Checking Transformation Model Properties with a UML and OCL Model Validator

Martin Gogolla, Lars Hamann, Frank Hilken

Database Systems Group, University of Bremen, Germany
{gogolla|lhamann|fhilken}@informatik.uni-bremen.de

Abstract. This paper studies model transformations in the form of transformation models connecting source and target metamodels. We propose to analyze transformation models with a UML and OCL tool on the basis of an implementation of relational logic on top of Kodkod. Within this approach it is feasible to prove transformation model consistency, i.e., to automatically construct a valid metamodel model instance. Certain properties implied by the transformation model, e.g. whether a particular property is preserved by the transformation, can be inspected as well. As an example, the paper uses the well-known transformation between ER schemata and relational database schemata.

1 Introduction

Model transformations are a central ingredient in Model-Driven Engineering (MDE). As for any kind of software artifact, quality improvement techniques like validation and verification of properties are essential for the success of MDE. Verification for model transformation [1] is thus gaining more attention.

This paper discusses model transformations in form of transformation models [2] that are descriptive and direction-neutral characterizations of mappings between a source and target metamodel. We propose to check model transformation consistency and implied properties by applying a so-called model validator that searches for model instances within a finite search space of possible instances. We employ the tool USE that allows to automatically construct system states (object diagrams) for UML and OCL models. With USE models can be validated against informal expectations. And, for example, model consistency or model consequences can be verified by automatically building system states.

Our work has links to related approaches. Different notions of consistency and instanciability as considered here have also been proposed in [4]. Our work is based on Alloy [7] and Kodkod [9]. The implementation of the model validator that we employ is grounded on a translation of UML and OCL concepts into relational logic as described in [8]. Application of transformation models using the same example as employed here, however with different underlying metamodels and focusing on transformation refinement, has been studied in [3]. The

application of Alloy in the context of graph transformations has been recently proposed in [10]. Additional material for the example can be found in [6].

The rest of this paper is structured as follows. Section 2 describes the running example which is used throughout the paper. Section 3 sketches how to apply the model validator in the context of the example. Section 4 shows how transformation models can be inspected with regard to consistency. Section 5 applies our technique for checking transformation model consequences. Section 6 shortly discusses translation and solving times. Section 7 closes the paper with a conclusion and future work.

2 Transformation Model Example

The running example in this paper is the well-known transformation between Entity-Relationship (ER) and relational database schemata. We study this transformation in form of a transformation model as introduced in [5, 2]. A transfor-

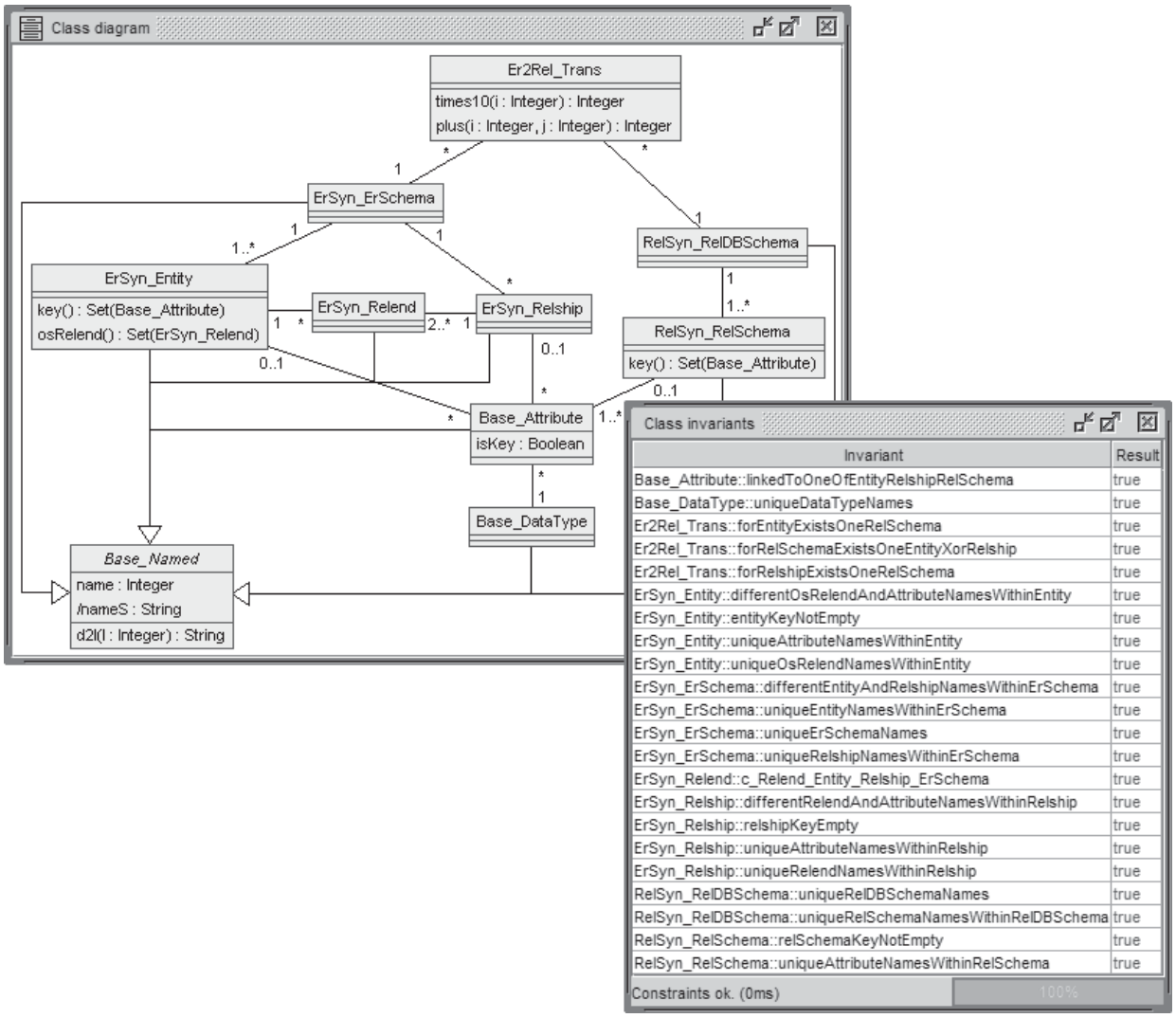

Fig. 1. Class diagram and invariants for example transformation model.

mation model is a descriptive model where the relationship between source and target is purely characterized by the (source,target) model pairs determined by

the transformation. A transformation model consists in our approach of a plain UML class diagram with restricting OCL invariants. Typically, there is an anchor class for the source model, an anchor class for the target model, and a connecting class for the transformation. There are OCL invariants for restricting the source metamodel, for the target metamodel, and for the transformation.

In Fig. 1 the class diagram and the invariant names for the example are pictured. All details of the example can be found in [5]. The example transformation model has four parts: a base part with datatypes and attributes for concepts commonly employed in the ER and relational model; a part for ER schemata (`ErSchema`) with the concepts `Entity`, `Relationship`, and `Relend` (relationship end); a part for relational database schemata (`RelDBSchema`) incorporating relational schemata (`RelSchema`); finally, a part for the transformation (`Trans`). [5] discusses also the semantics. Therefore, some classes here are marked in their names as belonging to the syntax (`ErSyn`, `RelSyn`).

We have used the terms source and target, but transformation models are direction-neutral due to the central employment of associations. We will say that we 'transform a source ER schema into a target relational database schema', but formally the class diagram does not indicate any direction. In our view, transformation models can be looked at as a form of bidirectional transformations.

Currently our model validator does not support the computation of strings in a satisfactory way. In particular, we need string computations for relational attributes in connection with ER attribute names and relationship end names. Through this, we can establish a connection between the source and the target model. Thus, in contrast to [5], we model names (for example, of entities or attributes) as integers and have to pose certain restrictions on the use of the underlying integers and strings. However, through a derived attribute `nameS`, we are able to represent the 'integer names' formally as string values. For example, we will calculate: `20 = 2*10+0` ≅ '2.concat(0)' ≅ `'C'.concat('A') = 'CA'`.

3 Applying the USE Model Validator

We explain the application of the USE model validator by showing how the tool has to be configured in order to construct a model transformation between an example ER schema and a corresponding relational database schema. The needed configuration is shown in Fig. 2 and the resulting generated object diagram, which captures both schemata, is pictured in Fig. 3.

In a model validator configuration, the population of (a) *classes*, (b) *associations*, (c) *attributes* and (d) *datatypes* is determined. Classes, attributes and datatypes are displayed in the configuration table in black-on-white, and associations in black-on-light-grey. (a) A *class* needs an integer upper bound for the maximal number of objects in that class, and an optional lower bound may be given. (b) *Associations* may also have a lower and upper bound for the number of links or their population may be left open and be thus determined through the (up-

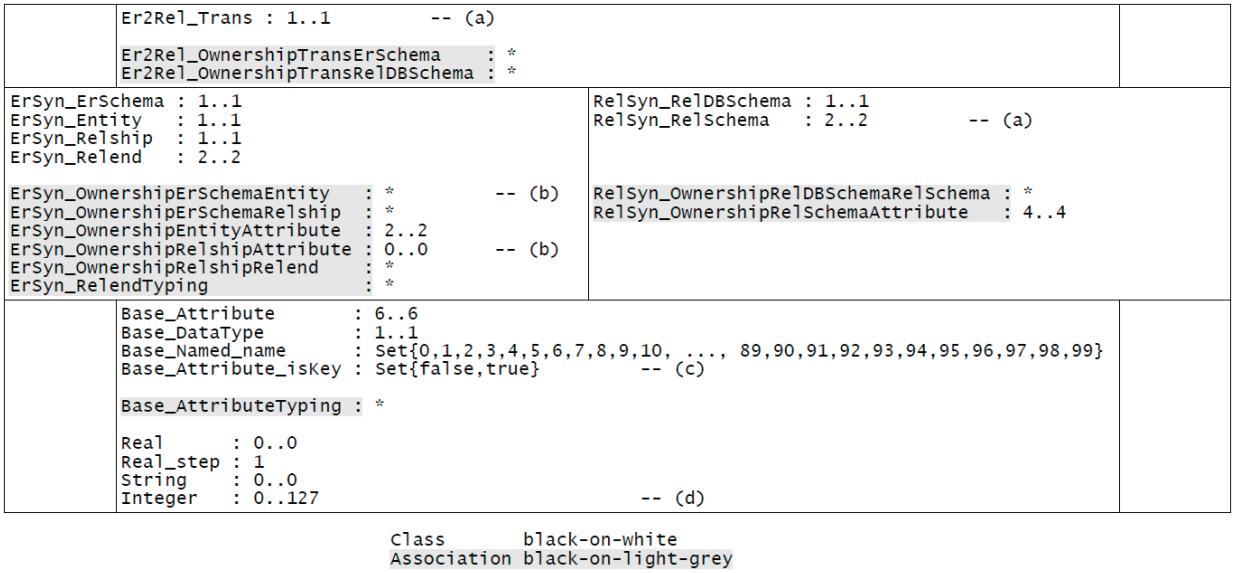

Fig. 2. Configuration for ER schema with binary relationship.

per bounds for the) participating classes. (c) *Attributes* may be determined by specifying an enumeration of allowed values or by the set of values derived from the value set of the corresponding datatype. (d) The numerical *datatypes* Integer and Real may be configured through an enumeration (e.g., Set{42,44,46} or Set$\{3.14, 6.28, 9.42\}$) or with lower and upper bounds for the interval of allowed values with an additional step value for Real (for example, resulting in Set{-8..7} or Set$\{-1, -0.5, 0, 0.5, 1\}$). The datatype String may be determined by an enumeration (e.g., Set{'UML','OCL','MDE'}) or through a lower and upper bound for the number of automatically generated String literals (resulting in, for example, Set{'String1',...,'String7'}).

The example configuration requires (among other restrictions) the following: (a) there is exactly one transformation object (in class `Er2Rel_Trans`), and there are exactly two relational schemas (in class `RelSyn_RelSchema`); (b) the links in association `ErSyn_OwnershipErSchemaEntity` between `ErSyn_ErSchema` and `ErSyn_Entity` are not explicitly restricted, but only implicitly through the upper bounds of the participating classes, and there is no link in the association `ErSyn_OwnershipRelshipAttribute`, which means that in the constructed ER schema there will be no relationship attribute; (c) the attribute isKey is allowed to take values from the enumeration Set{false,true} (recall that in UML and OCL more than two truth values are available); (d) the datatype Integer is allowed to take values from the interval [0..127].

The automatically generated transformation in Fig. 3 is displayed in form of the constructed object diagram and in form of a visual resp. textual domain-specific representation of the ER schema (in traditional ER notation) resp. the relational database schema (as textual SQL table declarations). In particular, the two relationship ends `E` and `J` of the relationship `HE` are represented as attributes `ED` and `JD` in the relational schema `HE`, because the attribute `D` constitutes the key in entity `HD` and in the relational schema `HD`. If there would be a composed key in the entity `HD`, say attributes `DA` and `DB`, the relational schema `HD` has to contain

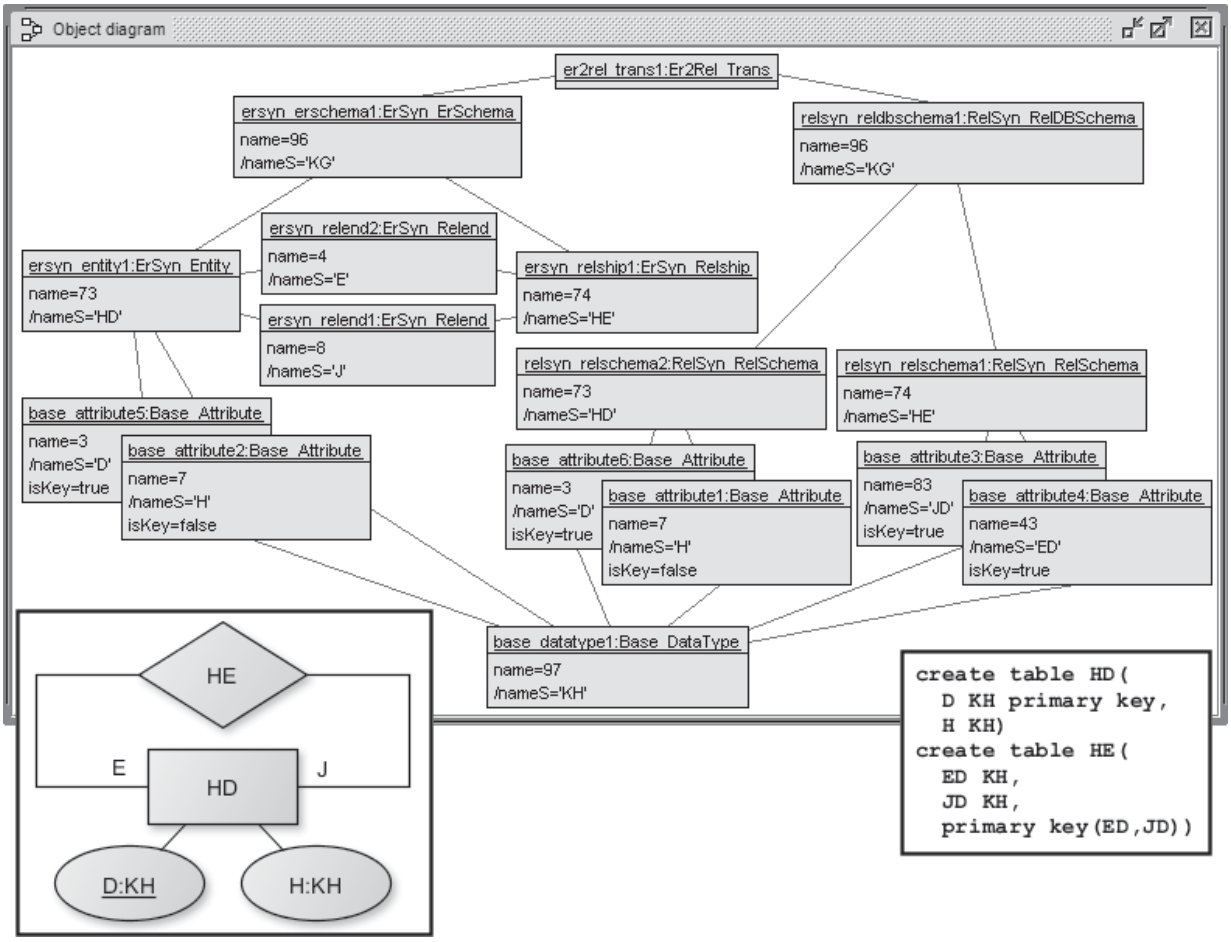

Fig. 3. Generated ER and relational database schema with binary relationship.

four attributes `EDA`, `EDB`, `JDA`, and `JDB`. Thus, the key attribute names on the relational side have to be composed from the relationship end and attribute names from the ER side.

4 Checking Transformation Model Consistency

The configuration table in Fig. 4 shows three configurations needed to check for (1) weak consistency, (2) class instanciability, and (3) class and association instanciability. With option (1) we mean that at least one valid object diagram can be found, even with no objects at all or empty populations for a single class, provided the model and the configuration allows this; option (2) means all classes are instantiated with non-empty populations; option (3) means that all classes and all associations are instantiated. The three options are determined by the first, second, and third column of the parts displayed in Fig. 4 with white-on-dark-grey. The essential differences between the three options are displayed in the following table.

	weak consistency	class instanciability	class and association instanciability
Class	0..UpperBound	1..UpperBound	1..UpperBound
Association	0..*	0..*	1..UpperBound

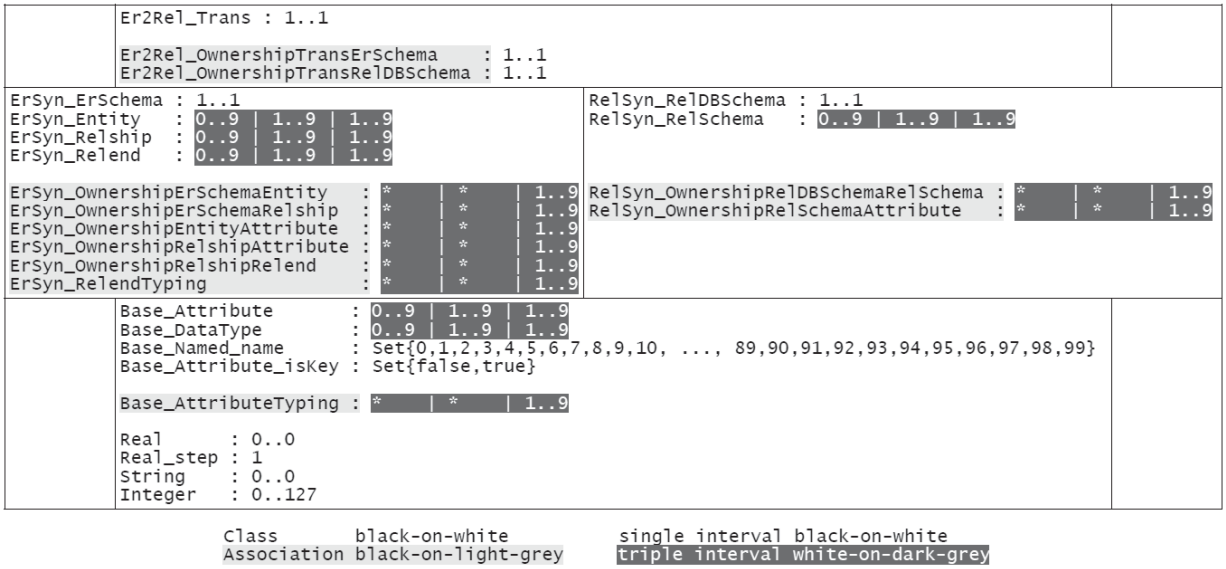

Fig. 4. Configurations for transformation model consistency and instanciability.

The first option states an upper bound for class population, but no lower bound for class population, and no restriction for association population. The second option additionally requires all classes to be populated by at least one object. The third option additionally demands that all associations must be populated by at least one link. As UpperBound we have chosen in the example the integer 9 which results in manageable execution times. Depending on the considered transformation model, this number may be different. However, the table shows the principle approach to handle consistency and (the two kinds of) instanciability in our context.

The object diagrams together with the ER schemata and the SQL table definitions in Figs. 5, 6, and 7 show the results of applying the USE model validator in the example transformation model for proving (by example) weak consistency, class instanciability, and class and association instanciability. As the requirements increase, more and more concepts from the metamodel have to be employed by the model validator. In Fig. 5 only entities and attributes are generated on the ER side (indeed only one entity and one attribute), in Fig. 6 additionally relationships and relationship ends show up, and finally in Fig. 7 also relationship attributes occur. The 'Object count' window and the 'Link count' window in Fig. 5 are the means in the USE tool to shortly check the class and association population in an overview. In the weak consistency case there are no relationships and no relationship end objects, and accordingly there are no links for relationships. For the case described in Fig. 6 (class instanciability), the 'Link count' window would show (if displayed) zero links for the association `ErSyn_OwnershipRelshipAttribute` indicating that there are no relationship attributes.

5 Checking Implications of the Transformation Model

Below an additional ad-hoc invariant is specified. The invariant expresses a connection between the key attributes on the ER and the relational side. This

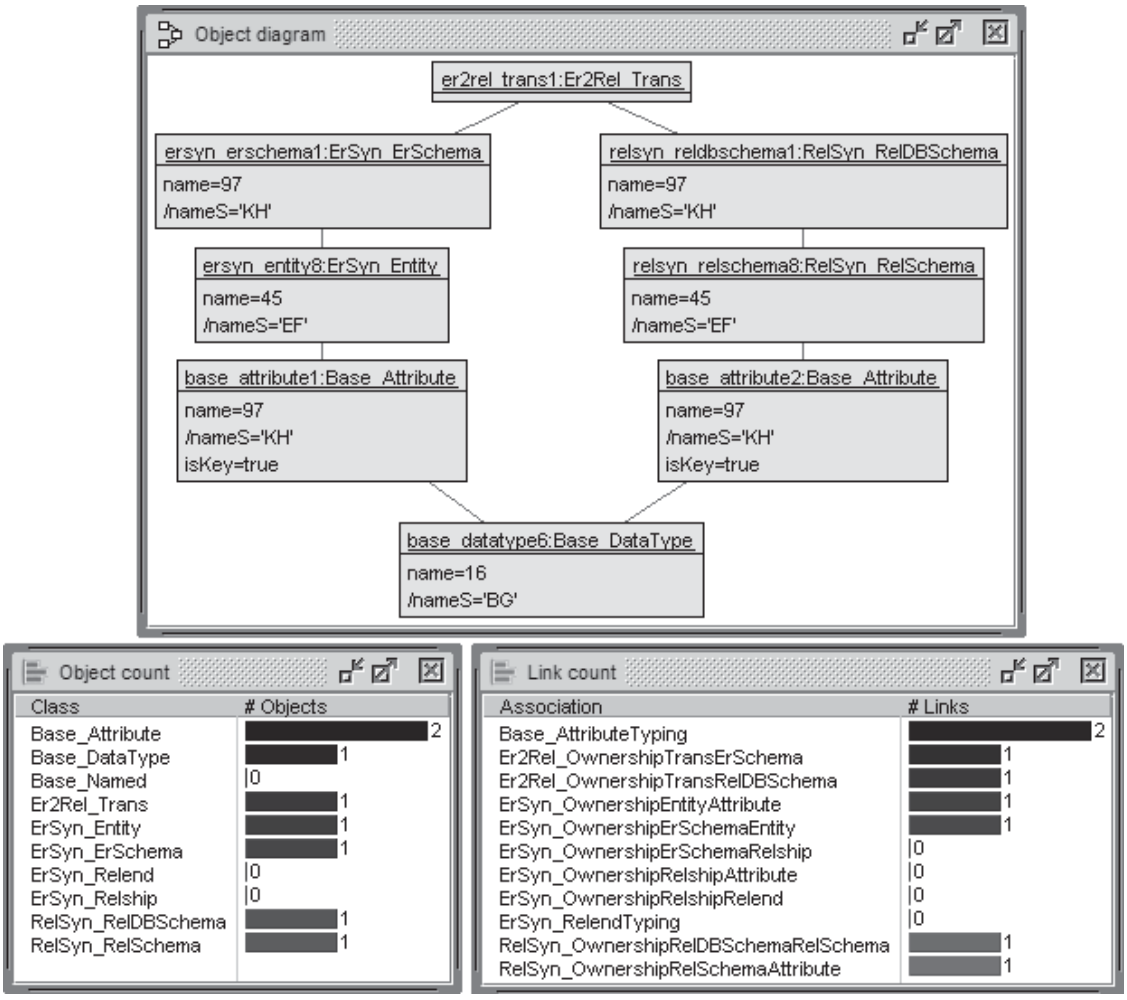

Fig. 5. Generated proof example for weak consistency.

invariant is added to the model invariants, and the model validator is started with the weak consistency configuration from Fig. 4 which is the least restrictive configuration (potentially allowing the maximum set of object diagrams). The model validator reports that the model with the additional invariant is not satisfiable.

```
context self:Er2Rel_Trans
  inv erHasOnlyKeyAttrs_relHasSomeNonKeyAttrs:
  self.erSchema.entity.attribute->
    union(self.erSchema.relship.attribute)->
      select(a|a.isKey=false)->isEmpty() and
  self.relDBSchema.relSchema.attribute->
      select(a|a.isKey=false)->notEmpty()

unsatisfiable: erSchema.hasOnlyKeyAttrs() and
                 not relDBSchema.hasOnlyKeyAttrs()

valid:         erSchema.hasOnlyKeyAttrs() implies
                 relDBSchema.hasOnlyKeyAttrs()
```

Of course, the model validator has checked only a finite number of object diagram candidates which are determined by the stated upper bounds. If we conclude

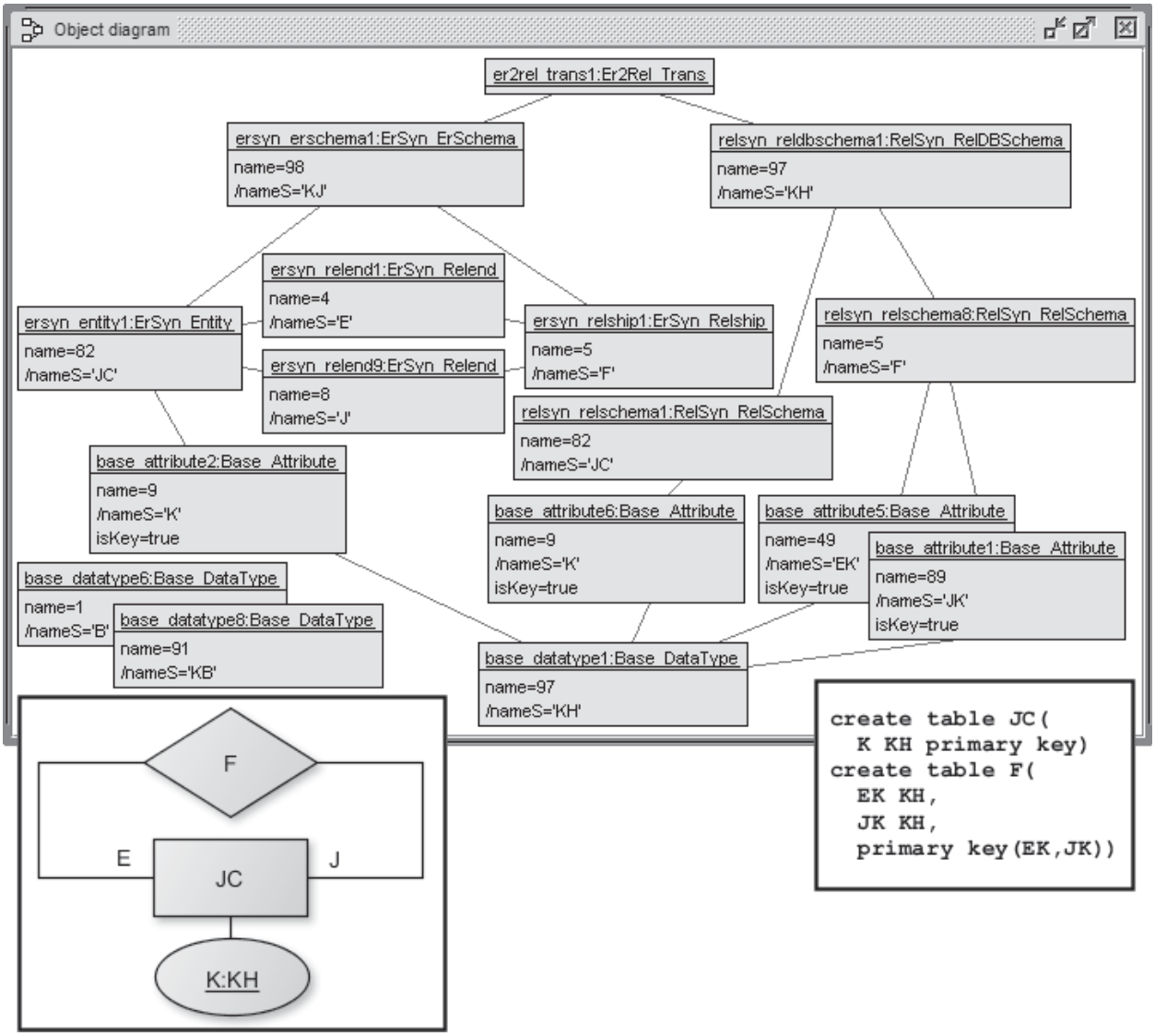

Fig. 6. Generated proof example for class instanciability.

from this fact that the model together with the added invariant is generally unsatisfiable, then the negated invariant must be valid for the transformation model. We argue that the unsatisfiability ends up in showing that the following property can be formally deduced: if there are only key attributes on the ER side, then there are also only key attributes on the relational side, i.e. non-key attributes cannot exist on the relational side. This is an example where we can show that a property, which can be formulated on the source and target side of the transformation, is preserved when considering the direction from the source to the target. We think that the strength of our approach is its genericity: the very same properties can be checked to any transformation model, whichever the considered metamodels are; another strength can be seen in the relative ease of using our approach.

6 Translation and Solving Times

The table below states translation and solving times. The translation time is the time for translating the configuration into a Kodkod relational formula.

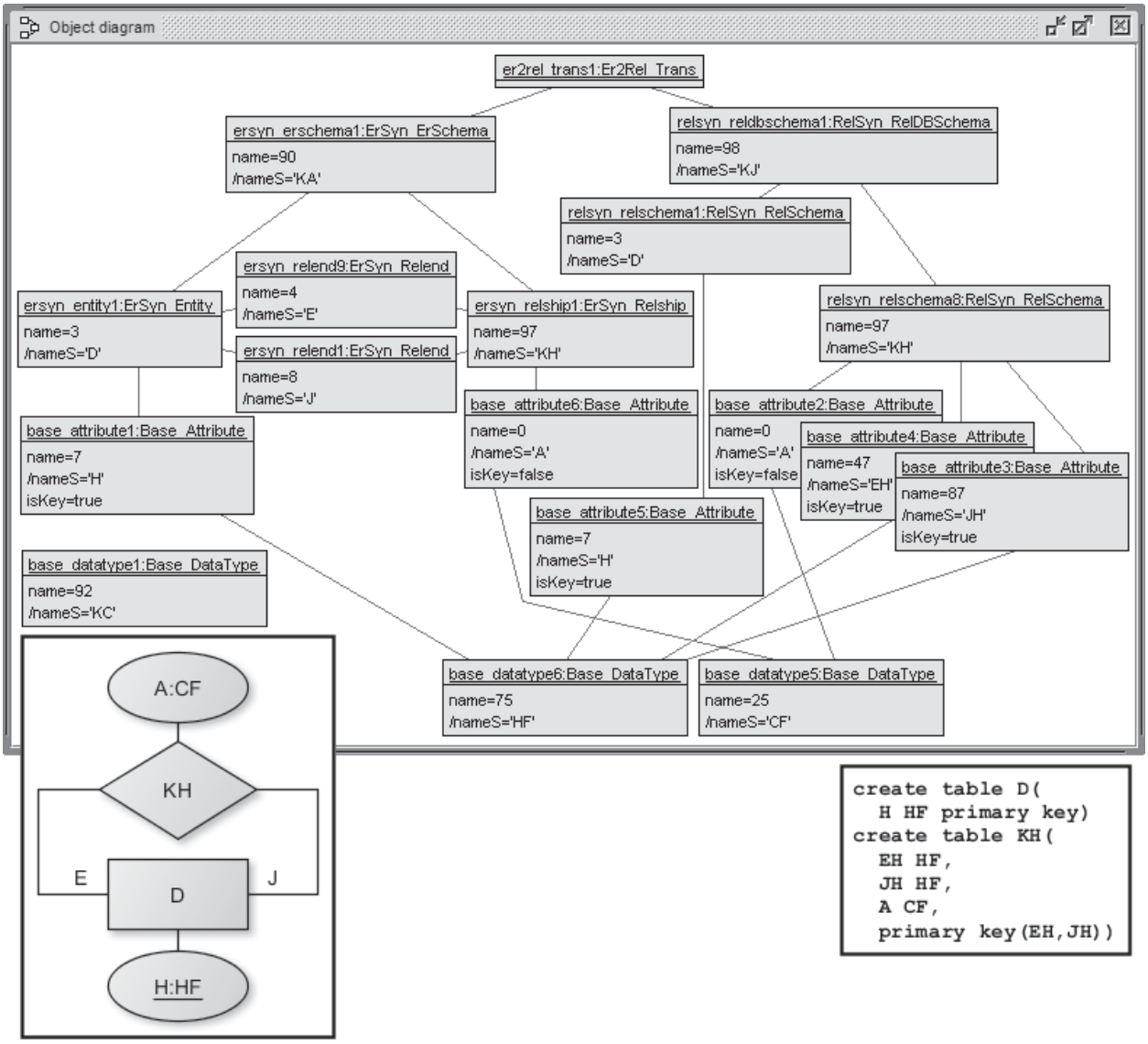

Fig. 7. Generated proof example for class and association instanciability.

The solving time is the time the Kodkod solver needs to compute the answer. The most complex task took about 8 minutes on a standard laptop. We used the Kodkod default solver, but faster solvers are available. Please be aware of the fact that the underlying transformation model has about 20, partly non-trivial OCL constraints. The most complex constraint (see `forRelSchemaExistsOneEntityXorRelship` in [6, page 8]) requires that for every attribute in the relational database schema there either has to be an equivalent entity attribute or a relationship attribute in the ER schema. This constraint is about 20 lines long and has five nested quantifiers.

	Translation [ms]	Solving [ms]
ER schema with binary association	2.152	2.481
Weak consistency	187.002	2.355
Class instanciability	208.258	177.388
Class and association instanciability	184.345	320.705
Implied property ('ER keys' vs. 'rel. keys')	182.957	58.305
Larger example from [6]	12.714	374.650

7 Conclusion

We have presented an approach for automatically checking transformation model features: consistency, class instanciability, class and association instanciability, and property preservation by the transformation model. The implementation of our model validator is based on the relational model finder Kodkod.

Future work could consider to study invariant independence, i.e., minimality of transformation models. One can also deliver to the model validator a partial solution and let the validator automatically complete the transformation. Furthermore, checking for unique results (constructing further results beyond first found solutions) is feasible. The handling of strings must be improved. Last but not least, larger case studies must check the practicability of the approach.

References

1. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In Antoniol, G., Bertolino, A., Labiche, Y., eds.: Proc. Workshops ICST, IEEE (2012) 921–928
2. Bezivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Proc. 9th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2006), Springer, Berlin, LNCS 4199 (2006) 440–453
3. Büttner, F., Egea, M., Guerra, E., de Lara, J.: Checking Model Transformation Refinement. In Duddy, K., Kappel, G., eds.: Proc. Inf. Conf. ICMT. LNCS 7909, Springer (2013) 158–173
4. Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In: ICST Workshops, IEEE Computer Society (2008) 73–80
5. Gogolla, M.: Tales of ER and RE Syntax and Semantics. In Cordy, J.R., Lämmel, R., Winter, A., eds.: Transformation Techniques in Software Engineering, IBFI, Schloss Dagstuhl, Germany (2005) Dagstuhl Seminar Proceedings 05161. 51 pages.
6. Gogolla, M., Hamann, L., Hilken, F.: Checking Transformation Model Properties with a UML and OCL Model Validator: Additional Material. Technical report, University of Bremen (2014) `http://www.db.informatik.uni-bremen.de/publications/intern/GHH2014addon.pdf`.
7. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
8. Kuhlmann, M., Gogolla, M.: From UML and OCL to Relational Logic and Back. In France, R., Kazmeier, J., Breu, R., Atkinson, C., eds.: Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012), Springer, Berlin, LNCS 7590 (2012) 415–431
9. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2007). (2007) LNCS 4424, 632–647
10. Wang, X., Büttner, F., Lamo, Y.: Verification of Graph-based Model Transformations Using Alloy. In Hermann, F., Sauer, S., eds.: Proc. Workshop Graph Transformation and Visual Modeling Techniques (GTVMT'2014), ECEASST, Electronic Communications, `journal.ub.tu-berlin.de/eceasst/`. To appear. (2014)