Automatizing the Evaluation of Model Matching Systems

Kelly Garcés, Wolfgang Kling, Frédéric Jouault

Automatizing the Evaluation of Model Matching Systems

Kelly Garces and **Wolfgang Kling** and **Frédéric Jouault** [1]

Abstract. Model-Driven Engineering (MDE) and the Semantic Web are valuable paradigms. This paper sketches how to use a scientific and technical result around MDE in the Semantic Web. The work proposes a mechanism to automatize the evaluation of model matching algorithms. The mechanism involves *megamodeling* and a Domain Specific Language named AML (AtlanMod Matching Language). AML allows to implement matching algorithms in a straightforward way. We present how to adapt the mechanism to the ontology context, for example, to the Ontology Alignment Evaluation Initiative (OAEI).

1 Introduction

Over the last decade, whereas the software engineering community has advanced the Model-Driven Engineering (MDE) paradigm, the Web, AI, and database communities have promoted the Semantic Web. MDE suggests to develop a model of the system under study, which is then transformed into an executable software entity [12]. The Semantic Web, in turn, aims to express Web information in a precise, machine-interpretable form, ready for software agents to process [12].

Even though MDE has foundations and applications different from the Semantic Web's, researches have raised the question: how the scientific and technical results around MDE can be used productively in the Semantic Web, and vice versa? [21] elucidates the potential of ontologies technologies in MDE. Given a classical MDE transformation scenario, [21] proposes further transformations into the ontology working context. The idea is to enable model checking, logics-based model analysis, and reasoning. [12], in turn, details how to use MDE for ontology development on the Semantic Web. The authors suggest MDE to automate the semantic markup of Web resources which has been done more or less manually.

Just as [12], we investigate how MDE can contribute to ontology development. In particular, our work is about matching systems evaluation. The Semantic Web defines *matching* like an operation computing alignments between ontologies. Alignments are the more longstanding solution to the interoperability problem. Because the number of matching systems is rapidly increasing, it is necessary to pursue efforts on an extensive evaluation [9].

To illustrate what a matching system evaluation consists of, let us consider to the Ontology Alignment Evaluation Initiative (OAEI). Every year since 2004, the OAEI develops a campaign that evaluates ontology matching systems presented by participants. Fig. 1 illustrates a classical evaluation. The matching system f takes as input two ontologies (o and o'), an initial alignment (A), parameters (p), and resources (r), and yields as output a new alignment (A'). R is a reference alignment (or *gold standard*), the one that participants must find. The evaluation mechanism calculates matching metrics (M) by comparing R to each A'. The OAEI uses such metrics to figure out strengths and weaknesses of proposed matching systems.

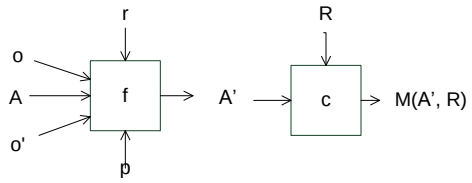


Figure 1. Ontology matching systems evaluation (Adapted from [19])

Each campaign involves three phases: preparation, execution, and evaluation. During the preparation phase, the OAEI defines a set of test cases to compare matching systems and algorithms on the same basis. A given test case includes ontologies (coming from different domains) and their corresponding alignments. In the execution phase, the participants use their algorithms to match all test cases. Finally, the OAEI organizers check the results obtained in the evaluation phase.

Inspired from the OAEI framework, we have developed a mechanism that automatizes the evaluation of model matching systems. Instead of ontologies, our mechanism involves systems that match metamodels (i.e., formalisms to represent models). In addition, the mechanism itself is based on MDE.

In a nutshell, our approach tackles two issues of matching algorithm evaluation.

1. **Not enough test cases**. Whereas it is relatively easy to pairs of metamodels, the availability of reference alignments is restricted. Instead of asking experts for reference alignments, our mechanism automatically extracts them from model transformations.
2. **Low evaluation efficiency.** Evaluation is a time-consuming task. By using a *megamodel* (i.e., a map), our mechanism executes matching algorithms over test cases in an automatic way. Such automatization can reduce the time spent during evaluations, and increase the confidence on results.

We believe that these problems are relevant to ontology matching systems evaluation too. The aim of this paper is to present the whole mechanism and draw out how the Semantic Web community can take advantage of it. Besides megamodels, our approach uses AML programs. AML (AtlanMod Matching Language) is a Domain Specific Language (DSL) to implement matching algorithms. In an experimentation, our mechanism evaluates an AML program over 30 test cases in 40 minutes.

The paper structure is as follows: Section 2 presents an overview of the mechanism. The subsequent three sections describe the mechanism in detail. Each of them briefly outlines how to integrate our

[1] École des Mines de Nantes Email: kelly.garces, wolfgang.kling, frederic.jouault@emn.fr

approach to the ontology context. Section 6 shows and discusses the results of applying the approach. Section 8 concludes the paper.

2 Mechanism Overview

Fig. 2 shows the artifacts manipulated by our mechanism along the evaluation phases. The artifacts are stored in a model repository.

- **Preparation.** The mechanism constitutes a given test case using two metamodels, m and m', and a transformation $tm2m'$ written in terms of them. A reference alignment R is extracted from $tm2m'$ (block e in Fig. 2).
- **Execution.** We develop a matching algorithm f with AML.
- **Evaluation.** The mechanism executes f which computes a candidate alignment A' from m, m', A, r, and p. Finally, the block c compares A' to R and derives matching metrics $M(A', R)$.

A megamodel is a kind of map of a model repository. Our approach uses a megamodel to constitute many test cases, launch a set of matching algorithms, and compute metrics. Next sections describe our mechanism in detail.

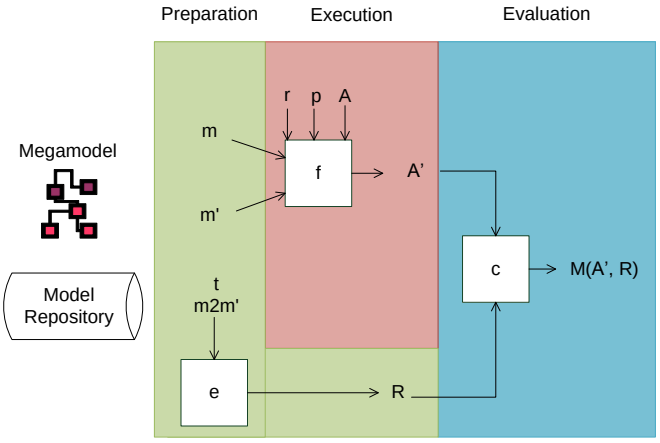


Figure 2. A mechanism to automatize matching system evaluation

3 Preparation: Getting Test cases from Model repositories

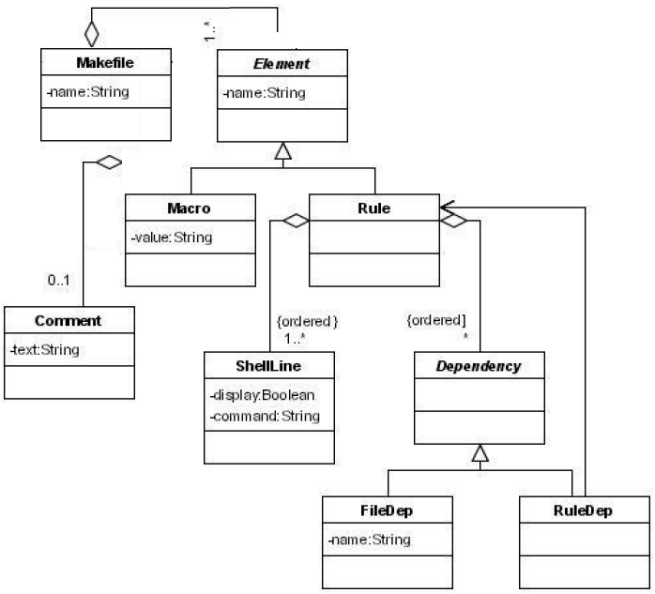
In this phase, our mechanism prepares a set of test cases. Each test case consists of a pair of metamodels and a reference alignment.

3.1 Discovering test cases

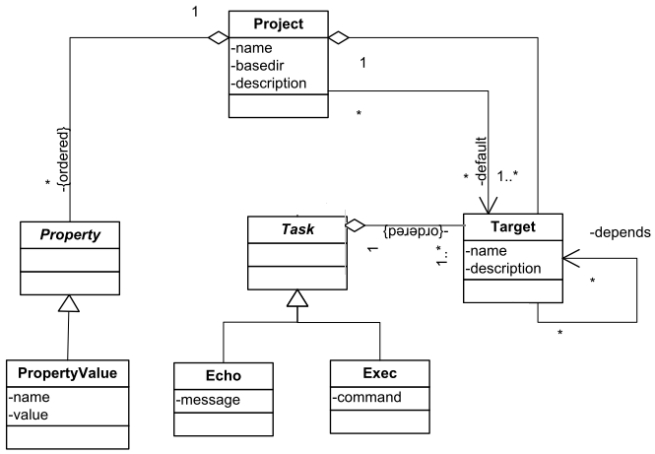
The megamodel is the cornerstone of our approach, it provides a roadmap of all the modeling artifacts stored in a repository, as well as, the relations between them [23]. Note that in MDE everything is a model (metamodels, transformations, etc). Before working with the megamodel it has to be populated. Since a model repository is continuously growing, we have implemented a strategy to automatically populate the megamodel. The strategy consists of the following steps:

1. A program parses a set of files containing metadata. The metadata describes, for example, in terms of what metamodels a transformation is written. The output of the parsing is a text file.
2. A textual syntax tool [6] translates the text file into a megamodel.
3. Because the megamodel refers to more models than the ones we are interested in, a program refines the megamodel to keep only the records related to transformations and their corresponding metamodels.

Fig. 3 shows the metamodels of an extracted test case. In general, the elements of a metamodel are classes and structural features (i.e., attributes, and references). For instance, the Ant metamodel contains the class `Project`, the attribute `message`, and the reference `default`.



(a) Make metamodel



(b) Ant metamodel

Figure 3. Metamodels of a test case

3.2 Extracting reference alignments from transformations

Having the megamodel, the mechanism extracts reference alignments from transformations. A transformation is a program that translates models, conforming to source metamodels, into models, conforming to target metamodels. A transformation may be available

in various flavors, i.e., being developed by programmers geographically distributed, or using model transformation languages different each other. Thus, we should define the extraction scope by answering the questions: 1) in what languages the available transformations are developed?, and 2) what patterns are embedded in them?

For example, we have found the following patterns in ATL [2] transformations (Listing. 1 shows a concrete example):

- Each transformation contains a set of rules (lines 1-9).
- Each rule consists of a pair of patterns, i.e., `inPattern` and `outPattern`. The `inPattern` matches a class of m (line 3), and the `outPattern` a class (or set of classes) of m' (line 5).
- Each `outPattern` is composed of a set of bindings. A binding initializes a rightClass property using a structural feature of leftClass. We have characterized four types of bindings:
  1. `rightStructuralFeature` ← a `leftStructuralFeature` (line 6).
  2. `rightStructuralFeature` ← an OCL [20] iteration expression which includes a `leftStructuralFeature` (line 7).
  3. `rightStructuralFeature` ← an operation (i.e., a helper expression) involving a `leftStructuralFeature`.
  4. `rightStructuralFeature` ← a variable referring to another `rightClass` created in the `outPattern`.

Note that the more we figure patterns out, the better is the quality of reference alignments. The quality can be further increased by extracting alignments not only from one transformation but also from a set of transformations $tm2m'$ if they are available. Alignments conform to one extension of the AMW core metamodel [2]. AMW (AtlanMod Model Weaver) is a tool to manipulate model-based alignments.

Listing 1. Excerpt of the transformation Make2Ant

```
1 rule Makefile2Project{
2     from
3         m : Make!Makefile
4     to
5         a : Ant!Project(
6             name       <- m.name,
7             targets    <- m.elements -> select(c | c.
                  oclIsKindOf(Make!Rule))
8         )
9 }
```

There are ways to bring the gap between modeling artifacts and ontologies. Tools like EMFTriple [13] can translate metamodels to ontologies. A transformation moreover can transform our alignments to a more standard alignment format (e.g., the Alignment API [8]).

4 Execution: Implementing and Testing Matching Algorithms with AML

The execution phase typically involves implementation, testing, and deployment[3] of matching algorithms. In our approach, the execution only embraces development and testing. We drop the deployment to the evaluation phase.

Instead of GPLs, we propose a DSL to implement and test matching algorithms. In general, DSLs improve programmers productivity and software quality. Often, this is achieved by reducing the cost of initial software development as well as maintenance costs. The improvements - programs being easier to write and maintain - materialize as a result of domain-specific guarantees, analysis, testing techniques, verification techniques, and optimizations [18].

According to [22], the matching operation has been investigated from the early 1980s. We promote the use of a DSL gathering all the expertise gained over the last twenty years. The DSL has to capture a significant portion of the repetitive tasks that a expert needs to perform in order to produce an executable matching algorithm.

A first approximation to such DSL is the AtlanMod Matching Language (AML) which is based on the MDE paradigm [11]. AML instruments the fundamental behind the most recent matching algorithms: to assemble multiple components each one employing a particular matching technique [15]. Internally, an AML algorithm is a chain of model transformations, each one implementing a matching technique. Input/output flow along the chain as models.

Roughly speaking, an AML program is a text file containing three sections [11]:

1. **Import section** enables reusing of pre-existing AML algorithms.
2. **Matching rule declaration** allows to specify a concrete matching technique. This conforms to one of 5 kinds of techniques:
   - **Creation** establishes an alignment between the element a (in metamodel m) and the element b (in metamodel m') when these elements satisfy a condition.
   - **Similarity** computes a similarity value for each alignment prepared by the creation techniques. One function establishes the similarity values by comparing particular metamodel aspects: labels, structures, and/or data instances.
   - **Aggregation** combines similarity values by means of one expression. An expression often involves:
     - n, the number of matching techniques providing alignments.
     - $\sigma_i(a, b)$ similarity value computed by the matching technique i
     - w_i weight or importance of the matching technique i, where $\sum_{i=1}^{n} w_i = 1$
   - **Selection** selects alignments whose similarity values satisfy a condition, e.g., thresholding.
   - **User-defined** represents matching techniques capturing functionality beyond our classification.
3. **Model flow block** specifies how several matching techniques interact to each other to deliver alignments.

Listing. 2 shows the excerpt of an AML algorithm named *LevenshteinBothMaxSim*. For the sake of brevity we present the model flow block and omit matching rule declarations [4]. *LevenshteinBothMaxSim* involves the following number of matching rules: 3 creation, 1 label-based similarity, 1 structure-based similarity, 1 aggregation, 2 selection, and 1 user-defined. They have been inspired by [3][4][16][9]. We have chosen such techniques because they report good results. Below we describe their functionality:

Creation *TypeClass*, *TypeReference*, and *TypeAttribute* create alignments between two metamodel elements conforming to the same metametamodel type (class, reference or attribute).

[2] The AtlanMod Transformation Language [14].

[3] We call deployment the stage that executes algorithms in order to produce the final matching results.

[4] The reader interested in that may check [10]

Similarity

- *Levenshtein* returns a value depending on the edition operations that should be applied to a label to obtain the other one. *Levenshtein* reuses the SimMetrics API [1].
- *Propagation*, *SF*, and *Normalization* instrument the algorithm Similarity Flooding described in [17]. This algorithm strengthens previously computed similarities based on the metamodel structure. *Propagation* associates two alignments (*l1* and *l2*) if there is a relationship between the linked elements, for example, l1 links classes, and l2 links attributes, classes contain attributes. *SF* propagates the similarity value from *l1* to *l2* because of their relationship, e.g., containment. *Normalization* divides all the similarity values by the maximal similarity value.

Aggregation The *Merge* matching technique puts together the alignments returned by other techniques. We have implemented it by means of an AML aggregation construct.

Selection *BothMaxSim* selects a given alignment if its similarity value is the highest among the values of alignments of two sets (e.g., `leftSet` and `rightSet`). The `leftSet` contains all the alignments linking a left concept, and the `rightSet` the alignments linking a right concept. *ThresholdMaxDelta* moreover selects an alignment when its similarity satisfies the range of tolerance $[Threshold-Delta, Threshold]$. While *BothMaxSim* provides 1:1 alignments, *ThresholdMaxDelta* provides n:m. These matching techniques (along with the thresholds) have been borrowed from [3].

User-Defined Whereas *Propagation* levers the execution of *SF* (a similarity matching technique), *Propagation* is user-defined.

Listing 2. LevenshteinBothMaxSim model flow

```
1  modelsFlow{
2      tp = TypeClass[map]
3      typeRef = TypeReference[map]
4      typeAtt = TypeAttribute[map]
5
6      merged = Merge[1.0:tp, 1.0:typeRef, 1.0:typeAtt]
7
8      nam = Levenshtein[merged]
9
10     filtered = ThresholdMaxDelta[nam]
11     prop = Propagation[filtered]
12     sf = SF[filtered](prop)
13     norm = Normalization[sf]
14
15     tmpresult = WeightedAverage[0.5 : norm, 0.5:nam]
16
17     result = BothMaxSim[tmpresult]
18 }
```

Besides the techniques mentioned above, we have implemented other sophisticated heuristics, for example, *MSR* (Measures of Semantic Relateness), which extracts semantic similarity between two labels based on a large corpora, e.g., Google or Wikipedia. Besides creation techniques, our library has 10 techniques [5].

AML is build on top of ATL, one of the most popular transformation language. AML inherits the ATL declarative nature which allows to straightforwardly implement matching rules. In addition, the model flow block enables to easily assembly matching techniques and tune algorithms. With respect to testing features, users can verify the correctness of alignments by displaying them in the AMW graphical interface [2]. AML moreover provides a functionality that compares computed alignments to gold standards and gives the results in HTML format.

[5] See http://wiki.eclipse.org/AML

Note that AML allows to develop user-defined matching techniques in ATL (to go beyond pre-established DSL semantics) and even invoke Java code. Thus, it is possible to integrate the vast set of matching algorithms developed in the past.

5 Evaluation: Deploying and Assessing AML Algorithms

Based on a megamodel, our mechanism automatically deploys algorithms and computes metrics. The megamodel indicates the algorithms to be executed, the required inputs, and the reference alignments. Our mechanism computes four matching metrics [9]: $Precision(R, A') = \frac{R \cap A'}{|A'|}$, $Recall(R, A') = \frac{R \cap A'}{|R|}$, $Fscore(R, A') = \frac{2 * Recall(A', R) * Precision(A', R)}{Recall(A', R) + Precision(A', R)}$, and $Overall(A', R) = Recall(A', R) * (2 - \frac{1}{Precision(A', R)})$. This moreover updates the megamodel with records associating A', R, and $M(A', R)$.

The use of DSLs and megamodels facilitates the evaluation phase. Since there is an unique programming interface, launching the algorithms become easier. By using the megamodel, our mechanism can execute algorithms over a large set of test cases and to obtain results by itself. This promotes a more confident evaluation of performance and accuracy.

The ontology community has contributed several tools to compute sophisticated matching metrics (e.g., PrecEvaluator [19]). We can take advantage of these tools by translating our alignments to their formats.

6 Experimentation: Results and Discussion

The approach has been tested over the "ATL Transformation Zoo" [5]. This repository has so far 103 ATL transformation projects contributed by the m2m Eclipse community [7]. Our mechanism automatically populated the megamodel with 1064 entities, 185 of them are metamodels and 156 are transformations. It took around 20 minutes to build the megamodel.

Figure 4. Automatic modeling artifact discovery results

After filtering the megamodel, for obtaining only the data matter of our interest, we obtained 30 test cases. Fig. 5 shows the results of applying the *LevenshteinBothMaxSim* algorithm to the test cases. The algorithm spent 40 minutes matching the pairs of metamodels. Fig. 5 reports low fscores (lower than 0.5). There are two reasons behind these results:

1. **The suitability of AML algorithms**. The space of matching algorithm is quite large, and sometimes it is difficult to find an algorithm that accurately matches a certain pair of metamodels. Thus, it is quite natural that *LevenshteinBothMaxSim* provides good results for few pairs of metamodels. We can see better the accuracy of an algorithm as we compare its results to other algorithms'.
2. **The quality of reference alignments**. When transformations are complex or incomplete, our mechanism may not extract enough reference alignments.

- **Complex** means that a given transformation implements imperative patterns (rules with only `outPattern`). In contrast, our mechanism relies on declarative patterns (as described in Section 3.2, rules with `inPattern` and `outPattern`). One example is the transformation Measure2Table which contains 6 rules, all of them imperative. The complexity of this transformation explains why our mechanism does not obtain any reference alignment, and why the fscore is zero.
- **Incomplete** means that a transformation lacks items. The justification is that developers write transformations in terms of the data instances that they expect to have in the source and target models. If the data instances are not relevant, developers do not written rules associating certain metamodel concepts. Thus, even if an AML algorithm gives good results, it is possible to obtain low fscores when reference alignments are poor.

Reference Alignment	Pres	Rec	Overall	Fscore
XML2Ant	0,3	0,08	-0,11	0,13
KM32CONFATL	0,03	0,22	-6,89	0,05
OWL2XML	0,07	0,06	-0,76	0,06
A2B	0,5	0,5	0,0	0,5
Ant2XML	0,22	0,14	-0,37	0,17
Make2Ant	0,04	0,13	-2,69	0,06
XML2Make	0,13	0,07	-0,43	0,09
XHTML2XML	0,04	0,02	-0,48	0,03
Table2TabularHTML	0,11	0,25	-1,75	0,15
Measure2Table	0,0	0,0	0,0	0,0
XHTML2XML	0,04	0,02	-0,48	0,03
TypeA2TypeB_v4	0,8	0,67	0,5	0,73
ATOM2RSS	0,06	0,19	-2,71	0,09
ATOM2XML	0,05	0,1	-1,9	0,06
RSS2ATOM	0,06	0,22	-3,17	0,1
RSS2XML	0,04	0,17	-3,67	0,07
XML2ATOM	0,05	0,02	-0,46	0,03
XML2RSS	0,17	0,09	-0,36	0,12
TypeA2TypeB_v1	0,25	0,33	-0,67	0,29
TypeA2TypeB_v2	0,75	0,6	0,4	0,67
SimpleSBVR2SimpleUML	0,1	0,2	-1,6	0,13
Syntax2SimpleSBVR	0,46	0,61	-0,1	0,53
TT2BDD	0,4	0,43	-0,21	0,41
SVG2XML	0,08	0,08	-0,77	0,08
HTML2XML	0,1	0,1	-0,76	0,1
Table2TabularHTML	0,11	0,17	-1,17	0,13
Tree2List	0,33	0,33	-0,33	0,33
Tree2List_usingATLResolveAlgorithm	0,33	0,25	-0,25	0,29
OWL2XML	0,07	0,06	-0,76	0,06
XML2XSLT	0,67	0,29	0,14	0,4

Figure 5. Results

In the future, we wish to apply a larger set of AML algorithms to the test cases extracted from the ATL transformation Zoo. With respect to the quality of reference alignments, we think that it is acceptable: only one transformation out of thirty has produced an empty reference alignment, i.e., *Measure2Table*, the one containing imperative rules. In general, ATL transformations contain declarative patterns which allows to extract reference alignments much more easily. We believe that transformations are a good source of reference alignments. So that, we can get an intuition about matching algorithm accuracy as we use them.

7 Acknowledgments

The authors would like to thank the ANR project FLFS for the provided funding, as well as Jean Bezivin and Pierre Cointe for discussions that have contributed to the formation of the views reflected in this paper.

8 Conclusion

We presented a mechanism to automatize the evaluation of model matching systems. Our early results are motivating. Firstly, the mechanism extracts complete test cases from open-source repositories. The number of candidate test cases raises every year. Secondly, the use of a DSL makes it possible to reuse matching techniques. Thus, developers can improve their productivity. Finally, by using a megamodel, our mechanism extensively evaluates matching algorithms over extracted test cases. We outlined the tools needed to adapt our mechanism to the ontology context. They are mostly based on model transformations. In the near future, we wish to join an evaluation initiative, like the OAEI. The idea is to test our mechanism, notably, the development of further matching algorithms.

REFERENCES

[1] Sam Chapman, *SimMetrics*, `http://sourceforge.net/projects/simmetrics/`, 2009.
[2] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Erwan Breton, and Guillaume Gueltas, 'AMW: a generic model weaver', in *Proceedings of the 1re Journée sur l'Ingénierie Dirigée par les Modéles*, (2005).
[3] Hong Hai Do, *Schema Matching and Mapping-based Data Integration*, Ph.D. dissertation, University of Leipzig, 2005.
[4] Anhai Doan, Pedro Domingos, and Alon Halevy, 'Reconciling schemas of disparate data sources: A machine-learning approach', in *In SIGMOD Conference*, pp. 509–520, (2001).
[5] Eclipse/M2M. Atl transformations zoo. `http://www.eclipse.org/m2m/atl/atlTransformations`, 2009.
[6] Eclipse.org, *TCS project*, `http://www.eclipse.org/gmt/tcs/`, 2008.
[7] Eclipse.org, *Model to Model (M2M)*, 2009.
[8] Jerome Euzenat, *An alignment API*, `http://alignapi.gforge.inria.fr/`.
[9] Jerome Euzenat and Pavel Shvaiko, *Ontology Matching*, Springer, Heidelberg (DE), 2007.
[10] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin, 'Adaptation of models to evolving metamodels', Technical report, INRIA, (2008).
[11] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin, 'A Domain Specific Language for Expressing Model Matching', in *Proceedings of the 5ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM09)*, Nancy, France, (2009).
[12] Dragan Gašević, Dragan Djurić, and Vladan Devedzic, *Model driven architecture and ontology development*, Springer Verlag, pub-SV:adr, 2006.
[13] Guillaume Hillairet, *EMFTriple*, `http://code.google.com/p/emftriple/`, 2009.
[14] Frédéric Jouault and Ivan Kurtev, 'Transforming models with ATL', in *Proceedings of the Model Transformations in Practice Workshop, MoDELS 2005*, Montego Bay, Jamaica, (2005).
[15] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon S. Rosenthal, 'etuner: tuning schema matching software using synthetic scenarios', *The VLDB Journal*, **16**(1), 97–122, (2007).
[16] S. Melnik, H. Garcia-Molina, and E. Rahm, 'Similarity flooding: A versatile graph matching algorithm and ist application to schema matching', in *Proc. 18th ICDE*, San Jose, CA, (2002).
[17] Sergey Melnik, *Generic Model Management: Concepts and Algorithms*, Ph.D. dissertation, University of Leipzig, 2004.
[18] Marjan Mernik, Jan Heering, and Anthony Sloane, 'When and how to develop domain-specific languages', *CSURV: Computing Surveys*, **37**, (2005).
[19] OAEI, *Towards a methodology for evaluating alignment and matching algorithms*, `http://oaei.ontologymatching.org/doc/oaei-methods.1.pdf`.
[20] OMG, *OCL 2.0 Specification, OMG Document formal/2006-05-01*, `http://www.omg.org/docs/ptc/05-06-06.pdf`, 2006.
[21] Fernando Silva Parreiras, Steffen Staab, and Andreas Winter, 'On marrying ontological and metamodeling technical spaces', in *ESEC/SIG-*

SOFT FSE (Companion), eds., Ivica Crnkovic and Antonia Bertolino, pp. 439–448. ACM, (2007).

[22] Erhard Rahm and P Bernstein, 'A survey of approaches to automatic schema matching', *The VLDB Journal*, **10**(4), 334–350, (2001).

[23] Eric Vpa, Jean Bzivin, Hugo Brunelire, and Frdric Jouault, 'Measuring model repositories', in *Proceedings of the 1st Workshop on Model Size Metrics (MSM'06) co-located with MoDELS'2006*, (2006).