Exploring ER and RE Syntax and Semantics with Metamodel Object Diagrams

Martin Gogolla
University of Bremen, Computer Science Department
Database Systems Group, D-28334 Bremen, Germany

ABSTRACT

This paper explains how a metamodel transformation between two well-known database models work: Entity-Relationship (ER) database schemas are transformed into Relational (RE) database schemas hand in hand with the transformation of ER states into Relational states. The schemas determine the syntax of the datamodels, and the set of associated states fix the semantics of the datamodels. Often, when database models are considered, one formally only treats syntactical aspects, i.e., schemas, and handles the semantics, i.e., states, merely informally. Our approach formally handles syntax and semantics of database models and their transformation within a single, uniform framework. The approach thus precisely describes properties of the datamodels and properties of the transformation. Our metamodel transformations do not simply convert ER schemas into Relational schemas, but they respect the semantics in a formal way and they can also be used the other way round in order to transfrom Relational schemas into ER ones. This opens the approach for re-engineering as well.

1. INTRODUCTION

Within the database community the Entity-Relationship (ER) and the Relational (RE) database models have been studied and used for decades. Both models are subject to introductory courses in database and software engineering education. A typical course will introduce the main concepts of both database models in an informal way, explain how to transform ER schemas into Relational database schemas and will deepen the subject by practical exercises using a design tool and a database system.

Within the software engineering community, metamodeling has gained recently much attention, in particular in connection with the Unified Modeling Language (UML) [1] and the Model Driven Architecture (MDA) [2]. This paper proposes another metamodel formulated with UML and OCL [3] for the ER and RE database models. We will use only such UML features available also in the Meta Object Facility (MOF) which is a subset of UML. In contrast to known approaches, this paper however describes with its metamodel not only the concepts of the database schemas but also models a formal interpretation, i.e., a formal semantics, for ER and RE database schemas: The paper formally connects database states to database schemas. Furthermore, the transformation between the database models is also described in a formal way, such that semantical properties of the transformation can be expressed on the basis of the formal semantics of the database models. We are not aware of another approach handling these two classical database models with respect to syntax (schemas) and semantics (states) and their transformation in a rigorous and uniform way. In particular, we are not aware of an approach being able to express the ultimate goal of the schema transformation process, namely the equivalence between the database states for the different models, in a formal and explicit way. Although both datamodels are well-known, the paper gives new insights into the models and their transformation by making usually implicit assumptions explict which is particular useful for non-experts and beginners.

At this point we would like to draw the attention to the fact that above we made use of the notion *model* in two different contexts: *Database model* and *metamodel*. A *database model* denotes a collection of interrelated concepts for describing certain classes of databases: The ER database model knows, for example, relationships, and instances; the Relational database model talks, for example, about attributes, and tuples. The *metamodel* which we develop in this paper is a collection of UML (MOF) classes, associations, and OCL constraints. This *metamodel* covers the two *database models* and their *transformation*.

Our work has connections to related approaches. In [4, 5] deficiencies of the UML metamodel architecture with respect to syntax and semantics are discussed. Similar to our approach, in [6] a framework for general transformations is described with UML and OCL. [7, 8] study a general metamodeling framework but do not explicitly distinguish between syntax and semantics. A graph-based, visual method for metamodeling mathematics is put forward in [9]. Metamodels for datamodels [10] and database languages have been studied with emphasis on various aspects like database evolution [11], database design and conceptualization [12, 13], object-orientation [14] or re-engineering database schemas [13, 15]. The idea of having a strict sep-

Proc. Metainformatics'2005 (MIS'2005), ACM Int. Conf. Series, P. Nürnberg (Ed.), ACM, New York, 2007.

aration between syntax and semantics (without giving detailed examples) can also be found in [16]. Our approach has also connections to other work done within the interdisciplinary metainformatics area. For example, in [17, 18] metamodels for structural computing and collections therein have been put forward. These metamodels and their transformation could be described with our technique. In [19] a general frame for technical spaces is discussed. Our approach may be understood as one concrete instance of this general frame.

The speciality of our present approach is however that we describe with our metamodel not only the concepts of database schemas but also give in detail their formal interpretation and thus we could formally introduce correctness criteria for metamodel transformations. Our paper continues work started for an EER model in [20, 21] and being continued with UML (MOF) as a description language in [22, 23, 24]. Full details of all our OCL constraints (being dropped in this paper according to space restrictions) can be found in [25].

The rest of this paper is structured as follows. Section 2 will give an overview on the resulting UML (MOF) metamodel mainly in form of a class diagram. First, a package diagram will state the overall metamodel structure. Second, both database models (ER and RE) are sketched. Third, the concepts for the transformation are introduced. Section 3 concentrates on treating common concepts from the ER and RE database models and the ER syntax and the ER semantics; then the RE syntax and the RE semantics are handled; hereafter concerns which are common to both datamodels are studied; thereafter the transformation is detailed. Section 3 discusses these topics in form of object diagrams exploring the class diagram from Sect. 2. Section 4 turns to the basic idea behind the modeling method in this paper and emphasizes the role of tool support. Section 5 discusses typical questions and section 6 finishes the paper with a conclusion.

2. OVERVIEW BY CLASS AND PACKAGE DIAGRAMS

2.1 Package Diagram with Class Names

Figure 1 shows the packages we have used and their dependencies. The package Base contains fundamental classes, associations, and constraints with elements common to both the ER and RE datamodel. The package ErSyn includes elements describing the syntax of the ER datamodel whereas the package ErSem covers elements relevant for the semantics of the ER datamodel. Analogously the package RelSyn and RelSem are responsible for the syntax and semantics of the RE datamodel. The package DataMods contains remaining syntactical and semantical elements only expressible for both datamodels together. In DataMods further invariants but no new classes or associations are introduced. The package Er2Rel describes modeling elements relevant for the transformation between the ER datamodel and the RE datamodel.

A closer look at the dependencies in Fig. 1 should help in understanding the interconnections between and the structure of the packages. The package Base contains modeling elements for ER and RE attributes and data types and describes syntactical as well as semantical aspects. Both modeling elements, attributes and data types, occur in the ER and in the RE datamodel. Therefore these elements are considered separately. In order to describe the semantics of both data models one has to know their syntax beforehand, thus the semantic packages ErSem and RelSem rely on the syntax packages ErSyn and RelSyn. The packages DataMods and Er2Rel finally unite the two different lines for the ER and RE model in order (1) to express restrictions that each apply to both datamodels together and (2) to express the transformation between the datamodels.

2.2 Modeling ER and RE Syntax and Semantics

The class diagram in Fig. 2 gives an overview on central classes and associations by showing four clouds: In the left part a solid grey and a solid black cloud, in the right part a dashed grey and a dashed black cloud. The two solid left clouds model the syntax of the datamodels, the two dashed right clouds the semantics. The top grey clouds describe the ER datamodel, the bottom black clouds the RE datamodel. The ER and the RE datamodel share some concepts, namely the parts in the middle talking about data types, attributes and their semantics. The four clouds correspond to five packages in Fig. 1, namely to Base, ErSyn, ErSem, RelSyn, and RelSem. The two intersections of the clouds correspond to Base, the remainder of the solid grey cloud to ErSyn, the remainder of the dashed grey cloud to ErSem, the remainder of the solid black cloud to RelSyn, and the remainder of the dashed black cloud to RelSem. We have used only binary associations, because our description is supposed to be MOF compliant, and MOF only knows binary associations.

Syntax of the ER datamodel: This part introduces the classes ErSchema, Entity, Relship, Relend, Attribute, and DataType. ErSchema objects consist of Entity and Relship objects which in turn may possess Attribute objects typed through DataType objects. Relend objects represent the connection points between the Relship objects and the Entity objects. All elements on the syntax side require an attribute expressing their name. Therefore, we have factored out this attribute, build an abstract class Named and made all classes on the syntax side subclasses of this class Named. The class Named is not shown in Fig. 2.

Semantics of the ER datamodel: In this part we set up the classes ErState, Instance, Link, RelendMap, AttrMap, and Value. The interpretation is as follows. An ErSchema object is interpreted by possibly many ErState objects. An Entity is given semantics by a set of Instance objects, and a Relship by a set of Link objects. DataType objects are given life through a set of Value objects. Relend and Attribute objects are interpreted by a set of RelendMap objects and AttrMap objects, respectively.

Syntax of the Relational datamodel: This part shows the classes RelDBSchema, RelSchema, Attribute, and DataType. RelDBSchema objects consist of RelSchema objects which possess Attribute objects typed through

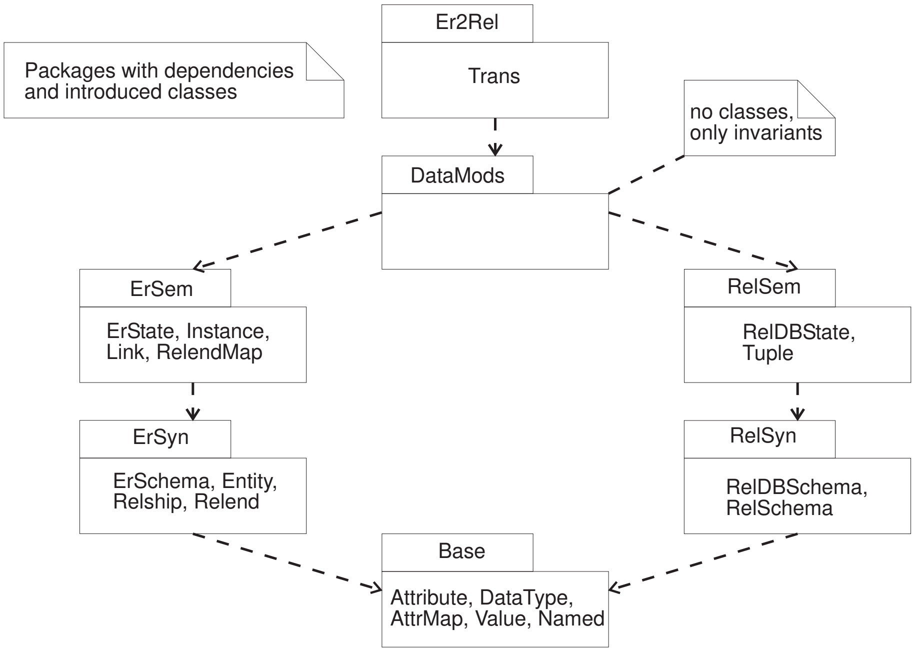


Figure 1: Package Diagram with Class Names

DataType objects. The difference in the notions RelDBSchema (Relational database schema) and RelSchema (Relational schema) is that a RelDBSchema consists of several RelSchemas or in other words that a collection of RelSchemas constitutes a RelDBSchema.

Semantics of the Relational datamodel: The last part utilizes the classes RelDBState, Tuple, AttrMap, and Value. RelDBSchema objects are interpreted by a set of RelDBState objects. Each RelDBState object consists of a set of Tuple objects. Tuple objects in turn consist of a set of AttrMap objects assigning a Value object to an Attribute within a Tuple and a RelDBState. A Tuple belongs to exactly one RelSchema as the multiplicity 1 on the association to RelSchema guarantees.

The class diagram and the resulting textual description follows certain naming and layout conventions apart from the principles *Syntax-Left-Semantics-Right* and *ER-Top-RE-Bottom*. Role names are not stated explicitly, but are always determined by the UML (MOF) and OCL rule that unspecified role names consist of the respective class name with the first letter being given as a lower case letter. All associations within the syntax side are displayed with straight lines and the same holds for the associations within the semantics side, but the associations between syntax and semantics are shown with bent lines. These bent associations always have multiplicity 1 0..* and express that for every syntactical class in the left there is a semantical class in the right and that one syntactical object from the left is interpreted by a set of semantical objects from the right. These associations may be seen as a typing mechanism, because a semantical object is always associated with exactly one syntactical object, its type. Exactly as a syntax class from the left possesses a corresponding semantics class on the right, each syntax association from the left has a corresponding semantics association on the right. In addition the semantics side on the right has three associations between ErState and RelendMap, between ErState and AttrMap and between RelDBState and AttrMap which do not have corresponding syntactical parts on the left.

A further observation concerns implicitly present part-of relationships. Due to the multiplicities and constraints on the syntax side, every object on the left will be connected to a schema, and due to the multiplicities and constraints on the semantics side every object on the right will be connected to a state. This means any object originating from the left side will be connected directly or indirectly to either an ErSchema or a RelDBSchema object, and any object originating from the right side will be linked directly or indirectly to an ErState or a RelDBState object. The only exception to this rule are DataType and Value objects which may exist independently from schema objects.

2.3 Transformation between ER and RE

The class diagram in Fig. 3 describes the transformation between the ER datamodel and RE datamodel and shows the constituents of the package Er2Rel. It introduces one additional class Trans which represents transformation objects. A transformation object can express with its links that one ErSchema is transformed into a RelDBSchema together with the corresponding ErState objects transformed into RelDBState objects via the association between Trans and ErState and the association between Trans and RelDBState as well as via the association between ErState and RelDBState. This addition association between ErState and RelDBState expresses which ER state correspond to which RE state. This is not already expressed in the pure set of states associated

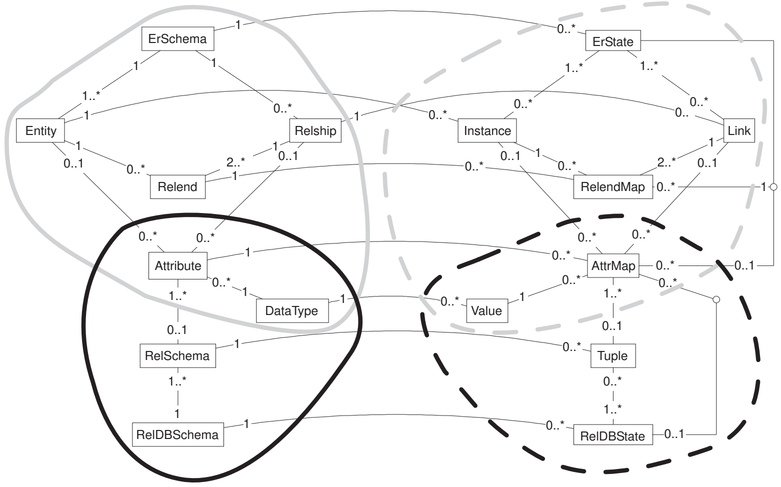


Figure 2: Class Diagram Modeling the ER and RE Datamodel

with the transformation. Indeed, this last association could also more precisely be expressed as a ternary association with Trans as an additional class. Remember however, we use MOF, which offers only binary associations for modeling, and then this situation would have to be modeled with an additional class making the scenario much more complicated than the simpler class diagram which we have chosen.

By expressing the transformation by a transformation class and intentionally not, e.g., by operations we do not introduce any direction into the transformation. One tends to think of the connection between the ER and RE model as *transforming an ER schema into a RE database schema* (see our package name Er2Rel), but the opposite direction is worth to be considered for re-engineering purposes as well. Our design decision to represent transformations as classes together with appropriate associations does not impose any transformation direction. Forward and backward engineering techniques are covered in principle by this direction-neutral design.

Thus it is only one view on the class diagram in Fig. 3 that it represents a transformation from the top ER part to the bottom RE part. Another view is that a transformation from the bottom RE part to the top ER part is described. And yet another view is that the left syntax part is transformed into the right semantics part. That transformation is a relationship between syntactical elements, in this case pairs of schemas, and semantical elements, in this case pairs of states.

3. EXPLORATION WITH OBJECT DIAGRAMS

This section explains the details of our language and transformation metamodel shown in Figs. 2 and 3 by considering an example scenario. Object diagrams are means to communicate complex designs with concrete examples to experts and also to non-experts. Figure 4 shows this scenario which we want to study. Put in simple words, the scenario says: Diana marries Charles. The figure is structured into four parts starting on the top: The two top parts show the ER version of the scenario, the two bottom parts the Relational version; for the ER version as well as the Relational version first a database schema and afterwards a state is shown.

The ER database schema displays one entity Person having two attributes, namely passport number and gender, with the key attribute passport being underlined and one reflexive relationship Marriage with indicated relationship end names wife and husband and one relationship attribute date. The ER state has two instances for entity Person, diana and charles, and assigns a passport and a gender value to both. The ER state also pictures a link between the two instances expressing that their marriage took place on 1981/07/29.

The Relational database schema possesses two Relational schemas originating from the transformation of the ER database schema. The entity Person is described by a Relational schema Person having the same attributes as the entity. The relationship is represented by a Relational schema Marriage having in particular two attributes, one for each relationship end, where the attribute names must utilize the relationship end names and the name of the key attribute: wife_passport and husband_passport. The facts from the ER state, i.e., two instances and one link, are represented in the Relational state by three tuples.

3.1 Base

We now want to represent the four parts from Fig. 4 as instantiations of the metamodel we have sketched before. We start with the package Base. Since the ER database schema and the RE database schema both utilize the data types Integer and String, we first introduce these data types. Both data types are pictured in Fig. 5 as objects of class DataType. In Fig. 5 also Value objects are shown which represent the semantical interpretation of DataType objects. One data type can have a set of values associated with it. The five Value objects have been given object identifiers which already in-

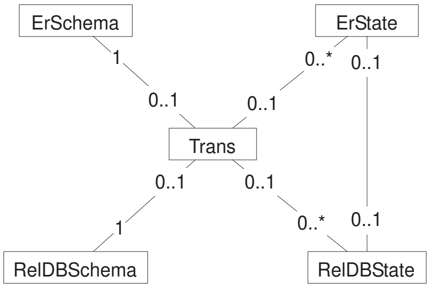


Figure 3: Class Diagram Modeling the Transformation between ER and RE

dicate their content attribute. Alternatively, we could have denoted these objects, for example, with object identifiers value1 to value5. Then, we had to use these object identifiers in the object rectangles before the colons instead of the object identifiers from Fig. 5, but the content attribute would be as in Fig. 5.

Constraints have to restrict the possible object diagrams allowed by the class diagram in the package Base. For example, there will be a constraint guaranteeing a unique representation of each value. Thus it will be forbidden to have another Value object called, for instance, i_123B being also linked to data type Integer and having also content='123'. In the package Base also the class Attribute and AttrMap are present. Thus, for example, the attribute passport from the ER schema or the attribute passport from the RE database schema belong to this package. Concrete objects for this part will be displayed later in connection with the database schemas and database states.

3.2 ER Syntax

The ER database schema is shown as an object diagram in Fig. 6. All object identifiers (the part before the colon in the object rectangles) end with Er whereas the forthcoming RE database schema will show Rel as endings. The ER schema itself being not explicitly present in Fig. 4 is called PMEr as a shorthand for Person-Marriage. This object diagram introduces one ErSchema object, one Entity object, two Attribute objects for the entity, one Relship object, two Relend objects for the relationship ends, and one relationship Attribute object. Links establish proper connections so that the graphically represented ER schema is captured formally as an object diagram.

Constraints have to restrict the class diagram so that only meaningful object diagrams are allowed to be drawn. For example, with respect to the class diagram only it would be allowed to have an additional link in Fig. 6 between the attribute dateER and the entity PersonER. This has to be excluded.

3.3 ER Semantics

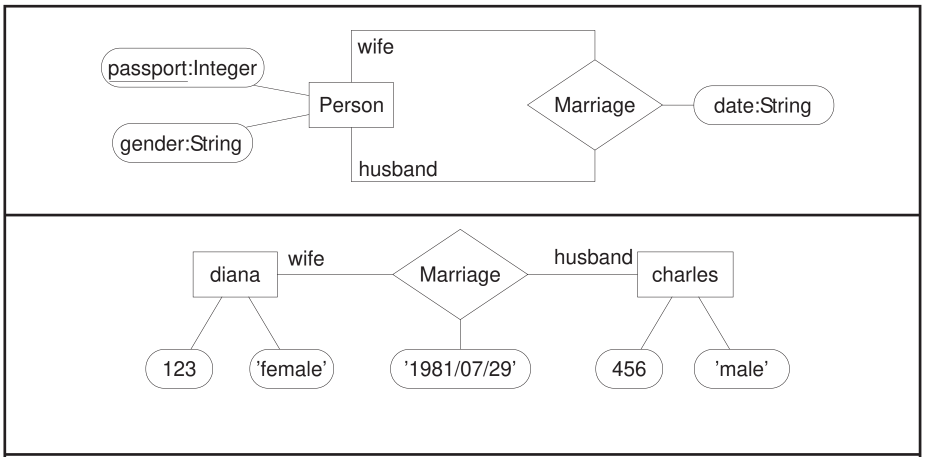
Following the overview diagram in Fig. 4 we now come to the ER state being given in Fig. 7. First, we identify the objects: One ErState object, two Instance objects, one Link object, two RelendMap objects for the link ends, and five AttrMap objects for the attribute values. Second, links establish the proper connections.

As before, constraints must restrict the possible object diagrams. For example, the above object diagram would become invalid, if the link between dianaPassportEr and i_123 would be replaced by a link between dianaPassportEr and s_female: Attribute typing restrictions would be violated.

The lower part of the object diagram with objects of class AttrMap and Value belongs to the package Base. Remember that data types and values as well as attributes and attribute maps are also used by the RE datamodel. The object diagram shows semantical objects only, but remember that all semantical objects are typed by syntactical objects. Thus all objects in the above ER state diagram possess links not shown here, but present in the complete object diagram. This fact is emphasized in Fig. 8.

In the left of Fig. 8, part of the objects belonging to the ER schema are shown and, in the right of the figure, part of the objects belonging to the complete ER state are displayed. The links in the middle are all typing links. For example, the semantical object stateEr is typed by the syntactical object PMEr, the semantical object dianaEr is typed by the syntactical object PersonEr and the semantical object i_123 is typed by the syntactical object Integer. This figure explains the role the links for AttrMap objects play: The AttrMap object dianaPassportEr with its links to stateEr, dianaEr, passportEr, and i_123 expresses that in the given ER state stateEr the instance dianaEr receives with respect to the attribute passportEr the value i_123. An analogous modeling as for AttrMap is chosen for RelendMap.

Up to now we have considered scenarios where an ER schema is associated with a single state only. But in general, an ER schema may be interpreted by many ER states. Figure 9 shows an object diagram involving one ER schema and two ER states. The Instance object representing Diana as well the AttrMap object assigning the name attribute are shared between the two states. This object diagram also explains why each AttrMap object must be linked to its four connected objects, i.e., an ErState, an Instance, an Attribute, and a Value. One can check: If one of these four links would be missing, one could not express the change of Diana's address properly under the assumption that a single Instance object has to represent Diana in both states. One might wonder whether AttrMap objects can receive their ER state connec-




```
Person(passport:Integer,gender:String)
Marriage(wife_passport:Integer,husband_passport:Integer,date:String)
```

```
 Person | passport | gender
--------+----------+----------
        | 123      | 'female'
        | 456      | 'male'

 Marriage | wife_passport | husband_passport | date
----------+---------------+------------------+--------------
          | 123           | 456              | '1981/07/29'
```

Figure 4: Example Scenario *Diana marries Charles*

tion not in an implict way through the respective Instance object. But it must be expressible that the AttrMap object nameAttrMap is part of two states, and therefore there must be links to the ER states.

Figure 10 shows a variation of the previous scenario. The ER schema is the same as before (except the object identifiers), however the two states are different. As before, the two different states share one Instance object, but both attribute values have changed between the first and the second state, even the key attribute value. Although both states show completely different values, both states share the common Instance object and thus Diana is present in both states, in the first state under the name 'Diana Spencer', in the second state under the name 'Diana Mountbatten-Windsor'.

3.4 RE Syntax

Next we come to the Relational database schema which is pictured in Fig. 11. As in the case of the ER database schema respective objects are shown, in this case, one RelDBSchema object, two RelSchema objects, and five Attribute objects. Recall that a Relational database schema can consist of many Relational schemas. Proper links are established the correct object connections. All object identifiers in the RE part end with Rel in contrast to the objects in the ER part which all ended with Er.

In order to sketch an example for a needed constraint, imagine we would not have name='wife_passport' and name='husband_passport' for wifeRel and husbandRel but we would have name='passport' for both objects. Then one would have two different attributes with the same name in one RE schema. This has to be excluded.

Comparing the ER database schema and RE database schema one finds that both are represented by ten objects and twelve links, however the RE schema has a flat structure with two Relational schemas whereas the ER database schema exhibits its non-flat structure because the relationship depends on the entity.

Analogously to the representation of the ER state as an object diagram, the RE state could be represented. Due to space limitations we refrain from showing the details.

3.5 Common Datamodel Concerns

The package DataMods introduces new invariants but no new classes. One of these invariants will guarantee on the schema side that an attribute either belongs to an ER entity or an ER relship or a RE schema. The other invariants will require on the state side (1) that an attribute map is either linked to an ER instance or an ER link or a RE tuple and (2) that an attribute map is either linked to an ER database state or a RE database state.

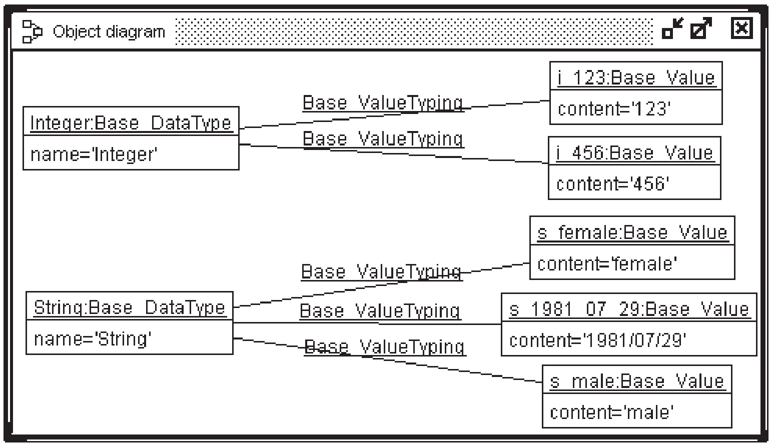


Figure 5: Object Diagram with Data Types and Values

3.6 Transformation

What is now still missing is the transformation object which connects the schemas and the states by proper links. The corresponding screenshot displaying the ErSchema object, the ErState object, the RelDBSchema object, the RelDBState object, and the Trans object can be found in Fig. 12. In this simple scenario with only one state for the ER and RE datamodel the additional link between the states seems unnecessary. However, if more than two states were captured such links are needed to express which ER state corresponds to which RE state.

A constraint violation would occur, if we would drop, for example, the link between PMEr2PMRel and stateRel, because then there would be a RE state connected via the ER state to the transformation object, but not connected directly to the transformation object.

4. MODELING METHOD

4.1 Basic Modeling Method

As indicated with the package diagram in Fig. 13, the method behind our approach is to divide a language into a syntax and semantics part and to describe a transformation between two languages as a direction-neutral affair. All packages will in general include classes, associations, and constraints. We expect that the semantics of the languages will depend on their syntax. The transformation will rely on the syntax and the semantics of the two languages. Transformation properties could be further distinguished into syntax and semantics properties. We regard it as important that formal properties of

- the syntax of the languages to be transformed,
- the semantics of the languages to be transformed and,
- the transformation itself are described in a uniform way within one language.

Only with an explicit formulation of the semantics, a transformation can make requirements about the relationship between the semantics of the first language, the semantics of the second language and the transformation. Apart from a better understanding of the underlying domain, a uniform description opens the possibility of a coherent reasoning mechanism for languages and their transformation.

Above, we have coined the notion that a transformation is a direction-neutral affair. We expect that transformations are described with the same modeling features as the languages, i.e., with classes, association, and constraints. Our transformation between ER and RE was insofar a special case as we needed only a single class in the transformation package. In general, a transformation package may involve more classes and associations on its own. We think that in order to be general, it is a good idea to start with transformation classes and to describe properties of transformations as invariants. The advantage we see in this is that no direction is imposed for the transformation. In a later reification step, the transformations may be turned into appropriate operations.

Our above transformation between ER and RE was also in another respect a special case of the general situation: Our aim for ER and RE was an equivalence between the two database state spaces, i.e., between the two semantic domains, and our constraints exactly required such an equivalence. However, with respect to the transformation properties, the approach is flexible. If we would like to do so, we could, for example, require that the semantics of the second language can be embedded in the semantics of the first language but not necessarily the other way round. The transformation properties merely depend on what is stated in the transformation constraints.

4.2 Tool support

The last important ingredient for our method is tool support. It is necessary to validate the underlying class diagram, i.e., the underlying model, including all constraints. We have done this with our OCL tool USE. Complex models like the one we have presented cannot be developed in a paper-and-pencil fashion. There are too many right or wrong design alternatives which can be taken and there are too many possible dead ends in the development. The USE tool makes it possible to explore informal and formal model and meta-

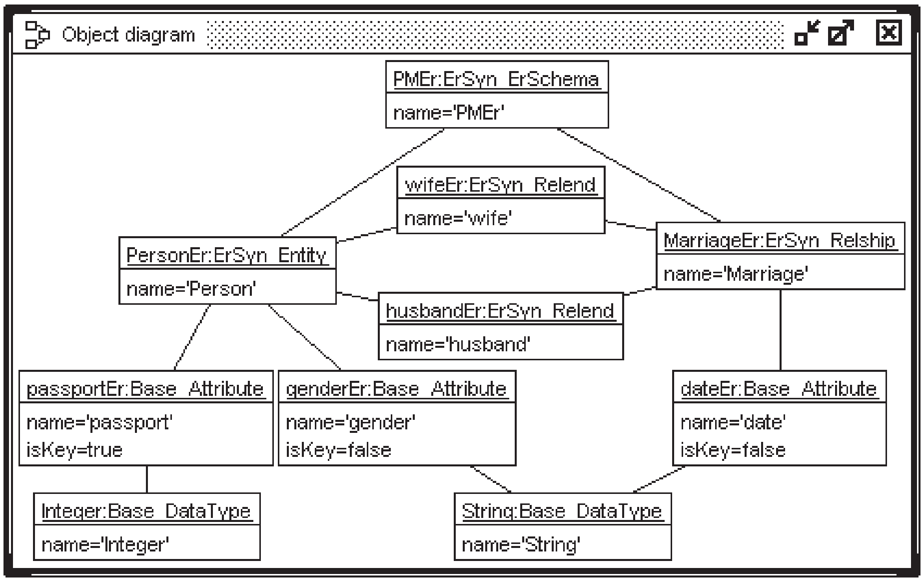


Figure 6: Object Diagram with ER Schema

model properties.

You constantly need feedback about the modeling decisions which you have taken. Our tool (like others) gives support for this in that design decisions can be checked immediately through instantiation by building an object diagram and checking in particular whether the object diagram meets the constraints. The object diagram has not necessarily to be a complete one, it suffices to build an instantiation for the part you are currently working on. Building such object diagrams gives spontaneous feedback about the classes, associations, and constraints and helps in error detection and recovery, i.e., it helps in model debugging. Having tool support in this sense allows you to trace abstract design decisions down to concrete objects which you can handle and check on your system.

5. DISCUSSION

In this section we want to debate typical questions that may show up during discussions about the subject of this paper.

What are the business cases of the approach? A 'business case' for our approach for metamodeling datamodels could be the tuning of the general database development process for the special needs within a particular development process. With our approach it is possible to define a particular data model, i.e., special data structures with special constraints. The enforcement to use exactly this data model could be required in a particular development process, for example, by a particular customer for a particular application. Another 'business case' of our approach is the possibility to formulate schema queries in the same way one formulates queries on the domain level. For example, one could query an ER schema with a query like 'which are the attributes of entities being connected to entity Person through binary relationships'.

What role plays 'semantics' in the approach? The notion of 'semantics' is used in this paper on two levels: Each UML model has a 'semantics' and we use the term 'semantics' also within the UML model which we develop, i.e. in our modeling domain. Let us exaplin this a bit more. We call a UML class diagram together with an OCL constraint set a UML model. First, the semantics of a UML class diagram and an OCL constraint set is the set of object diagrams corresponding to the class diagram and satisfying the OCL constraints. Second, in our application domain, we develop a UML model for two datamodels and divide a datamodel into (A) a syntax part, namely the datamodel schemas, and (B) a semantics part, namely the datamodel states belonging to the datamodel schema. Note that we give a UML model (class diagram and constraints) for both the datamodel syntax and the datamodel semantics. Thus the term semantics appears twice.

How do constraints influence the semantics? The role of 'constraints' is important but note that the semantics of a UML model (the set of corresponding object diagrams) is basically already determined by the class diagram. The role of the constraints is to restrict the set of object diagrams further. Thus it would not be correct to say that 'the constraints determine the semantics of our approach' (and that the class diagrams determine the syntax). The formal constraints are not given in form of OCL expressions in this paper but are explained informally when discussing the object diagrams.

Why is it possible that a transformation, a dynamic thing, can be captured by a class diagram, a static thing? We describe a transformation, for example, the transformation between an ER schema and a Relational schema, by a transformation class and corresponding associations. The aim of this is that a transformation object is linked to the source ER schema and the target Relational schema. In the transfor-

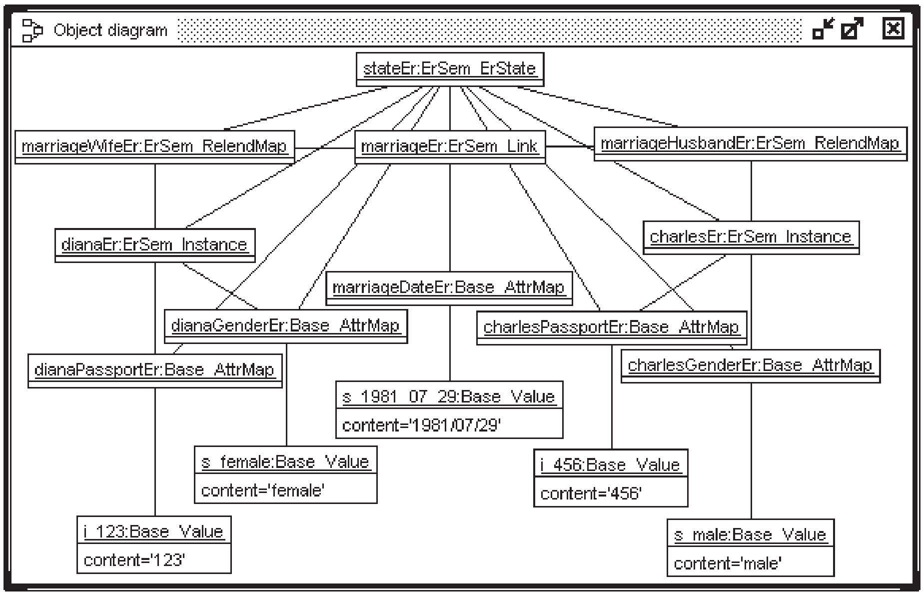


Figure 7: Object Diagram with ER State

mation class all properties of the transformation are stated as invariants. These invariants may however be understood as pre- and postconditions for an operation transforming the source into the target. The advantage of stating the requirements as invariants is that these invariants may be understood also the other way round, namely as conditions for transforming a source Relational schema into a target ER schema: The transformation is stated in a direction-neutral way. Future work will show the details of presenting the invariants equivalently as pre- and postconditions of appropriate operations.

What role plays tool suppport in the approach? The underlying UML class diagram and the OCL constraints for the datamodels and the transformation are quite complex. Our experience shows that this complexity requires tool support in order to understand the consequences of design decisions, for example, the consequences of a particular constraint. Our USE tool allows to check and understand complex constraints by testing with object diagrams.

6. CONCLUSION

In this paper have employed UML (MOF) as a meta-description language for two classical database approaches and their transformation. We have formally characterized the syntax of the datamodels, i.e., the database schemas, and we have covered the semantics of the datamodels, i.e., the database states. We have also formally described the transformation. It is possible to formulate the equivalence of the described database state spaces with formal correctness criteria [25].

The method behind this concrete model transformation example is more general. For a precise transformation between languages one has to know the syntax and semantics of both languages. Only if one knows these four ingredients, one can state criteria about the properties and the correctness of the transformation.

Topics for future research include:

- Formulation of our ER and RE example with operations having pre- and postconditions instead of using only invariants.
- Study of other database models, for example, historical database models like the Hierarchical or Network database model and newer approaches like the Object-Oriented datamodel, Semi-Structured (XML-like) datamodels or Ontology-Based database models a la RDF and OWL.
- Instantiating the method with further examples, e.g., hierarchical and flat statecharts, a classical compiler transforming a small imperative language into an appropriate asembler, or hypertext languages.
- Advanced tool support for metamodeling.

Acknowledgements

Thanks to the referees and the attendees of Metainformatics'2005 for their constructive questions. The Metainformatics'2005 organizers realised a particular familiar atmosphere.

7. REFERENCES

[1] OMG, ed.: OMG Unified Modeling Language Specification, Version 2.0. OMG (2004)
[2] OMG, ed.: OMG Model-Driven Architecture Specification, Version 2.0. OMG (2004)
[3] OMG, ed.: OMG Object Constraint Language Specification, Version 2.0. OMG (2003)

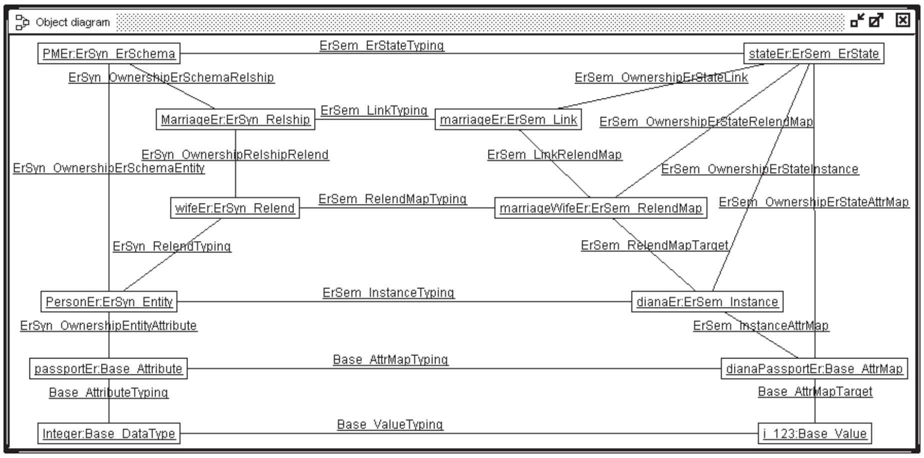


Figure 8: Object Diagram for Interplay between Syntax and Semantics

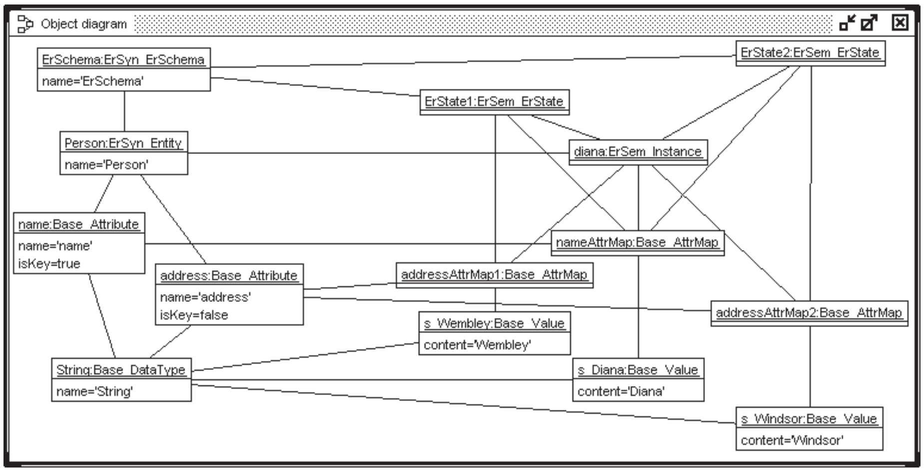


Figure 9: Object Diagram for *Diana moves from Wembley to Windsor*

[4] Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software **20** (2003) 36–41

[5] Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. ACM Trans. Model. Comput. Simul. **12** (2002) 290–321

[6] Akehurst, D., Kent, S., Patrascoiu, O.: A relational approach to defining and implementing transformations between metamodels. Software and System Modeling **2** (2003) 215–239

[7] Clark, T., Evans, A., Kent, S.: The Metamodelling Language Calculus: Foundation Semantics for UML. In Hußmann, H., ed.: FASE. LNCS 2029, Springer (2001) 17–31

[8] Clark, T., Evans, A., Sammut, P., Willans, J.: Transformation Language Design: A Metamodelling Foundation. In Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G., eds.: ICGT. LNCS 3256, Springer (2004) 13–21

[9] Varró, D., Pataricza, A.: VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics). Software and System Modeling **2** (2003) 187–210

[10] Halpin, T.: Comparing Metamodels for ER, ORM and UML Data Models. In Siau, K., ed.: Advanced Topics in Database Research, Vol. 3. Idea Group (2004) 23–44

[11] Tresch, M., Scholl, M.H.: Meta Object Management and its Application to Database Evolution. In Pernul, G., Tjoa, A.M., eds.: 11th International Conference on the Entity-Relationship Approach. LNCS 645, Karlsruhe, Germany, Springer-Verlag (1992) 299–321

[12] Welzer, T., Eder, J.: Meta data model for database design. LNCS 720 (1993) 677–692

[13] Atzeni, P., Torlone, R.: Management of Multiple Models: A Metamodel for Conceptual Models. In Kanellakis, P., Schmidt, J., eds.: DBPL, Morgan Kaufmann (1991) 169–181

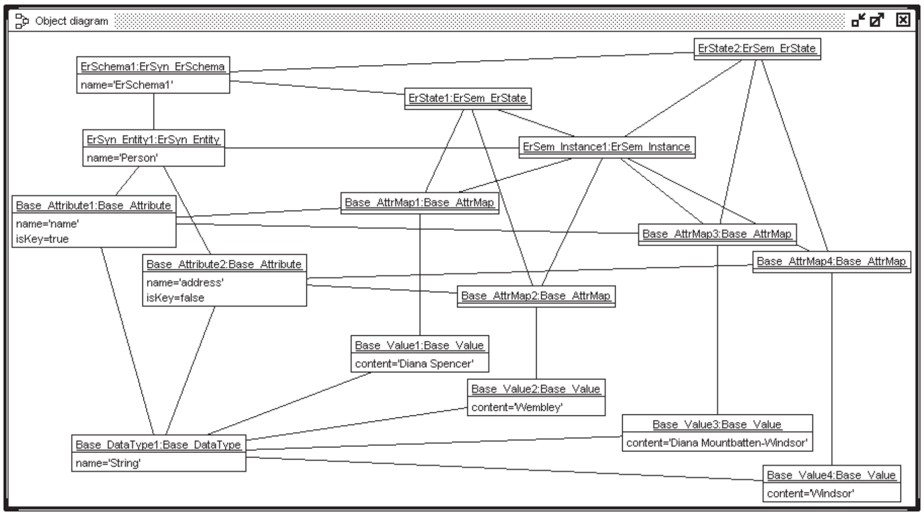


Figure 10: Object Diagram for *Diana moves and marries*

[14] Jodlowski, A., Habela, P., Plodzien, J., Subieta, K.: Extending OO Metamodels towards Dynamic Object Roles. In Meersman, R., Tari, Z., Schmidt, D., eds.: CoopIS/DOA/ODBASE. LNCS 2888, Springer (2003) 1032–1047

[15] Jeusfeld, M., Johnen, U.: An Executable Meta Model for Re-Engineering of Database Schemas. Int. J. Cooperative Inf. Syst. **4** (1995) 237–258

[16] Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"?. IEEE Computer **37** (2004) 64–72

[17] Anderson, K.: Towards Lightweight Structural Computing Techniques with SmallSC. In Wiil, U., Nürnberg, P., Hicks, D., eds.: Preliminary Proc. Metainformatics'2005, University of Southern Denmark (2005)

[18] Lepthien, W., Anderson, K.: A Metainformatical View of Collaborations. In Wiil, U., Nürnberg, P., Hicks, D., eds.: Preliminary Proc. Metainformatics'2005, University of Southern Denmark (2005)

[19] Bezivin, J., Kurtev, I.: Model-Based Technology Integration with the Technical Space Concept. In Wiil, U., Nürnberg, P., Hicks, D., eds.: Preliminary Proc. Metainformatics'2005, University of Southern Denmark (2005)

[20] Gogolla, M.: An Extended Entity-Relationship Model - Fundamentals and Pragmatics. Springer, Berlin, LNCS 767 (1994)

[21] Gogolla, M.: Towards Schema Queries for Semantic Data Models. In Revell, N., Tjoa, A., eds.: Proc. 6th Int. Conf. and Workshop on Database and Expert Systems Applications (DEXA'95), ONMIPRESS, San Mateo (1995) 274–283

[22] Gogolla, M., Lindow, A., Richters, M., Ziemann, P.: Metamodel Transformation of Data Models. In Bezivin, J., France, R., eds.: Proc. UML'2002 Workshop in Software Model Engineering (WiSME 2002), http://www.metamodel.com/wisme-2002 (2002)

[23] Lindow, A., Gogolla, M., Richters, M.: Ein formal validiertes Metamodell für die Transformation von Schemata in Informationssystemen. In Bauknecht, K., Brauer, W., Mück, T., eds.: Proc. GI Jahrestagung (GI'2001), Band 1, Workshop Integrating Diagrammatic and Formal Specification Techniques, Austrian Computer Society, Wien (2001) 662–669

[24] Gogolla, M., Lindow, A.: Transforming Data Models with UML. In Omelayenko, B., Klein, M., eds.: Knowledge Transformation for the Semantic Web. IOS Press, Amsterdam (2003) 18–33

[25] Gogolla, M.: Tales of ER and RE Syntax and Semantics. In Cordy, J., Lämmel, R., Winter, A., eds.: Proc. Dagstuhl Seminar on Transformation Techniques in Software Engineering, Schloss Dagstuhl Int. Conf. and Research Center for Computer Science, http://www.dagstuhl.de/05161/ (2005)

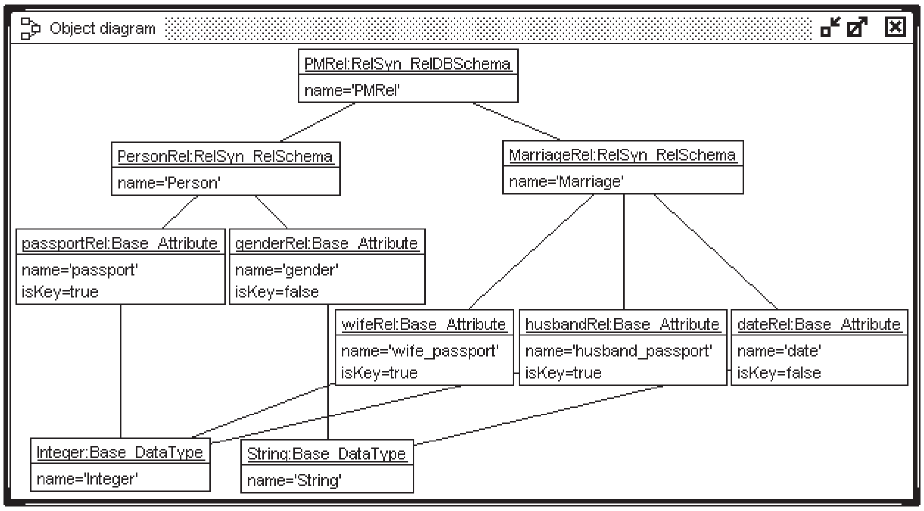


Figure 11: Object Diagram with RE Schema

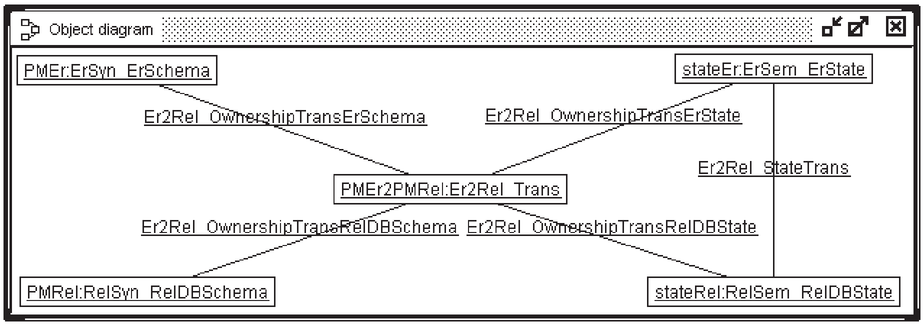


Figure 12: Object Diagram with Transformation

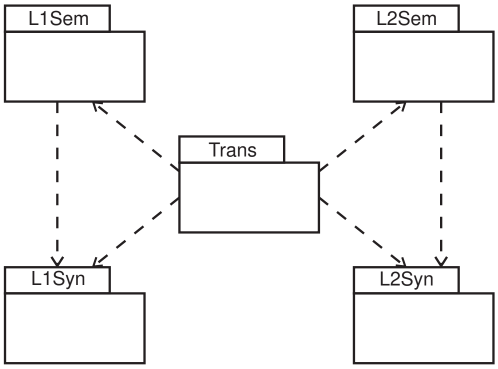


Figure 13: Package Structure of Underlying Modeling Method