Model Transformations? Transformation Models!

Jean Bezivin[(A)], Fabian Büttner[(B)], Martin Gogolla[(B)],
Frederic Jouault[(A)], Ivan Kurtev[(A)], Arne Lindow[(B)]

University of Nantes, Computer Science Department & INRIA[(A)]
University of Bremen, Computer Science Department & TZI[(B)]

Abstract. Much of the current work on model transformations seems essentially operational and executable in nature. Executable descriptions are necessary from the point of view of implementation. But from a conceptual point of view, transformations can also be viewed as descriptive models by stating only the properties a transformation has to fulfill and by omitting execution details. This contribution discusses the view that model transformations can be abstracted as being transformation models. As a simple example for a transformation model, the well-known transformation from the Entity-Relationship model to the Relational model is shown. A transformation model in this contribution is nothing more than an ordinary, simple model, i.e., a UML/MOF class diagram together with OCL constraints. A transformation model may transport syntax and semantics of the described domain. The contribution thus covers two views on transformations: An operational *model transformation* view and a descriptive *transformation model* view.

1 Introduction

Today it is well accepted that models play an important role in software development. Standards like UML including OCL and the recent QVT (Queries, Views, Transformations) [OMG05] underpin a trend called model engineering [Bez05] which can be seen as a discipline within software engineering.

QVT is a family of languages for the description of model transformations. It is designed to formalize transformations from one model to another model. Source and target models may be formulated in different modeling languages. Many QVT language features are operational in nature. A main intention of QVT seems to formulate transformations which can be executed.

> *A model can tell* ***what*** *something does (specification) as well as* ***how*** *the function is accomplished (implementation). These aspects should be separated in modeling. It is important to get the* ***what*** *correct before investigating much time in the* ***how****.* *[RBJ05, p. 22]*

QVT is strong on the *how* in transformations. This contribution concentrates on the *what* in transformations. QVT focuses on the process and means of going from the source model to the target model. This contribution focuses on the properties of the source and target models and by this characterizes the transformation without going into the details of the transformation process. Transformations are viewed from a modeling perspective as transformation models.

The structure of the rest of the contribution is as follows. Section 2 discusses how transformation models may emerge from model transformations. Section 3 puts forward an example for a transformation model. Section 4 elaborates on advantages and disadvantages of the two views on transformation and model. The contribution is finished with concluding remarks in Sect. 5.

2 From Model Transformations to Transformation Models

In our view, the basic idea of model transformation is presented in Fig. 1 where (at the bottom) a transformation operation Mt takes a model Ma as the source model and produces a model Mb as the target model. This operation Mt is probably the most important operation in model engineering. Being models, Ma and Mb conform to metamodels MMa and MMb. Usually, the transformation Mt has complete knowledge of the source metamodel MMa and the target metamodel MMb. Furthermore, the metamodels MMa and MMb conform to a metametamodel, in this figure, OMG's MOF which in turn conforms to itself.

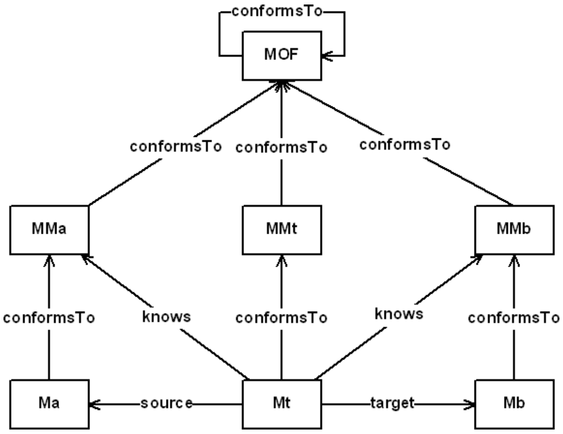

Fig. 1. From a *Model Transformation* Mt to a *Model Transformation Metamodel* MMt

The question of interest discussed in this contribution is concerned with several views on the operation Mt. In fact, one might ask whether *operation* is the right term at all. Our first proposal to view Mt consists in stating that Mt could be a program written in a given (programming) language like Java which upon execution causes the output Mb from the input Ma. Alternatively, if Mt is a transformation expressed in XSLT, then the structure of Mt would be different, but its execution on top of an engine like Saxon would produce a similar effect. Such a view on transformations with focus on execution is understandable because the most important motivation for model transformation is the generation of code from (UML) models.

Before stating a second view on Mt, let us emphasize that we want to work within the realm of model engineering: We want to develop software concentrating on and with the help of models and metamodels; we do not want to focus on code or programs. Model engineering in particular means that one has to ask: Is there a model or metamodel for the thing one is working with; if so, what does the model or metamodel look like? Therefore, it seems natural to introduce a metamodel MMt for Mt. The model transformation Mt must be conformant to MMt, and, if we want to restrict ourselves to a three-level metamodeling stack, then the model transformation metamodel MMt must again conform to our top-level metamodel MOF.

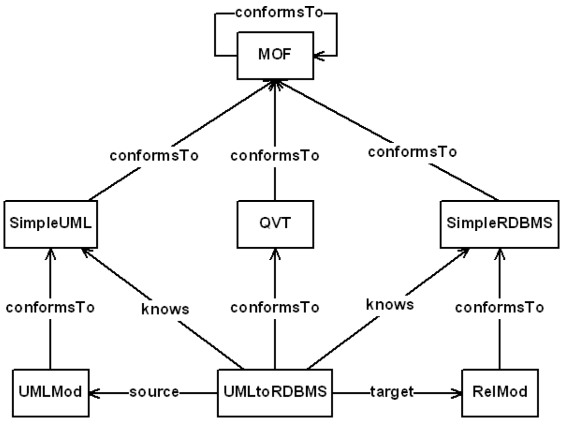

Fig. 2. QVT Example Transformation

Example: In Fig. 2, the above abstract considerations are made more concrete by considering the QVT standard and the example treated there. QVT (which conforms to the MOF) is the metamodel of the model transformation, i.e., a model transformation metamodel. The model transformation UMLtoRDBMS, the example from the QVT standard, describes in an operational way how simple UML class diagrams may be transformed into Relational database schemata.

For our third view on the operation Mt as shown in Fig. 3, we point to the fact that different model transformations Mt1 and Mt2 may work on the same source

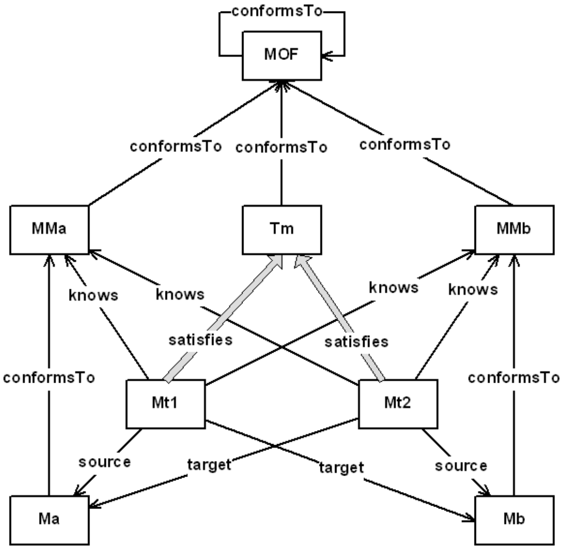

Fig. 3. Model Transformations Abstracted to a Transformation Model

and target and may produce similar results. However, these model transformations may be syntactically different viewed as instantiations of the model transformation metamodel MMt (which is not shown in Fig. 3). In order to emphasize the commonalities between these different model transformations, we propose to identify the commonalities by a *model transformation model* Tm, shortly denoted by the term *transformation model*, which abstracts from the technical realization details of the different model transformations and summarizes and concentrates the similarities. We expect that the different model transformations all satisfy what is required in this transformation model. This satisfies relationship is indicated by the thick grey arrows. Having set this context, we state the hypothesis which we would like to discuss further in this contribution:

Model transformations can be abstracted to a transformation model.

The reader may check, that the three highly related notions *model transformation*, *model transformation metamodel* and *model transformation model*, for short denoted as *transformation model*, mean different things to us. As indicated in Fig. 3, the transformation model again conforms to our metametamodel, in our case MOF. Speaking in technical terms, this means that we only employ MOF features for the formulation of our transformation model.

3 Er to Rel: A Transformation Model Example

We want to show the usefulness of the concept *transformation model* through a proof by example. The example chosen here is the well-known transformation from the Er database model to the Relational database model. This example is also used (with a bit different terminology) in the current QVT proposal [OMG05], in [Bez05] and other works on model transformation [CESW04]. Because it is well-known, it is well-suited to demonstrate ideas and technical details of transformation principles.

3.1 Technical details of the Example Transformation Model

As indicated above, we employ MOF for the formulation of transformation models. Thus, a transformation model is nothing more or less than a MOF model: We need a moderate class diagram and many OCL constraints. These language features are supported by our system USE [RG01,GBR05] in which we have completely realized this transformation example and which we employ as a MOF compliant validation system.

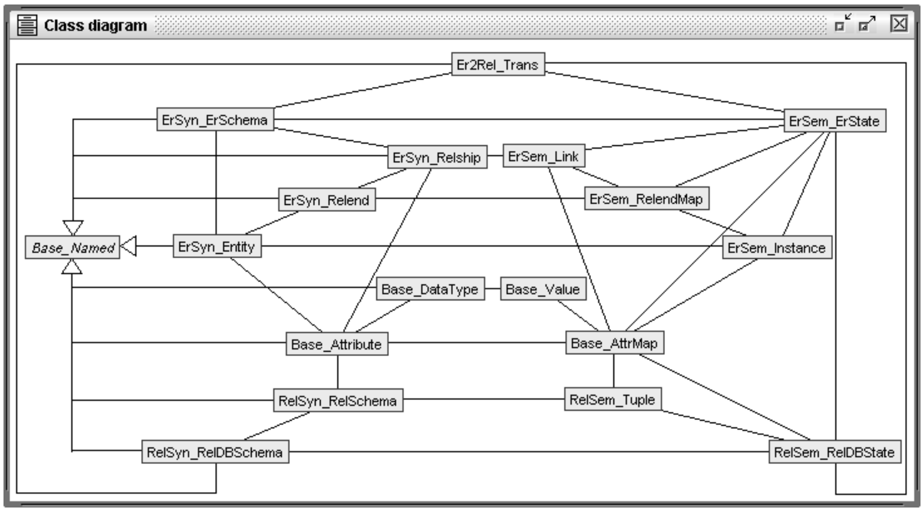

Fig. 4. Class Diagram for Transformation Model

The class diagram in Fig. 4 shows the six parts of the transformation model: Class names starting with Base are shown in the middle, ErSyn in the upper left, ErSem in the upper right, RelSyn in the lower left, RelSem in the lower right, and Er2Rel in the top. Generalization and associations are pictured as well. ErSyn describes the syntax of the Er model, namely Er database schemas; ErSem describes the semantics of the Er model, namely Er database states; RelSyn describes the syntax of the Relational model, namely Relational database schemas; RelSem describes the semantics of the Relational model, namely Relational database states. For example, in the Er semantics part, the assignments of attribute values to instances is handled and a constraint is stated that the key attributes have to uniquely identify the instances.

All syntax classes (Er and Rel) can be found in left, all semantics classes (Er and Rel) in the right; all Er classes (Syn and Sem) are in the upper part, whereas the Rel classes (Syn and Sem) are in the lower part. In case the reader is interest in details like association multiplicities or constraint details, the full description in [Gog06] can be consulted; we will illustrate this transformation model in the following by some simple object diagrams and by sketching the transformation constraints.

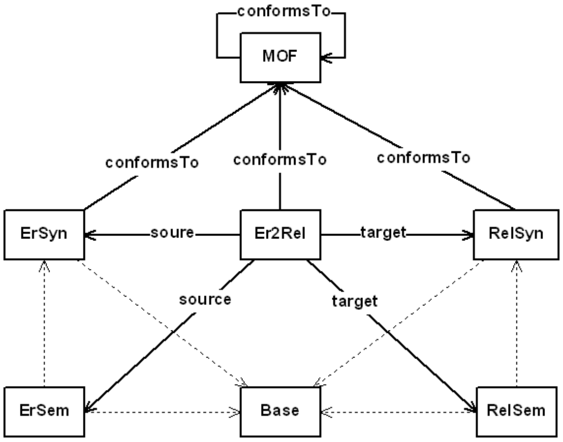

Fig. 5. Example Transformation viewed as a Transformation from Er to Rel

Figure 5 shows the six parts of the class diagram similar to the previously mentioned example transformation in Fig. 2 from the QVT standard. The dashed arrows indicated dependencies.

Structuring a transformation into a source metamodel, a target metamodel, and a metamodel part for the actual transformation is not new. This idea is present, for example, in the QVT standard [OMG05] and the triple graph grammar approach [KS06].

In our approach we constrain all three components with OCL constraints, i.e., the source, the target, and the actual transformation. As shown in Fig. 6, in addition, we divide source and target metamodels into a syntactic and a semantic part. This enables us to formulate transformation properties expressing syntactic and also semantic characteristics.

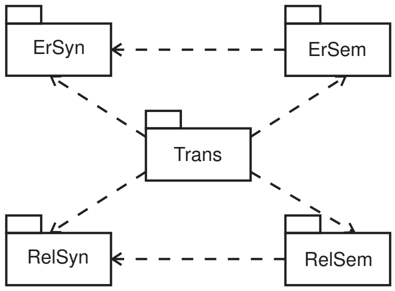

Fig. 6. Syntax, Semantics, and Transformation

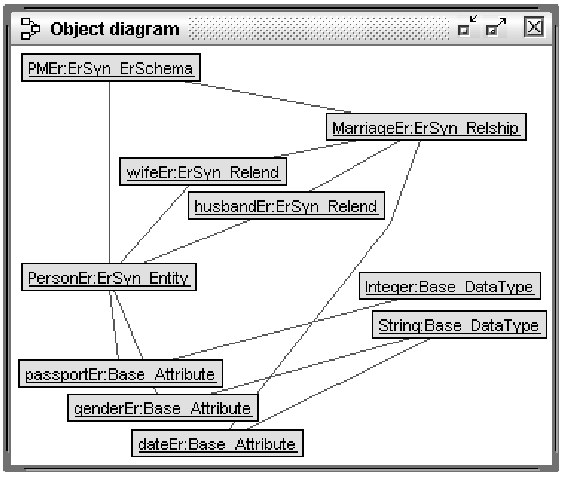

Fig. 7. Er Syntax

Figure 7 shows an Er database schema PMEr (PersonMarriage Er version) modeling an entity Person and a reflexive relationship Marriage together with three attributes and two relationships ends.

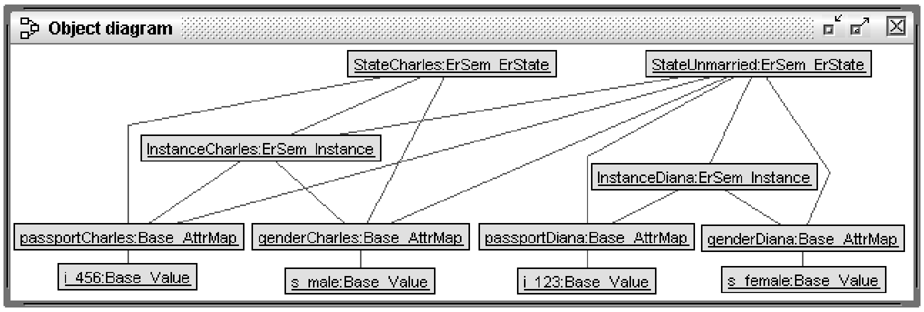

Fig. 8. Er Semantics

Figure 8 pictures two Er database states. The first state (StateCharles) incorporates one Instance (Charles) and AttrMap objects assigning attribute values to instances; the second state (StateUnmarried) has two Instances (Charles, Diana) and attribute assignments. In order to make the presentation simple, both states do not have links. We emphasize that the two database states are part of a single (larger) object diagram for the complete transformation model.

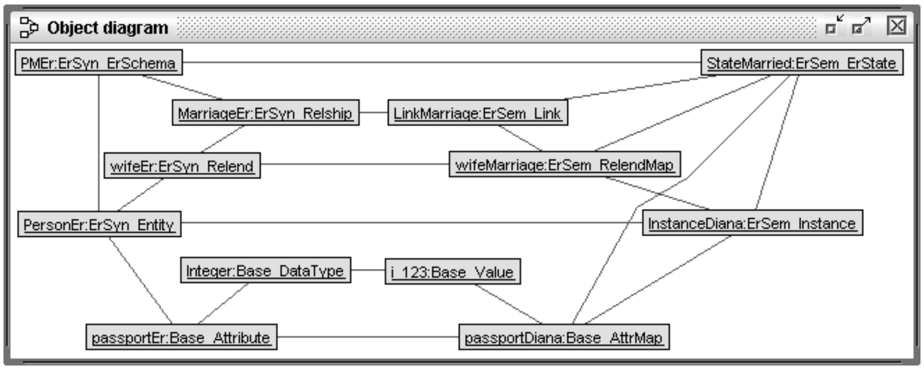

Fig. 9. Interplay between Syntax and Semantics

Figure 9 displays the interplay between syntax and semantics with a (partial) object diagram. A syntactical thing from the left is associated and interpreted by semantic things from the right. To make the presentation comprehensible, each Er syntax concept is associated with only one Er semantic object. This part shows a third database state with a marriage link (the husband is ignored in the display).

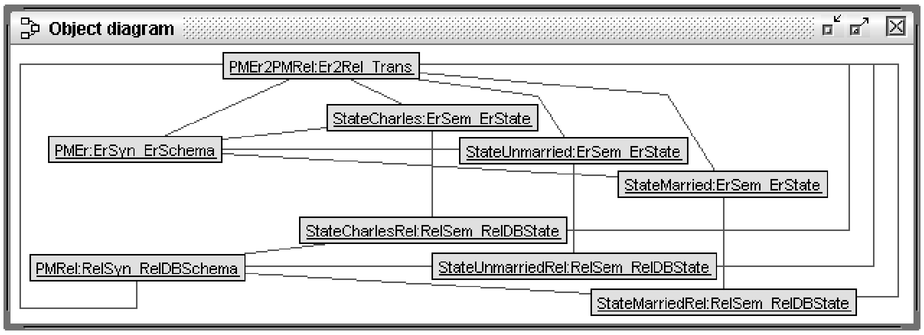

Fig. 10. Transformation

Figure 10 shows a Trans(formation) object which connects the schemas (the syntax parts) and the states (the semantics parts). In general, a transformation object will connect source and target objects by links expressing that the source may or must be transformed into the target (depending on the stated multiplicities and constraints). One schema is associated (in this example object diagram) with three database states. This transformation model covers syntax and semantics of the two classical database models. As will be explained below, the model covers the transformation and its properties as well. Database dynamics is captured insofar that more than one state can be associated with a single database schema. In the example, one can think of the first state having only the Charles instance, the second state having Charles and Diana as unmarried instances, and the third state with a marriage link between Diana and Charles.

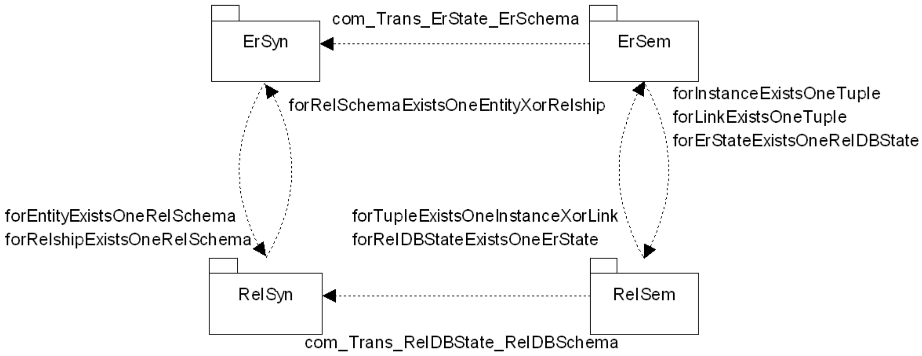

Fig. 11. Overview on Transformation Constraints

Figure 11 gives an overview on the probably most interesting part of the transformation model: the constraints for the transformation. The figure involves the four central areas (Er and Rel; Syn and Sem) with dependencies, constraint names and indication of the 'direction of the constraint'. We explain three constraints in more detail.

forRelSchemaExistsOneEntityXorRelship: This constraint 'goes from' the Relational syntax part to the Er syntax part. It requires that for a Relational schema from a transformed Relational database schema a uniquely determined entity or relationship in the Er schema with the same characteristics exists.

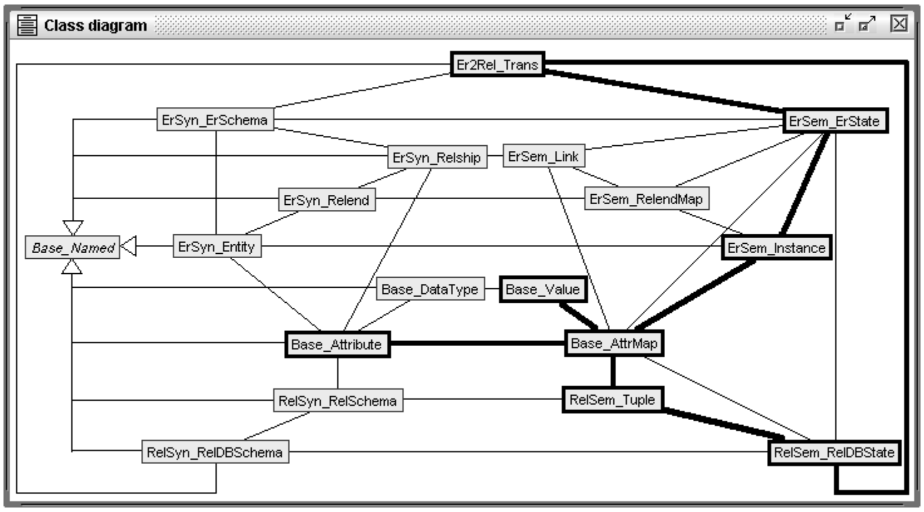

Fig. 12. Class Diagram Illustrating Constraint forInstanceExistsOneTuple

forInstanceExistsOneTuple: This constraint 'goes from' the Er semantics part to the Relational semantics part. It requires that for an instance from an Er state occurring in a transformation an equivalent tuple in the Relational state exists.

```
context self:Er2Rel_Trans inv forInstanceExistsOneTuple:
  self.erState->forAll(erSt | self.relDBState->one(relSt |
    erSt.instance->forAll(i | relSt.tuple->one(t |
      i.attrMap->forAll(amEr |
        t.attrMap->one(amRel |
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value))))))
```

com_Trans_ErState_ErSchema: This constraint ‘goes from’ the Er semantics part to the Er syntax part. Constraints starting with ‘com’ are commutativity constraints requiring the commutativity of two different evaluation paths in the class diagram. This one requires that an Er state which is connected to a Trans(formation) object must also be linked to the Er schema being associated to the Trans(formation) object.

3.2 Explanation for Calling the Example a Transformation Model

Semantic properties: We have modeled the transformation with a class and corresponding associations holding source and target object. By doing so, semantic properties of the transformation can be formulated because we can access source and target and retrieve their properties. In the example, a bijection between database state spaces is described. But by dropping certain constraints, this requirement could be relaxed to achieve only an injection. For example, we could only require that each Er database state has a corresponding equivalent Relational database states but not the other way round. The required properties of the transformation rely merely on the stated constraints and are under control and responsibility of the transformation developer. Only the properties of the transformation are stated, not the realization of the transformation.

Alternatives: In the example, we have decided to make the transformation deterministic. In general however, transformation alternatives can be allowed in a single transformation model. For example, the transformation model may allow two or more alternative Relational schemas to be associated with one Er schema.

4 Model Transformation versus Transformation Models

Executability: Model transformations can directly and efficiently be executed. There is an international standard for them, QVT, and commercial and open source implementations and systems like UMT, MTL, ATL, GMT or BOTL are available (see the overview on transformation systems in [Wan05]).

Direction freeness: Transformation models may be seen as transformations in multiple directions. Please check Fig. 13 which is nearly identical to Fig. 5 except the central source and target decorations. Apart from the direction (Er to Relational) which we have already discussed, the transformation model may be seen in two other directions: As a transformation from the Relational database model to the Er database model and as a transformation from syntax to semantics. In technical terms, a transformation direction has not to be fixed in the model. This is based on the use of direction-free minimal MOF language features: Classes, associations, attributes, and invariants.

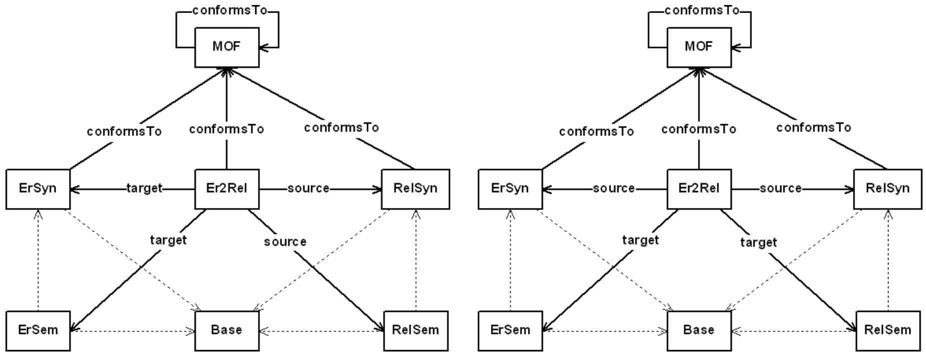

Fig. 13. Two Further Views on Example Transformation (Different Source/Target)

Uniformity: Transformation models provide uniformity between the model description language and the language for transformations. If one has simple models, for example, UML class diagrams with OCL constraints, then the use of this language for transformations reliefs the development from the burden of introducing another language like QVT. In particular in early project development phases, it might be advisable to concentrate on transformation properties by expressing them in transformation models instead of realizing them already by model transformations.

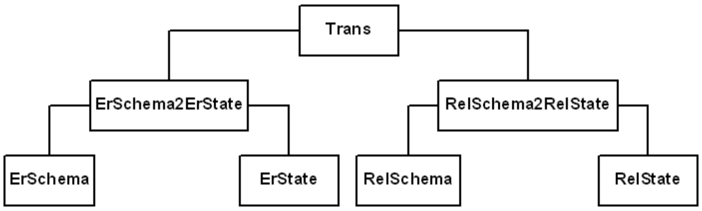

Fig. 14. Example for Higher-Order Transformation

Higher-order transformations: Uniformity of the model and transformation language also allows for higher-order transformations, i.e., transformations that work on transformations. Our example could be understood and realized as such a higher-order transformation: As shown in Fig. 14, assigning semantics to the schemas could be seen as two basic transformations realized through two classes ErSchema2ErState and RelSchema2RelState and appropriate associations; the transformation from the Er model to the Relational model could then be realized in a higher-order style by a third class Trans(formation) with associations to the two transformation classes ErSchema2ErState and RelSchema2RelState.

Transformations of Transformations: Working with transformation models provides for the possibility for rewriting transformation models exactly as they were ordinary models. Thus refactorings and improvements for general models [ZLG05,GSMD03] and UML models [SGJ04,CW04,BSF02,SPTJ01] would be applicable.

Validation and completions: Standard transformation models can be validated and checked with standard UML and OCL validation tools [GBR05,Chi01]. Model finders (like Alloy [JSS00], to some extent USE [RG01,GBR05]) can be employed for finding completions of partially given transformations. In the example, if only the database schemas and the Er state is provided, a model finder could search for the resulting Relational state without explicitly describing it. Tools and approaches based on formal reasoning [ABB^{+}00,JSS00,KFdB^{+}05] can check transformations models w.r.t. formally derivable properties. Such formal reasoning capabilities could be used for formally checking the compatibility of two modeling languages.

Complete language descriptions: Transformation models allow complete descriptions of modeling languages w.r.t. syntax and semantics and their transformation properties to be described within a single framework. This is in contrast to mainstream modeling languages like UML which do not formally describe semantic domains.

5 Conclusion

In this contribution we have discussed model transformations and transformation models. We have put our work into the context of UML, OCL, MOF and QVT. The main benefits we see for transformation models are direction freeness, uniformity, higher-order transformations, and powerful possibilities for validation and verification. The benefits of model transformations lie in the efficient execution and the availability of practically useful systems.

Further work has to investigate to what extent available transformation systems can be used for transformation model purposes. Further examples for transformation models, in particular transformation models between modeling languages, have to be developed. It seems that syntax and semantics of hierarchical and flat statecharts as well as advanced and basic UML class diagrams can be characterized as transformation models. Lastly, the connection between model transformations and transformation models on the one hand and domain specific languages and profiling of modeling languages on the other hand has to be explored.

Acknowledgments

A subset of the authors have been partially supported by the IST European project ModelWare (Contract 511731).

References

[ABB+00] W. Ahrendt, T. Baar, B. Beckert, M. Giese, E. Habermalz, R. Hähnle, W. Menzel, and P. H. Schmitt. The KeY approach: Integrating object oriented design and formal verification. In M. Ojeda-Aciego, I.P. de Guzmán, G. Brewka, and L. M. Pereira, editors, *Proc. 8th European Workshop Logics in AI (JELIA'2000)*, LNCS 1919, pages 21–36. Springer, 2000.

[Bez05] J. Bezivin. On the Unification Power of Models. *Software and System Modeling*, 4(2):171–188, 2005.

[BSF02] M. Boger, T. Sturm, and P. Fragemann. Refactoring browser for UML. In M. Aksit, M. Mezini, and R. Unland, editors, *NetObjectDays*, volume 2591 of *Lecture Notes in Computer Science*, pages 366–377. Springer, 2002.

[CESW04] T. Clark, A. Evans, P. Sammut, and J.S. Willans. Transformation language design: A metamodelling foundation. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *ICGT*, volume 3256 of *LNCS*, pages 13–21. Springer, 2004.

[Chi01] D. Chiorean. Using OCL Beyond Specifications. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Proc. UML'2001 Workshop Rigorous Development*, pages 57–68. LNI, GI, Bonn, 2001.

[CW04] Alexandre L. Correa and Cláudia Maria Lima Werner. Applying refactoring techniques to UML/OCL models. In T. Baar, A. Strohmeier, A.M.D. Moreira, and S. J. Mellor, editors, *UML*, volume 3273 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2004.

[GBR05] M. Gogolla, J. Bohling, and M. Richters. Validating UML and OCL Models in USE by Automatic Snapshot Generation. *Journal on Software and System Modeling*, 4(4):386–398, 2005.

[Gog06] M. Gogolla. Tales of ER and RE Syntax and Semantics. In J.R. Cordy, R. Lämmel, and A. Winter, editors, *Transformation Techniques in Software Engineering*, number 05161 in Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, 2006.

[GSMD03] P. Van Gorp, H. Stenten, T. Mens, and S. Demeyer. Towards automating source-consistent UML refactorings. In P. Stevens, J. Whittle, and G. Booch, editors, *UML*, volume 2863 of *Lecture Notes in Computer Science*, pages 144–158. Springer, 2003.

[JSS00] D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: The Alloy constraint analyzer. In *Proc. Int. Conf. Software Engineering (ICSE'2000)*, pages 730–733. ACM, New York, 2000.

[KFdB+05] M. Kyas, H. Fecher, F. S. de Boer, J. Jacob, J. Hooman, M. van der Zwaag, T. Arons, and H. Kugler. Formalizing UML models and OCL constraints in PVS. *Electr. Notes Theor. Comput. Sci.*, 115:39–47, 2005.

[KS06] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars: A Survey. In R. Heckel, editor, *Proc. SegraVis School Foundations of Visual Modelling Techniques*, *ENTCS 148*, pages 113–150. Elsevier, 2006.

[OMG05] OMG, editor. *MOF QVT Final Adopted Specification*. OMG, 2005.

[RBJ05] J. Rumbaugh, G. Booch, and I. Jacobson. *The Unified Modeling Language Reference Manual, Second Edition*. Addison-Wesley, Reading, 2005.

[RG01] M. Richters and M. Gogolla. OCL - Syntax, Semantics and Tools. In T. Clark and J. Warmer, editors, *Advances in Object Modelling with the OCL*, pages 43–69. Springer, Berlin, LNCS 2263, 2001.

[SGJ04] H. Schippers, P. Van Gorp, and D. Janssens. Leveraging UML Profiles to generate Plugins from Visual Model Transformations. In *Proc. ICGT Workshop Software Evolution through Transformations*, 2004.

[SPTJ01] G. Sunyé, D. Pollet, Y. Le Traon, and J.-M. Jézéquel. Refactoring UML models. In M. Gogolla and C. Kobryn, editors, *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2001.

[Wan05] W. Wang. *Evaluation of UML Model Transformation Tools*. Technical University of Vienna, Business Informatics Group, Master Thesis, 2005.

[ZLG05] J. Zhang, Y. Lin, and J. Gray. Generic and Domain-Specific Model Refactoring using a Model Transformation Engine. In S. Beydeda, M. Book, and V. Gruhn, editors, *Model-Driven Software Development*, pages 199–218. Springer, 2005.