Metamodel Transformation of Data Models
Position Paper

Martin Gogolla, Arne Lindow, Mark Richters, Paul Ziemann
University of Bremen, Computer Science Department
Database Systems Group, D-28334 Bremen, Germany

Abstract

This paper studies syntax and semantics of the Entity-Relationship (ER) and Relational data model and their transformation. The ER model may be regarded as a platform independent model and the Relational model as a prototypical platform specific model. The paper studies the transformation between these models and proposes to express that transformation again as a model.

Keywords: Platform independent model (PIM); platform specific model (PSM); Entity-Relationship model; Relational model; model transformation; mapping between different technological spaces; validation of model transformation.

1 Introduction

Today, metamodeling [OMG99] has become an important technique in software development. For example, the UML [OMG01] is described in a semi-formal way by a metamodel. Another nice use of metamodeling has been made in connection with information systems and databases within the Common Warehouse Metamodel [OMG00]. This paper also concentrates on aspects of information systems. We study two classical data models, namely the ER and Relational data model, and show how one can completely formalize syntax and semantics and the transformation between the two data models. Central parts of our transformation model have been validated with our UML and OCL tool USE. We are not aware of an approach where both syntax and semantics of data models and their transformation have been studied within a single framework. The MOF provides such a framework.

The structure of the rest of this paper is as follows. Section 2 gives an overview on the resulting architecture. Section 3 explains our running example. Section 4 and 5 show how syntax and semantics of the ER and Relational data model are represented. In Sect. 6 aspects of the transformation are mentioned. The paper closes with a short discussion in Sect. 7.

2 Overview

We want to define syntax and semantics of the ER and the Relational data model and their transformation. As pictured in Fig. 1, we do so by introducing a class ErSchema representing the ER syntax and a class ErState representing the semantics of an ER schema, i.e. we associate the possible database states with the database schema.

Analogously, we do this for the Relational data model: A class RelDBSchema represents the syntax of a Relational database schema and a class RelDBState represents the semantics of a Relational database schema, i.e. we again associate the possible database states with the database schema.

The transformation aspect is reflected by two associations, namely ErSchema2RelDBSchema and ErState2RelDBState. An ER schema is linked to the Relational database schema which represents the translated schema. Each ER state is associated with the Relational database state representing the same information. Note that the syntax on the left side and the semantics on the right side are connected by associations pictured with bent lines. We will follow this layout convention throughout the paper.

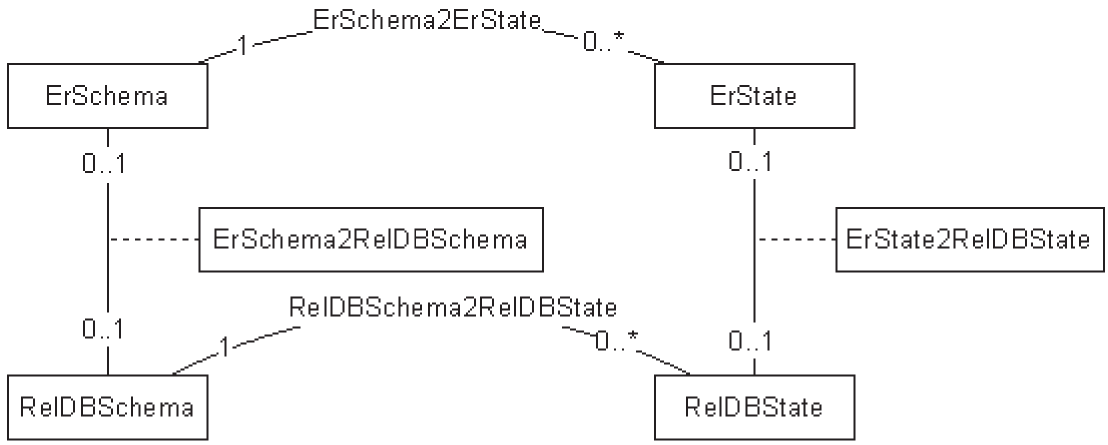
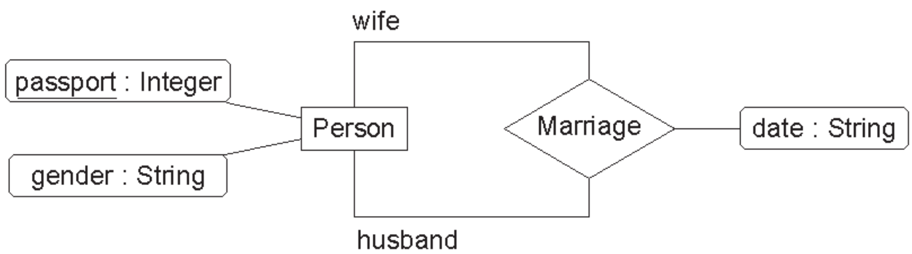


Figure 1: Metamodel for Transformation

3 Example Schemata

Throughout the paper we use the simple ER schema and the equivalent Relational database schema given in Fig. 2.




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

Figure 2: Example Schemata

4 Syntax and Semantics of ER Model

In Fig. 3 we show the metamodel for the ER data model. The syntax is shown in the left part and the semantics is given in the right part. As in Fig. 1 giving the transformation metamodel, syntax and semantics are connected by associations pictured with bent lines.

The syntax part expresses that an ErSchema consists of at least one Entity instance and possibly zero or more Relship instances. A Relship instance possesses two or more Role instances which in turn are typed through an association to the class Entity. Entity and

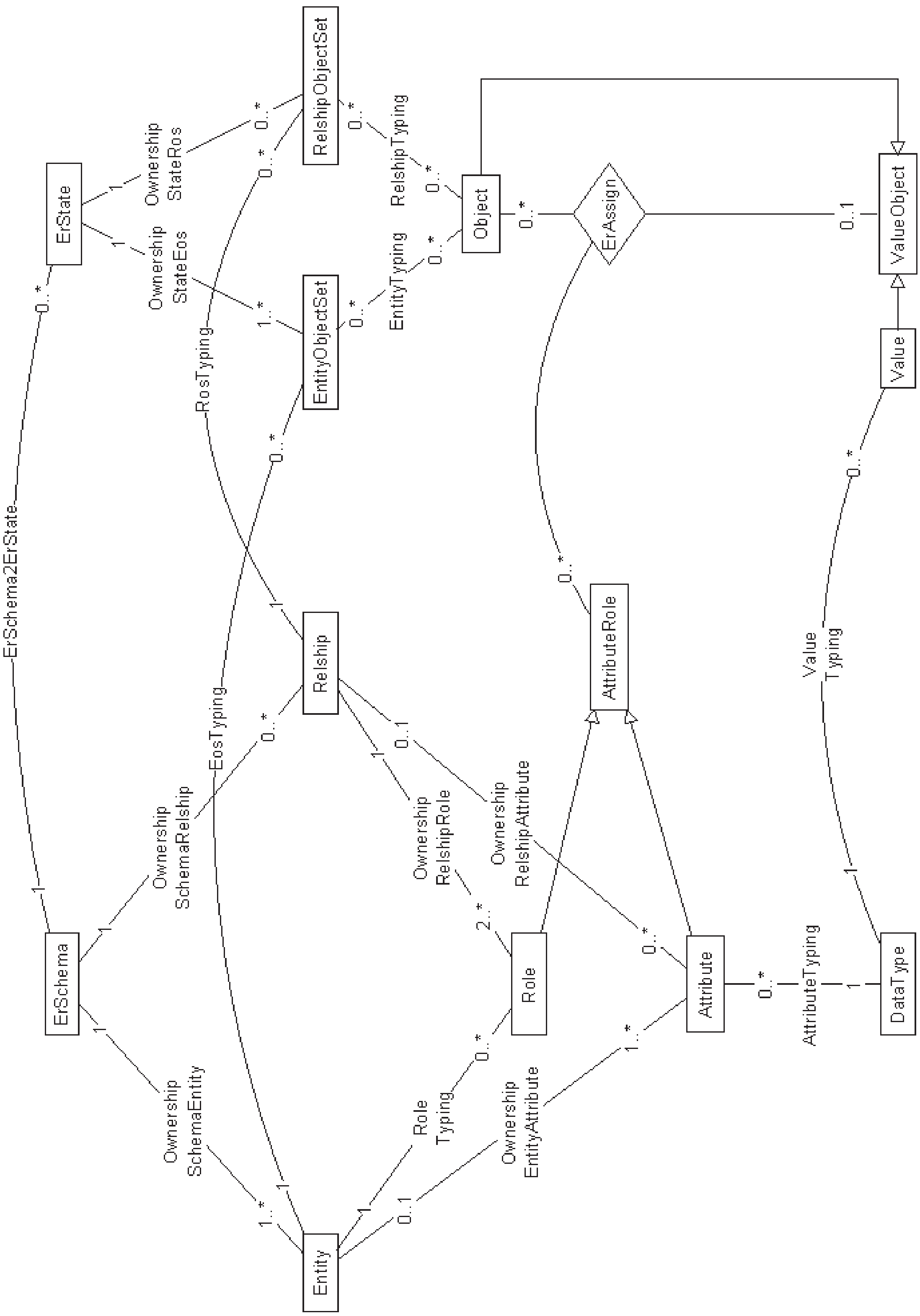


Figure 3: Metamodel for ER Model

Relship instances may possess Attribute instances which are typed through an association to the class DataType.

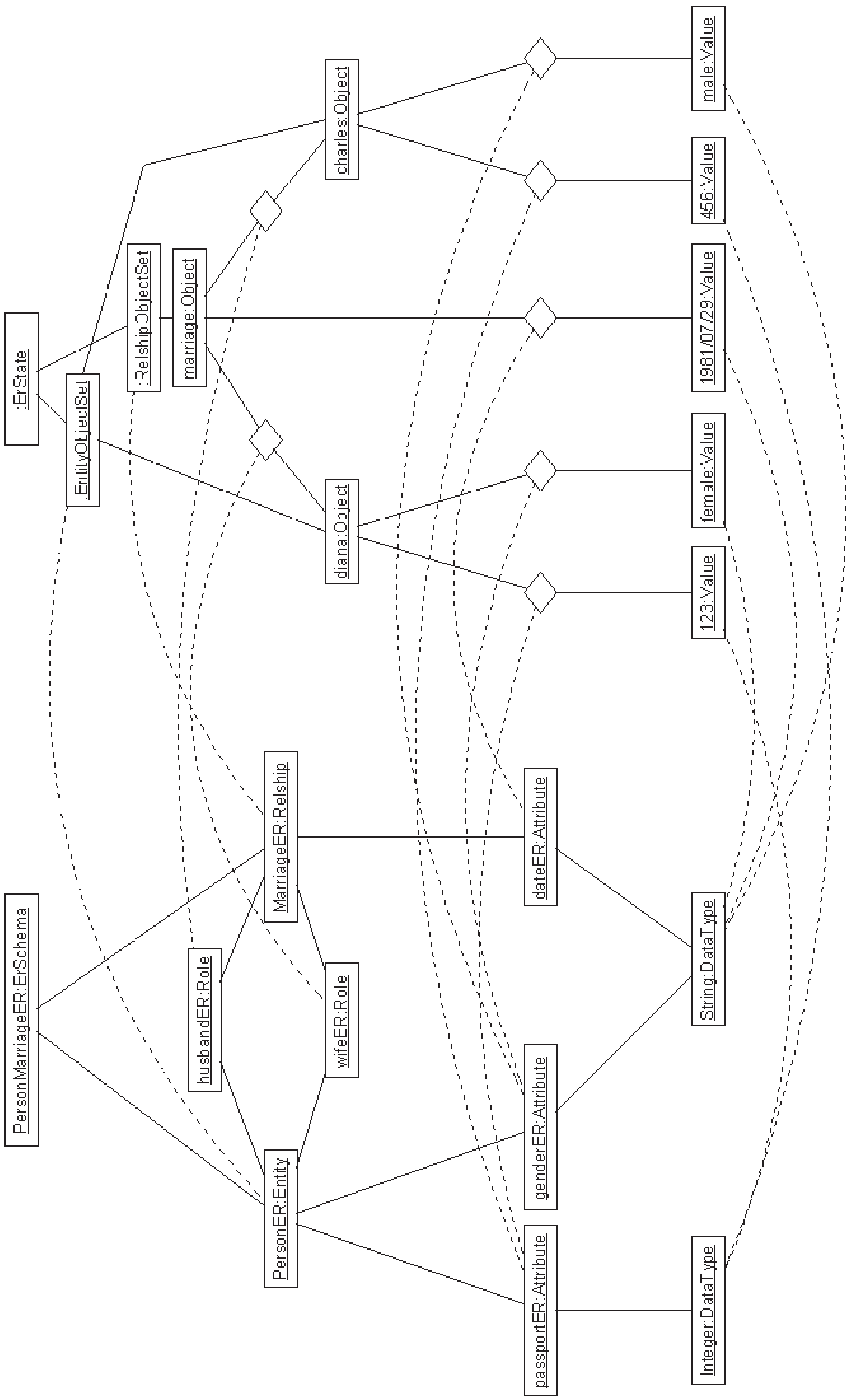


Figure 4: Sample Schema and State for ER Model

The semantics part explains that an ErState consists of EntityObjectSet and RelshipObjectSet instances. An EntityObjectSet instance represents a set of objects where each object stands for an Entity instantiation instantiation, e.g., one could have an Entity instantiation Person and a Person instantiation diana. A RelshipObjectSet instance represents a set of objects where each object stands for a Relship instantiation instantiation. EntityObjectSet and RelshipObjectSet are typed by associations into the syntax part. The corresponding links via the associations EosTyping and RosTyping connect an EntityObjectSet resp. RelshipObjectSet instance to the Entity resp. Relship instance of the ErSchema. Both, EntityObjectSet and RelshipObjectSet, consist of Object instances. First, an Object instance may represent an Entity instantiation instantiation; in this case, the Object instantiation belongs to an EntityObjectSet via the association EntityTyping. Second, an Object instance may represent a Relship instantiation instantiation; in that case, the Object instantiation belongs to a RelshipObjectSet via the association RelshipTyping. Given an Attribute or Role instance in the schema, Objects are assigned ValueObject instances representing the assignment of the attribute or the role in the given ER state. ValueObject is a generalization of Value and Object. These ideas are illustrated by the example in Fig. 4.

Figure 4 shows in form of an object diagram how the example ER schema is represented. The figure also gives an example state with two Person entity instantiations and one Marriage relationship instantiation. We have used dashed lines for connections between the syntax and the semantics parts in order to better identify the connection visually.

Note that in the ER part of the metamodel (and also in the following Relational part) many constraints must be stated in order to guarantee proper instantiation. For demonstration purposes we only mention one constraint for the ER part: An object representing an entity must have assignments for all its attributes.

```
context Object inv assignmentsForAllAttributes:
  self.entityObjectSet->notEmpty implies
    self.entityObjectSet.entity.attribute->forAll(a|
      self.attributeRole->includes(a))
```

5 Syntax and Semantics of Relational Model

The syntax part for the Relational data model in the left of Fig. 5 expresses that a Relational database schema, a RelDBSchema instance, consists of at least one Relational schema, a RelSchema instance, which in turn may possess one or more Attribute instances. As in the ER part, Attribute instances are typed through an association to the class DataType.

The semantics of the Relational data model given in the right of Fig. 5 connects Relational database states, RelDBState instances, to one or more Relation instances which in turn consist of zero or more Tuple instances. Tuple instances can be connected to Value instances. Such links are typed by a connection to an Attribute instance of the schema part. Note that the associations between the syntax and the semantics part are shown again by bent lines.

Figure 6 shows how the example Relational database schema is represented. The figure also gives an example state with two relations, one relation with two tuples, the other relation with one tuple.

6 Transformation from ER to Relational Model

The transformation invariants are the most challenging expressions in the metamodel. The transformation of ER schemata into Relational schemata is represented by the class ErSchema2RelDBSchema. One ErSchema2RelDBSchema instance is linked to one ERSchema in-

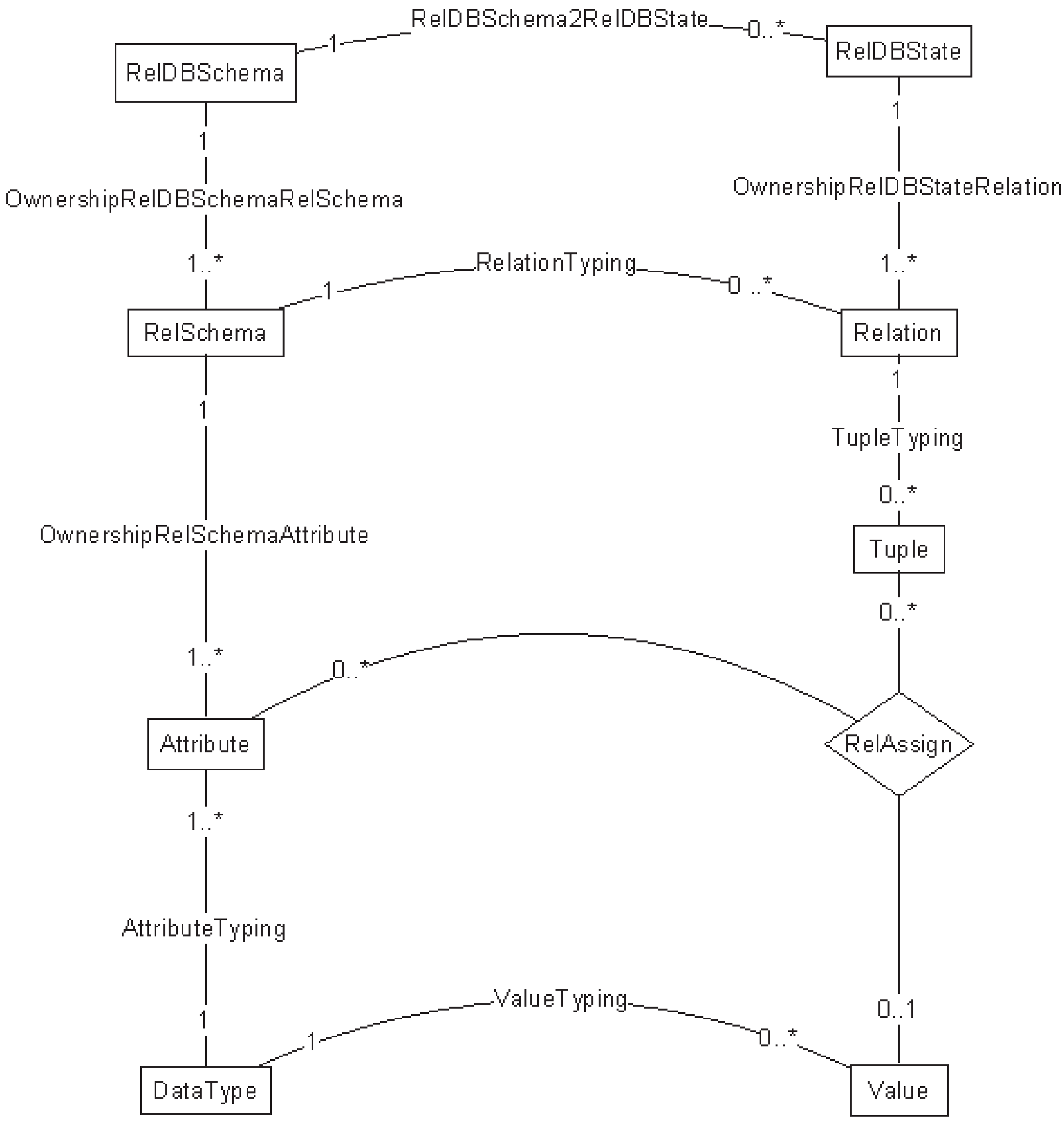


Figure 5: Metamodel for Relational Model

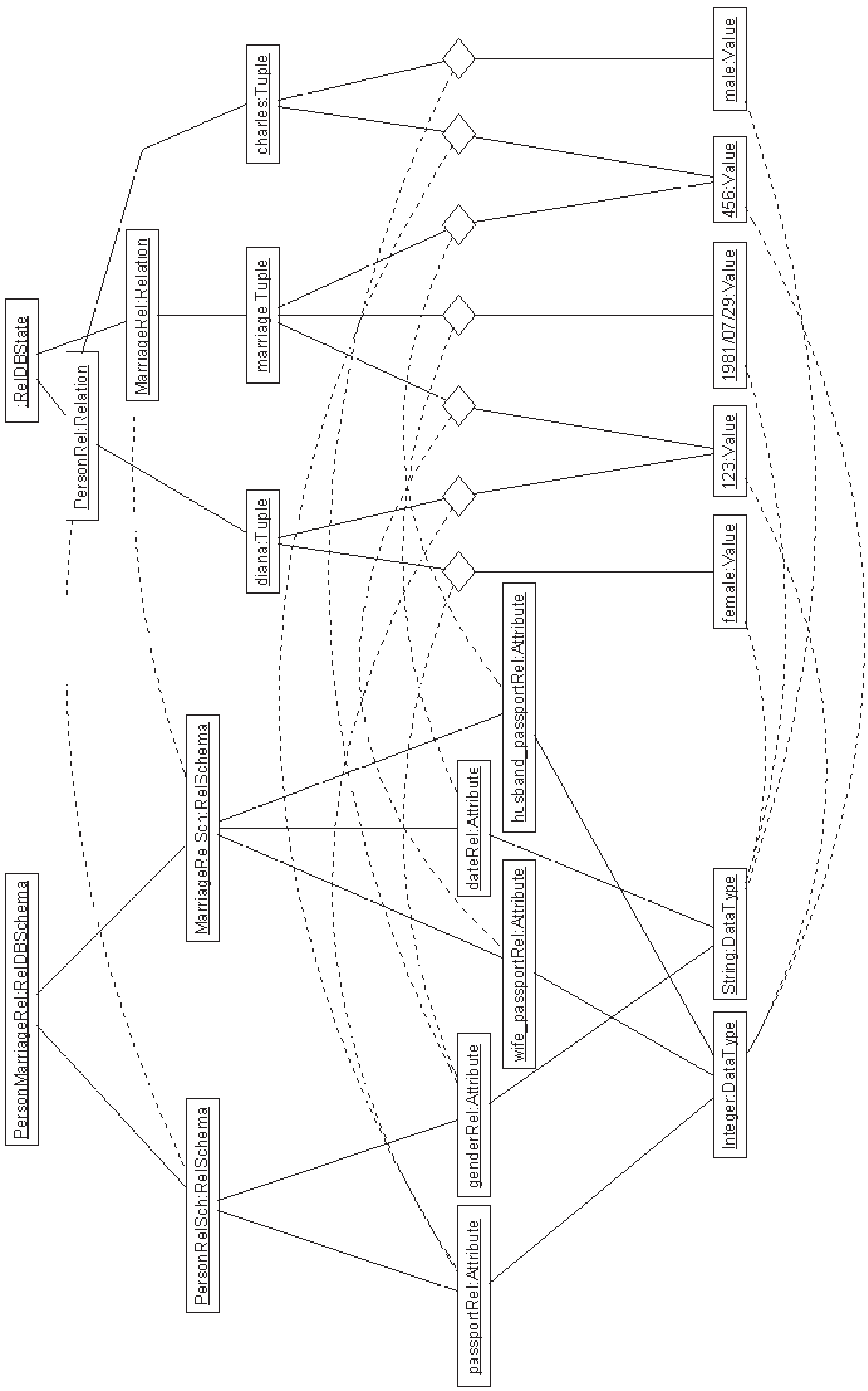


Figure 6: Sample Schema and State for Relational Model

stance and one RelSchema instance. The purpose of the ErSchema2RelDBSchema invariants is to guarantee that the RelDBSchema instance is the result of transforming the ERSchema instance into the Relational data model. The transformation is restricted by various invariants: `relationExistsForEntity`, `relationExistsForRelship`, and `entityXorRelshipExistsForRelation`. As the names indicate, the first two are responsible for the task that for each entity and each relationship in the ER schema there exists a corresponding Relational schema in the Relational database schema, and the last invariant assures that for each relational schema there is an entity or a relationship. We only show the invariant `relationExistsForEntity` in detail as an OCL constraint:

```
context ErSchema2RelDBSchema inv relationExistsForEntity:
  self.erSchema.entity->forAll(e|
    self.relDBSchema.relSchema->exists(rl|
(A)   e.name=rl.name and
(B)   e.attribute->forAll(ea| rl.attribute->exists(ra|
        ea.name=ra.name and ea.dataType=ra.dataType and
        ea.isKey=ra.isKey))))
```

Up to now we have not discussed attributes of the metamodel. All classes posses a String-valued attribute name and the class Attribute has a Boolean-valued attribute isKey indicating whether that attribute contributes to the key or not.

The invariant `relationExistsForEntity` requires that for each entity `e` in the considered ER schema there is a Relational schema `rl` in the Relational database schema such that (A) the entity `e` and the Relational schema `rl` have the same name and (B) for each attribute `ea` of the entity there is an attribute `ra` in the Relational schema having the same name, the same data type, and the same key property.

7 Conclusion

We have sketched how syntax and semantics of the ER and Relational model and their transformation can be understood as platform independent and platform specific models. We think that the description of the data models and their transformation is a good example how ideas from Model Driven Architecture can be employed to formulate and formally analyse concepts which are used in everyday practical work. The formal representation of syntax and semantics within a metamodel offers possibilities for further semantical checks such as formal statements about information equivalence on the different data model levels.

References

[OMG99] OMG, editor. *Meta Object Facility (1.3).* OMG, 1999. www.omg.org.

[OMG00] OMG, editor. *The Common Warehouse Metamodel Specification.* OMG, 2000. www.omg.org.

[OMG01] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.4.* OMG, September 2001. OMG Document formal/01-09-68 through formal/01-09-80, www.omg.org.