Towards combining Model Matchers for Transformation Development

Konrad Voigt

SAP Research CEC Dresden, Chemnitzer Str. 48, 01187 Dresden, Germany

Abstract. The theory of model transformation has been studied extensively during the last decade and is now well understood. Several transformation languages have been developed and implemented. Recently, model matching has been proposed to offer support for transformation development. The task of model matching aims at finding semantic correspondences between model elements, thus facilitating semi-automatic mapping generation.
However, current model matching approaches mostly concentrate on label-based model similarity and are isolated. Further, they show deficits with respect to quality, performance and language independence.
We tackle these issues by proposing a novel approach using a combination of matchers in a common framework. Thereby, schema matching techniques are adapted and extended to suit our needs. Complementing our approach, we propose model specific matchers addressing new aspects of similarity. Our configurable framework allows an interpretation of combined matching results, thus increasing the number and quality of mappings found.

1 Introduction

Model transformation in the context of Model Driven Development (MDD) has been widely studied in the past. During the last years several transformation languages have been proposed resulting in different engines. According to [1], a model transformation in context of MDD is ”the process of converting one model to another model of the same system”. This process is performed according to a transformation definition, which we also refer to as *mapping*. A mapping describes the way a source model is transformed to a target model. Transformation languages are supported by tools like smart editors and debuggers. However, the languages face the problem of their complexity. They are powerful in expressiveness but lack simplicity.

Considering a set of models which have to be transformed, the following steps are part of developing a transformation. First, a mapping has to be identified, then it has to be specified in a transformation language, which is finally executed. Mappings consist mostly of one-to-one relations and common patterns, such as nesting or concatenation of elements. Applying proposal generation for mappings accelerates the task of transformation development and reduces errors and effort in implementing transformations [2]. In the last two years *model matching* (also named metamodel alignment) has been proposed as an approach for supporting mapping specification. The term model matching refers to an identification of semantic correspondences between metamodel elements and originates from the fields of database- and XML-schema matching [3–5]. Since schemas and metamodels share similarities, model matching levers the concepts to metamodels. It can be applied by generating proposals for mappings based on matching results [6, 2, 7].

Despite their feasibility today's model matching approaches leave room for improvement in terms of quality and performance (execution time) of the matching process. Furthermore, the proposed generation of transformations is specific to one language, thus constituting no generic approach. The generated transformations also lack an optimization in order to increase readability and usability for a transformation developer.

In this paper we demonstrate a novel approach on model matching-based on a framework combining different matching techniques, also called *matchers*. Additionally, we propose combining several matchers for an improved matching result. These techniques differ from existing ones in taking additional information into account, like instances, existing mappings, graph structures etc., thus allowing for an increased match quality for an automatic approach. Therefore, we position our work in the middle, shifting the approach closer to an automatic model mapping than existing approaches as can be seen in Figure 1. It depicts today's approaches on mappings, which are manual. The semi-automatic way is illustrated by the diamond in the middle, which we address to shift more towards the automatic way.

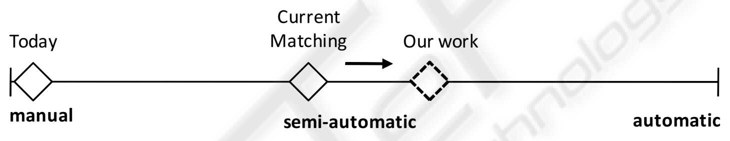

Fig. 1. Degree of manual involvement in mapping specification.

We structure our paper as follows; in section 2 we show two motivating examples for model matching for model transformation. The subsequent section 3 deals with the issues in current model matching addressing the related work; followed by a description of our approach proposing a matcher combination framework for matching improvement (section 4). We finally summarize and conclude our paper in section 5.

2 Motivating Examples

In the following we will show two motivating examples for model transformation and matching-based support in model transformation development.

2.1 Service Engineering

Our example is positioned in the area of model driven service engineering and service marketplaces [8]. Consider two parties offering services to potential customers. For example the company SAP[1] and IBM[2]. Both use a proprietary approach of describing a service. Assuming each of these approaches is model based, hence the companies use their own tools and metamodels for offering and describing services. This setting is depicted in Figure 2, illustrating the two parties, a partial description of a service and the services themselves (instances of the description).

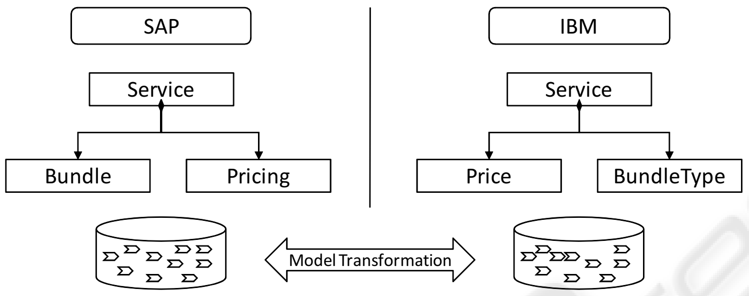

Fig. 2. Example of model transformation as integration of different service descriptions.

Since both companies are part of a competitive market, they do not agree on a common description language nor adjust their tools to external descriptions. One can see, that the same entity of information is described – a service. There are common attributes of a service like bundling and pricing. However, the descriptions are slightly different in name, structure, type etc. In order to enable SAP to offer a service in IBM's marketplace, a service's description needs to be transformed. This can be done using a model transformation based on metamodels. Hereby model transformation development can be supported by model matching, because the metamodels have similarities in name, structure, type etc. While finding similarities between the metamodels, transformation proposals can be generated and ease the development of the transformations. For each new marketplace the recurring challenge of integrating service descriptions arises. To summarize, the example is constituted of the following points:

1. Different metamodels of the same domain represent identical information (e.g. service descriptions)
2. There is a need for synchronization of information
3. Models are isolated and not changeable, so there is no possibility for an integrated model
4. Model transformation is used for synchronization
5. Models potentially include similar structures, similar names, similar concepts, thus model matching can support the transformation development

[1] `http://www.sap.com`.
[2] `http://www.ibm.com`.

2.2 Metamodel Evolution

Another example for matching-based support of model transformation is the area of metamodel evolution. Consider a metamodel with corresponding instances, which is subject of change. This includes actions like removing elements, restructuring, renaming, adding elements etc. For example removing a metamodel element could lead to invalid instances, since the corresponding metamodel element for old instances has been removed. A model transformation can be used to transform the old instances into new ones with respect to the new metamodel version. This setting is a perfect situation for model matching, because two metamodel versions are very similar, since both are of the same domain. Again model matching can be applied for generating transformation proposals.

Both examples for model matching in model transformation development demonstrate, that model matching supports the recurring task of mapping specification. Subsequently we will describe model matching approaches and reveal their issues.

3 Issues in Current Model Matching

Nowadays model matching approaches are in an early stage. There are several works addressing support of model transformation development, which can be separated into metamodel- and example (instance)-based approaches.

Metamodel-based Approaches. Model matching has been done first by Lopes et al. [6, 9]. They apply a fix-point computation based approach for determining similarity between metamodels. Based on this information they propose to generate transformations. According to [6] their approach seems to be label-based. It uses information from names, data types and enumerations, to propagate computed similarity through containment child elements. In addition they identify the problem of optimizing the generated transformation code, because of its complexity.

Fabro and Valduriez [2] tackled the problem of model matching using a similarity flooding approach. Again they use label-based similarities which are propagated. Their results lead to a verbose generation of concrete mappings. Their work shows a promising approach on proposal generation, which again raises the need for optimization.

According to the approach proposed by Falleri [7] metamodels to be matched are converted into directed labelled graphs. These graphs are used to apply a similarity flooding algorithm, whereby the similarity computation is done via label-based similarity. This similarity is propagated through the encoded graph until a fix-point is reached. This approach lacks an evaluation and is label-based.

Example-based Approaches. Wimmer et al. [10] follow an example-based approach for transformation generation. They consider label-based similarity of instance values in order to determine and generate a transformation between metamodels. They consider only linguistic aspects and concentrate on instance values, which raises the need for instances to cover all possible mappings. Finally they generate transformations.

Varro and Balogh [11] use a similar approach as Wimmer et. al. They apply inductive logic in order to derive mappings based on instance data. They explicitly state their assumptions including a complete coverage of mappings by the instance data.

All of the current model matching approaches rely on label-based similarity. Considering internationalized metamodels in different languages, like English and Chinese, these approaches will fail. The published approaches lack an evaluation, but experiments have shown that the number of found matches is below a half of all matches, relative to a complete mapping specification. Therefore, they seem to leave room for improvement. Additionally, the approaches do not consider the performance (execution time) of the matching algorithms implemented. This leads to execution times for matching in ranges of minutes considering models with more than 100 elements. This is not feasible for using matching techniques for an acceleration of model transformation development. Today's model matching approaches are based on one matching algorithm that is performed sequentially without a reuse or combination of matching results. For a generation of transformation proposals they consider only one specific transformation language. Thus removing flexibility and the choice for a language one is most familiar with.

Furthermore, the code generated is not suitable for further processing by a developer, since a lot of rules (more than 100 compared to 4 manually developed [2]) are generated. Additionally, these rules lack of readability and usability. The subsequent section deals with our approach to address the issues described.

To summarize, we have identified the following issues, investigating the state-of-the-art:

1. Current model matching approaches make use of labels and are therefore restricted to natural language (e.g. English or Chinese)
2. Current model matching approaches have to be increased in quality and performance
3. Current code generation targets only model transformation and is limited to one programming language
4. Generated transformation code is not developer-friendly

4 Improving Model Matching for Model Transformation Development

We identified the issues and needs for improving model matching and transformation generation. To address these issues we propose an approach independent from a specific transformation language, in order to provide a generic solution for model transformation development. This section describes our approach; first giving an overview followed by detailed descriptions of our main ideas:

1. Improved model matching based on a matcher combination framework
2. Model matching by additional metamodel specific matchers
3. Model transformation generation and optimization

Model matching is used to create mapping proposals based on element similarity of metamodels to be mapped. We propose to apply a matcher combination framework and additional metamodel specific matchers. We represent the generated proposals in a pivot model for model transformation, which serves as a base for a generation of executable model transformations.

Figure 3 depicts an overview of our matching-based approach. The metamodels to be mapped (and their instances) are passed onto the matching component (1). After the matchers have been applied, generated proposals are presented to a model transformation developer (2). This developer edits the proposals and afterwards executable transformations are generated of them (3) and finally optimized.

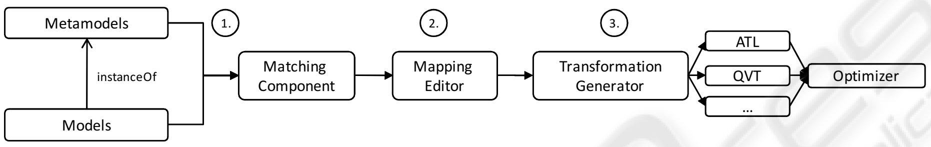

Fig. 3. Process of matching-based support for model transformation development.

4.1 Combining Matcher Framework and Metamodel Matchers

In order to increase the matching result quality we propose to use a matcher combination framework. This framework provides a base for combining results of model matchers. For this purpose, we adopted the COMA++ approach proposed by Do et. al. [4, 5] as outlined in [12]; thus taking advantage of their results. Figure 4 shows the concept of this combining framework; it receives metamodels (with additional models) as an input producing mapping proposals as an output. The processing of the given metamodels is done by applying different matchers each leading to a specific matching result.

These results are placed in a matrix having the source and target elements on the axis and the similarity values as content. Arranging the matrices along the elements being matched leads to a similarity cube containing all similarity values. These values are combined in the similarity cube using heuristics and different matching strategies. Optionally matchers can be applied again and finally the resulting match is created. This approach allows for a combination of matching results from different approaches and even grants a possibility for importing matching results from other tools.

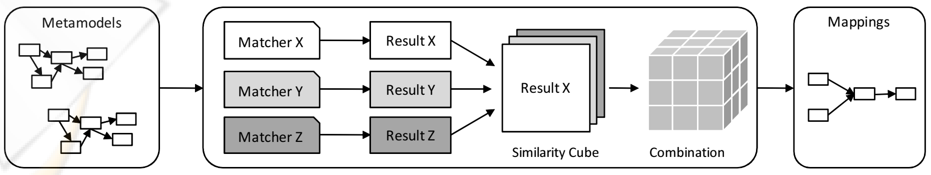

Fig. 4. Overview of our model matcher combination framework.

Nowadays model matching approaches rely on the similarity flooding algorithm, which uses fix-point calculation for similarity calculation. The result is computed by only one matcher with a fixed order of matching algorithms. Applying our approach allows a combination of matchers and matching results while caching intermediate results. Both results can be reused across different matchers and do not have to be computed again. This is one argument for an increased performance. Furthermore, we propose to design the matchers themselves with respect to performance.

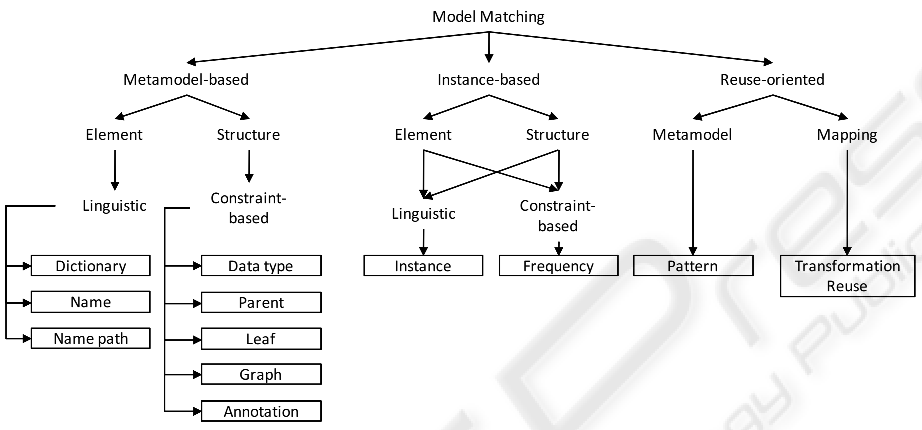

Fig. 5. Classification of model matching techniques.

Figure 5 depicts our classification of matching approaches, which is based on the schema matching classification of Rahm and Bernstein [13]. We adapted and extended it to suit model matching, finally we classified our matchers. The lowest level, highlighted by boxes, shows the proposed matching techniques, the other nodes (classes) are described in the following. The top-level class *metamodel-based* describes matchers based on the information provided by metamodels. It is divided into *element*- and *structure*- based matchers. The element-based matchers use *linguistic* similarity computed from the names of elements, whereas the structure-based ones use *constraint-based* information like types, cardinalities etc. The *instance-based* matchers rely on information given by the metamodel instances (models), which is again structured like the metamodel-based matchers. Finally the *reuse-oriented* matchers takes advantage of available information from existing metamodels (e.g. shared elements) or former mappings, this knowledge is used for similarity computation.

Adopting the XML-schema based COMA++ we selected a set of matchers. The selection is based on the classification, and coverage of all classes and the schema matching experiences, opting for matchers with best results. Furthermore, we propose additional matchers to take advantage of the greater expressiveness of metamodels (compared to XML-schema). The matchers are as follows:

Dictionary Matcher. Using a dictionary allows for matching-based on synonyms or translations. This matcher uses a given database of words to compute a linguistic similarity.

Name Matcher. This matcher targets the linguistic similarity of metamodel elements. It splits given labels into tokens following a camel case approach. Afterwards a token similarity based on a string similarity is computed.
Name Path Matcher. This matcher performs a name matching on the containment path of an element. This supports to distinguish sublevel-domains in a structured containment tree even if leaf nodes do have equal names.
Data Type Matcher. In contrast to the type system provided by XML, metamodels and in particular EMF allows a broader range of types. For example EMF allows defining data types based on Java classes. An extended data type matcher uses these concepts allowing an improved matching result.
*Parent Matcher.*Based on similar containment parents this matcher determines a similarity between children. It follows the rational that having similar parents, indicates a similarity of elements.
*Leaf Matcher.*This matcher computes a similarity based on similar containment children. If metamodel elements have similar children, then a similarity of these elements can be derived.
Graph Matcher. This matcher applies graph similarity algorithms in order to derive a similarity for elements being part of a matching graph structure. Hereby it takes advantage of typed relationships between elements like inheritance, aggregation, etc. It follows the rational of computing the biggest common sub-graph of two given graphs.
Annotation Matcher. Annotation of metamodel elements can also be used for similarity computation. In contrast to schema matching this does not only address documentation but specific aspects, like mapping descriptions. For example EMF[3] makes use of annotations for describing customization of automatic schema mapping generation. This specific information can be used for a better matching result.
Instance Matcher. This matcher uses instance data (models) for computing an element similarity. The matcher examines a set of instances of the metamodels to be matched. If instance values are similar, an element similarity can be concluded.
Frequency Matcher. This matcher examines the distribution of instantiated elements of metamodels to be matched by using instance data. A similarity based on the frequency of instances can be computed.
Pattern Matcher. To reduce metamodel heterogeneity and redundancy of metamodel elements, patterns of these are reused across different metamodels. This includes elements like data types, constraints, etc. This matcher uses these common elements for similarity computation.
Transformation Reuse Matcher. Based on a central mapping repository and associated metamodels this matcher evaluates an existing knowledge base of mappings. The matching is performed by looking up elements in the repository similar to elements being matched. If there is an existing relation between them, a similarity is derived for the matching elements.

First experiments with model matching have shown, that the matching quality depends on the type of metamodels and the matcher configuration. For example consider two metamodels to be matched which are defined on the same level of abstraction containing different representation of the same information, so they are very similar in

[3] Eclipe Modeling Framework – `http://www.eclipse.org/emf/`

their names. Here a name matcher has significant weight, because the names for the same concept are similar. In contrast consider two metamodels in different languages; here a name-based matcher will fail. Classifying the type of model transformation intended allows a derivation of a specific weighting and matcher configuration for the matching framework. This covers the configuration and specific selection of matchers to be applied.

4.2 Model Transformation Generation

Today's approaches use simple mapping models which serve as a base for code generation. However, the mapping models capture only static information, because they allow only links between model elements. We propose to investigate several transformation languages in order to define a non-executable pivot metamodel based on their commonalities. This metamodel is the foundation for transformations being generated and optimized with respect to a developer. For validation and acceptance purposes we propose to start with QVT [14] for transformation generation.

5 Summary and Conclusions

We proposed an approach on improving model matching for model transformation development. This is implemented by a matcher combination framework, metamodel specific matchers and model transformation generation. We presented a classification and concepts for metamodel specific matchers. Namely they are: an instance matcher, a graph matcher, an annotation matcher, a data type matcher, a frequency matcher, a pattern matcher, a transformation reuse matcher, and a matcher configuration based on model transformation type classification. We proposed a matcher and framework design dedicated to improved performance and scalability of model matching.

The main goal of our idea is to lower the effort in model transformation development and to reduce possible errors. Our proposal of using a combining matcher framework with additional matchers leads to:

1. Metamodel language independent matching by applying additional matchers
2. Increased quality of matching results by applying a matcher combination framework, additional matchers and a reuse of matching results
3. A combination of isolated matchers and even an import of matching results of external approaches (as a virtual matcher)
4. Increased matching performance by applying a matcher combination framework and designing performant matchers

A first prototypical implementation using an existing framework and combining the instance matcher and data type matcher indicate the feasibility of our approach, which has to be refined and evaluated further. The proposed framework can also serve as a basis for integrating existing matching approaches in order to improve matching quality.

The evaluation and development of both will be based on practical use cases of model transformation in the area of service engineering and model evolution. This will allow for an evaluation of feasibility as well as quality of our proposals using common measurements like precision, recall and F-measurement. Hereby it is worth to investigate a benchmark for model matching consisting of a suite of sample data. A study of criteria influencing the quality of matching results, which allows for reliable statements regarding the matching quality, e.g. a complete automatic generation of transformations is part of the future work as well. This also includes model transformation scenario classification for matcher configuration. Furthermore, it is worth to explore model matching in other areas like trace development, model search and modelling proposal generation as promising future work.

Acknowledgements

The work was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference ”01MQ07012”. The author takes the responsibility for the contents.

References

1. Object Management Group (OMG): MDA Guide Version 1.0.1. (2003) OMG document omg/2003-06-01.
2. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. Proceedings of SAC ’07 (2007) 963–970
3. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: The VLDB Journal. (2001) 49–58
4. Do, H.H.: Schema Matching and Mapping-based Data Integration. VDM Verlag Dr. Mueller e.K. (2006)
5. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proceedings of SIGMOD ’05. (2005) 906–908
6. Lopes, D., Hammoudi, S., Abdelouahab, Z.: Schema matching in the context of model driven engineering: From theory to practice. In: Proceedings of SCSS05. (2006) 219–227
7. Falleri, J.R., Huchard, M., Lafourcade, M., Nebut, C.: Metamodel matching for automatic model transformation generation. In: Proceedings of MoDELS ’08. (2008) 326–340
8. Cardoso, J., Voigt, K., Winkler, M.: Service engineering for the internet of services. In: Enterprise Information Systems X, Springer (2008)
9. Lopes, D., Hammoudi, S., de Souza, J., Bontempo, A.: Metamodel Matching: Experiments and Comparison. In: Proceedings of the International Conference on Software Engineering Advances (ICSEA’06), Tahiti, French Polynesia, IEEE Press (2006)
10. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: Proceedings of HICSS ’07. (2007) 285b
11. Varró, D., Balogh, Z.: Automating model transformation by example using inductive logic programming. In: Proceedings of SAC ’07. (2007) 978–984
12. Voigt, K.: Generation of language-specific transformation rules based on metamodels. In: Proceedings of the 1st IoS PhD Symposium 2008 at I-ESA’08. (2008)
13. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10** (2001) 334–350
14. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Object Management Group (2007) ptc/07-07-07.