A Benchmark for Model Matching Systems: The Heterogeneous Metamodel Case

Manuel Wimmer and Philip Langer

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology
{wimmer,langer}@big.tuwien.ac.at

Abstract

Recently, several model matching tools emerged that compute the corresponding elements of two different models. Having a closer look at concrete matching scenarios, one may find different cases such as matching different versions of one model, e.g., model evolution, or matching heterogenous models that do not have a common predecessor. Especially, the latter case is interesting as a first step when similar but at the same time potentially heterogenous metamodels have to be bridged for exchanging models between different tools.

Most model matching tools provide means for matching heterogenous metamodels. In particular, several heuristics are implemented in matching algorithms to reason about heterogenous structures and terminologies used in the metamodels to be matched. However, which set of heuristics is appropriate for heterogeneous metamodel matching and how to weight them when aggregating the different results to achieve correspondences with high quality is still an open challenge. To address this issue, we propose a benchmark consisting of real world metamodels and manually defined expected correspondences that allows to evaluate automatically the quality of the output of model matching tools.

1 Introduction

More than 10 years ago, a new software development approach was officially launched—the *Model Driven Architecture* (MDA)—intended to support *Model Driven Engineering* (MDE). As indicated by their names, models play the key role in MDA as well as in MDE and are becoming more and more first-class citizens in the software development process.

With the rise of MDA, the landscape of available tools for model-driven software development is growing steadily. These tools support a wide range of tasks like model creation, simulation, checking, and code generation–to name just a few. Coming along with this multitude of tools, heterogeneity increases, which complicates or even prevents tool interoperability due to different syntax, semantics, exchange formats, etc.

The efficient exchange of models, however, constitutes an important prerequisite for effective software development processes based on MDE. Unfortunately, the transformation of one model to another model and the development of integration solutions is a cumbersome if performed manually. An approach to tackle this problem—at least semi-automatically—is to consider the metamodels of the modeling languages and to identify correspondences between them. These correspondences may be valuable input for developing model transformations to exchange models between tools, merging two metamodels into an unified metamodel, or "just" to understand the differences between two metamodels.

Current model matching tools may be employed to compute the correspondences between metamodels. However, based on experiments with ontology matching tools [5], it seems challenging to compute correspondences with good quality between heterogeneous metamodels, i.e., to produce a less amount of false negatives and false positives, but at the same time a high amount of true positives. One key in matching heterogenous metamodels is to use an appropriate combination of heuristics for reasoning on the structures of the metamodels as well as additional information such as linguistic knowledge to reason on name heterogeneities. Current tools offer several matching algorithms and their configuration. However, currently less knowledge exists about the performance of these tools for the heterogenous metamodel matching case.

In this paper, we aim at addressing this challenge by proposing a *benchmark* based on *real-world metamodels* that allows to evaluate automatically the *quality* of model matching tools for the case of computing correspondences between heterogenous metamodels. To make this benchmark easily applicable, we provide the correspondences that should be found by the tools in a standardized format coming from the area of ontology alignment [9] that is straight-forward

to produce. The benefit of using this format is the automatic evaluation of the quality of the computed correspondences.

The benchmark is available as open source project at Eclipse labs.[1] We already applied the benchmark successfully to a list of Ontology matching tools back in 2007 [5]. In the meantime, several dedicated model matching tools emerged [6], thus we believe that now it is a good point in time to revive this benchmark proposal. For instance, we recently evaluated EMF Compare that is the out-of-the-box model comparison tool for the Eclipse Modeling Framework (EMF).[2] We provide this evaluation as a reference case how to instantiate this benchmark proposal at our benchmark website.

2 Goal of the Benchmark

The goal of the benchmark proposed in this paper is to provide the basis to compare current model matching tools for the heterogenous metamodel matching case. Further applications are also possible such as finding appropriate configurations of the matchers used within the tools. With the proposed benchmark, we aim at assessing the following questions about model matching tools:

1. *Correctness*: What is the ration between correctly and falsely detected correspondences? This property is also referred to as *precision*.

2. *Completeness*: What is the ration between correspondences to be found and correctly detected correspondences? This property is also referred to as *recall*.

The benchmark should allow an automatic evaluation of the two stated properties. Thus, it has to cover the metamodels to be matched, the correspondences that have to be found, i.e., the gold standard, as well as a component that is able to compare the actual computed correspondences with the expected ones to compute the two stated properties. In the following, we explain how we designed the benchmark to fulfill these three points.

3 Design of the Benchmark

We now provide details about the matching scenarios that are comprised by the benchmark, as well as the measures and how they are computed for giving a statement on the quality of the computed correspondences. Furthermore, we discuss the formats used to represent the metamodels as well as the correspondences.

Matching scenarios. The benchmark consists of ten matching scenarios involving well-known structural modeling languages from the fields of object-oriented modeling and data modeling. These scenarios cover different challenges for model matching tools. While some modeling languages use a very similar terminology for their metamodel elements, other languages are very heterogenous with respect to the used terminology. This is in particular true for scenarios where two languages coming from different modeling fields are matched, e.g., data modeling vs. object-oriented modeling. Furthermore, some metamodels are small ones having simple structures, others are larger metamodels involving complex containment and inheritance structures to represent the modeling languages.

Metamodels. We assembled five different modeling languages to define ten matching scenarios (each scenario matches two different metamodels). In particular, we provide the metamodels of the UML class diagram[3] (version 1.4 and 2.0), the modeling language of the Eclipse Modeling Framework (Ecore)[4], the Extended Entity-Relationship Language (EER)[5], and the data modeling part of the Web Modeling Language (WebML)[6]. For all these metamodels, we provide Ecore-based representations as well as OWL-based representations (that have been automatically produced by our metamodel lifter component [4] to evaluate ontology matching tools as well). It has to be noted that although the UML class diagram version 1.4 and 2.0 may look as a language evolution example, it is actually not the case. The UML 2.0 metamodel has not been defined by modifying the UML 1.4 metamodel, but from scratch.

Table 1 summarizes the main characteristics of the modeling languages by means of counting the metamodel elements according to their types. As the numbers suggest, the metamodels can be categorized by their size in small, medium, and large. UML 1.4, UML 2.0, and Ecore are using inheritance relationships heavily resulting in a large inheritance depth, in contrast to WebML and EER. Furthermore, the UML metamodels make use of multiple inheritance. Finally, Table 1 states the origin of the terminology which is used for naming the metamodel elements. It is worth mentioning that the UML metamodels and Ecore use *object-oriented terminology* (OO for short), in contrast to WebML and EER, which use *database terminology* (DB for short).

Correspondences. For each scenario, we developed manually the set of correspondences that is expected as the result of the manual matching task act-

[1] `http://code.google.com/a/eclipselabs.org/p/m3-benchmark`

[2] `http://www.eclipse.org/emf/compare`

[3] `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`

[4] `http://www.eclipse.org/emf`

[5] based on [1]

[6] `big.tuwien.ac.at/projects/webml`

Table 1: Modeling languages used in the benchmark.

	UML 2.0 CD	UML 1.4 CD	Ecore	WebML	EER
#Class	40	33	18	6	7
#Attribute	18	31	31	8	5
#Containment	23	8	9	3	4
#Reference	52	29	25	4	7
#Enumeration	3	6	0	2	0
#EnumLiteral	11	18	0	15	0
#AllModelElements	158	143	83	53	23
Size	large	large	middle	small	small
Taxonomy					
#SuperClass	17	11	7	1	1
#SubClass	36	28	16	4	2
#Multiple Inheritance	9	3	0	0	0
Inheritance Depth	6	5	5	1	1
Terminology	OO	OO	OO	DB	DB

ing as the gold standard. The correspondences are expressed in the INRIA Alignment Format[7] [3], which is a commonly agreed format for representing correspondences in the ontology matching community. The INRIA Alignment API[8] [2] provide dedicated support to interact with this format.

Listing 1: Example for the INRIA Alignment Format

```
<Alignment>
    <uri1>http://UML1.4</uri1>
    <uri2>http://UML2.0</uri2>
    ...
    <Cell>
        <entity1 rdf:resource='http://UML1.4#
            ↪Class'/>
        <entity2 rdf:resource='http://UML2.0#
            ↪Class'/>
        <measure rdf:datatype='http://www.w3.org
            ↪/2001/XMLSchema#float'>
            0.7479
        </measure>
        <relation>=</relation>
    </Cell>
    <Cell>
        <entity1 rdf:resource='http://UML1.4#
            ↪Element_name'/>
        <entity2 rdf:resource='http://UML2.0#
            ↪NamedElement_name'/>
        <measure rdf:datatype='http://www.w3.org
            ↪/2001/XMLSchema#float'>
            0.8923
        </measure>
        <relation>=</relation>
    </Cell>
    ...
</Alignment>
```

An example for expressing correspondences in the INRIA Alignment Format is shown in Listing 1. The correspondences are collected in one XML file that consists of a set of *Cells*. Each cell is representing one correspondence. The metamodel elements that correspond to each other are referred by the *entity1* and *entity2* elements using an URI. In our example, the *Class* meta-class in the UML 1.4 is mapped to the *Class* meta-class in UML 2.0. Additional meta-information is provided by the *measure* element and the *relation* element. The *measure* element is referring to the confidence that the correspondence holds. This value is of special importance in cases where many correspondences are computed for one metamodel element. The *relation* element is describing the kind of the identified correspondence. While in generic model matching, normally no special type is used, in the metamodel matching case, also more specific types such as sub-type or super-type relationships may be computed. However, in the current benchmark, we use simple equals correspondences, only, and also the measures are not further interpreted.

We defined not only correspondences between metaclasses, but also between their features, i.e., attributes and references. As a convention to refer to features in the alignment files, we use *Class.name.concat('_').concat(Property.name)*. Besides this convention, the cells in the alignment files are analogous for properties as for classes.

Measures. To measure the quality of the matching tools, we reuse measures stemming from the field of information retrieval [8] to compare the manually determined correspondences M (also called expected correspondences) to the automatically found correspondences A. The primary measures are *precision* and *recall*, which are negatively correlated. Thus, we use a common combination of the primary measures, namely *F-measure*.

The measures are based on the notion of *true positives* ($tp = A \cap M$), *false positives* ($fp = A \setminus M$), and *false negatives* ($fn = M \setminus A$). Based on the cardinalities of these sets the aforementioned measures are defined as in [7, 8] as follows:

- $Precision = \frac{|tp|}{|A|} = \frac{|tp|}{|tp|+|fp|}$
- $Recall = \frac{|tp|}{|M|} = \frac{|tp|}{|tp|+|fn|}$
- $F\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$

Precision reflects the share of relevant correspondences among all the automatically found correspondences. *Recall* reflects the frequency of relevant correspondences compared to the set expected correspondences. As these two measures may be thought of

[7] http://alignapi.gforge.inria.fr/format.html
[8] http://alignapi.gforge.inria.fr/index.html

as probabilities, their values may range from 0 to 1, whereas the higher value, the better. *F-measure* takes both precision and recall into account to overcome some over- or underestimations of the two measures. Formally the F-measure is in our case the equally weighted average of the precision and recall measure.

For evaluating the benchmark itself, we reuse the INRIA Alignment API that provides an evaluation framework for correspondences expressed in the INRIA Alignment Format. More specifically, the API is able to compute basic values such as true negatives, false negatives, true positives as well as the aforementioned metrics for a set of expected correspondences and automatically computed ones. How to use the API for our benchmark is demonstrated for the results of EMF Compare on our project website.

Execution. For executing the proposed benchmark, the metamodels have to be matched by the tool under evaluation. This may require to transform the Ecore-based metamodels to tool-specific formats. Normally, a model-to-text transformation can act as an importer. However, by using Ecore, which is the defacto metamodeling standard in the MDE community, we expect that importers are already available in the majority of model matching tools. After running the model matching algorithms for the 10 scenarios, the produced correspondences have to be translated to the INRIA Alignment Format in case no specific exporter for this format is available. Again, a model-to-text transformation could serve for this purpose. In our reference case we also provide such a model-to-text transformation for EMF Compare. The produced correspondences can be automatically evaluated by an Eclipse project also provided at our website by just copying the produced files using pre-defined names to a special folder. More information on this is directly provided in the Eclipse project that is based on the INRIA Alignment API. The output of running the Eclipse project are the mentioned measures for each matching scenario as well as the average values for all scenarios.

4 Conclusion

In this paper, we have presented a benchmark and an evaluation framework for model matching tools considering the heterogenous metamodel matching case. The evaluation framework is generic in the sense that also other (meta)model matching cases may be evaluated, e.g., understanding metamodel evolution by matching metamodels having the same origin. Here we kindly invite the community to contribute further matching cases to cover the broad spectrum of (meta)model matching.

References

[1] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[2] J. David, J. Euzenat, F. Scharffe, and C. Trojahn dos Santos. The Alignment API 4.0. *Semantic Web*, 2(1):3–10, 2011.

[3] J. Euzenat. An API for Ontology Alignment. In *Proceedings of the ISWC'04*, pages 698–712. Springer, 2004.

[4] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting Metamodels to Ontologies - a Step to the Semantic Integration of Modeling Languages. In *Proceedings of MODELS'06*. Springer, 2006.

[5] G. Kappel, H. Kargl, G. Kramler, A. Schauerhuber, M. Seidl, M. Strommer, and M. Wimmer. Matching Metamodels with Semantic Systems - An Experience Report. In *Workshop Tagungsband der Datenbanksysteme in Business, Technologie und Web (BTW) Konferenz*, pages 38–52. Verlag Mainz, 2007.

[6] D.S. Kolovos, D. Di Ruscio, A. Pierantonio, and R.F. Paige. Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing. In *Proceedings of the CVSM Workshop @ ICSE'09*, pages 1–6. IEEE, 2009.

[7] D.L. Olson and D. Delen. *Advanced Data Mining Techniques*. Springer, 2008.

[8] G. Salton and D. Harman. *Information retrieval*. Wiley, 2003.

[9] P. Shvaiko and J. Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2013.