AUTOMS-F: A Framework for the Synthesis of Ontology Mapping Methods

Alexandros G. Valarakos, Vassilis Spiliopoulos, and George A. Vouros

Abstract. Effective information integration is still one of today's emerging research goals. The explosive growth of heterogeneous information sources makes the task harder and more challenging. Although ontologies promise an effective solution towards information management and coordination, it would be a surprise if two independent parties have constructed the same ontology to manage information for the same domain. Hence, to integrate information effectively, ontology mapping methods are invaluable. This paper presents the AUTOMS-F framework, which aims to facilitate the development of synthesized methods for the efficient and effective automatic mapping of ontologies. AUTOMS-F is highly extendable and customizable, providing facilities for supporting the rapid prototyping of synthesized mapping methods, adapting some well established programming design patterns. The paper presents the AUTOMS mapping method as an evaluated case of AUTOMS-F's potential.

1 Introduction

During the last years the world is faced with the information overload phenomenon: Information is growing exponentially, is being provided in various forms and is stored in decentralized systems that range from inter-/intra-organization systems to those operating over the World Wide Web. Meanwhile, the need for transparent and bidirectional communication between these decentralized systems is more vital than ever before, as the exploitation of the available information is required for the right decision at the right time. To effectively deal with information heterogeneity, state-of-the-art approaches utilize ontologies. Ontologies formalize a conceptualization of

Alexandros G. Valarakos, Vassilis Spiliopoulos, and George A. Vouros
AI Lab, Information and Communication Systems Engineering Department,
University of the Aegean, Samos 83 200, Greece
e-mail: alexv,vspiliop,georgev@aegean.gr

S. Schaffert et al. (Eds.): Networked Knowledge - Networked Media, SCI 221, pp. 45–59.
springerlink.com

a certain domain by defining specific elements (concepts and properties) and the relations among them. Ontologies provide the key technology for the fulfilment of the Semantic Web vision, where - in contrast to what is happening today - provided information will not be mainly targeted to humans, but will be machine understandable and exploitable, as well: Innovative Semantic Web applications are expected to be able to deal effectively with the information overload phenomenon and manage available information successfully.

In spite of the fact that ontologies provide a formal and unambiguous representation of domain conceptualizations, it would be a surprise if two independent parties would have constructed the same ontology to manage information even for the same domain. This is true, because ontologies are mainly developed in a decentralized fashion and are freely provided in the World Wide Web for being used in numerous applications. This heterogeneity introduces ambiguity on the appropriateness of information and restrains interoperability between different information sources. Simple examples of ontologies heterogeneity include ontologies which use different lexicalizations for the same ontology elements: For example car and vehicle may denote the same class of entities. More complicated situations appear in cases where ontologies formalize different conceptualizations of the same domain, comprising different elements, and being structured (in terms of ontology elements relations) in different ways.

True interoperability, data integration and effective management of information will be admittedly achieved through reaching an agreement, by producing a single and well-agreed ontology or by coordinating source ontologies so that each party uses its own ontology, but refers to the information of the other party, by exploiting concept and relation mappings between the two ontologies. Ontology Mapping is of increasing importance towards this goal. Specifically, given two ontologies O_1 and O_2, mapping one of them to the other involves computing pairs of elements with highly similar intended meaning.

Towards this goal, state-of-the-art ontology mapping systems exploit synthesized mapping methods, each one targeting different kinds of ontological features, by utilizing different similarity strategies. All these efforts have as common goal the optimum synthesis of individual (atomic) mapping methods, in order to maximize their efficiency. In the context of the Ontology Alignment Evaluation Initiative (OAEI) [5], for instance, all participating systems (especially the best performing ones), heavily focus on the effective and efficient synthesis of individual mapping methods. As a result, the investigation of the optimum synthesis of individual mapping methods is of paramount importance. Therefore, for the proper investigation of the best performing synthesis of atomic methods and for the production of ontology mapping systems that achieve the effectiveness needed in real-world applications, solid, generic, expandable and configurable ontology mapping frameworks must exist, facilitating the development and evaluation of synthesized methods.

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 the automatic mapping of domain ontologies. The main concern of AUTOMS-F is the provision of facilities for the advanced, flexible and rapid synthesis of several ontology mapping methods. As already stated, 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 [3] 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 and Requirements

A mapping between two ontologies is expressed by a one-to-one function between (matching) ontology elements (i.e., ontology concepts and properties). Therefore, establishing a mapping [8] between ontology elements 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 (match) measure.

The majority of the mapping methods can be described by the generic mapping process [9] depicted in Fig. 1. The discrete steps of this process are as follows:

1. *F*eature Engineering: Ontologies are transformed into an internal representation. This step selects a fragment of the ontology to be processed.
2. *S*earch 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. *S*imilarity Computation: This step computes the similarity of the previously selected pairs. Many different similarity metrics may be utilized by a single method.
4. *S*imilarity Aggregation: In this step all similarity metrics, which may exploit different ontological features, are aggregated into a single one.

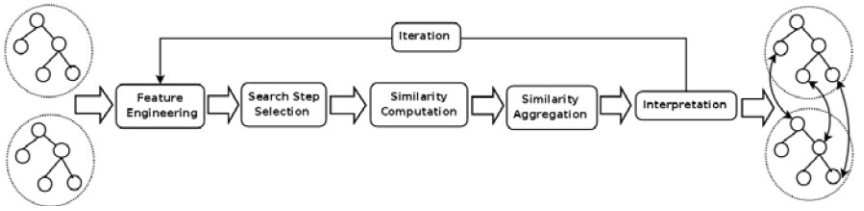

Fig. 1 The commonly accepted discrete steps of the generic mapping process

5. *I*nterpretation: This step concludes to a set of matching pairs by exploiting the aggregated similarities computed in the previous step (e.g., a trivial case is the use of threshold value(s)).
6. *I*teration: The whole process may be repeated several times, by propagating and updating the assessed similarities, taking into account the structure of the input ontologies.

Any framework that aims to facilitate the development of ontology mapping methods must support the development of the generic steps exposed in Fig. 1. AUTOMS-F, aiming to the provision of a generic framework for the development of mapping methods, in accordance to the steps proposed, poses a number of requirements:

1. According to the *F*eature 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. The manipulation of the input ontologies must abstract from their specific representation formalism. Thus, ontologies in various representation formalisms, such as xml dialects, plain texts, rdfs, owl etc., must be handled.
3. According to the *S*earch Step Selection step, a method may examine only a subset of the candidate matching pairs, while different methods should be able to select different subsets of pairs, under well-defined conditions.
4. Moreover, a method may be applied to the candidate matching pairs produced by other methods.
5. According to the *S*imilarity Computation step, different mapping methods may need to compute different similarity measures for the assessment of matching pairs.
6. Also, a mapping method must be able to re-examine the results of other methods, supporting the development of more effective (in terms of correct mappings) mapping methods.
7. According to the *S*imilarity Aggregation step, the synthesis of different mapping methods and the aggregation of their corresponding similarity measures must be robust, expandable and easily supported by the framework.
8. According to the *I*nterpretation step, the matching pairs may be produced based on the aggregated similarity values assessed, and after the

application of a selection policy, aiming at choosing the best matching element pairs of the input ontologies.

Concerning the requirements of the framework's Application Programming Interface (API) the following are required:

1. *S*implicity: The API should be the result of an abstract specification of the ontology mapping process, and should be independent of the particular implementation of the constituent mapping methods and their specific configurations. Moreover, it must support the development of easily configurable and extensible systems, reducing effectively the time and cost of development.
2. *F*lexibility: It must cleanly separate the implementation of the above mentioned distinct steps of the mapping process, resulting in an easily configurable and extensible API, supporting reusability and thus, reducing the development cost and time.

3 AUTOMS-F: Architecture and Implementation

AUTOMS-F is an open source toolkit implemented using the Java programming language. It provides a basic framework for developing customized and synthesized ontology mapping methods. The framework is accessible by a comprehensive API.

In this section, we firstly present the conceptualization of AUTOMS-F, exposing its main components. Then, we present the AUTOMS-F components in accordance to the steps of the generic mapping process presented in section 2. Secondly, we specify key programming issues concerning the implementation of AUTOMS-F, towards the rapid and effective development of synthesized ontology mapping methods.

3.1 Framework's Conceptualization

AUTOMS-F, aiming at the satisfaction of the requirements stated in section 2, is broken into operation-specific component parts. The main types of components defined in AUTOMS-F and which are further detailed in the paragraphs that follow, are: 1) The *M*apping Method, 2) the *M*apping Task, 3) the *M*apping Association Tree, 4) the *P*arser, 5) the *C*oncept Property Selector, 6) the *A*ggregation Operator, 7) the *S*imilarity Method, 8) the *P*air Selector and 9) the *R*esult Renderer. These types of components are sufficient for describing an ontology mapping task according to the presented mapping process. They constitute the backbone of the framework and their specific implementation leads to different specifications of the ontology mapping process. Their manipulation/implementation is achieved through the AUTOMS-F's API, resulting to individual mapping methods.

3.1.1 Mapping Method and Mapping Task

The *m*apping method is the central component of AUTOMS-F. This component aggregates all the necessary information that is exploited in the various steps of the mapping process: a) The elements of the input ontologies selected to participate in the candidate matching pairs, b) the metric used for assessing the similarity between the elements in the candidate matching pair, c) the logic used for combining the results of the various *m*apping methods, resulting in a new set of assessed matching pairs, d) the logic used for selecting valid matching pairs form the resulting ones, and e) the representation format that will be used for visualizing the valid matching pairs.

A *m*apping method can be associated with other *m*apping methods. When a *m*apping method is associated with at least another *m*apping method or another association of *m*apping methods, then this association constitutes a mapping task (or synthesized *m*apping method). A *t*ask specifies the synthesis of different (atomic or synthesized) *m*ethods.

Tasks, due to their recursive definition specify a hierarchical tree of arbitrary complexity, which is named the Mapping Association Tree (MAT). Fig. 2 depicts an example of MAT that consists of 2 *m*apping tasks (T_1 and T_2), each with 2 *m*apping methods (m_1, m_2 and m_4, m_5, respectively), and 2 *m*apping methods (m_3 and m_6) that are siblings to these *t*asks. A *m*apping task is depicted by a rectangular, whereas a *m*apping method is depicted in oval. The specific configuration of a *m*ethod or *t*ask is shown by the corresponding symbols attached to it, e.g., P_1 for parser, etc (these are further explained in the next subsections).

The root *m*apping method is always a *m*apping task (TR) since it is always associated with other *m*ethods. The MAT defines a hierarchical structure that among others specifies the execution order of *m*apping methods. A

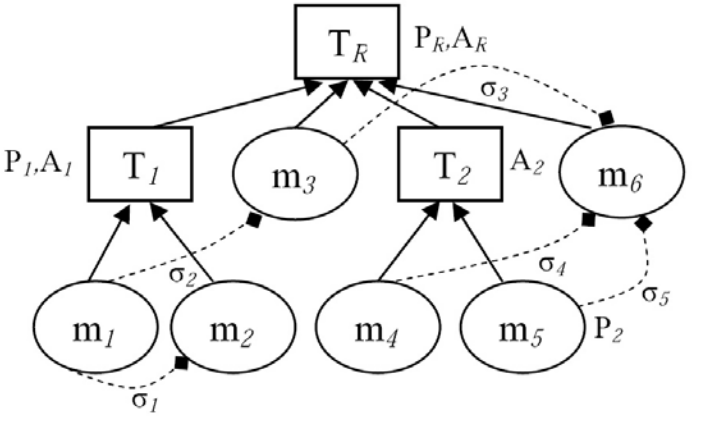

Fig. 2 An example of a Mapping Association Tree: Tasks (T), methods (m), parsers (P), aggregator operators (A) and concept-property selectors (σ)

left-to-right depth-first execution order of the *m*ethods and *t*asks in the *M*AT has been adopted. Hence, according to Fig. 2, the method m_2 follows the execution of method m_1. The execution order of the *m*ethods and *t*asks in *M*AT is: m_1, m_2, T_1, m_3, T_2, m_4, m_5, m_6, TR. Moreover, this hierarchical structure implies inheritance relations that are exploited for usability and performance reasons, as it will be shown in the next subsections.

Similarity measures are not specified in *t*asks, since their role is to combine/manipulate the results produced by the subsequent *m*ethods and *t*asks (those that are rooted by this *t*ask in the *M*AT). The root *t*ask (TR) has a default manipulation *m*ethod which unifies the results produced by its subsequent *m*ethods and *t*asks. This is in contrast to the other *t*asks, which can be associated with different combination/manipulation *m*ethods.

3.1.2 The Parser

The *p*arser is responsible for collecting the candidate matching pairs of ontologies elements involved in the mapping process. This collection is an $(n \times m)$ similarity matrix, where n and m are the number of elements of the target and source ontology, respectively. AUTOMS-F's internal representation distinguishes ontology elements in concepts (C) and properties (P). Candidate matching pairs between concepts and properties of the two input ontologies come from the cartesian product of their respective sets. Hence, the candidate matching pairs of concepts is the $C_1 \times C_2 = (c_{11}, c_{21}), (c_{11}, c_{22}), \ldots, (c_{12}, c_{21}), \ldots, (c_{1n}, c_{2m})$, where C_1 is the set of concepts in the first ontology, and C_2 is the set of concepts in the second ontology. Therefore, c_{1i} and c_{2j} are concepts from the first and second ontology, respectively.

A *p*arser is assigned to a *m*apping task or *m*ethod. According to the *M*AT structure a *p*arser is inherited to subsequent *t*asks (i.e., *t*asks lower in the hierarchy) and *m*ethods (i.e., *m*ethods lower in the hierarchy) that have not been associated to any *p*arser. Supporting the *p*arsers inheritance property, and for consistency preservation reasons, we assume that *t*asks or *m*ethods in the *M*AT use *p*arsers that collect pairs of ontological elements that are supersets of the sets collected by subsequent *t*asks or *m*ethods parsers. Different *p*arsers can be defined at any level of the tree. Because of this, different *m*ethods 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 *t*ask, which always contains the super-set collection of ontological element pairs.

In the *M*AT example (Fig. 2) a *p*arser (PR) has been assigned to the root *t*ask (TR) which is inherited to its subsequent *m*ethods and *t*asks, given that no *p*arser is specified for them. Thus, the *p*arser is inherited to the *m*ethods m_3, m_4 and to the *t*ask T_2, in contrast to the *m*ethods m_1 and m_2 that inherit the *p*arser (P_1) that is assigned to the *t*ask T_2. Finally, *m*ethod m_5 is associated to the *p*arser P_5.

It is possible for a *m*ethod to have two different *p*arsers attached: one for collecting elements of the first input ontology and one for the second. This feature is useful in cases where the two input ontologies are represented in different formalisms: A situation usually appearing in integrating legacy systems (schema oriented databases) with ontology-based applications.

Whenever there is not an one-to-one correspondence between the internal representation of AUTOMS-F (which is an ontology based one: concepts and properties along with all of their features) and the input ontologies/schemata, a transformation method is employed for defining correspondences between the appropriate elements of the ontologies/schemata with the elements in AUTOMS-F internal representation. For, example, the user may explicitly define which xml tag (e.g., tag <description>) of the input ontology/schema corresponds to which ontology element (e.g., <rdfs:comment> element) of the internal representation of the AUTOMS-F. The framework provides the necessary infrastructure for extending and adapting this behavior as needed by the specific needs of the input ontologies/schemata and their implementation.

3.1.3 Similarity Method

A *s*imilarity method is assigned to every *m*apping method and it specifies the way a match between the candidate matching pairs is being computed. Every *s*imilarity method results to an $(n_1 \times m_2)$ similarity matrix, where n_1 and m_2 are the number of the elements (concepts or properties) of the two input ontologies, respectively. For each element a different similarity matrix is produced. The value of each matrix entry specifies the similarity of the specific pair of elements (assessed matching pair) to which the entry corresponds. The candidate matching pairs, to which the similarity method is applied, are produced by the *p*arser of the corresponding *m*apping method or by a *s*elector component (the concept-property selector component is explained in the next subsection) applied to the candidate matching pairs computed by the *p*arser of another mapping method.

Also, since the internal representation of the AUTOMS-F is entirely based on Jena's ontology model, every similarity method has access to a copiousness of features regarding the selected elements of the input ontologies and the ontologies themselves. The set of the available features, which is the minimum and complete set concerning the manipulation of an ontology, is provided by Jena's ontology model. For example, a *s*imilarity method can directly access the local name of a concept and the property names of the concepts that constitute its vicinity.

A sophisticated *s*imilarity assessment method exploits information beyond the one found in the candidate matching pairs. Therefore, a *s*imilarity method has direct access to the involved ontologies. The framework's API supports all available settings of Jena's [1] ontology models, supporting the creation of advanced *s*imilarity methods. In order to facilitate synthesis of *m*apping methods, every *s*imilarity method of a *m*apping method has direct access to a previously-executed *m*apping method's similarity matrix.

3.1.4 Concept-Property Selector

A *c*oncept-property selector is assigned to a *m*apping method for producing candidate matching pairs on which the method's *s*imilarity method will be applied. Since a *c*oncept-property selector and a *p*arser have the same effect (producing candidate matching pairs for the *s*imilarity method), when both of them exist in a *m*apping method the *c*oncept-property selectors override the *p*arser. It must be noticed that, in contrast to a *p*arser, the *c*oncept-property selector of a *m*apping method makes the combination of mapping methods results that do not belong to the same branch in the *M*AT, feasible. This feature makes possible the implementation of rules, such as, in similarity method m_2 exploit as candidate matching pairs only those that have not been assessed as such by the mapping method m_1.

Also, the way matching pairs are being selected by the *c*oncept-property selectors preserve the consistency of the results produced: Indeed, the candidate mapping pair set produced by a *c*oncept-property selector in a *m*ethod mi should be at least a subset of the set produced by their super *m*ethods or *t*asks.

The selection of the candidate matching pairs is based on a similarity matrix of a previously executed *m*apping method and the models of the involved ontologies. In Fig. 2 the dashed lines represent the *c*oncept-property selectors. The square at the one end of the line denotes the *m*ethod to which the *s*elector is assigned, whereas the other end of the line denotes the *m*apping method that provides the similarity matrix. As it is depicted in Fig. 2, σ_1 is assigned to the method m_2 using m_1's similarity matrix, σ_2 is assigned to the method m_3 using m_1's similarity matrix, σ_3 is assigned to the method m_6 using m_3's similarity matrix, and σ_3, σ_4 and σ_5 are assigned to the method m_6 using m_3's, m_4's and m_5's similarity matrices, respectively.

3.1.5 Aggregation Operator

An *a*ggregation operator is assigned to every *m*apping task in a *M*AT and it is responsible for specifying the way similarity matrices of direct subsequent methods or tasks are being combined. Hence, as can be seen in Fig. 2, the aggregator of the *t*ask T_1 combines the similarity matrices of the *m*ethods m_1 and m_2. The *a*ggregation operator of the *t*ask T_2 combines the similarities matrices attached to *m*ethods m_4 and m_5, whereas the *a*ggregation operator of the root *t*ask (TR) combines the similarities matrices of the *m*ethods m_3 and m_6 and the *t*asks T_1 and T_2. The root *t*ask (TR) is being related to a default *a*ggregation operator which selects the best similarity value amongst the values produced by the *m*apping methods and *t*asks in the *M*AT, for every assessed mapping pair. Also, the models of the ontologies involved are accessible by *a*ggregation operators so as to facilitate advanced aggregation techniques and tests.

3.1.6 Pairs Selector

Every *m*apping method and *t*ask is assigned a *p*air selector. A *p*air selector defines the criteria for selecting the best matching pairs from the matching pairs assessed by the *s*imilarity method. For example, a *p*air selector may define that the best matching pair is the one with the highest similarity value (resulting in one-to-one matching pairs) or define that the n% of the candidate matching pairs with the highest similarity value, are the best mapping pairs (resulting in one-to-many mapping pairs). A common strategy found in state-of-the-art mapping systems is the application of the *p*air selector only in the aggregated similarity matrix of the root *t*ask. The selected mapping pairs are passed to the *r*esult renderer component in order to be visualized. Also, the models of the involved ontologies are accessible by this component, enabling the development of advanced selection techniques, beyond the ones based on threshold values.

3.1.7 Result Renderer

A *r*esult renderer is responsible for the presentation of the mapping pairs. Every *m*apping method and *t*ask in the *MAT* is assigned with a *r*esult renderer. This facilitates the separate evaluation of each *m*ethod and *t*ask, leading to better decisions concerning their individual vs. synthesized deployment. Furthermore, this component is responsible for the storage of results.

3.2 Synthesizing Mapping methods

To the best of our knowledge, all the available frameworks adapt a sequential synthesis of mapping processes following the sequential execution order of the mapping processes. This results in a linear synthesis of atomic mapping methods. AUTOMS-F's mapping method adequately represents more complex synthesis patterns, such as the one presented in section 2: In contrast to other existing frameworks, AUTOMS-F facilitates a non-linear synthesis of the mapping methods and tasks, introducing the notion of *M*AT in combination with *s*electors and *a*ggregation operators. According to the above subsections, the synthesis of *m*apping methods in AUTOMS-F is supported in three ways:

1. By allowing a *m*apping method to have direct access to the similarity matrix computed by another method or task,
2. By combing the similarity matrices of *m*apping methods or *t*asks using specific *a*ggregation operators, and
3. By selecting candidate matching pairs from other methods or tasks, by exploiting *c*oncept-property selectors. These pairs are being used as input to the mapping methods.

The first and the third way facilitate a non-linear synthesis of *m*apping methods.

3.3 Implementation Issues

AUTOMS-F has been developed using the Jena Java Framework [1]. It has been implemented in Java for ensuring platform independency. A great concern during its development was the easy extensibility of the framework API, hence well-established programming design patterns [2] for ensuring usability, reuse, extensibility and abstraction were employed.

Fig. 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 these are the classes that can be further extended.

*T*he Strategy pattern - behavioural design pattern - is used in the MappingMethodImpl class to support the creation of different mapping methods. *T*he template method pattern in the SimilarityMethod class - a behavioural pattern - is used for the computation of the similarity of a pair of ontology 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

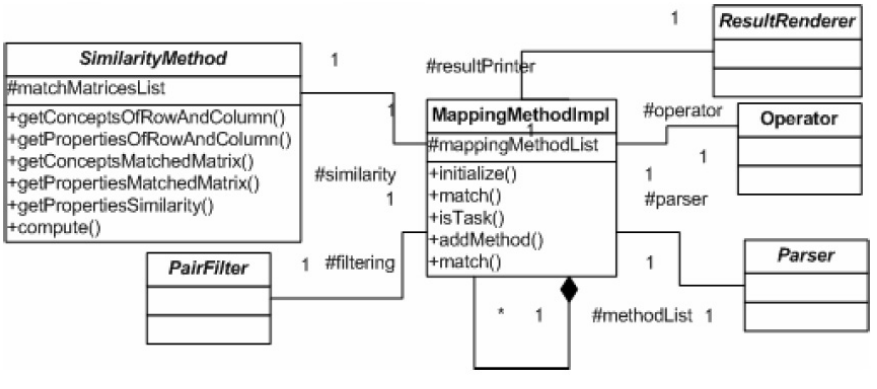

Fig. 3 UML diagram of the main AUTOMS-F classes, attributes and operations

class. Also, the *c*omposition pattern - a structural pattern - is exploited for the specification of the *M*AT.

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 *m*apping method. More advanced *m*apping methods can be developed by extending the SimilarityMethod class and overriding the methods that measure the similarity between ontology elements, i.e., concepts and properties. However, one may integrate a method into the framework by extending the SimilarityMethod class and overriding the compute operation that executes the similarity method. This means that the new class computes the mapping and the similarity matrix defined in the extended class. In this way, special attention is given to the manipulation of the ontology elements pairs, since the manipulation of the candidate matching pairs is left to the specific implementation of the method. Also, for the selection of the ontology elements we recommend the unified use of the framework's-based defined *p*arser: This ensures consistency between the produced candidate matching pairs.

4 A Case Study: The AUTOMS Ontology Mapping Tool

AUTOMS-F has been used for developing the AUTOMS ontology mapping tool. AUTOMS synthesizes 6 mapping methods [3]: The lexical, the semantic, the simple structural, the properties-based, the instances-based and the iterative structural methods. Fig. 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 *m*ethods in its level, as well as from the *t*ask that aggregates results from lower levels. AUTOMS uses the same *p*arser and *a*ggregation operator in any of its *t*asks. The *p*arser is defined in the TR *t*ask and the *a*ggregation operator of each *t*ask selects the best values of each assessed matching pair from the similarity matrices of its constituent *m*ethods and *t*asks.

The requirements of AUTOMS have been satisfied by the flexibility and extensibility provided by the framework. 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 [5]; however this did

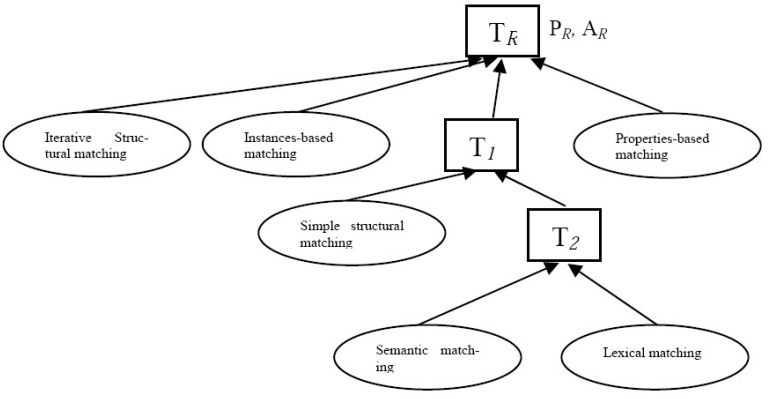

Fig. 4 AUTOMS's *M*apping Association Tree and its particular configurations

not affect the development of AUTOMS and AUTOMS-F proved to be a very robust and flexible framework. AUTOMS-F developers have been provided with an optimized version of their API, resulting in quite short (comparing to other tools of the OAEI contest) AUTOMS 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 was evaluated in the OAEI 2006 contest among 10 other systems, achieving very good results as far as its efficiency and effectiveness are concerned.

5 Related Work

To the extent of our knowledge the works that are related to AUTOMS-F are the following: The Alignment API [4] and the COMA++ system [10, 11]. The Alignment API has been used for the evaluation of the ontology mapping methods that participated in the Ontology Alignment Evaluation Initiative workshop [5]. It has been implemented using Java and provides an API for incorporating, evaluating and presenting the results of different mapping algorithms.

AUTOMS-F and the Alignment API are based on different semantic-web technologies. AUTOMS-F uses the Jena Java framework whereas the Alignment API uses the OWL API [6]. Moreover, the Alignment API executes mapping methods in a pipeline, in contrast to AUTOMS-F which defines an execution structure of the mapping methods - the *M*apping Association Tree - facilitating the effective synthesis of different mapping methods, as well as their parallel execution. The Alignment API supports the combination of two methods by means of the fixed operators [7] 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 his/her own *a*ggregation operators, combining more than two matrices. Also, AUTOMS-F supports the use of different *p*arsers, on each of the involved ontologies, for collecting their elements. Different *p*arsers can be applied in the context of a specific *m*ethod or *t*ask. Furthermore, AUTOMS-F incorporates *s*electors, which are built in components: This makes their exploitation very easy and straightforward. These facilities are not provided by the Alignment API. At the current version, AUTOMS-F does not provide any evaluation utilities, something that 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. The new version of AUTOMS-F will be available soon in www.icsd.aegean.gr/ai-lab under GPL licence.

The COMA++ system, although it provides the necessary interfaces for intergrading arbitrary mapping methods and taking advantage of its matching-pairs visualization features, it is not an extendible API framework. More precisely, it is not possible for the user to define its own *s*imilarity aggregation or *i*nterpretation (pair selection in AUTOMS-F) policies as presented in Fig. 1. For this purpose, a predefined list must be exploited. Finally, in comparison to AUTOMS-F it does not support advanced synthesis utilities such as the *M*apping Association Tree and the notion of *s*electors. To sum up, in contrast to AUTOMS-F and the Alignment API, the main focus of COMA++ implementation is not to provide the infrastructure for facilitating the building of mapping tools, but the development of a mapping tool.

6 Concluding Remarks and Future Work

AUTOMS-F addresses the ontology mapping problem providing advanced methods synthesis facilities. The framework provides solutions to integrating and combining different *m*apping methods that aim to solve the ontology mapping problem. AUTOMS-F successfully meets all the requirements specified in section 2 and smoothly implements all the steps of the generic mapping process (presented also in section 2) except the step concerning the iteration of the mapping process which constitutes future research work. AUTOMS [3] 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 synthesizing different mapping methods, increasing the benefits of deploying state of the art mapping technology.

We plan to extend AUTOMS-F in several ways. Firstly, execution threads will be added to the *m*ethods of a *t*ask at each task level, in order to decrease the execution time of systems that combine many different methods. Secondly, we will introduce a consistency checking method that will ensure consistency between the resulted matching pairs. Thirdly, we will investigate

a way to introduce iterative execution of the *m*apping methods and *t*asks in the framework preserving their synthesis capability, hence satisfying all the steps of the generic mapping process shown in Fig. 1. Fourthly, we will investigate the scalability issue: mapping between large ontologies and last but not least, we plan to add evaluation utilities for appropriate assessing and comparison of the implemented *m*apping methods and *t*asks.

Acknowledgements. This work is part of the 03ED781 research project, implemented within the framework of the Reinforcement Programme of Human Research Manpower (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund). The authors are grateful to K. Kotis for his valuable comments concerning the evaluation of the AUTOMS mapping tool.

References

1. Jena - A Semantic Web Framework for Java, `http://jena.sourceforge.net/` (accessed December 24, 2007)
2. Brandon, G.: The Joy of Patterns: Using Patterns for Enterprise Development. Addison-Wesley Pub.Co., Reading (2001)
3. Kotis, K., Valarakos, A., Vouros, G.: AUTOMS: Automating Ontology Mapping through Synthesis of Methods, OAEI (Ontology Alignment Evaluation Initiative), 2006 contest, Ontology Matching International Workshop, Atlanta USA (2006)
4. Alignment API and Alignment Server, `http://alignapi.gforge.inria.fr/` (accessed December 24, 2007)
5. Ontology Alignment Evaluation Initiative (2006), `http://oaei.ontologymatching.org/2006/newindex.html` (accessed December 24, 2007)
6. Bechhofer, S., Volz, R., Lord, P.: Cooking the semantic web with the OWL API. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 659–675. Springer, Heidelberg (2003)
7. The Alignment API documentation (2006), `http://gforge.inria.fr/docman/view.php/117/251/align.pdf` (accessed August 24, 2007)
8. INTEROP - Network of Excellence, State of the art and state of the practice including initial possible research orientations. Deliverable d8.1, NoE INTEROP, IST Project n. 508011 (2004)
9. Ehrig, M., Staab, S.: QOM - Quick Ontology Mapping. GI Jahrestagung (1), 356–361 (2004)
10. Aumueller, D., Do, H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proceedings of SIGMOD, Demonstration (2005)
11. Do, H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. In: Proceedings of VLDB (2002)