Metamodel Matching: Experiments and Comparison

Denivaldo Lopes
Federal University of Maranhão (UFMA)
São Luís, Brazil
Email: dlopes@dee.ufma.br

Slimane Hammoudi
ESEO
Angers, France
Email: shammoudi@eseo.fr

José de Souza and Alan Bontempo
Federal University of Maranhão (UFMA)
São Luís, Brazil
Email: {jgeraldo, abontempo}@dee.ufma.br

***Abstract*— Recently, Model Driven Engineering (MDE) has been proposed for supporting the development, maintenance and evolution of software systems. Model Driven Architecture (MDA), Software Factories and Eclipse Modeling Framework (EMF) are representative MDE approaches. These MDE approaches have some concepts and techniques in common such as modeling, metamodels and model transformation. However, other concepts and techniques should be envisaged such as metamodel matching. In this paper, we discuss some issues and provide some insights into metamodel matching. For this purpose, we use UML and the C# platform to illustrate our approach and to evaluate our Mapping Tool for MDE (MT4MDE) and Semi-Automatic Matching Tool for MDE (SAMT4MDE). Afterwards, comparisons with other tools are presented.**

I. Introduction

In the literature, some issues around Model Driven Engineering (MDE) have been studied and subject to intensive research, e.g. modeling languages [1], transformation languages [2][3], mapping between metamodels [4] and methodologies [5]. Another important issue for MDE is *metamodel matching*. However, there has been little research in *metamodel matching*. In the database domain, the corresponding term for *metamodel matching* is *schema matching*. For a long time, *schema matching* has been considered as an important issue for *schema management* [6][7][8]. Thus, we have some lessons to learn from database domain in order to improve an MDE approach.

In [4] [9], an approach separating *mapping specification* from *transformation definition* was introduced and implemented in a tool called Mapping Modeling Tool (MMT). In this approach, a *mapping specification* specifies the relationships between metamodels, while *transformation definition* contains the operational description of the transformation between models.

However, the manual creation of *mapping specification* is a fastidious and error-prone task. Generally, this task implies a search of *equivalent* or *similar elements* between two metamodels. In the database domain, this task is called *schema matching* [6]. In the MDE context, this can be called (meta)model matching. In [10], we present an algorithm for *(meta)model matching*, a tool for mapping modeling called MT4MDE, and another tool for semi-automatic metamodel matching called SAMT4MDE.

In this paper, we provide some insights into the *metamodel matching*, an illustrative example of the utilization of our approach and a tool for matching the UML metamodel and C# metamodel, a comparison of our tools (MT4MDE and SAMT4MDE) with other tools and approaches in the database domain.

This paper is organized in the following way. In section II, we discuss *schema matching* in the database domain. In section III, we present our approach to take into account *metamodel matching*. In section IV, we illustrate the application of our approach and tool for mapping UML to C#. In section V, we present a comparison between our approach and other approaches, providing their benefits and limitations. In section VI, we conclude this paper, presenting the future directions of our research.

II. Approaches for schema matching

In general, metamodels are created with a specific purpose[1] and by different groups of persons. Each purpose is determined in function of the domain, and each group of persons models a system in different ways. In the modeling task, each group abstracts, classifies and generalizes the reality based on its own knowledge. Consequently, different groups working in different contexts may create metamodels with different structures and terminologies [6], causing the *semantic distance* between these metamodels [11] [12].

A model can be transformed into another model, only if the metamodel of the former can be mapped into the metamodel of the latter. In order to map metamodels, the *equivalent* or *similar elements* must be identified, and the *semantic distance* should be minimized. The process of searching equivalent or similar elements is called *schema matching* [6]. And *"the notion of semantic distance was developed to cover the notion of how close is close enough"* [12]. A dual for *semantic distance* is *schema similarity* that is defined as *"the ratio between the number of matching elements and the number of all elements from both input schemas"* [8] ($SS = \frac{N_m}{N_t}$, where SS is the *schema similarity*, N_m is the number of matching elements and N_t is the number of all elements). *Semantic distance* can also be quantified as a numeral value (like *schema similarity*) or as a subset of a metamodel [4]. In fact, *"one of the primary purposes of automation in MDA is to bridge the semantic gap between domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework"* [11]. This *semantic gap*

[1]UML is a general-purpose modeling language, but it provides profiles as extension mechanism in order to be adapted to a domain.

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

is necessary, because it allows separate different abstraction levels, and it is inherent when two different groups are creating metamodels from different view points, e.g. metamodels used to model specific platforms. However, we need to graduality fill this gap in the transformation process inserting additional information.

In the literature, several *schema matching approaches* have been proposed [13]. Each *schema matching approach* has its own characteristics that were grouped in a taxonomy [6]. In addition, each approach have been evaluated through *match quality measures* [8].

In [6], the following characteristics are organized in a taxonomy:

- *Individual matcher approaches* use only one matching criterion. They are classified as:
  - *schema-only based*, when they consider only metamodels. They can be distinguished in:
    - *Element level*, the mapping is realized for each individual element. It can be classified as *linguistic* and *constraint-based*. *Linguistic* is based on *name similarity*, *description*, *global namespace*, while *constraint-based* is based on *type similarity* and *key properties*.
    - *Structure-level*, the mapping is realized taking into consideration the combinations of elements related in a structure. It is only classified as *constraint-based* that uses *graph matching*.
  - *instance/contents-based*, when they consider only instances (or models). It can also be classified as *element-level*. This last can be classified as *linguistic* and *constraint-based*. In this case, *linguistic* is based on *word frequencies* and *key terms* present in the element instances, while *constraint-based* is based on *value pattern* and *ranges* of the element instances.
- *Combining matchers* use multiple matching criteria. They can be classified as:
  - *Hybrid*, they combine multiple approaches to create only one matcher in order to produce a result, i.e. the creation of mapping between the elements.
  - *Composite*, they combine many results obtained from different approaches in order to produce the mapping between elements. This combination of results can be manual or automatic.

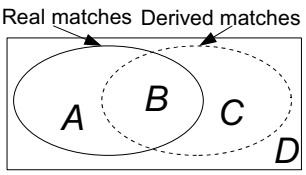

Fig. 1. Real matches and derived matches [8]

In Figure 1 [8], the interrelationships between metamodels are organized in sets that can be manually or automatically created. A set created manually can contain all needed matches (i.e. matched elements), while a set created automatically can contain valid and non valid matches. The former set is called *real matches*, and the latter is called *derived matches*. In addition, other subsets are defined as follows [8]:

- A (false negatives) are matches needed but not automatically identified.
- B (true positives) are matches which are needed and have also been correctly matched by the automatic match operation.
- C (false positives) are matches falsely proposed by the automatic match operation.
- D (true negatives) are false matches which have also been correctly discarded by the automatic match operation.

These sets can be used to create *match quality measures* as follows [8]:

Precision $= \frac{|B|}{|B|+|C|}$ reflects the share of real correspondences among all found ones.

Recall $= \frac{|B|}{|A|+|B|}$ specifies the share of real correspondences that are found.

F-Measure $= 2 * \frac{Precision * Recall}{Precision + Recall}$ represents the harmonic mean of *Precision* and *Recall*, i.e. $\alpha = 0.5$.

Overall $= Recall * \left(2 - \frac{1}{Precision}\right)$ quantifies the post-match effort needed for adding false negatives and removing false positives.

These *taxonomies* and *match quality measures* are used hereafter to evaluate and compare our approach and tool MT4MDE and SAMT4MDE with other approaches and tools (see section V).

III. Metamodel Matching

In the *model transformation* process [4], the creation of a transformation definition is preceded by the action of finding correspondences, i.e. *metamodel matching*. However, there has been little research in *metamodel matching* until now. In our research, we have applied some lessons from several researches into the database domain that have explored *schema matching* [6] [7] [14]. *Schema matching* and *Metamodel matching* have some similarities. For example, *intra-relationships* and *cross-kind-relationship implications* have been applied to *Schema matching* and can also be applied to *metamodel matching* [10].

Metamodel matching results in a *mapping model*, and a *mapping model* describes how two metamodels are related to each other. According to *model management algebra* [15], a *mapping* is generated using an operator called *match* which takes two models[2] as input and returns a *mapping* between them. We have modified this operator as follows: given M_a, M_b and $C_{M_a \to M_b}/M_c$, the *operator match* is formally defined as $Match'(M_a, M_b) = C_{M_a \to M_b}/M_c$.

In order to find *correspondent elements*, we use *cross-kind-relationship implications* [16]. In addition, we propose other *cross-kind-relationship implication* as follows:

1) if $C(p, q)$ and $I(r, q)$ then $C(p, r)$.

[2]In our approach, we prefer to employ the term metamodel in the definition of the term mapping.

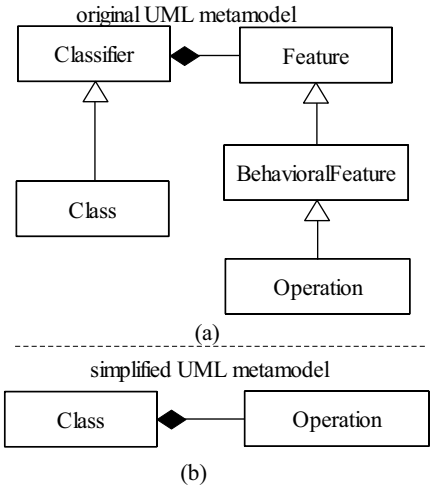

Fig. 2. Applying cross-kind-relationship implications in UML

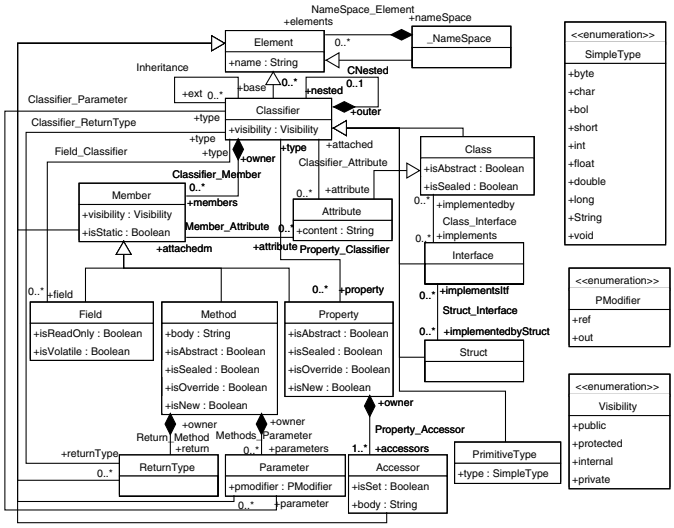

Fig. 3. A C# metamodel fragment [9] [18]

This seems contradictory, because r *Is-a* q, but q *Is-not-a* r. However, we understand that *Is-a* is a special relationship between r and q. So, if r exists is because a q exists, then r is linked to the existence of q. So, we can manipulate and replace q by r. Figure 2 was extracted from [1] and it is manipulated to illustrate the use of this *cross-kind-relationship implication*. Figure 2b, `Class` contains `Operation`. In fact, a `Class` inherits from a `Classifier` that has a `Feature`, and an `Operation` inherits from a `BehavioralFeature`, and a `BehavioralFeature` inherits from a `Feature` (see Figure 2a). The manipulation to obtain the Figure 2b from Figure 2a is presented as follows:

- If an *Operation*, *BehavioralFeature* and *Feature* exist such that *I(Operation,BehavioralFeature)* and *I(BehavioralFeature,Feature)*, then *I(Operation,Feature)*.
- If a *Class* and a *Classifier* exist such that *I(Class,Classifier)*, then *Class* also represents *Classifier*, and we can simplify by presenting only *Class*.
- If *C(Class,Feature)* and *I(Operation,Feature)*, then *C(Class,Operation)*.

IV. An illustrative example: matching the UML metamodel and the C# metamodel

In order to validate our approach, we have developed MT4MDE (Mapping Tool for Model-Driven Engineering) and SAMT4MDE (Semi-Automatic Matching Tool for Model-Driven Engineering). These tools are implemented as *plug-ins* for the environment Eclipse [17]. In this section, we illustrate the semi-automatic creation of a *mapping model* between the fragments of UML metamodel and a C# metamodel.

The UML metamodel [1] is well known and no more commentaries are needed. In this experiment, we use the C# metamodel proposed in [9] [18]. Figure 3 presents the fragments of the C# metamodel used in this experiment.

In Figure 3, we call attention to `Attribute`. That is a declarative information that can be attached to programs' entities (such as `Class` and `Methods`) and retrieved at runtime. All attribute classes derive from the `System.Attribute` base class provided by the .NET Framework. Although `Attribute` belongs to the C# API (i.e. model or M_1 layer), we have used it as part of the C# metamodel, in order to manipulate it in the metamodel layer (i.e. M_2 layer). C# `Attribute` does not have the same meaning of UML `Attribute`. In UML, an attribute is a feature within a classifier that describes a range of values whose type is a classifier, while, in C#, an attribute is used to include additional information to a classifier (such as `Class` and `Methods`).

The steps of utilization of our tools can be enumerated as follows:

1) A user creates an empty project in Eclipse.
2) The user copies the UML metamodel and C# metamodel into this project.
3) The user uses the MT4MDE wizard for creating a mapping model which conforms to our mapping metamodel [10]. This mapping model is initially empty.
4) In MT4MDE, the user invokes SAMT4MDE to find possible correspondent elements, i.e. *derived matches*.
5) After that SAMT4MDE returns the possible correspondent elements, the user can validate or refuse them. Figure 4 illustrates the execution of SAMT4MDE that found the possible correspondences between the UML metamodel and C# metamodel.
6) SAMT4MDE creates automatically the mapping model based on the validated correspondences (see Figure 5).
7) The user can complete the mapping model. The user creates the other correspondences that were not found by SAMT4MDE.
8) Once the mapping model is completed, our tool can calculate the *match quality measures*. The number of *real matches* is obtained from the final mapping model, the number of *false positives* is obtained from the number of refused correspondences, and the number of *true positives* is obtained from the number of validated

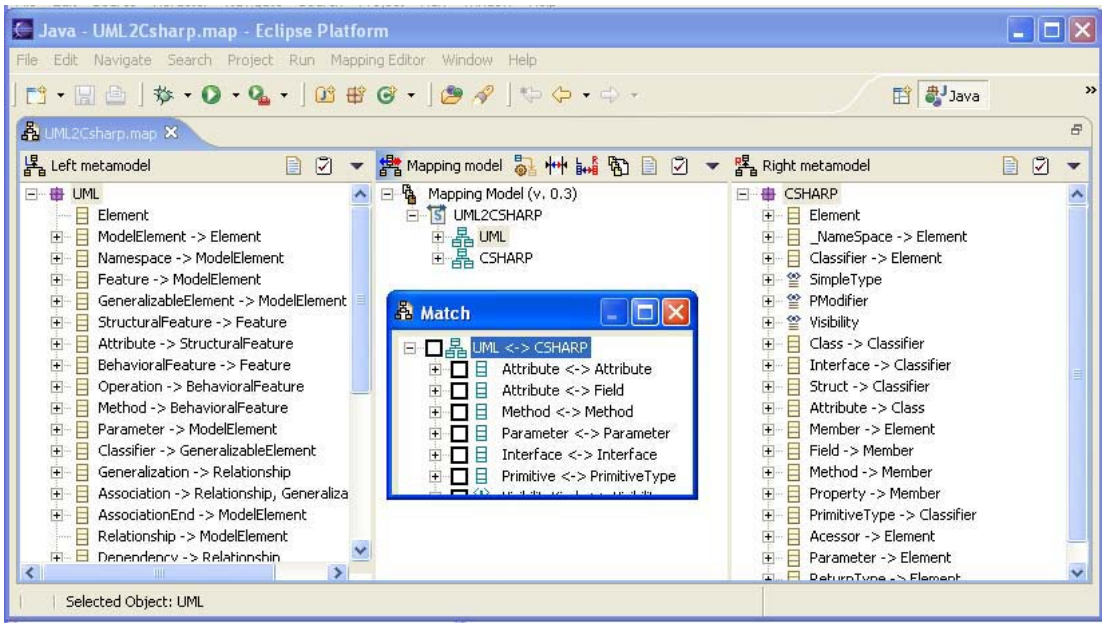

Fig. 4. The possible correspondences found by SAMT4MDE

correspondences.

9) Finally, the user uses the MT4MDE for creating automatically the transformation definition in a transformation language such as Atlas Transformation Language (ATL) [2].

The evaluation of our approach and tool through this illustrative example resulted in the following values [3]:

- *Schema similarity*
  $SS = 0.74$
- *Match quality measures*
  Precision $= 0.71$
  Recall $= 0.68$
  F-Measure $= 0.69$
  Overall $= 0.40$

In the ideal case, *Precision*=*Recall*=1.0, i.e. when the number of *false negatives* and *false positives* are both zero. In our experimentation, *Precision* $= 0.71$ demonstrates that 71% of derived matches were correctly determined using our *schema matching algorithm*. And *Recall* $= 0.68$ demonstrates that 68% of real matches were automatically found.

However, our tool has limitations. For example, we call attention to Figure 4 that shows our tool SAMT4MDE proposing that `Attribute` from UML may correspond to `Attribute` from C#. This mistake can be justified because a UML `Attribute` has a relationship[4] with a UML `Classifier`, and a C# `Attribute` has a relationship with a C# `Classifier`. In addition, the name of both elements are equal, but the meanings are different. In other words, both have similar structures, and a preliminary linguistic analysis determines that they are equal. In future research, we aim to avoid mistakes like this improving our tool with semantic analysis and machine learning.

We have used our tools in some use cases, e.g., creating a mapping model between UML and Java [10], between UML and WSDL, and between EDOC and WSDL.

V. Related Work and Discussion

In the database domain, some approaches and tools support the creation and edition of mappings between schemas such as Clio [7][19], Type Evolution Software Systems (Tess) [20] and Rimu Visual Mapper (RVM) [21].

Clio, Tess and RVM have three basic characteristics. First, the searching for correspondences between schemas in order to create a mapping. Second, a translating program is generated from this mapping. Third, this translating program is applied to transform a document in another document.

MT4MDE and SAMT4MDE constitute a set of tools providing the same characteristics as Clio, Tess and RVM. We separate the *mapping specification* from *transformation definition*. A *mapping specification* can be semi-automatically generated. Afterwards, a *transformation definition* is generated from this *mapping specification*.

Other schema match approaches and tools were evaluated and submitted to benchmarks such as SEMantic INTegrator (SEMINT) [22], Learning Source Descriptions (LSD) [23] and Cupid [24]. In [6], E. Rahm and P. A. Bernstein analyze these approaches and tools following a taxonomy. In [8], H.H. Do, S. Melnik and E. Rahm analyze these same approaches with the aid of the *match quality measures*.

Comparing our approach and tool with these approaches and tools, we can find some similarities and differences between them. SAMT4MDE uses an object oriented model as schema

[3] We used a fragment of the UML and Java metamodel in this experimentation.

[4] After the application of cross-kind-relationships implications

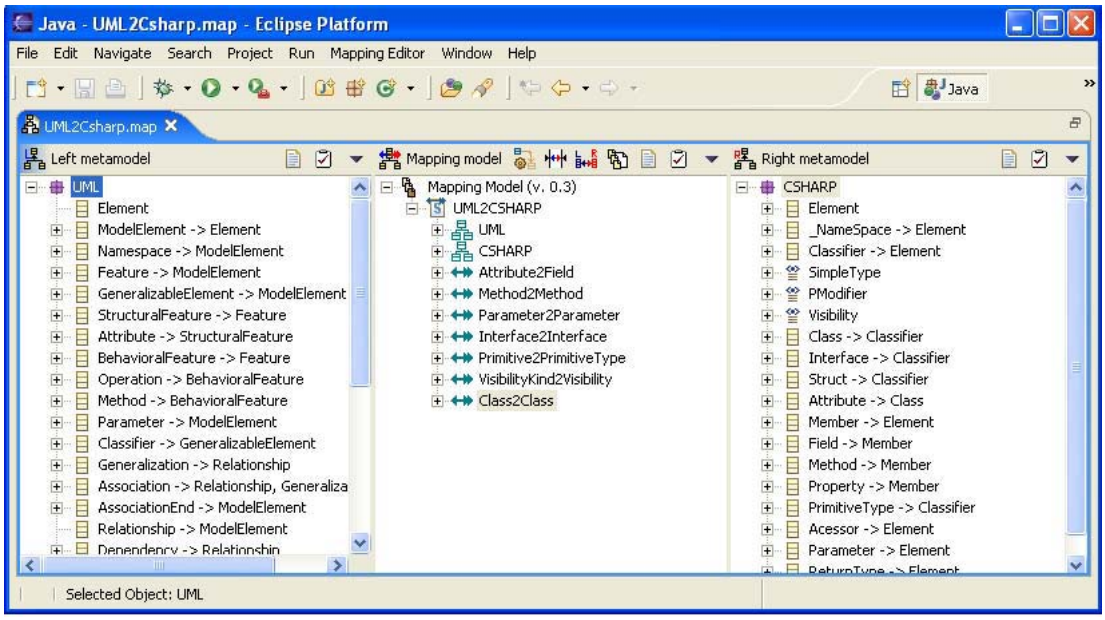

Fig. 5. A mapping model created automatically by SAMT4MDE

type, while the other approaches/tools use relational or XML as schema type. The metadata representation of SAMT4MDE is a metamodel (which conforms to Ecore), while the other approaches/tools have XML schema trees or extended Entity-Relationship (ER) models as metadata representation. All approaches with the exception of SEMINT have a granularity of matching at element or structure level. The other approaches provide a match cardinality of $1:1$, while our approach provides a match cardinality of $1:n$. The majority of these approaches have a combination of matches of type hybrid. Our proposition and tool provide several criteria based on the terminology, while the other approaches take benefits from terminology and artificial intelligence for searching correspondences. SAMT4MDE does not have instance-level match characteristic, while SEMINT provides character/numerical data pattern and value distribution, and LSD provides Whirl, Bayesian learners and list of valid domain values. SEMINT, LSD and Cupid are used in data base systems, more precisely, in data integration and data translation, while SAMT4MDE is used in the model transformation in order to develop software systems. SAMT4MDE has the advantage of being developed as a *plug-in* for *Eclipse*, so it is independent from MT4MDE and it can be integrated into other tools.

According to Table I, SAMT4MDE and Cupid allow only two schemas per match task ($2/1$ schemas/task), while SEMINT allows ten schemas per five match tasks ($10/5$ schemas/tasks) and LSD twenty-four schemas per twenty tasks ($24/20$ schemas/tasks). SAMT4MDE uses discrete values $\{-1, 0, 1\}$ as match result representation (i.e. different, similar or equal), while the others use continuous values in the range $[0, 1]$ to represent the similarity degree. We obtained average values for match quality measures that are close to those from SEMINT and LSD, through the execution of SAMT4MDE for creating a mapping model between UML and C# (see section IV) and another mapping model between UML and Java [10]. As presented in this paper, SS=0.74, Precision=0.71, Recall=0.68, F-Measure=0.69 and Overall=0.40 for the use case UML and C#, while, as presented in [10], SS=0.84, Precision=0.86, Recall=0.68, F-Measure=0.76 and Overall=0.57 for the use case UML and Java. Thus, we have obtained the following average values: SS=0.79, Precision=0.785, Recall=0.68, F-Measure=0.73 and Overall=0.49. These comparisons among our tools and SEMINT, LSD and Cupid demonstrated the quality of our tool. We note that SEMINT, LSD and Cupid are used in the database domain, and our tool is used in development of information systems.

VI. Conclusion

In this paper, we have presented our approach to take into account *metamodel matching* in the context of MDE. The study of *metamodel matching* in MDE is a promising trend to improve the creation of *mapping specification* and, consequently, *transformation definition*. Tools for *metamodel matching* are necessary to avoid error-prone factors linked to the manual creation of *transformation definition* and to evolve *mapping specification* when metamodels change. In fact, we have used the principles of MDE to develop MDE. First, the *metamodel matching algorithm* helps in the creation of *mapping specification*. Afterwards, a *mapping specification* is transformed in a *transformation definition*. So, a *mapping specification* is a PIM, and a *transformation definition* is a PSM.

The main contributions of this work are some insights in *metamodel matching*, some discussions about our *plug-in* for *metamodel matching tool* (SAMT4MDE), an illustrative example and an evaluation of our approach. An illustrative

	SEMINT [22]	**LSD** [23]	**Cupid** [24]	**SAMT4MDE**
Test problems				
Tested schema types	relational	XML	XML	Object-oriented model
#Schemas/#Match tasks	10/5	24/20	2/1	2/1
Avg schema similarity size	57	-	47	79
Match result representation				
Matches (element-level corresp.)	with similarity value in range [0,1]			with discrete values {0, 1 and -1}
Element repr.	node (attr.)	node	path	node
Best average match quality				
Prec./Recall	0.78/0.86	0.8/0.8	-	0.79/0.68
F-Measure	0.81	0.8	-	0.73
Overall	0.48	0.6	-	0.49
Evaluation highlights				
	Big schemas, No pre-match effort	Big schemas	Comparative evaluation of 3 systems	Representative metamodels (UML, Java and C# metamodel) for developing information systems

TABLE I

EVALUATION OF SAMT4MDE AND OTHER TOOLS (BASED ON [8])

example using UML and C# metamodels is presented to validate our approach. The values of *match quality measures* for the illustrative example demonstrate the potential of our approach and tools.

In future work, we will study and implement other *metamodel matching algorithms*, e.g. algorithms based on machine learning and other heuristics. In addition, we envisage to study the optimization of mapping models which seems to be another important issue in MDE.

ACKNOWLEDGMENTS

The work described in this paper was financed by **Fundo Setorial de Tecnologia da Informação (CT-Info), MCT, CNPq (CT-Info/MCT/CNPq)**.

REFERENCES

[1] OMG, *Unified Modeling Language Specification, Version 1.4*, September 2001.

[2] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui, "First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery," *2nd OOPSLA Workhop on Generative Techniques in the context of Model Driven Architecture*, October 2003.

[3] OMG, *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, November 2005, ptc/05-11-01.

[4] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault, "Generating Transformation Definition from Mapping Specification: Application to Web Service Platform," *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, no. LNCS 3520, pp. 309–325, June 2005.

[5] F. Fondement and R. Silaghi, "Defining Model Driven Engineering Processes," *WiSME@UML 2004*, October 2004.

[6] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[7] P. Andritsos, R. Fagin, *et al.*, "Schema Management," *In IEEE Data Engineering Bulletin*, vol. 25, no. 3, pp. 32–38, September 2002.

[8] H.-H. Do, S. Melnik, and E. Rahm, "Comparison of Schema Matching Evaluations," *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, pp. 221–237, 2003.

[9] D. Lopes, "Study and applications of the mda approach in web service platforms," Ph.D. thesis (written in French), University of Nantes, 2005.

[10] D. Lopes, S. Hammoudi, and Z. Abdelouahab, "Schema Matching in the Context of Model Driven Engineering: From Theory to Practice," *Proceedings of the International Conference on Systems, Computing Sciences and Software Engineering (SCSS 2005)*, December 2005.

[11] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic, "An MDA Manifest," *MDA Journal*, May 2004.

[12] H. T. Goranson, "Semantic Distance and Enterprise Integration," *Knowledge Sharing in the Integrated Enterprise - Interoperability Strategies for the Enterprise Architect*, vol. 183, pp. 39–52, October 2005.

[13] X. L. Sun and E. Rose, "Automated Schema Matching Techniques: An Exploratory Study," *Research Letters in the Information and Mathematical Sciences*, vol. 4, pp. 113–136, 2003.

[14] Y. Velegrakis, R. J. Miller, and L. Popa, "Mapping Adaptation under Evolving Schemas," *Proceedings of the 29th VLDB Conference*, pp. 584–595, September 2003.

[15] P. A. Bernstein, "Applying Model Management to Classical Meta Data Problems," *Proceedings of the 2003 CIDR*, pp. 209–220, January 2003.

[16] R. A. Pottinger and P. A. Bernstein, "Merging Models Based on Given Correspondences," *Proceedings of the 29th VLDB Conference*, pp. 826–873, 2003.

[17] Eclipse Project, "Eclipse," 2006, available at http://www.eclipse.org.

[18] J. Bézivin, S. Hammoudi, D. Lopes, and F. Jouault, "B2B Applications, BPEL4WS, Web Services and dotNET in the context of MDA," *Knowledge Sharing in the Integrated Enterprise - Interoperability Strategies for the Enterprise Architect*, vol. 183, pp. 225–236, October 2005.

[19] L. Popa, Y. Velegrakis, R. Miller, M. Hernandez, and R. Fagin, "Mapping Generation and Data Translation of Heterogeneous Web Data," *In International Workshop on Data Integration over the Web (DIWeb)*, May 2002.

[20] B. S. Lerner, "A Model for Compound Type Changes Encountered in Schema Evolution," *ACM Transactions on Database Systems (TODS)*, vol. 25, no. 1, pp. 83–127, March 2000.

[21] J. Grundy, J. Hosking, R. Amor, W. Mugridge, and Y. Li, "Domain-Specific Visual Languages for Specifying and Generating Data Mapping Systems," *Journal of Visual Languages and Computing*, vol. 15, no. 3-4, pp. 243–263, June-August 2004.

[22] W.-S. Li and C. Clifton, "SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data Knowl. Eng.*, vol. 33, no. 1, pp. 49–84, 2000.

[23] A. Doan, P. Domingos, and A. Y. Halevy, "Reconciling Schemas of Disparate Data Sources: a Machine-Learning Approach," *SIGMOD Rec.*, vol. 30, no. 2, pp. 509–520, 2001.

[24] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid." San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 49–58.