Matching Metamodels with Semantic Systems – An Experience Report*

Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber,†
Martina Seidl, Michael Strommer and Manuel Wimmer
Business Informatics Group
Womens Postgraduate College of Internet Technologies
Institute for Software and Interactive Systems
Vienna University of Technology, Austria
Email: lastname@big.tuwien.ac.at

Abstract: Ontology and schema matching are well established techniques, which have been applied in various integration scenarios, e.g., web service composition and database integration. Consequently, matching tools enabling automatic matching of various kinds of schemas are available. In the field of model-driven engineering, in contrast to schema and ontology integration, the integration of modeling languages relies on manual tasks such as writing model transformation code, which is tedious and error-prone. Therefore, we propose the application of ontology and schema matching techniques for automatically exploring semantic correspondences between metamodels, which are currently the modeling language definitions of choice. The main focus of this paper is on reporting preliminary results and lessons learned by evaluating currently available ontology matching tools for their metamodel matching potential.

1 Introduction

The rise of the *Semantic Web* [BLHL01] influences many areas of computer science. Not only the web engineering community feels the need for semantically enhanced technologies to increase the World Wide Web's machine processability but also researchers from a variety of fields follow this stream. Ontologies, which originate from knowledge representation, have experienced their renaissance and have become a mainstream research area. Nowadays, they are considered a very promising approach to handle any kind of information. They provide a vocabulary to describe a domain of interest and a specification of the used terms' meaning. Much effort is spent on the development of tools to efficiently process ontologies, including ontology creation tools, query tools, matching tools, and reasoning tools.

At about the same time the notion of "The Semantic Web" was born, a new software ap-

*This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.0

†This research has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

proach 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 become first-class citizens in the software development process. Models provide a very high level of abstraction on the one hand, on the other hand they are very well suited for visualization. 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. 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 very cumbersome, error-prone, and repetitive task if performed manually. An approach to tackle this problem at least semi-automatically is to consider the metamodels of the modeling languages, to identify correspondences between them in order to identify concrete mappings of metamodel concepts. To the best of our knowledge, there is no explicit tool support for matching metamodels.

If we take a look at the field of ontology engineering and semantic web, we find a multitude of techniques developed for various integration purposes like *ontology matching* that try to identify semantic correspondences between ontologies. Ontology matching supports handling of heterogeneous ontologies. The approach originates from former application domains of matching systems like database or XML schema integration. As metamodels and ontologies are closely related, it seems obvious to use those tools for the identification of semantic correspondences between metamodels.

Thus, we propose a novel application domain for traditional ontology and schema matching systems: the integration of modeling languages via their metamodels. The contributions of this paper are twofold: (1) we provide a collection of metamodels represented as ontologies, namely the metamodels of the UML class diagram (version 1.4 and 2.0), the modeling language of Eclipse's modeling framework (Ecore), the extended entity-relationship language (EER), and a web modeling language (WebML). (2) We report on our experiences when applying a set of ontology matching tools for metamodel matching and specifically focus on the tools' suitability for this task.

The remainder of this paper is structured as follows: The next section gives an overview of the model-driven engineering and ontology engineering technical spaces, and covers the description of our proposed Metamodel Matching Framework. Section 3 describes the setup and results of our metamodel matching experiment. Section 4 summarizes our lessons learned. Finally, conclusions and future work concerning specific extensions for metamodel matching are given.

2 On Models, Metamodels, Ontologies, and How They Match

We propose a framework based on ontologies for the matching of metamodels. The intention of using ontologies as means for representing metamodels is to benefit from the wide range of tools available for ontology processing. Roughly speaking, our framework consists of 2 components: (1) the *ModelWare* and (2) the *OntoWare*, namely the part which deals with models and metamodels and the part which is about the ontologies. Before going into the details of our framework, we introduce some basic terminology and concepts.

2.1 ModelWare and OntoWare

Recently, *Model Driven Engineering* (MDE) [AK03] has received considerable attention and is well on the way to become the new paradigm for software engineering. In MDE, models replace code as the primary artefacts in the software development process. Now developers are forced to focus on modeling the problem domain and not on programming one possible (platform-specific) solution. Thus, the abstraction from specific programming platforms by modeling at a platform-independent level and the definition of model transformations allow for the generation of platform-specific implementations. The *Model Driven Architecture* (MDA) [Gro03] by the Object Management Group (OMG) is a very prominent example of MDE. MDA is based on the OMG's standards, amongst them its modeling language, *The Unified Modeling Language* (UML) [Gro05], and its metamodelling language, namely the *Meta Object Facility* (MOF) [Gro04].

The usage of diverse domain-specific modeling languages and the usage of diverse versions of the same language rise the need for language integration in order to ensure interoperability between different tools. The first task in the integration process consists of finding semantically equivalent modeling concepts between two languages, i.e., finding equivalent metamodel elements. This task is known as *model weaving* and is typically done manually. The result of the model weaving task is a *weaving model* which incorporates all semantically equivalent links between elements of the metamodels. These weaving models can be used as input for deriving model transformations to realize the operational integration of models. Still, there are no approved attempts for the automation of the model weaving task and the derivation of model transformation code.

In the last few years, *ontologies* have gained enormous attention as they are considered as a very promising element in the creation of semantically enhanced technologies. The notion ontology subsumes a variety of terms and definitions. Ontologies aim to capture the consensual knowledge of a given domain in a generic and formal way [CFLGP06].

As the same domain knowledge can be modeled in various manners by different ontologies, it is necessary to find ways to detect and to express correspondences. *Ontology matching* is the task of manually, automatically or semi-automatically find the semantically equivalent elements between two ontologies. It can be seen as a set of rewriting rules which associates the elements of a source ontology with the elements of the target ontology. The result of the matching process is called *ontology mapping*. And this is our

starting point – we want to use those tools for the machting of metamodels.

What exactly is the difference between an ontology and a metamodel? Even though a detailed discussion of this question is out of scope of this paper, we will shortly reflect our point of view. Confusion between the terms ”ontology” and ”model” or ”ontology” and ”metamodel” arises easily as on the first glance the differences between those concepts do not appear very severe. One possible way to arrange the terms ”ontology” and ”model” is based on Lassila's and McGuinness' spectrum of ontologies [LM01] and the 3D matrix by A. Gruber et. al. [GWG06]. According to those authors ontologies range from simple informal catalogs over more formal schema representations to heavy-weight logic-based ontologies. Modeling languages like UML, ER, etc. can be seen as formal schemas and thus can be treated as ontologies. The difference lies in the expressiveness of the formalism. Due to this difference it is possible to make a more fine grained distinction between different kinds of ontologies.

Ontologies and metamodels can be distinguished from their domain of discourse. Ontologies are mostly used for modeling real world domains or systems by means of a schema and describing real world entities by means of individuals. In contrast, metamodels are used for defining modeling languages which are used to describe real world domains or systems. This means that the instances of metamodels are not representing real world entities, instead the instances are models. Furthermore, metamodels consist of containment relationships between elements, which describe the data structure for storing the models, however, this data storage aspect is not covered by currently used ontology languages.

2.2 Bridging ModelWare and OntoWare

As we have already discussed above, the integration of modeling languages is an absolute must in the technical space of the ModelWare. However, there are no automatic matching techniques available and therefore, the integration task is currently performed manually. Before implementing new methods for metamodel matching, which we see as a special kind of schema or ontology matching, we want to explore the already existing matching techniques of the OntoWare technical space. Thus, we propose a metamodel matching framework which is based on a transition from ModelWare to OntoWare by means of transforming Ecore-based [BSM+03] metamodels into OWL-based ontologies. After achieving this transition, we can reuse the ontology matching tools which process the ontologies actually representing the metamodels. After performing the matching task, we translate the ontology mapping into a weaving model. From this weaving model, we are able to derive the actually needed transformation rules to transform models conforming to the metamodel A into models conforming to the metamodel B. Note that the OntoWare layer is completely transparent to the user who can simply focus on the ModelWare.

Figure 1 illustrates the architecture of our framework on the left-hand side and provides a concrete example on the right-hand side where the lifting of a concrete metamodel (namely, the metamodel of the UML class diagram) to an ontology is shown. Detailed information on the lifting process can be found in [KKK+06].

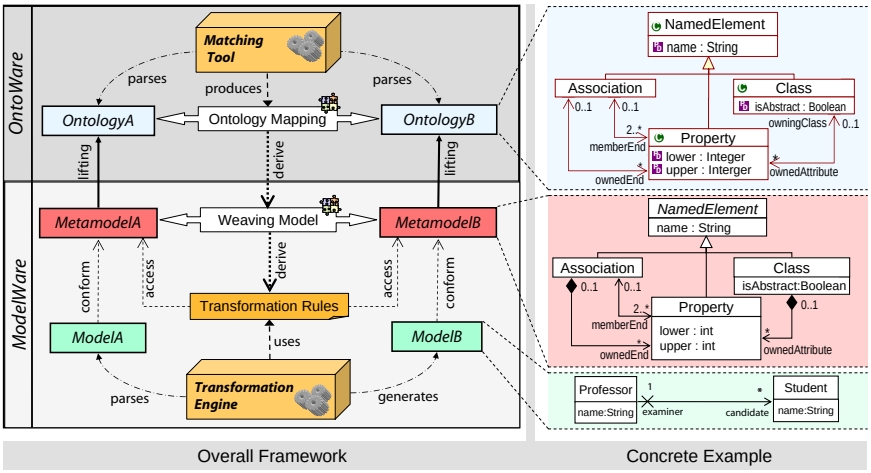

Figure 1: Metamodel Matching Framework

3 Metamodel Matching Experiment

This work is focused on the evaluation of ontology matching tools with respect to their suitability for metamodel matching. In the following, we discuss the selected matching systems, the kind and structure of the used metamodels, how we measure the quality of the matching results, and, finally, the results of the experiment.

3.1 Selected Matching Systems

In this work, we concentrate on evaluating *schema-based* matching tools, i.e, we do not consider *instance-based* matching techniques. This is due to the fact that we are using the data provided by metamodels (*schema-level*) and not data from models (*instance-level*) for finding equivalences between metamodel elements. The comprehension of instance-based matching techniques that is, the reasoning about models conforming to metamodels, remains a subject to future work. Another requirement the matching tools must fulfill is that they must at least allow to identify simple *one-to-one* equivalences between two ontologies. The last requirement is that the tools must be capable of reading in OWL input files. Currently, most tools support OWL Lite, only, which is not sufficient for metamodels represented in OWL. In particular, OWL DL is needed when using enumerations and cardinality restrictions greater than 1 in the metamodels. External information resources, such as dictionaries, thesauri or taxonomies, are very promising and would help to increase the quality of the automatically computed mappings. However, for our experiments we do not take advantage of domain-specific external information resources. As with the instance-based matching techniques, the definition of such external resources and the ap-

plication as well as user interaction during the matching task is left to future work. In view of this criteria the following four tools were chosen: Alignment API [Euz04], COMA++ [ADMR05], CROSI [KHRS05], and FOAM [ES05].

3.2 Measures for the Quality of Matching

To measure the quality of the matching tools, we reuse measures stemming from the field of information retrieval to compare the manually determined matches *M* (also called relevant matches) to the automatically found matches *A*. The primary measures are *precision* and *recall* [SM87], which are negatively correlated. Thus, we use a common combination of the primary measures, namely *F-measure* [vR79].

The measures are based on the notion of *true positives* (*tp* $= A \cap M$), *false positives* (*fp* $= A \cap \bar{M}$ *where* $\bar{M} = |tn| + |fp|$), and *false negatives* (*fn* $= M \cap \bar{A}$ *where* $\bar{A} = |fn| + |tn|$). *tn* stands for *true negatives*. Based on the cardinalities of these sets the aforementioned measures are defined similarly as in [SM87], [vR79] as follows:

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

Precision reflects the share of relevant matches among all the automatically retrieved matches given by A. This measure can also be interpreted as the conditional probability $P(M/A)$. A higher *precision* means that the matches found are more likely to be correct. If the number of false positives equals zero, all matches are to be considered correct.

Recall reflects the frequency of relevant matches compared to the set of relevant matches M. Again this measure can be expressed as a conditional probability which is given by $P(A/M)$. A high *recall* states that nearly all relevant matches have been found. Nothing is said about wrong matches contained in A.

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.

3.3 A Testset of Structural Modeling Languages

For the elaborate evaluation of the selected matching tools we developed a testset which consists of five structural modeling languages. These are frequently used in software en-

gineering, namely *UML 2.0*[1] (class diagram part), *UML 1.4*[2] (class diagram part), *Ecore*[3], *WebML*[4] (content model part) and *EER*[5]. 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 large, middle and small. Furthermore, Table 1 summarizes the main characteristics of the taxonomy expressed in the metamodels. UML 1.4, UML 2.0 and Ecore are heavily using inheritance relationships resulting in a large inheritance depth, in contrast to WebML and EER, which are only using some inheritance relationships resulting in a maximum inheritance depth of one. 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. In this context, it must be mentioned, that the UML metamodels and Ecore use *object-oriented terminology*, in contrast to WebML and EER, which use *database terminology*.

Table 1: *Testset*: Structural Modeling Languages

	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

These five metamodels are automatically transformed into OWL ontologies by our metamodel lifter component [KKK+06]. The metamodels (expressed in Ecore) and the corresponding ontologies (expressed in OWL) can be found at our *ModelCVS* [6] project web site. After establishing the metamodels and ontologies, we defined 10 matching scenarios (each scenario matches two different metamodels) and furthermore, we developed manual mappings for each scenario. In addition to the metamodels and ontologies, the mappings between the ontologies expressed in INRIA Alignment Format [Euz04] can also be found at the *ModelCVS* project site.

Figure 2 shows an excerpt of the Ecore and UML 2.0 metamodel in UML class diagram notation (cf. upper half of Figure 2) and as trees (cf. lower part of Figure 2). In addi-

[1] available at http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML
[2] available at http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML
[3] available at http://www.eclipse.org/emf
[4] available at http://www.big.tuwien.ac.at/projects/webML
[5] based on [Che76]
[6] http://www.modelcvs.org/

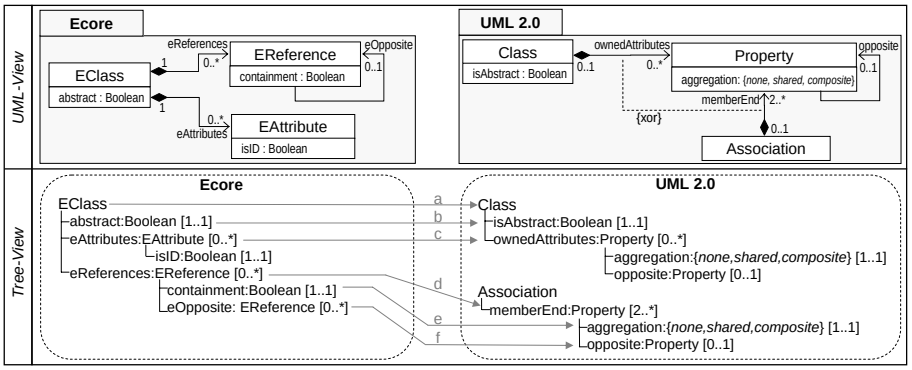

Figure 2: Mapping Example: Ecore to UML 2.0

tion to the metamodel elements, the lower part of Figure 2 illustrates the mappings *a-f* between the two metamodels. In Ecore *EClasses* can be abstract or concrete (cf. attribute *abstract*) and own a collection of *EAttributes* and a collection of *EReferences*. While *EAttributes* represent intrinsic attributes of *EClasses*, *EReferences* represent relationships between *EClasses* and may be composition relationships (cf. attribute *containment*). In UML *Classes* own *Properties* representing intrinsic attributes. However, *Properties* can also be owned by *Associations*. In such cases, *Properties* represent association ends (cf. *memberEnd* in Figure 2) also called roles. *Associatons* may be of type simple, aggregation or composition depending on the attribute *aggregation* of the contained *Properties*. The *xor*-constraint between *ownedAttributes* and *memberEnd* ensures that a property is either an attribute or an association end but not both at the same time. This *xor*-constraint leads to the duplication of the class *Property* in the tree-based view.

Mappings *a* and *b* in Figure 2 represent clear *one-to-one* correspondences. Mapping *a* specifies the equivalence between *EClass* in Ecore and *Class* in UML. Mapping *b* defines that the attribute *abstract* of *EClass* is equivalent to the attribute *isAbstract* of *Class*, in particular also data types and multiplicities of the two attributes match exactly. Mapping *c* defines that the reference *eAttribute* of *EClass* is equivalent to the reference *ownedAttributes* of *Class*. However, the attributes of the classes *EAttribute* and *Property* do not match at all. Mapping *d* defines that the reference *eReferences* is equivalent to *memberEnd*. In this case, the features of the classes *EReference* and *Property* are mappable (cf. Mapping *e* and *f*). However, two problems are associated with mapping *d* and *e*. First, the attribute *containment* of *EReference* is of type Boolean and the attribute *aggregation* of *Property* is of type Enumeration with the values *none*, *shared* and *composite*. This mapping requires a special integration rule, namely how the boolean values *true* and *false* are mapped to the three enumeration literals. Second, there is no mapping from an Ecore concept to the class *Association* of UML 2.0 resulting in the problem of missing container objects, when transforming *eReferences* to *memberEnds*, because properties which are *memberEnds* must be owned by an association. This circumstance further leads to the problem that the generated model is not conform to the associated metamodel.

In general, the Ecore and the UML 2.0 metamodel offer nearly equivalent modeling concepts, but in particular some heterogeneities, e.g., linguistic and structural, exist. Furthermore, this simple mapping example already points out that *zero-to-one* mappings are quite common between metamodels and complicate the derivation of model transformation considerably. Further information on *zero-to-one* mappings and solutions for deriving transformations in the field of schema integration may be found in [LN07].

3.4 Experimental Results

Due to lack of space, Figure 3 shows the results of the ontology matching tools used with their default settings. All experimental results are available at the *ModelCVS* project web site.

Figure 3 gives a detailed overview of the results gained with the default settings of the tools illustrated as a star glyph. Each axis of the glyph represents a mapping of two metamodels. The three values represent *precision*, *recall* and *F-measure*. Figure 3 allows for an easy comparision of the matching quality of the four tools and furthermore can identify which integration tasks deliver good or bad results. Some tools did not find any results for some matching tasks, therefore some axis of the star glyph are empty. The inner gray ring in Figure 3 describes the value of 0.5, which is in particular the most interesting value for the *F-measure*. An *F-measure* higher than 0.5 indicates a positive benefit, a value lower than 0.5 means a negative benefit.

In the following, we discuss the best and worst cases for our three measures when using the tools with their default settings. The highest precision value was achieved with *AlignmentAPI* for *UML1.4* to *UML2.0* (*precision*=0.96; *recall*=0.40; *F-measure*=0.57). The best *recall* and *F-measure* value was achieved with *COMA++* for *UML1.4* to *UML2.0* (*precision*=0.63; *recall*=0.58, *F-measure*=0.61).

To calculate an average value of *precision*, *recall* and *F-measure* we set the measurement values to zero for the cases when the tools did not deliver results. The average is calculated as the arithmetic mean by summarizing all values and divide it by the number of possible matching results (10). Each measure has a range between zero and one therefore the assumption of zero for the tools which did not find any result is passable.

The best average *precision* was achieved by *CROSI* (*precision*=0.33; *recall*=0.15; *F-measure*=0.18) and *FOAM* (*precision*=0.33; *recall*=0.09; *F-measure*=0.14). 33% of the found mappings of *CROSI* and *FOAM* are correct but *CROSI* found 15% of all possible and correct mappings instead of *FOAM* with 9%. Therefore, the benefit measured in terms of *F-measure* is better for *CROSI* (0.18) than for *FOAM* (0.14). The best average *recall* and *F-measure* was achieved by *COMA++* (*precision*=0.26; *recall*=0.25; *F-measure*=0.25). With the *precision* of 0.26 and recall of 0.25 we have a balanced result.

In our evaluation we have also experimented with modified settings of the tools resulting in 13 different tool settings (four tools with standard settings + nine modifications). The best *precision* of the nine modifications was achieved for *UML1.4* to *UML2.0* with modified *CROSI* (*precision*=1; *recall*=0.29; *F-measure*=0.45). Compared to the results of

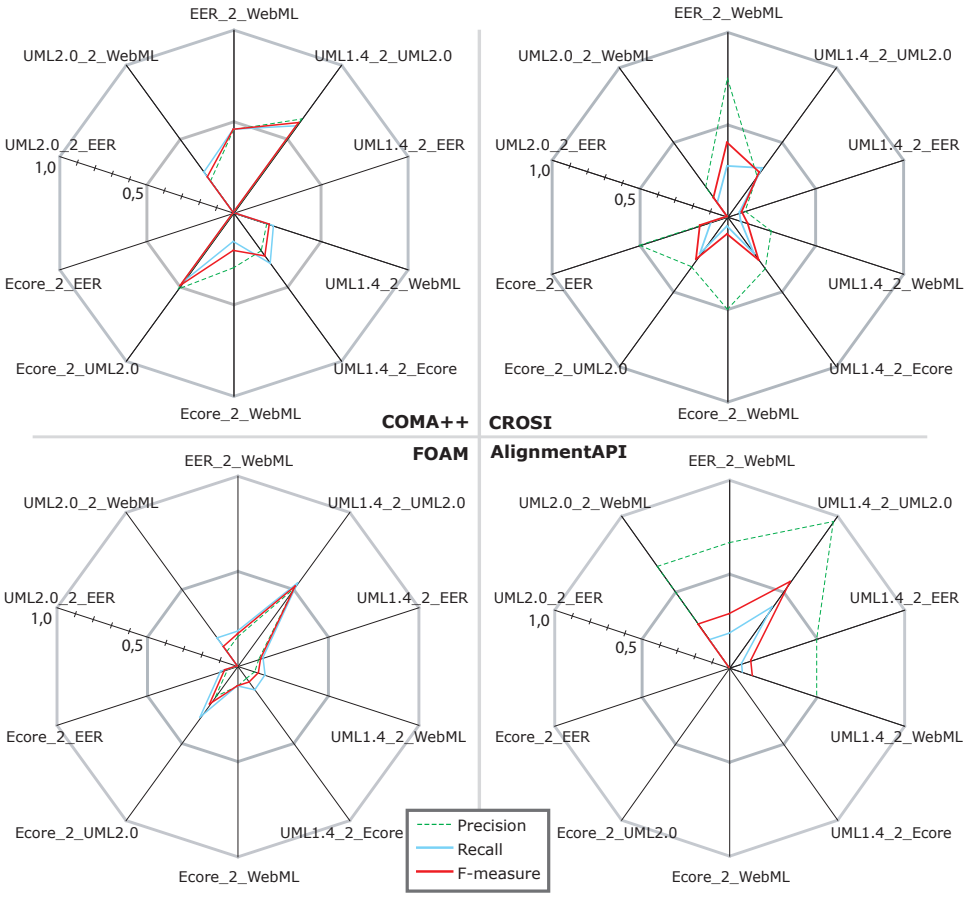

Figure 3: Matching results (standard settings)

CROSI with standard settings (*precision*=0.27; *recall*=0.32; *F-measure*=0.30) we can say that the modifications achieved a much better result. All found mappings were correct and only 3% less of all possible correct mappings were identified as with *CROSI's* default settings. The best *recall* and also the best *F-measure* was achieved with modified *COMA++* for *UML1.4* to *UML2.0* (*precision*=0.69; *recall*=0.57; *F-measure*=0.62), although these results do not differ so much from the results of *COMA++* with default settings (*precision*=0.63; *recall*=0.58; *F-measure*=0.61).

If we examine the average of the three measures of the 13 tool versions, the best *precision* over all integration scenarios was achieved with modified settings in *CROSI* (*precision*=0.83), but the *recall* and *F-measure* are very low (*recall*=0.12 and *F-measure*=0.20). The found mappings are mostly correct, but there were many mappings left that were not found by the tool. The best *recall* was achieved with a modified version of *COMA++* (*recall*=0.33). Despite of the low *precision* (*precision*=0.37), it leads also to the best *F-*

measure (*F-measure*=0.34). The benefit of the matching results of *COMA++* is much higher as the benefit of *CROSI* with its high *precision*. Summarizing, it can be said that modifying the settings often leads to a better *precision*. However, the higher the *precision* gets, the lower is the value of the *recall*, i.e., it is hard to increase the *F-measure* value considerably.

3.5 Conclusion of the Experiment

Besides the different modifications and combinations of different matching techniques which produce higher *precision* values but at the same time lower the *recall* values, the metamodels must fulfill some requirements in order to prove that matching tools are worth using. The 10 integration scenarios showed that the metamodels must have a common *terminology* and *taxonomy*, which is the case when matching *UML 1.4*, *UML 2.0* and *Ecore*. These combinations lead to the best results despite their size which obviously lead to a higher number of elements that have to be matched. Furthermore, good results are achieved when matching *WebML* with *EER*. These two metamodels also have a common terminology and both do not heavily use inheritance relationships. In contrast, matching *WebML* or *EER* with *UML 1.4*, *UML 2.0* or *Ecore* results in a very low *precision* and in a very poor *recall* which is mostly below 0.10. These results lead to the conclusion that ontology matching tools are not always appropriate for matching metamodels. Instead, the metamodels must fulfill some common properties which of course is not always the case when matching real-world metamodels.

4 Lessons Learned

In this section we present our lessons learned of the metamodel matching experiment presented in the previous chapter. In particular we discuss on the one hand the problems we have faced with the input schemas, produced mappings, and matching tools and on the other hand the usefulness of the generated mappings for deriving executable model transformation rules.

No support for enumerations: Enumerations frequently occur in metamodels, representing a domain of constants (expressed as Literals) which are used for typing attributes. However, Enumerations are only supported in *OWL DL* and not in *OWL Lite*. Currently, most matching tools support OWL Lite, only, thus ignoring Enumerations, Literals and the type information of Enumeration-typed attributes. The matching scenarios showed, that mappings between Literals of Enumerations are very helpful for deriving the required model transformation rules, because it must be exactly specified *how* the data is actually transformed. Furthermore, the mappings between Literals are often straight forward, i.e., simple *one-to-one* correspondences. Typically, in the UML 1.4 to UML 2.0 mapping scenario many correspondences between Literals and Enumerations are present.

Different modeling styles for Boolean-typed attributes: Particulary, in the *UML 1.4* to *UML*

2.0 matching scenario the mapping between Boolean-typed attributes and Enumeration-typed attributes is often required. This is due to the fact, that the majority of Enumerations consisting of two Literals can be represented as Boolean. For instance, the enumeration *ordering* = {*unordered, ordered*} is used for the type of the *order* attribute in UML 1.4. In UML 2.0 the equivalent attribute *isOrdered* is defined as a Boolean. In such cases the mappings between the possible values of the attributes must be also considered, i.e., the right value combinations must be identified. For example, the Literal *unordered* is mapped to *false* and the Literal *ordered* to *true*.

Need for value correspondences: The mapping examples showed that it is not possible to derive all necessary information for the model transformations based on the metamodel information, only. Besides metamodel definitions, instance values must be considered which are required for defining the model transformation rules. The following problem case exemplifies this requirement. In *Ecore* the *upper cardinality* has the value *-1* if the upper cardinality (of an association end) is unrestricted. In *WebML* the *upper cardinality* is defined as *N* for representing a unrestricted upper cardinality. Consequently, for the transformation rules not only the equivalent attributes are necessary. In addition, also the equivalent values of the semantically equivalent attributes are required for transforming the models. However, metamodels are not including such technical encoding conventions, hence, there is no way to reason about equivalent values on metamodel level. These values are contained in the models, only. Consequently, instance-based approaches are very interesting for deriving such value correspondences. Such an approach must reason about instances of the metamodels, i.e., comparing elements of models which conform to their metamodels.

Metamodel versioning is the winner: The matching scenario *UML 1.4* to *UML 2.0* is a special scenario, namely an example for *Metamodel Evolution*. This task offers the best matching quality in terms of used measures due to the high degree of name similarities between UML 1.4 and UML 2.0, one can say both share a common terminology. In particular, most of the transformation rules can be automatically derived from the ontology mappings for transforming models conforming to the smaller UML 1.4 metamodel into models conforming to the larger UML 2.0 metamodel. Only a few more complex transformation rules are necessary for the integration and are established manually.

Mappings between different types: The manually established mappings in the experiment showed that mappings between different types (e.g., *Attributes_2_References*) are quite common. These kind of mappings, however, represent one of the most challenging problems regarding their transformation into executable model transformation code. For instance, when deriving a transformation rule from an *Attribute_2_Reference* mapping, one must reason about how the value of the attribute is transformed to an object reference and how the value of the attribute is expressed within the referenced object. For instance, such a mapping is needed in the *UML 1.4* to *UML 2.0* integration scenario. In UML 1.4 the default value of an attribute is described as an attribute of the class *Attribute*. In contrast, in UML 2.0 the default value is not modeled as an attribute, instead it is modeled as a reference to the class *ValueSpecification*.

Name equivalence does not necessarily equate with conceptual equivalence and vice versa: When building our manual mappings, we found out that in some cases metamodel ele-

ments have the same name, but their semantics are quite different and the elements should not be mapped by equivalence links. One prominent example is the class *EnumLiteral* of the *Ecore* metamodel. *EnumLiteral* has an attribute *value* and also an attribute *name*. The *WebML* metamodel contains a class *DomainValue* which is semantically equivalent with *EnumLiteral* and has an attribute *value*. However, the mapping between *EnumLiteral.value* and *DomainValue.value* is not correct, instead *EnumLiteral.name* should be mapped to *DomainValue.value*. This is due to the fact that *EnumLiteral.value* is only a running counter for the literals. Instead, *EnumLiteral.name* and *DomainValue.value* both present constant values of an Enumeration. This case is not solvable without additional knowledge or exploring instances of the metamodels, i.e., the models.

No common taxonomy of modeling concepts: Building the manual mappings showed, that there is not a high heterogeneity between the concrete classes of metamodels, but between the abstract classes. Moreover, the design of the taxonomies in metamodels is mostly artificial and not based on a common ground resulting in different design possibilities. A very interesting case is the following example. When mapping *Ecore* to *UML 2.0* both metamodels have an abstract class *StructuralFeature* (in Ecore called *EStructuralFeature*, because Ecore uses always an 'E' as *prefix* for class names). When looking at the attributes of the two classes, it is obvious identifiable that these two classes do not share even one semantically equivalent attribute. This problem raises two important questions. First, is it appropriate to flatten the taxonomy hierarchy, duplicate all properties in the subclasses and delete the abstract classes before doing the matching task? This is possible, because abstract classes have no instances on the model level and this means there are no transformation rules required for this kind of classes. Second, should we map classes with *name equivalences* as *semantically equivalent* or at least as *semantically related* when they do not share the same properties? This means, is the terminology more important than the structural properties, or vice versa? For model transformations the structural properties are more important, because they must be set in the transformation rules. However, from an ontological point of view this question might have a different answer.

Are Metamodels no typical Ontologies? Summarized, it can be said that metamodels are composed of much more relationships between concepts and less attributes compared to typical input schemas for ontology matching tasks. Typical input schemas consist of a big taxonomy of concepts with attributes. However, relationships are often not present or only to a minimal extend. Our metamodel matching experiments showed that finding correspondences between relationships is the hardest task, however, the differentiation of containment and reference relationships in the reasoning tasks are promising, because metamodels make heavy use of containment references which are also very important information for the model transformation definitions. Furthermore, our matching experiments showed that the similarity between the different metamodels, even though they are in the same domain, is much lower than for typical matching scenarios such as used in the ontology matching contest [7].

[7] http://oaei.ontologymatching.org/2006/

5 Conclusion and Future Work

In this paper, we reported on our experience using ontology matching tools for producing semantic correspondences between metamodels. Summarizing it can be said, that the metamodels must have some common properties when matching tools should be used for the integration task, such as a common terminology or taxonomy. For the evaluation of currently existing ontology matching tools, we developed a framework for measuring the quality of automatically produced mappings which consists of five real-world metamodels (expressed in *Ecore* and *OWL*), mappings between them, and a tool for computing the match quality in terms of well-established measures. This framework is freely available on our project site and we hope that this would stimulate further research in the field of metamodel matching. This means, the framework should be reused from third parties to evaluate their ontology matching tools. The only precondition is an adapter which converts the produced results into the INRIA Alignment Format [Euz04]. The objective is that the most promising technique can then be implemented in the ModelWare technical space and directly executed on the metamodels.

Moreover, we have presented several lessons learned and open issues which are subject to future work for us in order to improve the metamodel matching potential. In particular, three questions are of special interest. (1) How to produce a reasoning graph for metamodel definitions? Is the resulting graph of the OWL definitions enough or do we need other types, e.g., are the lost *containment relationships* needed for deriving important mappings? (2) Why are the results of the ontology matching contest much better than our results – is there a big difference in the method or in the testsets? The last question would further lead to another, more general question: (3) Is there a structural difference between ontologies and metamodels? A possible next step would be the evaluation of ontology matching tools which support the flexible combination of different matching techniques, such as COMA++ supports. In this respect, we will have a look at different possibilities of combinations and find out which one is performing best for our testset.

6 Acknowledgments

We thank Jürgen Bauer and Patrick Größ for supporting us with the evaluation of various ontology matching tools and for setting up parts of the evaluation framework.

References

[ADMR05] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and Ontology Matching with COMA++. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pages 906–908. ACM Press, 2005.

[AK03] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific America*, pages 34–41, May 2001.

[BSM+03] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2003.

[CFLGP06] Oscar Corcho, Mariano Fernandez-Lopez, and Asuncion Gomez-Perez. Ontological Engineering: Principles, Methods, Tools and Languages. In *Ontologies for Software Engineering and Software Technology*. Springer, 2006.

[Che76] Peter Pin-Shan Chen. The Entity-Relationship Model-Toward a Unified View of Data. In *ACM Transactions on Database Systems*, volume 1, pages 9–36. ACM, March 1976.

[ES05] Marc Ehrig and York Sure. FOAM - Framework for Ontology Alignment and Mapping - Results of the Ontology Alignment Evaluation Initiative. In *Integrating Ontologies*, 2005.

[Euz04] Jérôme Euzenat. An API for Ontology Alignment. In *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, pages 698–712. Springer, 2004.

[Gro03] Object Management Group. MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, June 2003.

[Gro04] Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification. http://www.omg.org/docs/ptc/04-10-15.pdf, October 2004.

[Gro05] Object Management Group. UML Specification: Superstructure Version 2.0. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.

[GWG06] Andreas Gruber, Rupert Westenthaler, and Eva Gahleitner. Supporting Domain Experts in Creating Formal Knowledge Models (Ontologies). In *Proc. of the 6th International Conference on Knowledge Management (I-KNOW 2006)*, pages 252–260, 2006.

[KHRS05] Yannis Kalfoglou, Bo Hu, Dave Reynolds, and Nigel Shadbolt. CROSI Project, Final Report. Technical Report 11717, School of Electronics and Computer Science, University of Southampton, Southampton, UK, 2005.

[KKK+06] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting Metamodels to Ontologies - a Step to the Semantic Integration of Modeling Languages. In *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MODELS 2006)*. Springer, 2006.

[LM01] Ora Lassila and Deborah McGuinness. The Role of Frame-Based Representation on the Semantic Web. In *Linköping Electronic Articles in Computer and Information Science*, volume 6, 2001.

[LN07] Ulf Leser and Felix Naumann. *Informationsintegration*. dpunkt.verlag, 2007.

[SM87] Gerard Salton and Michael J. McGill. *Information Retrieval. Grundlegendes für Informationswissenschaftler*. McGraw-Hill, 1987.

[vR79] Cornelis Joost van Rijsbergen. Information Retrival. http://www.dcs.gla.ac.uk/Keith/Preface.html, 1979.