AUTOMS-F: A Java Framework for Synthesizing Ontology Mapping Methods

Alexandros G. Valarakos
(AI Lab, Inf. & Comm. Systems Engineering Dept., University of the Aegean, Samos, Greece
alexv@aegean.gr)

Vassilis Spiliopoulos
(AI Lab, Inf. & Comm. Systems Engineering Dept., University of the Aegean, Samos, Greece
and Inst. of Informatics & Telecomm., NCSR "Demokritos", Greece
vspiliop@aegean.gr)

Konstantinos Kotis
(AI Lab, Inf. & Comm. Systems Engineering Dept., University of the Aegean, Samos, Greece
kotis@aegean.gr)

George Vouros
(AI Lab, Inf. & Comm. Systems Engineering Dept., University of the Aegean, Samos, Greece
georgev@aegean.gr)

Abstract: Although ontologies promise an effective technology for information integration, it is often the case that two or more information providers do not share the same ontology. Several (semi)-automated ontology mapping methods have been developed towards solving this problem. This paper presents AUTOMS-F, a framework implemented as a Java API, which aims to facilitate the rapid development of tools for automatic mapping of ontologies by synthesizing several individual ontology mapping methods. Towards this goal, AUTOMS-F provides a highly extensible and customizable application programming interface. AUTOMS is a case study ontology mapping tool that has been implemented using the AUTOMS-F framework, and has been successfully evaluated in the international OAEI 2006 contest.

1 Introduction

Ontologies have been realized as the key technology to shaping and exploiting information for the effective management of knowledge and for the evolution of the Semantic Web and its applications. In such a distributed setting, ontologies establish a common vocabulary for community members to interlink, combine, and communicate knowledge shaped through practice and interaction, binding the knowledge processes of creating, importing, capturing, retrieving, and using knowledge. However, it seems that there will always be more than one ontology even for the same domain. In such a setting, where different conceptualizations of the same domain exist, information services must effectively answer queries, bridging the gaps between conceptualizations of the same domain. Towards this target, networks of semantically related information must be created at-request. Therefore mapping of ontologies is a

major challenge for bridging the gaps between agents (software and human) with different conceptualizations.

"Simple" cases of heterogeneity include ontologies that use different lexicalizations of the same ontology element (e.g. "car" and "road-vehicle"). More complicated situations appear in cases where ontologies are structured (in terms of concepts' relations) in completely different ways.

Information integration and effective management of information will be admittedly achieved through reaching an agreement, by producing a single, commonly-agreed and shared reference ontology, or by achieving coordination so that each party uses its own ontology, but with it also establishes concept and relation mappings with other ontologies. In any case, tools for supporting the ontology mapping task are of paramount importance. Specifically, given two ontologies O_1 and O_2, establishing a mapping between them involves computing pairs of ontology elements (one from O_1 and one from O_2) that have the same intended meaning.

AUTOMS-F (**AUT**omated **O**ntology **M**apping through **S**ynthesis - **F**ramework) is a Java application programming interface (API) that aims to facilitate the development of integrated tools for automatic one-to-one mapping of domain ontologies. The main concern of AUTOMS-F is the provision of facilities for the synthesis of several ontology mapping methods. The ultimate goal is to provide synthesized approaches realized as integrated tools that produce better results and performance measures than each of the synthesized individual mapping methods alone. The framework has been used for the implementation of the AUTOMS mapping method [Kotis, 06] which is described as a case study in the fourth section of this article.

The paper is structured as follows: Section 2 presents the ontologies mapping problem, the requirements and the assumptions made towards implementing AUTOMS-F. Section 3 describes AUTOMS-F in detail. Section 4 presents AUTOMS, a specific mapping tool implemented using AUTOMS-F, as a case study of using the proposed framework. Section 5 presents related work, and section 6 concludes the paper, sketching our future plans.

2 Problem Statement

A mapping between two ontologies is mainly expressed by a one-to-one function between ontology elements (i.e. the concepts and the properties of ontologies). Therefore, establishing a mapping between ontology elements [INTEROP, 04], involves the computation of pairs of elements whose meaning is assessed to be similar. Similarity in meaning can be computed using a number of metrics that exploit ontology elements' features. It is important to note that the *mapping process* does not modify the involved ontologies: It produces, as output, a set of *mapping pairs* together with their computed similarity measure.

The majority of the mapping methods/tools can be described by the generic mapping process [Ehrig, 04] depicted in Figure 1. The discrete steps of this process are as follows:

1. *Feature Engineering*: Ontologies are transformed into an internal representation. This step selects a fragment of the ontology to be processed.
2. *Search Step Selection*: Element pairs from the two input ontologies are being selected, with the one element belonging to the first ontology and the other

to the second. Depending on the mapping method, all element pairs or only a subset of them may be considered. The set of pairs constitute the search space of the method.

3. *Similarity Computation*: This step computes the similarity of the previously selected pairs. Many different similarity metrics may be utilized by a single method.
4. *Similarity Aggregation*: In this step all similarity metrics are aggregated into a single one.
5. *Interpretation*: This step concludes to a set of mapping pairs by exploiting the aggregated similarities computed in the previous step (e.g. this can be done using threshold value(s)).
6. *Iteration*: The whole process may be repeated several times, by propagating and updating the assessed similarities, taking into account the structure of the input ontologies.
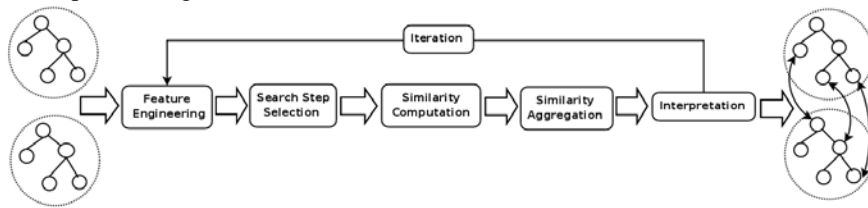

Figure 1: The generic and commonly accepted discrete steps of a mapping process.

We conjecture that any framework that aims to facilitate the development of ontology mapping tools must support the development of the generic steps shown in Figure 1. AUTOMS-F, aiming to the provision of a generic framework for the development of mapping tools, in accordance to the steps proposed, poses a number of requirements:

1. According to the *Feature Engineering* step, a mapping method may utilize only a subset of the available information provided by the input ontologies. Different mapping methods should be able to use different sets of features.
2. According to the *Search Step Selection* step, a method may examine only a subset of the possible mapping pairs, while different methods should be able to select different subsets of pairs, using well-defined conditions.
3. According to the *Similarity Computation* step, different mapping methods may need to compute different similarity measures for the assessment of mapping pairs.
4. According to the *Similarity Aggregation* step, the synthesis of mapping methods and the aggregation of their corresponding similarity measures should be robust, expandable and be easily supported by the mapping framework.
5. According to the *Interpretation* step, the mapping pairs may be produced based on the aggregated similarity values of custom mapping methods.

3 AUTOMS-F: Architecture and Implementation

AUTOMS-F is an open-source application programming interface (API) implemented in Java which aims to provide a basic framework towards synthesizing ontology mapping methods.

3.1 Framework's Conceptualization

The main concept of AUTOMS-F is the *mapping method*. A *mapping method* exploits all the information concerning the mapping process: The pair of ontologies involved and functional information regarding the execution of the mapping steps specified in section 2. This is done by linking the mapping method to:

a) one or more *mapping methods*. When a mapping method is associated with at least another mapping method or another association of mapping methods, then this association constitutes a "task" (or synthesized mapping method). A task specifies the synthesis of different (atomic or synthesized) methods. Tasks, due to their recursive definition specify a hierarchical tree of arbitrary complexity, which is named the "*mapping association tree*".

b) a *parser*. This is responsible for collecting the appropriate elements of the ontologies that are involved in the mapping process. This collection defines the potential mapping pairs and results in an ($n \times m$) similarity matrix, where n and m are the number of elements of the target and source ontology, respectively. The value of each matrix entry specifies the similarity of the specific pair to which the entry corresponds. A parser is associated to a task and is applied to its subsequent methods. Moreover, a parser is inherited to subsequent tasks (i.e. tasks lower in the hierarchy). In order to support parser inheritance property, we assume that tasks in the mapping association tree use parsers that collect pairs of ontological elements that are super-sets of the one collected by subsequent tasks' parsers.

c) a *similarity method.* Such a method specifies the way similarities between pairs of ontological elements are being computed. Such method results to an ($n \times m$) similarity matrix.

d) an *aggregation operator*. This is responsible for specifying the way similarity matrices are being combined. Aggregation operators, as well as results, are associated with the task that combines the specific methods.

e) a *pairs filter*. This defines the criteria for selecting the best mapping pairs from the resulted similarity matrix.

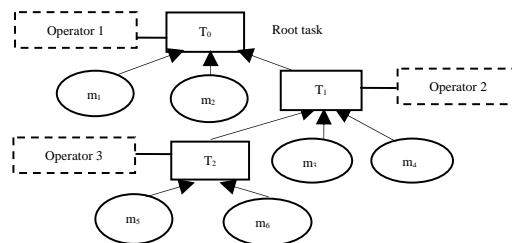f) a *results renderer*, which is responsible for the presentation of the *mapping pairs*.

Figure 2: An example of a mapping association tree.

According to the above, the synthesis of mapping methods in AUTOMS-F is supported in two ways: a) by allowing a mapping method to have direct access to the similarity matrix computed by another method, and b) by combing the similarity matrices of mapping methods using specific aggregation operators. Figure 2 depicts an example of a mapping association tree in which aggregation operators are shown with dashed line rectangulars: They are attached to tasks (rectangulars), and each task aggregates one or more methods (shown in ovals).

The "mapping association tree" specifies also the execution order of the mapping methods and the application of the operators and parsers. The top-most task in the tree is named the "root task". This is the T_0 task in Figure 2. We assume a right-to-left bottom-up execution order of the methods in a tree. Hence, according to Figure 2, the method m_5 follows the execution of method "m_6". "m_4" executes and may exploit the aggregated results that are produced by the methods "m_6" and "m_5". The aggregation of "m_6" and "m_5" methods is being preformed by an aggregation operator associated to task T_2.

Parsers are associated to *tasks* and apply to tasks' subsequent methods. However, different *parsers* can be defined at any level of the tree. Because of that, different methods may exploit different collections of element pairs: Generally, the similarity matrix of a method "contributes" to the computation of the similarity matrix of the root task, which always contains the super-set collection of ontological element pairs.

The root task comprises a *pair filter* that selects the pairs that specify "best mappings" (e.g. whose similarities are above a specific threshold value). Finally, the renderer presents the *mapping pairs*.

3.2 Implementation Issues

AUTOMS-F has being developed using the Jena Framework [Jena, 07]. It has been implemented in Java for ensuring platform independency and uses various well-established design patterns [Brandon, 02] for ensuring usability, reuse, extensibility and abstraction.
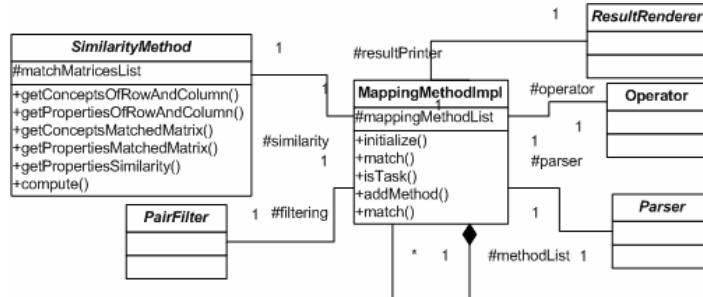

Figure 3: UML diagram of the main AUTOMS-F classes, attributes and operations.

Figure 3 depicts a UML diagram of the main classes of the framework according to its conceptualization (section 3.1). The "MappingMethodImpl" class is linked through an aggregation relation with itself and it aggregates at least one "MappingMethodImpl" class. Also, the same class is linked with exactly one of the following abstract classes:

"SimilarityMethod", PairFilter", "Parser", "Operator" and "ResultRenderer". The "MappingMethodImpl" class stores a list with the methods-tasks to which it is linked using the "mappingMethodList" attribute. This method is responsible for doing the necessary initializations ("initialize" operation) and for performing the mapping operations (the "match" operation of the "MappingMethodImpl" class). The "SimilarityMethod" class supports various manipulations of the similarity matrices to support the synthesis of methods. Due to space restrictions we present only some of the attributes and operations of the system. All the classes, except the "MappingMethodImpl" class, constitute hot spots for the framework, hence are the classes that can be further extended.

The Strategy pattern - behavioral design pattern - is used in the "MappingMethodImpl" class to support the creation of different mapping methods. The template method pattern in the "SimilarityMethod" class - a behavioral pattern - is used for the computation of the similarity of a pair of ontological elements. Thus, instantiating the framework, one can define - override - the methods that measure the similarity between a pair of ontological elements and leave the construction of the similarity matrix to the "SimilarityMethod" class. Also, the composition pattern - a structural pattern - is exploited for the specification of the "mapping association tree". AUTOMS-F, as it exploits Jena's model loader, can handle ontologies that are implemented in RDF, RDFS, OWL and DAML+OIL formalisms. The ontologies can be read from the local disk or be accessed through their URLs. An ontology element can be any of Jena's ontology class (OntClass) or property (OntProperty) objects. Hence, a method can retrieve any information about an ontology element, i.e. label, super-concepts, class properties etc.

AUTOMS-F contains samples of all the extensible classes resulting in a default mapping method. More advanced mapping methods can be developed by extending the "SimilarityMethod" class and overriding the methods that measure the similarity between ontology elements. Also, someone may integrate a method to the framework by extending the "SimilarityMethod" class and overriding only its "compute" operation, which is responsible for executing the similarity method. This means that the new class computes the mapping and the similarity matrix, which is defined in the extended class. Following the latter case, special attention should be given to the collection of the ontological elements pairs used: This must be consistent with the collection of the T_0 task. To ensure this consistency, we recommend the use of a framework-based defined parser and not the use of any user–specific manipulation of the ontological elements pairs.

4   A Case Study: The AUTOMS tool

AUTOMS-F has been used for the development of the AUTOMS ontology mapping tool which synthesizes 6 mapping methods [Kotis, 06]: The *lexical*, *semantic*, *simple structural*, *properties-based*, *instances-based* and the *iterative structural* method. Figure 4 depicts the association tree of AUTOMS and the position of the mapping methods in it. The lexical and semantic methods are executed first. Then the structural matching method follows by exploiting the results of the previously run methods, whose results have been aggregated by task T_2. Afterwards, AUTOMS executes the properties-based and instances-based mapping methods, and finally, the iterative

structural matching method is being executed by exploiting results from the other methods in its level as well as from the task that aggregates results from lower levels. AUTOMS uses the same parser and aggregation operator in any of its tasks. The parser is defined in the T_0 task and the operator of each task unifies the matrices of constituent methods.
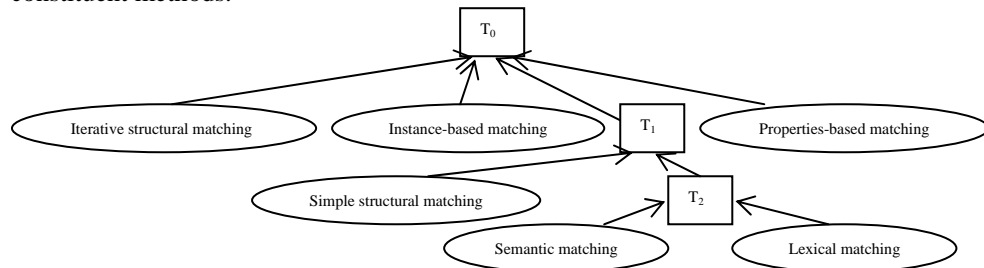

Figure 4: AUTOMS's mapping association tree.

The requirements of AUTOMS have been satisfied by the flexibility and extensibility provided by AUTOMS-F. The learning curve of the framework was rather short. In some cases, AUTOMS developers needed to extend the framework for capturing OAEI contest's requirements [OAEI, 06]; however this did not effect the development of AUTOMS, and AUTOMS-F proved to be a very robust and flexible framework. AUTOMS developers have been provided with an optimized version of AUTOMS-F, resulting to quite short (comparing to other tools of the OAEI contest) execution times. Scalability was a weak point at the time AUTOMS-F was used to develop AUTOMS, since very large ontologies (~30MB) provided by the OAEI organizers could not be loaded and parsed. AUTOMS tool was evaluated in the OAEI 2006 contest among 10 other systems, achieving very high precision and recall measures, providing evidence for the potential of AUTOMS-F to this direction. For a detailed view of contest's results and AUTOMS performance please visit OAEI 2006 results Web page at http://oaei.ontologymatching.org/2006/results/benchmarks/.

5   Related Work

To the best of our knowledge the only work that is related to the presented one is the Alignment API [4]. It is used for the evaluation of the ontology mapping tools that participated in the OAEI contest [5]. It has been implemented using Java and provides an API for incorporating, evaluating and presenting the results of different tools.

AUTOMS-F and the Alignment API are based on different technologies. AUTOMS-F uses the Jena framework whereas the Alignment API uses the OWL API [OWL API, 07]. Moreover, the Alignment API executes mapping methods in a pipe line, in contrast to AUTOMS-F which defines an execution structure of the mapping methods - the "mapping association tree" - facilitating the effective synthesis of different mapping methods as well as their parallel execution by exploiting the thread mechanism. The Alignment API supports the combination of two methods by means of fixed operators i.e. compose, join, inverse and meet which combine the result matrices of the constituent methods. However, these operators have not been fully

implemented as far as the version 2.5 is concerned. On the other hand, using AUTOMS-F one has the flexibility to define her own aggregation operators, combining more than two matrices. Also, AUTOMS-F supports the use of different parsers for collecting ontology elements. Different parsers can be applied in the context of a specific method or task. At the current version, AUTOMS-F does not provide any evaluation utilities, whereas the Alignment API does. In general, AUTOMS-F provides more hot spots than the Alignment API, thus making itself more extensible and customizable. Alignment API is under LGPL license. AUTOMS-F will be available soon in www.icsd.aegean.gr/ai-lab under GPL licence.

6   Concluding Remarks and Future Work

AUTOMS-F addresses the ontology mapping problem by providing advanced facilities for the development of synthesized ontology mapping methods. AUTOMS is an evaluated case of the framework's potential. Although the full automation of ontology mapping is still a challenge, AUTOMS-F provides a robust framework for the synthesis of different mapping methods, increasing the benefits of deploying state of the art mapping technology.

We plan to extent AUTOMS-F in several ways. Firstly, execution threads will be added to the methods of a task at each task level, in order to decrease the execution time. Secondly, we will introduce a consistency checking method that will ensure consistency between the mapping result pairs according to ontologies specification semantics. Thirdly, we will investigate a way to introduce iterative execution in the framework, hence satisfying all the steps of the generic mapping process shown in Figure 1. Finally, we will investigate scalability issues and evaluation utilities.

Acknowledgements

References

[Alignment API, 07] Alignment API, http://alignapi.gforge.inria.fr/

[Brandon, 02] G. Brandon, The Joy of Patterns, Addison-Wesley, ISBN 0-201-65759-7, 2002.

[Ehrig, 04] M. Ehrig, S. Staab, QOM - Quick Ontology Mapping, GI Jahrestagung (1), 2004.

[INTEROP, 04] INTEROP - Network of Excellence, State of the art and state of the practice including initial possible research orientations. Project n. 508011, 2004.

[Jena, 07] Jena Java Framework, http://jena.sourceforge.net/

[Kotis, 06] K. Kotis, et al., AUTOMS: Automating Ontology Mapping through Synthesis of Methods, OAEI 2006 contest, Ontology Matching International Workshop, USA, 2006.

[OAEI,   06]   OAEI: Ontology   Alignment   Evaluation   Initiative. http://oaei.ontologymatching.org/2006/

[OWL API, 07] OWL API, http://owl.man.ac.uk/api/readme.html/