it a $Threshold$ value. This threshold will be used as weight for all mappings in EMap. The weight for all mappings in the SMap is equal to $(1 - Threshold)$.
– The final mappings are produced by weighted aggregation of EMap and SMap. After that, we use Greedy Selection algorithm to extract the most stable mappings whose similarity score higher than the Threshold found above.

Next, in order to perform Semantic verification, in the current version of our system, we define two simple patterns to recognize the inconsistent mappings as follows:

– If two properties are discovered as similar but none of the classes in their domain are mapped and none of the classes in their range are mapped neither, then the mapping of these two properties are inconsistent and will be removed.
– If classes $(A, B), (A, B_1), (A_1, B)$ are matched and A_1 and B_1 are subsumed of A and B respectively, then $(A, B_1), (A_1, B)$ are removed.

1.3 Adaptations made for the evaluation

There are two factors that directly impact to the our system's performance. The first relates to matching by machine learning model. The training data and selected similarity metrics as learning attributes are important. We proposed a simple solution for this issue by selecting the most appropriate similarity metrics and training data according to their correlation with experts assessment. For more detail, see our paper [7]. The second issue relates to the threshold used as a filter in the selection module. Different tests require different thresholds. Our experiments show that if we set low threshold, we need to use strict semantic constraint to remove inconsistent mappings. In the current version of system, we propose a simple heuristic method (i.e., dynamic threshold and dynamic weighted aggregation) to deal with this problem. Besides, it satisfies the rules of the contests that all alignments should be run by the same configuration.

1.4 Link to the system and parameters file

The YAM++ system and parameter files can be download at:http://www2.lirmm.fr /∼dngo/ YAMplusplus.zip. See the instructions from SEALS platform [7] to test our system.

1.5 Link to the set of provided alignments (in align format)

The results on **Benchmark** and **Conference** tracks can be downloaded at: http://www2.lirmm.fr /∼dngo/ YAMplusplus_oaei2011.zip.

2 Results

In order to see the performance of our system, we will show the effectiveness and the impact of the listed modules above by comparing the results obtained from running system with three configurations. The basic configuration is the Terminological matcher only, which is based on a machine learning (**ML**) approach. Next, we extend it with the Extensional module, which is an Instance based matcher (**IB**), in order to have a full Element-level matcher. Finally, we add the Similarity Flooding (**SF**) module to the third configuration. All experiments are executed with JRE 6.0 on Intel 3.0 Pentium, 3Gb of RAM, Window XP SP3.

2.1 Benchmark

The Benchmark 2010 track includes 111 tests. Each test consists of source (reference) ontology and a test ontology, which is created by altering some information from the reference. The reference ontology contains 33 named classes, 24 object properties, 40 data properties, 56 named individuals and 20 anonymous individuals. The evaluation of our system's performance on this track with 3 configurations is shown in the Table 1.

The observation from this track is as follows. By using only machine learning (**ML**) approach, YAM++ achieved good result with very high precision (**0.99**) and F-Measure

[7] http://oaei.ontologymatching.org/2011/seals-eval.html

Configuration	**Precision**	**Recall**	**F-Measure**
ML	0.99	0.72	0.84
ML + IB	0.99	0.74	0.85
ML + IB + SF	0.98	0.84	0.90

Table 1. H-mean values on Benchmark 2010 track

(**0.84**). After adding Instance based (**IB**) matcher, both Recall and F-Measure increased with **2%** and **1%** respectively. These improvements were happened because many ontologies have common extensional data (instances). Finally, thanks to the process of propagation of similarity, both Recall and F-Measure increased with **10%** and **5%** respectively.

The Benchmark 2011 track includes 103 tests. Similar to Benchmark 2010, in this track, a source (original) ontology is compared to target ontologies which were obtained by altering some features from the original. The reference ontology contains 74 named classes, 33 object properties without extensional data (instances). The evaluation of our system's performance on this track with 3 configurations is shown in the Table 2.

Configuration	**Precision**	**Recall**	**F-Measure**
ML	0.98	0.51	0.67
ML + IB	0.98	0.51	0.67
ML + IB + SF	0.97	0.60	0.74

Table 2. H-mean values on Benchmark 2011 track

Similar to the Benchmark 2010 track, using Similarity Flooding method increases both Recall and F-Measure values with **9%** and **7%** respectively. The Instance based matcher did not improve the performance because in this track, ontologies don't support extensional data. That is why the matching results of running the first and the second configurations are the same.

2.2 Conferences

Conference track now contains 16 ontologies from the same domain (conference organization) and each ontology must be matched against every other ontology. YAM++ is able to produce all 120 alignments from those ontologies. Due to the heterogeneity of those ontologies, finding mappings between them is more difficult than that in Benchmark tracks. Besides, this track is an open+blind test because there are no reference alignments for most of those tests. In the Table 3, we can only report our results with respect to the available reference alignments. The observation on this track is similar to the Benchmark2011 track. Here, thanks to Similarity Flooding method, all of Precision, Recall and F-Measure values are improved.

3 General comments

This is the first time we participate to the OAEI campaign. We found that SEALS platform is a very valuable tool to compare the performance of our system with the others.

Configuration	**Precision**	**Recall**	**F-Measure**
ML	0.75	0.50	0.60
ML + IB	0.75	0.50	0.60
ML + IB + SF	0.78	0.56	0.65

Table 3. H-mean values on Conference 2011 track

Besides, we also found that OAEI tracks covers a wide range of heterogeneity in ontology matching task. They are are very useful to help developers/researchers to develop their semantic matching system.

3.1 Comments on the results

Generally, according to the results published in last competition OAEI 2010, our system is acceptable and comparable with other participants. However, there are still some weaknesses in our current system such as (i) semantic verification/constraint issue; (ii) problem of dealing with large scale ontology matching task and time performance. In the current version, we provided some simple solutions for these issues above. The preliminary results were quite good to encourage us to continue seeking better solutions.

4 Conclusion

In this paper, we present the overview of the YAM++ system and our results on Benchmark and Conference tracks. Our experiments prove that the combination of machine learning and similarity flooding approaches bring good results. Additionally, YAM++ system is fully automate, flexible and extensible.

References

1. Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. Yam: a schema matcher factory. In *CIKM Conference*, pages 2079–2080, 2009.
2. Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
3. Feiyu Lin and Kurt Sandkuhl. A survey of exploiting wordnet in ontology matching. In *IFIP AI*, pages 341–350, 2008.
4. C. Meilicke and H. Stuckenschmidt. Analyzing mapping extraction approaches. In *In Proc. of the ISWC 2007 Workshop on Ontology Matching, Busan, Korea*, 2007.
5. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.
6. DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. A flexible system for ontology matching. In *Caise 2011 Forum*, 2011.
7. DuyHoa Ngo, Zohra Bellasene, and Remi Coletta. A generic approach for combining linguistic and context profile metrics in ontology matching. In *ODBASE Conference*, 2011.
8. Giorgos Stoilos, Giorgos B. Stamou, and Stefanos D. Kollias. A string metric for ontology alignment. In *ISWC Conference*, pages 624–637, 2005.
9. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.