(An Example for)
Metamodeling Syntax and Semantics of Two Languages, their Transformation, and a Correctness Criterion

Martin Gogolla

University of Bremen, Computer Science Department
Database Systems Group, D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

Abstract. We study a metamodel for the Entity Relationship (ER) and the Relational data model. We do this by describing the syntax of the ER data model by introducing classes for ER schemata, entities, and relationships. We also describe the semantics of the ER data model by introducing classes for ER states, instances, and links. The connection between syntax and semantics is established by associations explaining that syntactical objects are interpreted by corresponding semantical objects. Analogously we do this for the Relational data model. Finally, we give a metamodel for the transformation of ER schemata into Relational database schemata. By characterizing the syntax and semantics of the languages to be transformed and also the transformation itself within the same (meta-)modeling language we are able to include equivalence criteria on the syntactical and on the semantical level for the transformation. In particular, we show that the semantical equivalence criterion requires that the ER states and the corresponding Relational states bear the same information.

1 Context

This paper is based on a comprehensive case study employing our tool USE allowing to check and to reason about properties of UML class diagrams and OCL constraints. The complete model currently covers 17 classes, 34 associations, and 58 constraints. This work uses ideas from earlier material on metamodeling an Extended Entity-Relationship approach [Gog94,Gog95], preliminary versions [GLRZ02] of the metamodel introduced here, and work concentrating only on syntactical aspects [LGR01,GL03].

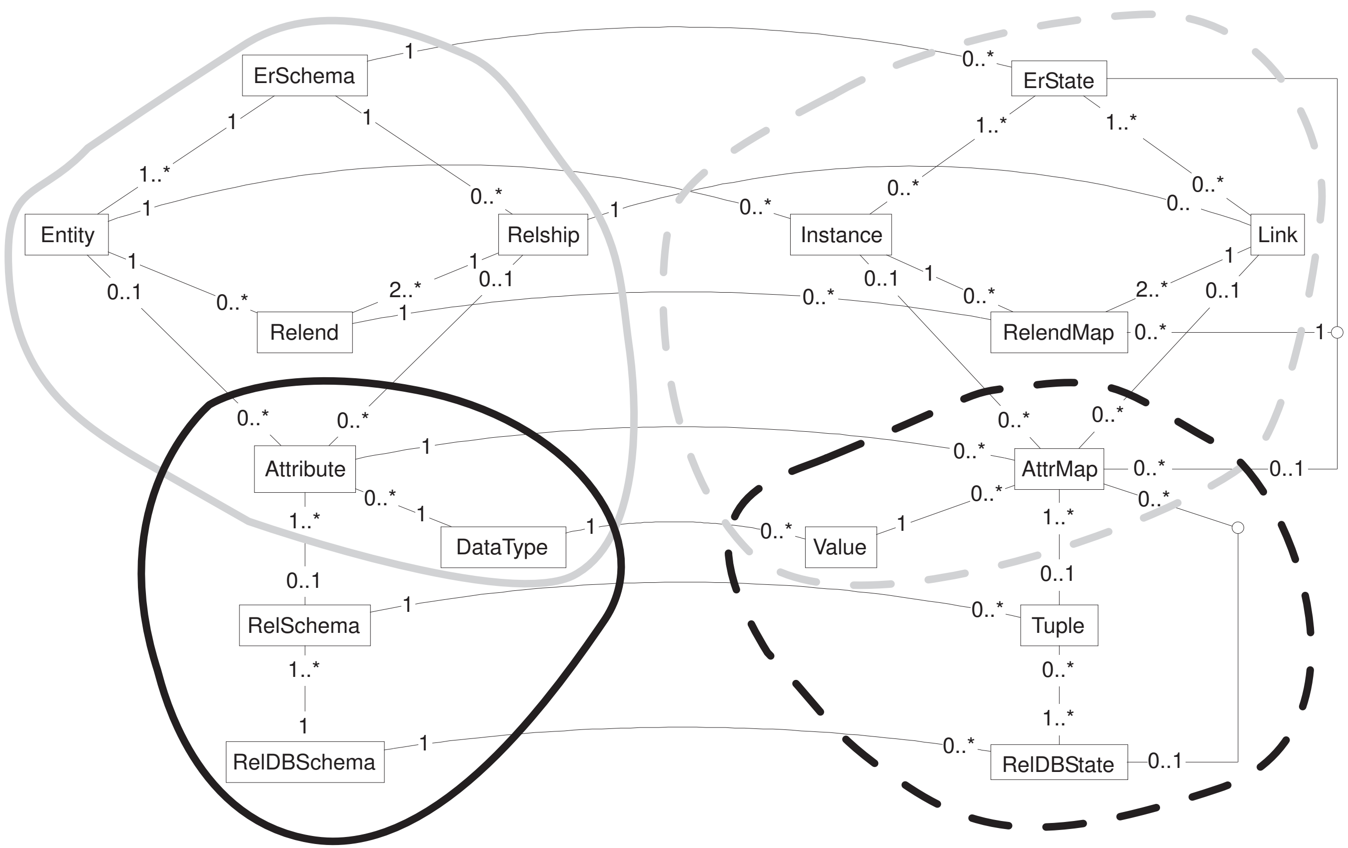

Fig. 1. Class Diagram Metamodeling the ER and Relational Data Model

2 Metamodeling Data Models

Consider the class diagram in Fig. 1. It shows 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 data models, the two dashed right clouds the semantics; the upper clouds describe the ER data model, the lower clouds the Relational data model. The ER and the Relational data model share some concepts, namely the parts in the middle talking about data types, attributes and their semantics.

Syntax of the ER data model: This part introduces the clases 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.

Semantics of the ER data model: 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 object, respectively.
In order to give more visual clues for differentiating, the associations from the syntax into the semantics are displayed by bent lines whereas the associations within the syntax and the associations within the semantics are displayed by straight lines.

Syntax of the Relational data model: Here the classes RelDBSchema, RelSchema, Attribute, and DataType are needed. RelDBSchema objects consist of RelSchema objects which possess Attribute objects typed through DataType objects.

Semantics of the Relational data model: 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 the Tuple.

Let us shortly mention the attributes and operations relevant for the class diagram but being not displayed. All classes in the (left) syntax part possess an attribute name of data type String. The class Attribute has an aditional boolean-valued attribute isKey indicating whether this attribute contributes to the key of the Entity or the RelSchema. The class Value owns the attribute content of data type String indicating the actual content of the Value object.

Concerning operations, the classes Instance, Link, and Tuple have an operation applyAttr with a State and an Attribute parameter returning the actual Value object of the Attribute. The class Link has an operation applyRelend with an ErState and a Relend parameter returning the actual Instance of the Relend. The classes Entity and RelSchema possess an operation key returning the set of its key attributes.

Apart from the shown multiplicities in the class diagram, all parts must be restricted by appropriate constraints. In the total we obtain about 50 constraints. But we show only one typical example from each of the four parts.

Syntax of the ER data model: Within one Entity, different Attributes have different names.

```
context self:Entity inv uniqueAttributeNamesWithinEntity:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)
```

Semantics of the ER data model: Two different Instances of one Entity can be distinguished in every ErState (where both Instances occur) by a key Attribute of the Entity.

```
context self:Instance inv keyMapUnique:
  Instance.allInstances->forAll(self2 |
    self<>self2 and self.entity=self2.entity
    implies
    self.erState->intersection(self2.erState)->forAll(s |
      self.entity.key()->exists(ka |
        self.applyAttr(s,ka)<>self2.applyAttr(s,ka))))
```

Syntax of the Relational data model: The set of key Attributes of a RelSchema is not empty.

```
context self:RelSchema inv relSchemaKeyNotEmpty:
  self.key()->notEmpty
```

Semantics of the Relational data model: The Attributes connected to the RelSchema of a Tuple are identical to the Attributes connected to the AttrMap of the Tuple. In other words, there are Attribute assignments for all Attributes of a Tuple.

```
context self:Tuple inv commutativityAttribute:
  self.relSchema.attribute=self.attrMap.attribute->asSet
```

The modeling is probably best explained by an example. Figure 2 shows an example scenario which is represented in Fig. 3 as an ER schema, in Fig. 4 as an ER state, in Fig. 5 as a Relational schema, and in Fig. 6 as a Relational state. These object diagrams are however not complete with respect to the links between syntax and semantics. For example, every object in Fig. 4 is typed by a link (belonging to one of the bent associations in Fig. 1) to an object in Fig. 3. Thus, e.g., the five Value objects in the bottom of Fig. 4 possess typing links to the two DataType objects in the bottom of Fig. 3. In order to make the presentation comprehensible we have ommitted these details.

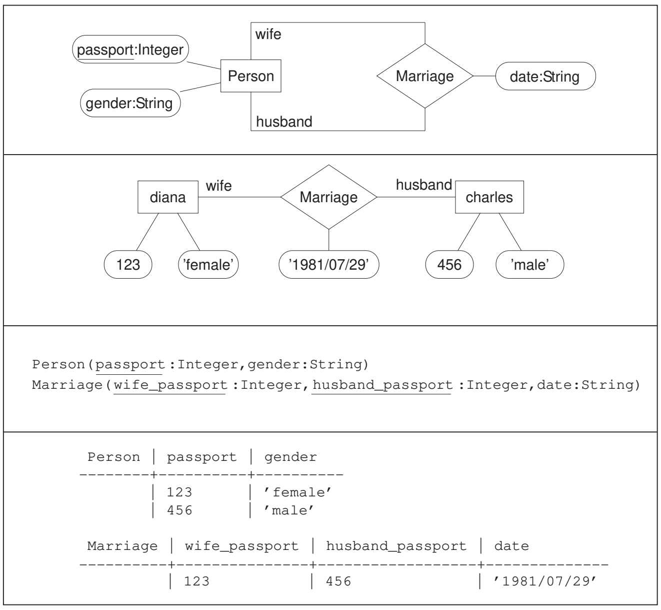

Fig. 2. Content of Example Scenario

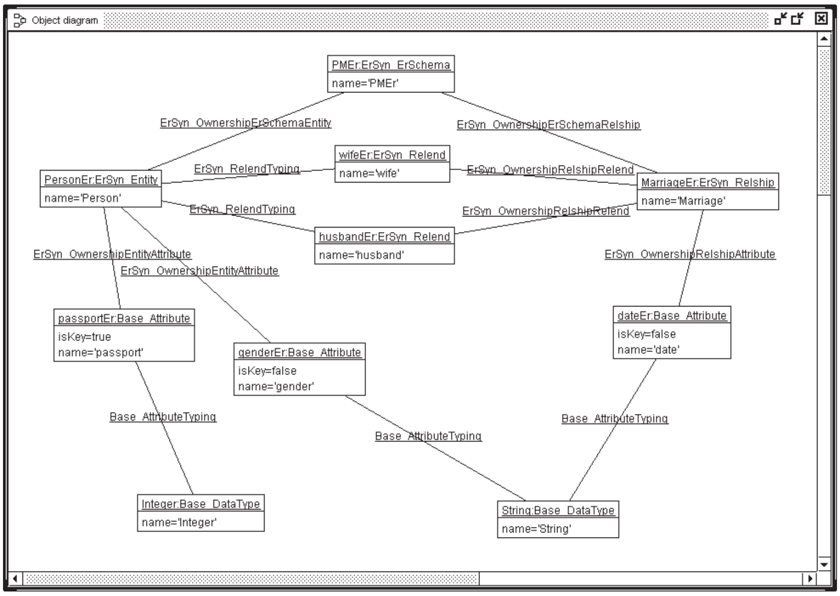

Fig. 3. Example Scenario as an ER Schema

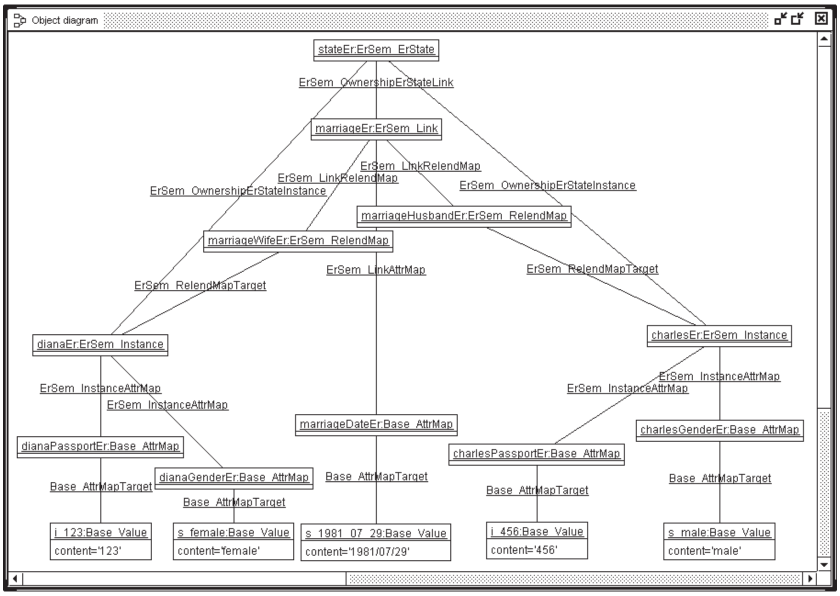

Fig. 4. Example Scenario as an ER State

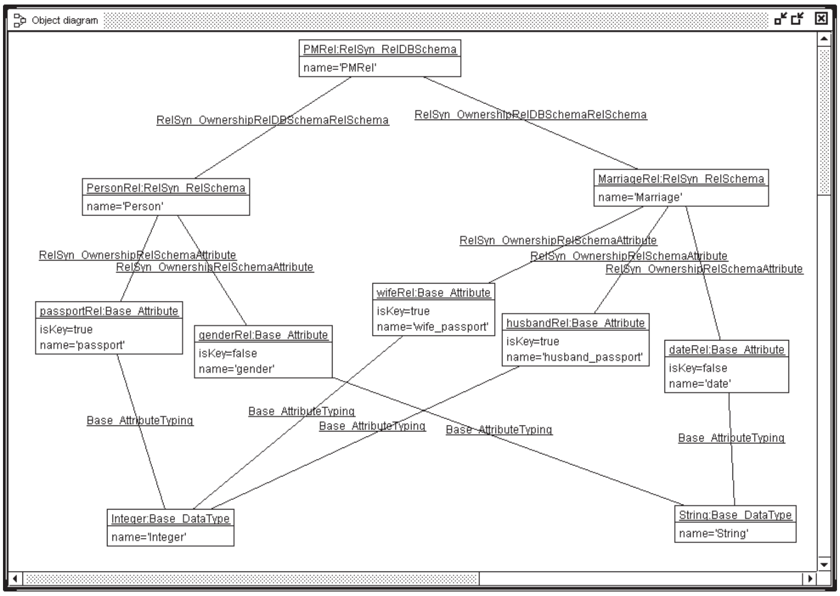

Fig. 5. Example Scenario as a Relational Schema

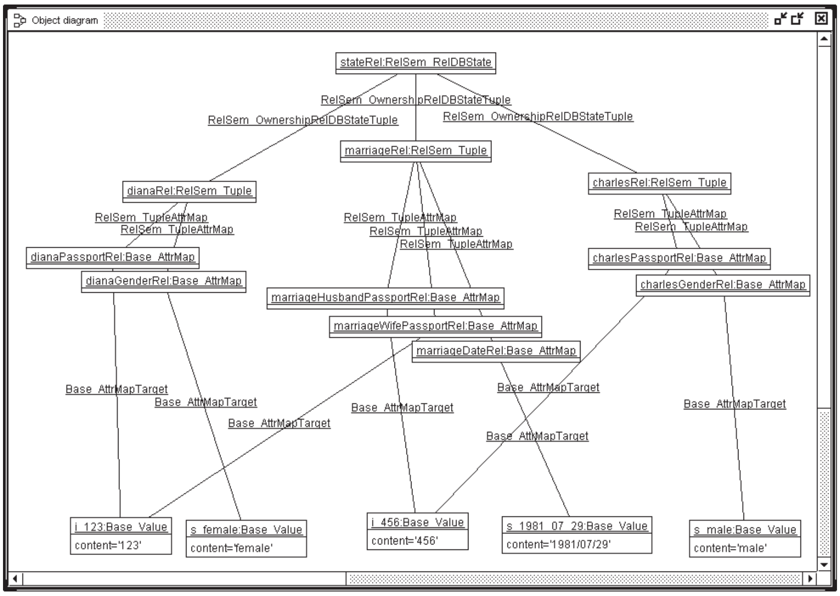

Fig. 6. Example Scenario as a Relational State

3 Metamodeling Transformations

Let us now turn to the transformation between the languages. We will model the transformation by a special class Trans which has associations to the respective syntax and semantics classes already introduced and which has constraints attached to it which guarantee desired properties.

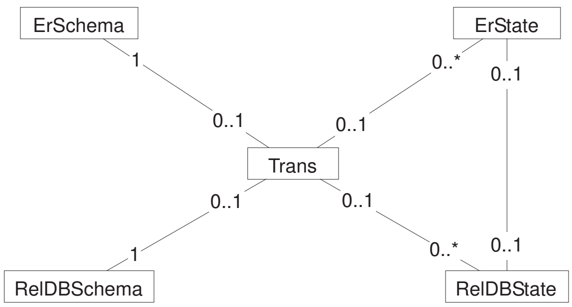

Fig. 7. Transformation Class and Associations

The class diagram in Fig. 7 shows the class Trans needed for the transformation and the respective associations. A Transformation object is associated with exactly one ER schema and exactly one Relational database schema. It additionally can be connected to ER and Relational states which in turn can be connected through an association expressing that both states are equivalent.

As for the syntax and semantics of the data models the Transformation class diagram must be restricted by another 10 constraints in order to allow only meaningful situations. We here discuss only two example constraints, one dealing with schema aspects and the other one dealing with state aspects.

3.1 Schema aspect

For every Entity in the ErSchema there is a RelSchema having the same name and Attributes with the same properties, i.e. name, DataType, and key property.

```
context self:Trans inv relSchemaForEntity:
  self.erSchema.entity->forAll(e |
    self.relDBSchema.relSchema->select(rl | -- existsOne
      e.name=rl.name and
      e.attribute->forAll(ea |
        rl.attribute->select(ra | -- existsOne
          ea.name=ra.name and ea.dataType=ra.dataType and
          ea.isKey=ra.isKey)->size=1))->size=1)
```

3.2 State aspect

For every Tuple in the RelDBState (1) there is either exactly one Instance such that for every AttrMap of the Tuple there is exactly one AttrMap in the Instance holding the same information or (2) there is exactly one link such that for every AttrMap of Tuple the following holds: (A) if the AttrMap belongs not to a key Attribute, there is exactly one AttrMap in the Link holding the same information, and (B) if the AttrMap belongs to a key Attribute, there is exactly one RelendMap in the Link and exactly one AttrMap of the RelendMap such that the AttrMap from the Tuple and the AttrMap from the Link hold the same information.

```
context self:Er2Rel_Trans inv tupleCorrespondsToInstanceXorLink:
  self.relDBState.tuple->forAll(t |
    self.erState.instance->select(i | -- existsOne
      t.attrMap->forAll(amRel |
        i.attrMap->select(amEr | -- existsOne
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value)->size=1))->size=1
    xor
```

```
self.erState.link->select(l | -- existsOne
  t.attrMap->forAll(amRel |
    ( not(amRel.attribute.isKey) implies
      l.attrMap->select(amEr | -- existsOne
        amEr.attribute.name=amRel.attribute.name and
        amEr.value=amRel.value)->size=1 )
    and
    ( amRel.attribute.isKey implies
      l.relendMap->select(rm | -- existsOne
        rm.instance.attrMap->select(amEr | amEr.attribute.isKey)->
        select(amEr | -- existsOne
          amRel.attribute.name =
            rm.relend.name.concat('_').concat(amEr.attribute.name)
          and
          amRel.value=amEr.value)->size=1)->size=1)))->size=1)
```

For a notion of equivalence between an ER state and a Relational state one has to consider two directions of the translation, namely first the direction from ER to Relational and second the direction from Relational to ER. The above constraint touches the second direction from Relational to ER (for each tuple there is an instance or a link). The first direction is also covered by the constraints but not shown here (for each instance and link there is a tuple). Both directions together establish a correctness criterion, namely the equivalence between the ER state space and the Relational state space. As far as we know, such a correctness criterion, i.e., the equivalence between the respective state spaces, is always the aim of translating data models but such a criterion has not been stated explicitly and completely formally in the literature up to now. We emphasize that our approach allows to discuss these questions only because we have rigorously described the data models and their transformation within a single framework, namely with the language of UML class diagrams and OCL constraints. However, the approach is not bound to UML and OCL.

4 Relationship to Language Engineering

We think that the underlying principal ideas lying behind the concrete translation and the concrete languages which we consider, i.e., the ER and Relational data models and their translation, can be used in other domains like classical compilers or modern approaches like model-driven development as well. We classify the central ingredients of our approach as follows:

(A) Formally describe the syntax and the semantics of your language (or perhaps more generally of your domain). Explicitly distinguish between syntactical and semantical aspects and explicitly establish the connection between syntax and semantics.

(B) If you need a second language (or domain) also describe this formally as you have done with the first language (or domain). The second language may of course be derived from your first language (or domain) by specialisation, e.g., by adding more constraints.

(C) Explicitly describe the translation by introducing concepts for the translation and by restricting it through appropriate constraints. Such restrictions can describe correctness criteria for the translation. Insist on using the same description mechanism for all three ingredients, i.e., the source language, the target language, and the translation. Using one single description mechanism enables you to check and to reason about your 'language engineering' system homogeneously.

(D) Test, test, test.

References

[GL03] Martin Gogolla and Arne Lindow. Transforming Data Models with UML. In Borys Omelayenko and Michel Klein, editors, *Knowledge Transformation for the Semantic Web*, pages 18–33. IOS Press, Amsterdam, 2003.

[GLRZ02] Martin Gogolla, Arne Lindow, Mark Richters, and Paul Ziemann. Metamodel Transformation of Data Models. In Jean Bezivin and Robert France, editors, *Proc. UML'2002 Workshop in Software Model Engineering (WiSME 2002).* Technical Report, University of Nantes, http://www.metamodel.com/wisme-2002, 2002.

[Gog94] Martin Gogolla. *An Extended Entity-Relationship Model - Fundamentals and Pragmatics.* Springer, Berlin, LNCS 767, 1994.

[Gog95] Martin Gogolla. Towards Schema Queries for Semantic Data Models. In Norman Revell and A Min Tjoa, editors, *Proc. 6th Int. Conf. and Workshop on Database and Expert Systems Applications (DEXA'95)*, pages 274–283. ONMIPRESS, San Mateo, 1995.

[LGR01] Arne Lindow, Martin Gogolla, and Mark Richters. Ein formal validiertes Metamodell für die Transformation von Schemata in Informationssystemen. In K. Bauknecht, W. Brauer, and T. Mück, editors, *Proc. GI Jahrestagung (GI'2001), Band 1, Workshop Integrating Diagrammatic and Formal Specification Techniques*, pages 662–669. Austrian Computer Society, Wien, 2001.