# Towards Model Transformation Generation By-Example*

Manuel Wimmer, Michael Strommer, Horst Kargl and Gerhard Kramler
Business Informatics Group
Institute for Software and Interactive Systems
Vienna University of Technology, Austria
Email: {wimmer|strommer|kargl|kramler}@big.tuwien.ac.at

*Abstract*— **With the advent of Model-Driven Engineering (MDE) several model transformation approaches and languages have been developed in the last 5 years. Most of these existing approaches are metamodel-based with metamodels representing both an abstract syntax of the corresponding modeling language and also a data structure for storing models. However, this implementation specific focus makes it difficult for users of modeling languages to develop model transformations, because metamodels do not necessarily define all language concepts explicitly which are available for notation purposes. Therefore, we propose a by-example approach for defining inter-model mappings representing semantic correspondences between concrete domain models which is more user-friendly then directly specifying model transformation rules or mappings based on the abstract syntax. The inter-model mappings between domain models can be used to generate the model transformation rules, by-example, taking into account the already defined mapping between abstract and concrete syntax elements. With this approach the user's knowledge about the notation of the modeling language is sufficient for the definition of model transformations regarding semantic correspondences. Hence, no detailed knowledge about the metamodel and the model transformation language is required.**

## I. Introduction

Software development is a complex task. Developers have tried to overcome complexities with different kinds of methodologies and technologies (e.g., OOP, CASE-Tools, general-purpose notation, etc.). One of the latest approaches is Model-Driven Engineering (MDE), which aims to define a framework for modeling, metamodeling and transformation between models. In particular, a model can be transformed into a semantically corresponding model (horizontal transformation) or into a model on another level of abstraction (vertical transformation). In this paper we explain a way to define horizontal model transformations, following a special kind of MDE, namely Model-Driven Architecture (MDA) [1], an Object Management Group (OMG) [2] initiative, with metamodels representing both an abstract syntax of the corresponding modeling language and also a data structure for storing models. However, this implementation specific focus does not ease the way for user-friendly development of model transformations, because metamodels do not explicitly define all language concepts, which are available for notation purposes. These concepts are hidden in the metamodel and often hard to discover. For example, in the core of the UML metamodel (defined in the UML Infrastructure [3]) the concept *attribute* is hidden in the class *Property*. Properties can only be attributes if the property has a relation *owningClass* to a class. When the user has to define model transformations these hidden concepts must be re-engineered by hand.

Therefore, we propose *Model Transformation By-Example* (MTBE) which is based on other by-example approaches like *Programming By-Example* (PBE) or *Querying By-Example* (QBE). PBE tries to generate code from user interactions with a GUI program (e.g., Microsoft Excel) with the help of a concrete example (e.g., coloring of cells). The generated code can be used to automatically replay the recorded actions. Based on the aforementioned by-example approaches, MTBE allows the definition of inter-model mappings representing semantic correspondences between concrete domain models on the M1 layer, which is more user-friendly, in contrast to directly specifying model transformation rules based on the metamodel (abstract syntax) on the M2 layer. However, the inter-model mappings can be used to generate the transformation rules by-example, taking into account the already defined mapping (notation) between abstract and concrete syntax elements. The notation includes the constraints how elements from the abstract syntax (metamodel) are related to the concrete syntax. Applying MTBE to EMF [4] and GMF [5] it is possible to reuse the already available constraints to derive ATL [6] transformations out of the mappings on the concrete syntax between the modeling languages. The user's knowledge about the notation of the modeling language is sufficient for the definition of model transformations regarding semantic correspondences. Hence, no details about the abstract syntax (metamodel) is required. However, it is essential to align two models, which represent the same problem domain, to automatically derive the transformation rules.

The main contribution of this paper is to lay out a by-example based approach for defining mappings on the M1 layer between concrete domain models which incorporates the notation of modeling languages and allows the generation of model transformation code based on the M2 layer. In addition, several issues are discussed when generating model transformation code from the user defined inter-model mappings.

* 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

Hence, the remainder of this paper is structured as follows: The next section gives an overview of shortcomings of current model transformation approaches, prerequisites for applying a by-example approach for deriving model transformation code, and a conceptual framework that can be used to realize MTBE. Section 3 covers an example application of MTBE, which deals with the alignment of the modeling languages *UML* and *ER*, and the generation of model transformation code for transforming UML models into ER models, and vice versa. Section 4 discusses some open issues of applying MTBE in practice concerning the user-friendly adaption of generated model transformation code, extending the language for user-defined inter-model mappings, reference examples for modeling domains and a prototypical implementation. Section 5 discusses related work and finally we outline conclusions and future work in Section 6.

## II. Motivation, prerequisites and conceptual framework for MTBE

In this section first the shortcomings of current model transformation approaches are discussed, subsequently the prerequisites of modeling languages for applying MTBE are briefly explained. Furthermore, this section is concluded by outlining a conceptual framework for generating model transformations by-example.

### A. *Shortcomings of current model transformation approaches*

In the MDE research field various model transformation approaches have been proposed in the past 5 years, mostly based on either a mixture of declarative and imperative rules such as ATL [6], or on graph transformations such as AGG [7], Fujaba [8], [9], and BOTL [10], or on relations MTF [11]. Moreover, the Object Management Group (OMG) has published a first version of QVT [12] which should become the standard model transformation language. Summarizing all these approaches, it can be said that state of the art for defining model transformations is to describe model transformation rules between a source and a target metamodel (M2), whereas the rules are executed on the instance layer (M1) for transforming a source model into a target model. Consequently, each of these approaches is based on the abstract syntax of modeling languages, i.e., on their metamodels, only, and the notation of the modeling language is totally ignored.

In collaboration with the Austrian Ministry of Defense and based on experiences gained in former integration scenarios [13], [14] we are currently realizing a system called ModelCVS [15], [16] which aims at enabling tool integration through transparent transformation of models between metamodels representing different tools' modeling languages. Hence, we developed various model transformation examples for tool integration purposes using some of the aforementioned approaches, and in doing so, we discovered two main issues which prevent the user-friendly definition of model transformations. On the one hand there is a gap between how the modeler reasons about aligning two models and how the corresponding rules are defined in order to be executable by

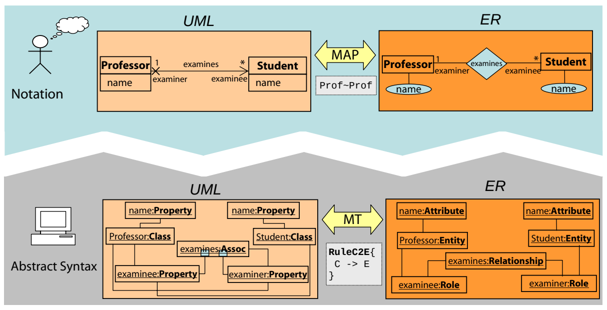


Fig. 1. Gap between user intention and computer representation

the computer, and on the other hand not all concepts of a modeling language supported by the concrete notation are explicitly represented in the metamodel. In the following we discuss these two issues in more detail.

*Issue 1*: There is a huge gap between the user's intention of aligning two languages and the way model transformation rules are defined for being automatically executable by the computer. Mostly, the user reasons on models representing real world examples shown by concrete notation elements and mappings between semantically corresponding model elements. However, this way of thinking is not appropriate for defining model transformations with currently available model transformation languages, because they support defining model transformation rules based on the abstract syntax, only.

Figure 1 illustrates this problem by an alignment scenario for UML and ER models. The upper half of figure 1 depicts that for the user it is appropriate to reason on models representing real world examples expressed in concrete notation of the modeling language to find the semantic equivalent parts. In contrast, the lower half of figure 1 shows the same domain model in abstract syntax visualized as an UML object model. As one can see, the abstract syntax is designed for the computer in order to process the models efficiently and not for the visualization of the domain knowledge in an easy understandable way. Hence, when trying to understand a domain model in abstract syntax one has to explore more model elements compared to the concrete notation representation, and furthermore, one has to know all relevant details of the metamodel, i.e., the language definition. Moreover, this problem is further aggravated by the following issue.

*Issue 2*: The aim of metamodeling lies primarily in defining modeling languages in an object-oriented manner leading to efficient repository implementations. This means that in a metamodel not necessarily all modeling concepts are represented as first-class citizens. Instead, the concepts are frequently hidden in attributes or in association ends. We call this phenomenon *concept hiding*. For an in-depth discussion of concept hiding and how concepts can be hidden in metamodels see [17].

As an example for concept hiding in metamodels consider figure 2. In the upper part it shows a simplified version of the UML metamodel kernel which is defined in the UML

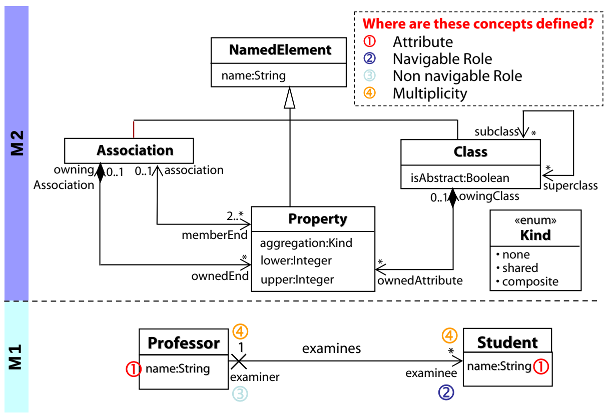


Fig. 2. Concept hiding in metamodels

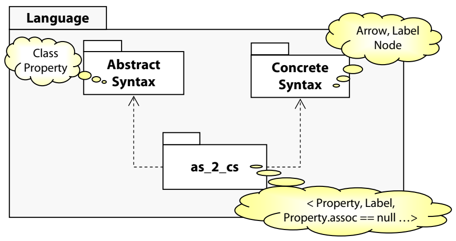


Fig. 3. Relationship between abstract and concrete Syntax

Infrastructure [3]. In the lower part a domain model is shown in concrete UML syntax as defined by the notation tables in the UML Superstructure [18]. As one can see in figure 2, the metamodel covers more than 10 modeling concepts but uses only four classes. Hence, most of the modeling concepts are implicitly defined, only. It is left as an exercise to the reader to find out *where* and *how* the concepts *attribute*, *navigable role*, *non-navigable role*, and *multiplicity* are defined in the metamodel.

These two issues mainly circumvent the user-friendly definition of model transformations. Therefore, we propose an orthogonal and extending approach to existing model transformation approaches for defining semantic correspondences in the concrete syntax of the models and the automatic generation of model transformations for the abstract syntax. This procedure allows a more user-friendly development of model transformations. Before going into details about the by-example approach we have to discuss which tasks are currently involved when model transformations are developed.

In general, before actually formalizing the model transformation rules in a model transformation language the user has to acquire knowledge about semantic correspondences between the concepts of the modeling languages as incorporated in their metamodels. One appropriate way to gain this knowledge is to start modeling the same problem domain with both modeling languages. By comparing the two resulting models the semantic correspondences between model elements can be easily found which again can be used to derive the correspondences between the metamodel elements. In addition, these models entail another benefit - they can be deployed for testing purposes as input for the expected model transformation and for comparing the output of the model transformation execution.

After clarifying all necessary semantic correspondences the user has to implement the gained mapping knowledge in the model transformation rules. For this task the user has to understand how the notation is represented in abstract syntax elements and how missing concepts in the abstract syntax can be reconstructed, e.g., by setting attribute values and links to other objects. Here comes MTBE into play. First, the mappings are explicitly definable between the domain models shown in concrete syntax which allows also the documentation of the semantic equivalences. Second, these mappings are a good starting point for automatically generating the required model transformation code which is more efficient in contrast to current approaches where the user has to implement all of them by hand.

## B. *Prerequisites for MTBE*

This subsection discusses the prerequisites for establishing model transformations by example to realize the aforementioned benefits. The primary idea of MTBE is to exploit the concrete notation of modeling languages, which is well known by the user, for defining mappings between semantically corresponding model elements on the M1 layer. In order to further discuss the by-example approach for model transformations, the interrelationships between the *abstract syntax*, *concrete syntax* and the *mapping* between them, which describe the notation of the modeling language, have to be clarified. In accordance with MMF [19] and GMF [5] figure 3 depicts how these three parts of a formal language definition are interrelated in terms of an UML package diagram.

The package *abstract syntax* summarizes elements of the abstract syntax, i.e., the metamodel. For example for UML these would be concepts such as *property*, *class* and *association*. In contrast, the package *concrete syntax* covers graphical elements, e.g., ellipse, label, and rectangle, which can be further combined to more complex forms, e.g., *ClassRectangle*, *AttributeLabel*. Finally, how elements of the abstract syntax are mapped to elements of the concrete syntax is defined in the package *as_2_cs* which mainly consists of triples of the following form:

$$Triple :=< as\_E, cs\_E, const(as\_E)? > \quad (1)$$

The first part *as_E* stands for an element of the *abstract syntax* package, the second *cs_E* for an element of the *concrete syntax* package and the last *const(as_E)* stands for an optional constraint, e.g., defined in OCL, that defines under which conditions, i.e., links and attribute values of an instance of *as_E*, *as_E* is represented by *cs_E*. The optional constraint part (*const(as_E)*) of the triple is the most relevant part for this work.

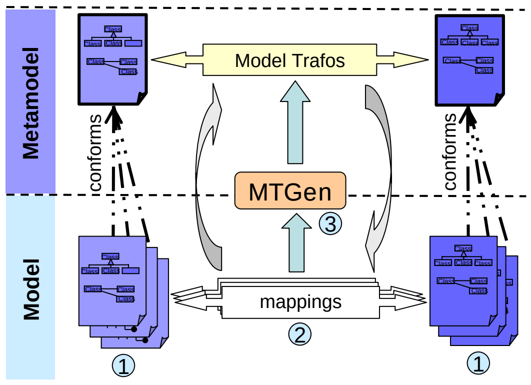


Fig. 4. MTBE framework

In case no constraint is defined, there is a *one-to-one* mapping between an abstract syntax element and a concrete syntax element, i.e., the concept defined in the metamodel is directly represented by one concrete notation element. However, the other case is the more interesting in context of solving the concept hiding problem. The presence of a constraint defines a new sub-concept for the notation layer, which is not explicitly represented by one of the metamodel classes. Consequently, when defining model transformations based on the abstract syntax, the constraints for these sub-concepts must be defined by the user in the query part or when going the other way round by setting the property values correctly in the generation part of the transformation rules. This is a tedious and error-prone task that requires excellent knowledge about the metamodel.

With MTBE this circumstance can be improved by incorporating the existing constraints defined in the triples (cf. 1) of the *as_2_cs* package (cf. figure 3) into the model transformation generation process in order to minimize the effort for re-engineering and defining these constraints by hand.

### C. Conceptual Framework for MTBE

This subsection discusses a conceptual framework for MTBE at a glance. The key focus of this framework is the automatic generation of transformation programs regarding semantic correspondences between two languages as can be seen in figure 4. In this framework the model transformation generation process requires 3 steps, that are explained in the following.

*Step 1*: The initial step is the definition of models of the same problem domain in both modeling languages (cf. left and right of the lower half of figure 4). The user can decide if a single model, which covers all aspects of the languages, or several examples, each focusing on one particular aspect. Presumably the second approach is more preferable. The requirements on the models are twofold. First, certainly they must conform to their metamodels, and second, the available modeling constructs of the modeling language should be covered by the examples.

*Step 2*: The second step in the framework is that the user has to align the domain models (M1) by defining semantic correspondences (mappings) between model elements of the left and right side (cf. middle of the lower half of figure 4). For simplicity, it is assumed that the models on the left and on the right side represent the same problem domain, as explained in step 1. In the current state of our work we assume full equivalence mappings, only. However, the introduction of other mapping kinds is subject to future work. Concerning the example models, general reference models for several modeling domains such as structural, interaction, and process modeling can ease the development of the required examples. However, this part is also subject to future work.

*Step 3*: After finishing the mapping task, the third step is that the *Model Transformation Generator* (cf. MTGen in figure 4) takes the user-defined mappings as input and generates a model transformation program which is based on metamodel elements. The resulting model transformation programs can transform any source model, which conforms to the source metamodel, can be transformed into a target model which conforms to the target metamodel. However, as further explained in section 3, this generation process may need some user interactions for resolving ambiguities in the mappings, arising from heterogeneities concerning the extend of the modeling languages. Another necessary condition for the transformation generation process is that the MTGen needs access to the package *as_2_cs* (cf. figure 3) in order to compute all necessary conditions for the query and property values for the generation parts of the transformation rules.

Subsequently, the generated model transformation programs can be tested on the models, that were already used for defining the mappings. This is another benefit of the by-example approach, as the input and output models for testing the model transformations are already available and no extra work for developing test cases is necessary. The target models generated by the transformation program can be compared to the already existing models. When some differences between the two models arise, the user can decide if the mappings on M1 should be revised and a newer version of the model transformation program should be generated or if the mappings on M1 are correct and the model transformation needs some fine-tuning directly in the transformation code.

## III. MTBE BY-EXAMPLE

In this chapter we will exemplify the operating mode of MTBE by a concrete example. In order to do so, consider the situation in which we have two UML classes *Professor* and *Student* as well as a one-to-many relationship between them. This simple UML class diagram is depicted in the upper left corner of figure 5. In addition, the same problem domain is also modeled in terms of an ER diagram that can be found in the upper right of figure 5. In the upper half both models are represented in concrete syntax, whereas the lower part of figure 5 represent the same models in abstract syntax, which is the type of representation the computer uses for model transformation execution. For simplicity and higher readability the models in abstract syntax are represented as UML object diagrams. The example models shown in figure 5 are quite simple, however, they are sufficient to show the most important aspects of our proposed MTBE approach.

In the following subsections the steps 2 and 3 of the MTBE framework (cf. figure 4) are discussed in more detail. Step 2 has to be carried out by users themselves and concerns the alignment of two domain models shown in concrete syntax (cf. subsection III-A). Step 3 is split into 4 sub-steps, to give an in-depth discussion of the work the MTGen has to do. In particular, we explain how the abstract syntax is analyzed to collect all necessary data for the model transformations. Therefore, we interpret the models shown in abstract syntax as object models consisting of objects, attribute values and links, because these models can be seen as instances of the metamodel, which again can be seen as a simple class diagram. Consequently, we first explain the creation of objects (cf. subsection III-B), then the placement of attribute values (cf. subsection III-C), and finally the linking of objects (cf. subsection III-D). By collecting the data of these three sub-steps, it is possible to derive all necessary information in order to define the query parts (e.g., the *from* part of ATL rules) and also the generation parts (e.g., the *to* part of ATL rules) of the model transformation rules (cf. subsection III-E).

## A. Mapping Definitions

First, the user has to define mappings between model elements of the two concrete domain models as shown in figure 5. These mappings are illustrated by thin dotted lines between elements of the two models in figure 5. For the sake of clarity, we omitted some of the mappings as this helps to focus on those mappings that are of special interest for our algorithms explained in the next subsections. As mentioned before in chapter II-C mappings specified by users are solely full equivalence mappings, i.e. one-to-one mappings. Furthermore, these mappings can be regarded as bidirectional in contrast to other by-example transformation approaches, e.g., [20]. Hence, model transformation code can be generated for both directions, namely from UML to ER, and vice versa.

Second, we need to know how the model elements shown by the concrete syntax correspond to the model elements shown by the abstract syntax. These definitions are provided by the package as_2_cs as described in section II-B. The links between concrete syntax and abstract syntax are illustrated in figure 5 as thin solid arrows for the right and for the left side, respectively. Again we left out some of the links to focus on the mapping definitions which are relevant for the following discussions.

## B. Object Creation

As we defined semantic correspondences between model elements of the two domain models in the previous step, we can now move on to the object creation process. Assume first that we want to transform the UML class diagram into an ER diagram. Therefore, the algorithm has to analyze the abstract syntax of both models and additionally the user-defined mappings. In particular, the algorithm has to check if a certain type of object in the UML model is mapped to a certain type of object in the ER model. If this is the case, there is also a *full equivalence mapping* on the abstract syntax layer and a simple transformation rule without a condition can be generated for this object type. For example, objects of type *class* are mapped to objects of type *entity* (cf., mapping *a* in figure 5), only. However, some objects of the same type are mapped to different object types depending on their attribute values and links. In this case, an additional mapping operator is available for the abstract syntax layer, namely *conditional equivalence mapping*. The conditions for the conditional equivalence links are derived from the *as_2_cs* package, i.e., the concept hiding is resolved, and finally these conditions manifests in the query part of the model transformation rules. For example, *property* objects of the UML class diagram are mapped to both *attribute* objects (cf., mapping *b* in figure 5) and *role* objects (cf., mapping *d* in figure 5) of the ER model. Taking the constraints *property.owningClass != null* and *property.association == null* of the *as_2_cs* package into account, we can assure that only an ER *attribute* is generated when the *property* actually represents an attribute in the UML class diagram. The same procedure can be applied for properties representing roles.

After completion of this step we have created all necessary objects for an ER diagram from a UML class diagram, which are the basis for our next steps to be performed. The same procedure can also be applied for a ER diagram to an UML class diagram transformation as our transformations can be generated in either direction.

## C. Placement of Attribute Values

This step constitutes the placement of attributes values for the created objects. In contrast to the object creation step where primary the query parts of the transformation rules were relevant, this subsection focuses on the generation parts, i.e., how to set the attribute values. First of all, we have to differentiate between two different kinds of attributes which occur in metamodels, namely *ontological attributes* and *linguistic attributes*.

Ontological attributes represent semantics of the real world domain which can be incorporated by the user by setting the values explicitly in the concrete syntax. Examples for ontological attributes are *Class.name* and *Attribute.name*. In order to set the ontological attributes in the generation part of the transformation rules we use heuristics, e.g., string matching. In our example, we can conclude that the *name* of a *class* should be the *name* of an *entity* when considering the class *professor* and the entity *professor* (cf., mapping *b* in figure 5), because these two attributes have the same value.

Linguistic attributes are used for the reification of modeling constructs which cannot be set explicitly by the user in the concrete syntax, e.g., *Class.isAbstract* or *Property.aggregation*. Hence, these attributes have predefined ranges of values as they are fixed elements of the language definition. When dealing with linguistic attributes in context of MTBE we need to exploit the information stored in the as_2_cs mappings, because in these mappings the concepts become explicit by defining

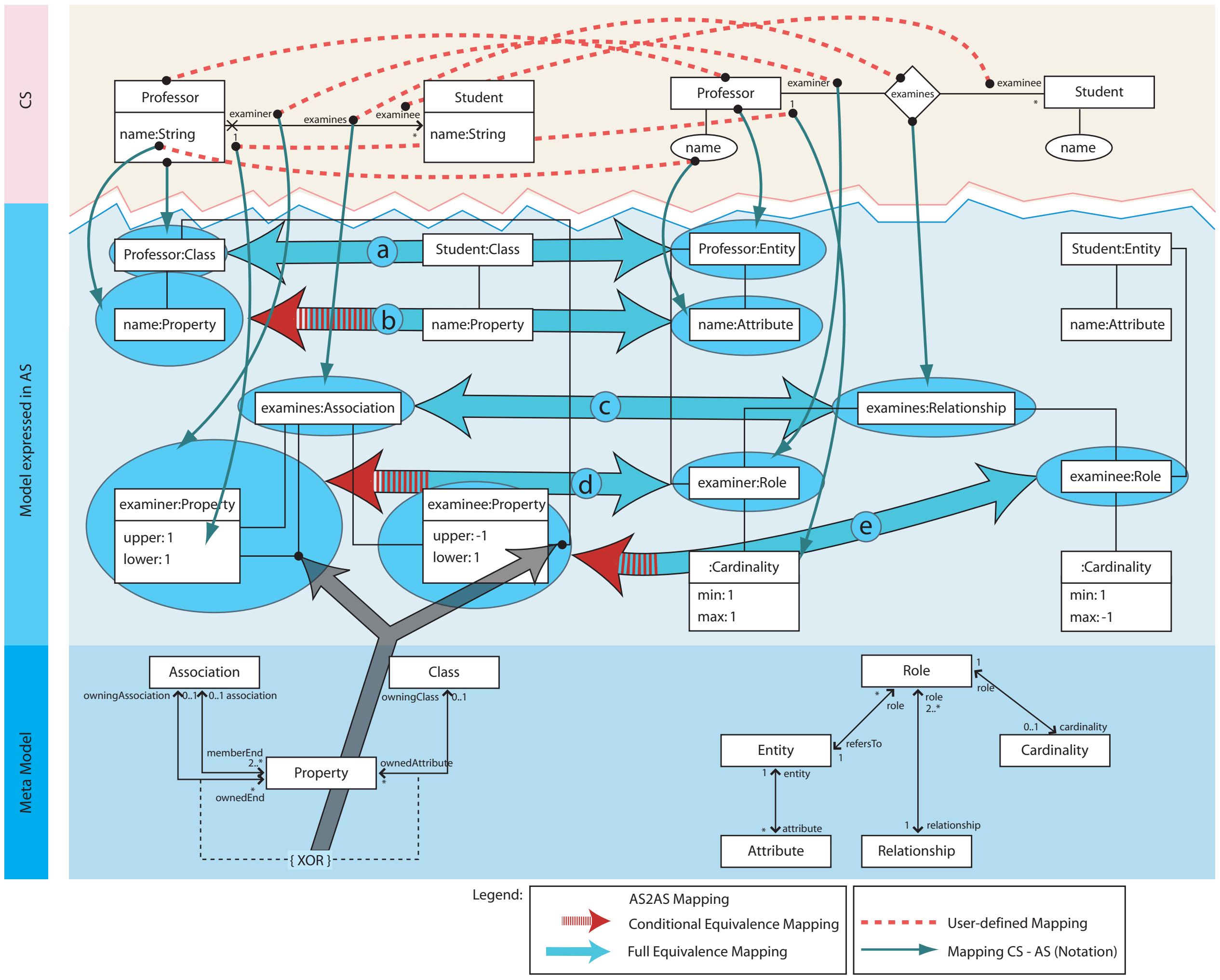


Fig. 5. MTBE for UML2ER and vice versa

the required condition, i.e., how the values have to be set to fulfill the requirements for the sub-concept. For example, when transforming an ER attribute to an UML property, we also have to set the linguistic attributes of the property class (e.g. *Property.aggregation*) which can be done by incorporating the information stored in the as_2_cs mapping.

## D. Linking Objects

Finally the links between the created objects have to be deduced from the metamodel, from our triples of the *as_2_cs* mappings containing OCL constraints, from the user-defined mappings, and user interaction as the last choice when the last three mentioned options are not sufficient. This part of the transformation step is obviously the most interesting one, as most difficulties arise at this stage. In particular, the user-defined mappings on the concrete syntax can result in unambiguous mappings, i.e., mappings that are controversial and it is not automatically decidable which case should be chosen for the general model transformation. Especially *0..1* associations in combination with *xor*-constraints in the metamodel are relevant in this context, as they might entail some hidden concepts. Another reason of unambiguous mappings is the heterogeneity of the expressiveness of the modeling languages.

In the following the creation of object links is described, whereby we classify some interesting cases regarding to multiplicity of the association ends of the metamodel, namely *1..1*, *0..1* and *0..1* in combination with *xor*-constraints.

*1) Unambiguous mappings:* Concerning unambiguous mappings, two interesting cases are here stated, namely association ends with multiplicity *1..1* and *0..1*.

- *1..1 association ends*: We encounter such association ends in our ER metamodel between *Entity* and *Attribute* as can be seen in the bottom of figure 5. In addition, when looking at the middle of figure 5, one can see, that each ER attribute is linked to an ER entity as this is the mentioned constraint of the ER metamodel. Furthermore, one can see that each UML attribute is linked to an ER attribute and that the containing UML class is linked to the containing ER entity, respectively. Consequently, if we transform an UML *property*, that is actually an attribute, into an ER *attribute*, we can automatically create the link between entity and attribute.

- *0..1 association ends*: This kind of association ends in the metamodel allows concept hiding, as is done in the UML metamodel for the class *Property*. A property can either be a special kind of role or an attribute belonging to a certain class. As we will see, association ends of this kinds are not as easy decidable as 1..1 association ends are, because the links are not required on the abstract syntax layer and can vary, also within the same example. However, in case of unambiguous mappings, i.e., the link is always or never present on the abstract syntax layer, a general model transformation rule can be derived. For example, ER relationship has two links to its roles and UML association has two links to its properties. Furthermore, ER relationship is mapped to UML association (cf., mapping *c* in figure 5) and the ER roles are mapped to the UML properties of the association (cf., mappings *d* and *e* in figure 5). When going from ER to UML, we can deduce that each corresponding association should have links to the properties which correspond from the roles of the ER relationship. However, the second possible kind of link between association and role (cf. concerning association end *owningAssociation* in figure 5) is not automatically decidable as we see in the next subsection.

*2) Ambiguous mappings:* In this part we describe an example that shows that especially for object linking some ambiguities can occur which have to be resolved by user interactions.

In figure 5 two user-defined mappings are shown, which are the source for ambiguity mappings on the abstract syntax layer. In this example the role *examinee* in the ER model is mapped to the navigable role *examinee* in the UML class diagram, but the role *examiner* is mapped to the non-navigable role *examiner* in the UML class diagram. Now we want to discuss the impacts on the abstract syntax layer mappings. The problem arises that it is not decidable which general transformation rule should be derived, because one role of the ER model is mapped to an UML property, which has a link to an class object (cf., mapping *e* in figure 5), and another role of the ER model is mapped to an UML property which has instead an link to an association object (cf., mapping *d* in figure 5). This unambiguity results from the metamodel of UML where an *xor*-constraint exists between *owningAssocation* and *owningClass*, as can be seen in figure 5, and from the fact that UML differentiates between navigable role and non-navigable role without supporting the general role concept. As our definition of the ER metamodel does not allow for two different kinds of roles as the UML metamodel does, we cannot derive an general transformation rule. Instead the user must decide on how to deal with roles from the ER model in the UML model. This example shows that in general it is not possible to automatically derive all model transformation rules, not even between modeling languages, which share the same modeling domain. Instead, for some rules the user has to interact and decide, which alternative is appropriate, such as in our example to generate navigable roles in the UML model for roles of the ER model.

### *E. ATL Rule Generation*

At last all gathered information can be aggregated to generate proper ATL transformations. In the following, two examples are shown just to give an idea how the query parts and generation parts of the ATL transformations are generated.

The first example as presented in listing 1 is a transformation from *ER attributes* to *UML properties*, which actually represent UML attributes in the concrete syntax. Note that the generation part of this rule is the most interesting part, because the attribute value assignments for ontological and linguistic attributes have been automatically generated.

Listing 1. Attribute2Property

```
module ER2UML;
create OUT : UML from IN : ER;

rule A2P {
    from a : ER!Attribute
    to p : UML!Property (
          name <- a.name,
          aggregation <- 'none',
          ...
          owningClass <- a.entity
    )
}
```

The second example is an ATL rule that incorporates the condition of the abstract to concrete syntax mapping for UML in its query part in order to produce *ER attributes* for *UML properties* which are actually representing *UML attributes* on the concrete syntax layer, only.

Listing 2. Property2Attribute

```
module UML2ER;
create OUT : ER from IN : UML;

rule P2A {
    from p : UML!Property (
           p.owningClass.oclIsUndefined()
           = false and
           p.association.oclIsUndefined()
    )
    to a : ER!Attribute (
           name <- p.name,
           entity <- p.owningClass
    )
}
```

## IV. Open Issues in MTBE

Concerning open issues, we particularly strive for first, user-friendly adaptation of the generated model transformation code, second, extension of the language of the user defined mappings, third, creation of reference examples, and fourth, prototypical implementation of graphical mapping editors.

*1) User-friendly adaption of generated model transformation code*: The MTBE framework in its current state has one major drawback concerning the one-step model transformation generation based on the abstract syntax when the user needs to adapt the generated transformation by hand. This drawback is due to hidden concepts in the metamodel, that are explicit in the concrete syntax. Hence, the user has to deal with the used constraints from the notation when adapting ATL rules. We are to tackle this problem by applying higher-order transformations as introduced in [21] in combination with using models of models whereas each particular model has a particular purpose as introduced in [22].

In particular, we combine these two techniques in a two-step model transformation generation process with an intermediate layer as illustrated in figure 6.

*Step 1*: Starting from the mappings between concrete domain model elements, in a first step, a model transformation is generated in which the concepts available in the notation are explicitly represented to hide complexities of the original metamodel from the user. For this intermediate step, a metamodel has to be generated from the original metamodel which in addition to concepts from the original metamodel covers concepts introduced by the notation. Hence, the purpose of this generated metamodel is explicit knowledge representation allowing easier development of model generation code. The generation of the extended metamodel is realized by automatically transforming the original metamodel combined with mapping conditions of the package as2cs into a new metamodel which explicitly represents the concepts.

*Step 2*: In a second step, the transformation code adapted with additional user extensions is transformed into a model transformation which operates on the abstract syntax. For this step we adopt the fact that also a transformation is a model, which allows the transformation, of a transformation to reduce to model transformation. In the transformation of the transformation, the sub-concepts introduced by the notation are reduced to their super-concepts and expressed in the transformation rules with complex conditions in the query parts.

*2) Extension of the mapping language*: The presented mapping language for the concrete syntax consist of one operator, namely full equivalence mapping. Consequently, when MTBE should also support model transformations, which go beyond semantic equivalence, the mapping language has to be extended by other operators, such as string manipulation, conditional equivalence mapping, and nested mappings. Therefore, we have to look at further examples to derive a requirement catalog for a MTBE mapping language in order to determine further mapping operators.

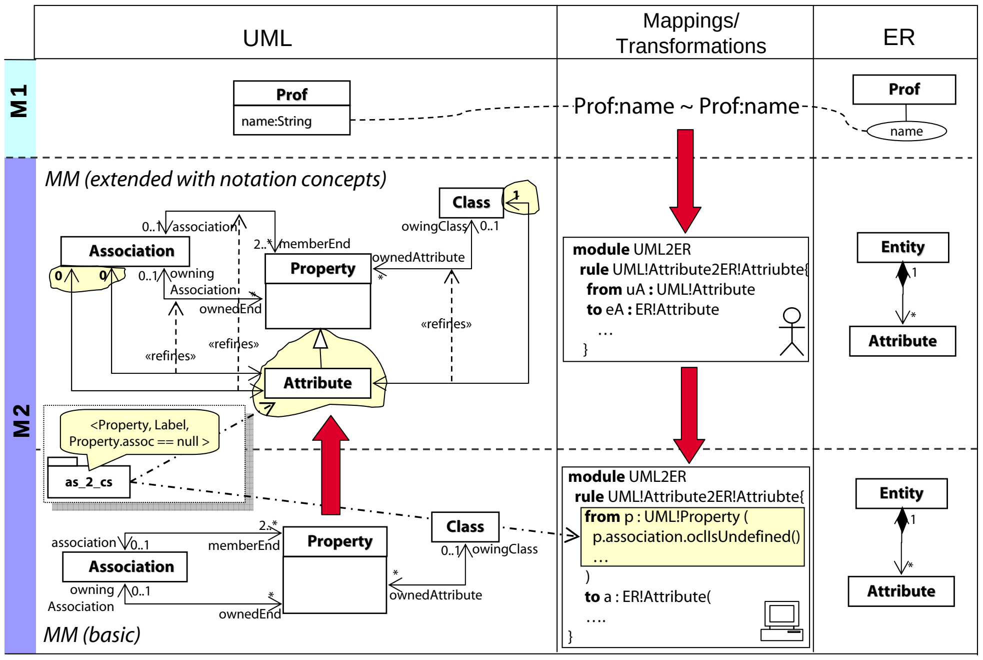
*3) Creation of reference examples for specific modeling domains*: One of the main goals of our ModelCVS project is to build reference models representing a well described problem domain for various modeling domains to support the integration task. These reference models can be reused for the presented by-example approach as domain models, which are aligned by the user. Hence, the user need not conceive domain models from scratch, instead the reference models can be used as starting point for most prominent modeling languages and modeling domains. However, we also plan to explore in more detail how the domain models should look like and which undesirable situations are possible when models are aligned at M1 which share the same problem domain.

*4) Prototypical implementation*: Concerning the implementation of tool support for MTBE, emerging technologies such as the Graphical Modeling Framework (GMF) [5] for defining and generating graphical editors can be used. GMF separates between the concrete syntax and abstract syntax as discussed in subsection II-B. However, the standard editor generator is only capable of generating model editors for a single language. Therefore, we plan to extend the standard editor generator of GMF to support the generation of a mapping editor. This extended generator should take two language definitions (e.g., UML and ER) and the mapping language as input and should produce a mapping editor which is capable of defining domain models in both languages and also to map the model elements of the domain model.

## V. Related Work

With respect to our approach of defining inter-model mappings between domain models (M1) and the derivation of model transformation code from these mappings we distinguish between three kinds of related work: first, related work concerning on linking model elements between models within a separate model (model weaving), second, declarative and example-based transformation rules mainly supported by graph transformations and third, related by-example approaches starting from their origin approach, namely query-by-example.

In general, our approach of defining similarities between modeling languages and models is related to model transformation. Model transformation in the context of MDE is a rapid emerging topic as can be seen in the model transformation workshop at the MoDELS/UML 2005 conference. One of the first and nowadays one of the most matured approaches is the ATLAS Model Weaver (AMW) [23] and the ATLAS Transformation Language ATL [6]. The idea behind model weaving is to define a relationship between a left model (or metamodel) and a right model (or metamodel) with certain kind of mapping operators which can also be user-defined. This approach is related to the mapping between two concrete domain models of two different modeling languages, however, the difference lies in the representation of the models and in the level of the mappings. AMW works with the abstract syntax representation of a model, while our approach works with mappings between models represented with the concrete syntax of the modeling languages. The benefit of mapping examples shown in concrete syntax is the absence of hidden concepts which occur quite often in metamodels. Our work is also different to the AMW in that the model transformation generation process of the AMW currently focuses on using mappings between metamodels (M2 mappings) and therefore



```
module UML2ER
  rule UML!Attribute2ER!Attriubte{
   from uA : UML!Attribute
   to eA : ER!Attribute
     …
  }
```

```
module UML2ER
 rule UML!Attribute2ER!Attriubte{
   from p : UML!Property (
     p.association.oclIsUndefined()
     …
   )
   to a : ER!Attribute(
   ….
}
```

Fig. 6. 2-step transformation generation

based on the abstract syntax as input to derive ATL code [24], while our approach aims at generating model transformation code from M1 mappings. Hence, we have shifted the definition of the mappings from the abstract syntax to the concrete syntax and from the metamodel layer to the model layer.

Our proposed MTBE approach follows the main principles of the query-by-example (QBE) approach introduced in [25]. The aim of QBE is to have a language for querying and manipulating relational data. This is achieved by skeleton tables, which consists of example rows filled out with constants, constraints, and variables, combined with commands. Commands describe what to do with the selected tuples that match the defined queries, such as deletion or selection of the tuples. In order to operate on relational data stored in DBMS, real queries (e.g., SQL scripts) are derived from the skeleton tables and can be executed on relational models. Lechner et al. [20] follow this original approach of QBE, but with extensions for defining scheme transformers, which is demonstrated in the area of web application modeling with WebML [26]. Therefore, the original QBE approach is extended by introducing in addition to the query part (WebML model before transformation) also a generation part (WebML model after transformation) in the template definitions. Finally, XSLT code is generated to transform the WebML models which are represented within the accompanying tool *WebRatio* as XML files.

Our work reuses the main idea of the aforementioned by-example approach [20], but our work is different to this work in that first, we propose the use of real world examples instead of using abstract examples, second, we introduce bi-directional mappings in contrast to uni-directional template based examples, third, our domain for applying a by-example approach is the modeling technical space [27], while the others are based on relational data, and fourth, we also consider the abstract syntax to concrete syntax mappings to tackle the problem of implicitly defined modeling concepts and are therefore able to make them explicit.

Other by-example based approaches related to our proposed MTBE approach are programming by-example [28], [29], and [30] as well as XSLT style sheet generation by-example [31]. The objective of these approaches is to facilitate the end user to be able to perform tasks which normally need more knowledge, e.g., knowledge about programming languages like Visual Basic, Java or even XSLT. The way PBE tries to to achieve this objective is to record the users actions (e.g., by a trace model) maybe in more than one iteration, and generate a program from the trace models to automatically perform the afore manually performed task by the computer.

The difference to the programming by-example approaches is that we statically define the mappings between two models instead of the iterative adaptation of the examples to get the resulting code, in our case the ATL code.

## VI. Conclusion and Future Work

In this paper we have introduced a by-example approach for defining semantic correspondences between domain models (M1) shown in their concrete notation, that allows the derivation of model transformation code. This approach tackles concept hiding in metamodels, which results in complex query and generation parts of model transformation rules. Furthermore, the user can reason about semantic correspondences in a notation and with concepts the user is familiar with. Hence, complex details of the metamodel resulting from the need for efficient API and repository implementations are hidden from the user when defining transformations.

We have presented relevant issues concerning MTBE, however, various extensions of this work are required in the future, e.g., application on larger modeling languages, also from other modeling domains and full elaboration of the so far gained insights. In particular, MTBE requires proper tool support and methods guiding the mapping and transformation code generation tasks in order to fulfill the requirements for the user-friendly application of MTBE. Therefore, the next step is the implementation of a prototype in order to further evaluate our proposed approach in the large.

## References

[1] *MDA Guide Version 1.0.1*, omg/2003-06-01 ed., June 2003. [Online]. Available: http://www.omg.org/docs/omg/03-06-01.pdf

[2] (2006, June) Object management group omg. [Online]. Available: http://www.omg.org/

[3] OMG, *UML 2.0 Infrastructure Specification*, OMG, November 2003. [Online]. Available: http://www.omg.org/docs/ptc/03-09-15.pdf

[4] F. Budinsky, D. Steinberg, E. M. Raymond, E. Timothy, and J. Grose, *Eclipse Modeling Framework*, person education, inc. ed. Addison Wesley, August 2003.

[5] (2006, June) Graphical modeling framework gmf. [Online]. Available: http://www.eclipse.org/gmf/

[6] F. Jouault and I. Kurtev, "Transforming models with atl," in *Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, NfC, MDD, WUsCAM, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers, LNCS 3844*, J.-M. Bruel, Ed. Springer Berlin / Heidelberg, 2006, pp. 128–138.

[7] G. Taentzer, "Agg: A tool environment for algebraic graph transformation." in *AGTIVE*, ser. Lecture Notes in Computer Science, M. Nagl, A. Schürr, and M. Münch, Eds., vol. 1779. Springer, 1999, pp. 481–488.

[8] J. Niere and A. Zündorf, "Using fujaba for the development of production control systems." in *AGTIVE*, ser. Lecture Notes in Computer Science, M. Nagl, A. Schürr, and M. Münch, Eds., vol. 1779. Springer, 1999, pp. 181–191.

[9] ——, "Testing and simulating production control systems using the fujaba environment." in *AGTIVE*, ser. Lecture Notes in Computer Science, M. Nagl, A. Schürr, and M. Münch, Eds., vol. 1779. Springer, 1999, pp. 449–456.

[10] P. Braun and F. Marschall, "Transforming object oriented models with botl." *Electr. Notes Theor. Comput. Sci.*, vol. 72, no. 3, 2003.

[11] IBM, "Model Transformation Framework," http://www.alphaworks.ibm.com/tech/mtf.

[12] *QVT-Merge Group: Revised submission for MOF 2.1 Query/View/-Transformation*, version 2.0 formal/05-07-04 ed., OMG, 2005. [Online]. Available: http://www.omg.org/docs/ad/05-07-01.pdf

[13] M. Wimmer and G. Kramler, "Bridging grammarware and modelware." in *MoDELS Satellite Events*, ser. Lecture Notes in Computer Science, J.-M. Bruel, Ed., vol. 3844. Springer, 2005, pp. 159–168.

[14] A. Schauerhuber, M. Wimmer, and E. Kapsammer, "Bridging existing web modeling languages to model-driven engineering: A metamodel for webml," 2nd International Workshop on Model-Driven Web Engineering, (MDWE'06), Palo Alto, California, 2006.

[15] E. Kapsammer, H. Kargl, G. Kramler, G. Kappel, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "On models and ontologies - a semantic infrastructure supporting model integration." in *Modellierung*, ser. LNI, H. C. Mayr and R. Breu, Eds., vol. 82. GI, 2006, pp. 11–27.

[16] G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W.Retschitzegger, and W. Schwinger, "Towards a semantic infrastructure supporting model-based tool integration," *1st International Workshop on Global integrated Model Management, Shanghai, 22 May 2006*, 2006.

[17] 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," ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy, 2006.

[18] *UML Superstructure Specification, v2.0*, version 2.0 formal/05-07-04 ed., OMG, August 2005. [Online]. Available: http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf

[19] T. Clark, A. Evans, S. Kent, and P. Sammut, "The mmf approach to engineering object-oriented design languages." in *In Proc. Workshop on Language Descriptions, Tools and Applications (LDTA2001)*, 2001.

[20] S. Lechner and M. Schrefl, "Defining web schema transformers by example." in *DEXA*, ser. Lecture Notes in Computer Science, V. Maršík, W. Retschitzegger, and O. Stepánková, Eds., vol. 2736. Springer, 2003, pp. 46–56.

[21] F. Jouault, "Loosely coupled traceability for atl," in *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, Nuremberg, Germany, 2005. [Online]. Available: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/ECMDATraceability05.pdf

[22] A. Borgida and J. Mylopoulos, "Data semantics revisited." in *SWDB*, C. Bussler, V. Tannen, and I. Fundulaki, Eds., vol. 3372, 2004, pp. 9–26.

[23] M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas, "Amw: a generic model weaver," in *Proceedings of the 1re Journe sur l'Ingnierie Dirige par les Modles (IDM05)*, 2005. [Online]. Available: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/IDM_2005_weaver.pdf

[24] F. Jouault and I. Kurtev, "On the architectural alignment of atl and qvt," in *Proceedings of ACM Symposium on Applied Computing (SAC 06), model transformation track*, Dijon, Bourgogne, France, 2006, to appear. [Online]. Available: http://www.sciences.univ-nantes.fr/lina/atl/www/papers/ATLandQVT-PRELIMINARY%20VERSION.pdf

[25] M. M. Zloof, "Query-by-example: the invocation and definition of tables and forms." in *VLDB*, D. S. Kerr, Ed. ACM, 1975, pp. 1–24.

[26] S. Ceri, P. Fraternalia, A. Bongio, M. Bramilla, S. Comai, and M. Matera, *Designing Data-Intensive Web Applications*. Morgan-Kaufmann, 2003.

[27] I. Kurtev, M. Aksit, and J. Bézivin, "Technical Spaces: An Initial Appraisal," *CoopIS, DOA2́002 Federated Conferences, Industrial track, Irvine*, 2002.

[28] A. Repenning and C. Perrone, "Programming by example: programming by analogous examples," *Commun. ACM*, vol. 43, no. 3, pp. 90–97, 2000.

[29] R. S. Amant, H. Lieberman, R. Potter, and L. Zettlemoyer, "Programming by example: visual generalization in programming by example," *Commun. ACM*, vol. 43, no. 3, pp. 107–114, 2000.

[30] J. Edwards, "Example centric programming," *SIGPLAN Not.*, vol. 39, no. 12, pp. 84–91, 2004.

[31] K. Ono, T. Koyanagi, M. Abe, and M. Hori, "Xslt stylesheet generation by example with wysiwyg editing," in *SAINT '02: Proceedings of the 2002 Symposium on Applications and the Internet*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 150–161.

[32] M. Nagl, A. Schürr, and M. Münch, Eds., *Applications of Graph Transformations with Industrial Relevance, International Workshop, AGTIVE'99, Kerkrade, The Netherlands, September 1-3, 1999, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1779. Springer, 2000.