Checking UML and OCL Model Consistency: An Experience Report on a Middle-Sized Case Study

Martin Gogolla, Lars Hamann, Frank Hilken, Matthias Sedlmeier

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

Abstract. This contribution reports on a middle-sized case study in which the consistency of a UML and OCL class model is checked. The class model restrictions are expressed by UML multiplicity constraints and explicit, non-trivial OCL invariants. Our approach automatically constructs a valid system state that shows that the model can be instantiated and thus proves consistency, i.e., shows that the invariants together with the multiplicity constraints are not contradictory.

1 Introduction

In the context of Model-Driven Engineering (MDE) assuring quality of software models by validation and verification approaches is of central concern. Thus testing and proving techniques and their combination is highly relevant, as several recent case studies from different domains and with different methodological aims demonstrate [11, 1, 2, 9, 3]. The work done here is carried out in the context of the design tool USE (UML-based Specification Environment) developed for UML (Unified Modeling Language) and OCL (Object Constraints Language). USE [5, 8, 7] is employed here for automatically checking the consistency (instantiability) of a UML class model enriched by OCL invariants. USE contains (what we call) a 'model validator' that constructs a system state on the basis of a configuration, which describes a search space for possible system states [10]. With this functionality and on the basis of a transformation into the relational logic of Kodkod [12], it is possible to check the UML and OCL model for consistency, for implications, for invariant independence and for possible completions of partially described system states, among other possible uses.

Here, we discuss a middle-sized case study with complex OCL constraints. We show that it is feasible to automatically check consistency for middle-sized UML and OCL models. Thus, by building a *test* case, i.e., an object model that instantiates the class model, we *prove* a property, namely model consistency or instantiability. The case study is a transformation model that describes the syntax (schemata) and the semantics (states) of the ER (Entity-Relationship) and the relational data model as well as the transformation between these two data models as the object model in Fig. 1 and the class model in Fig. 2 sketches.

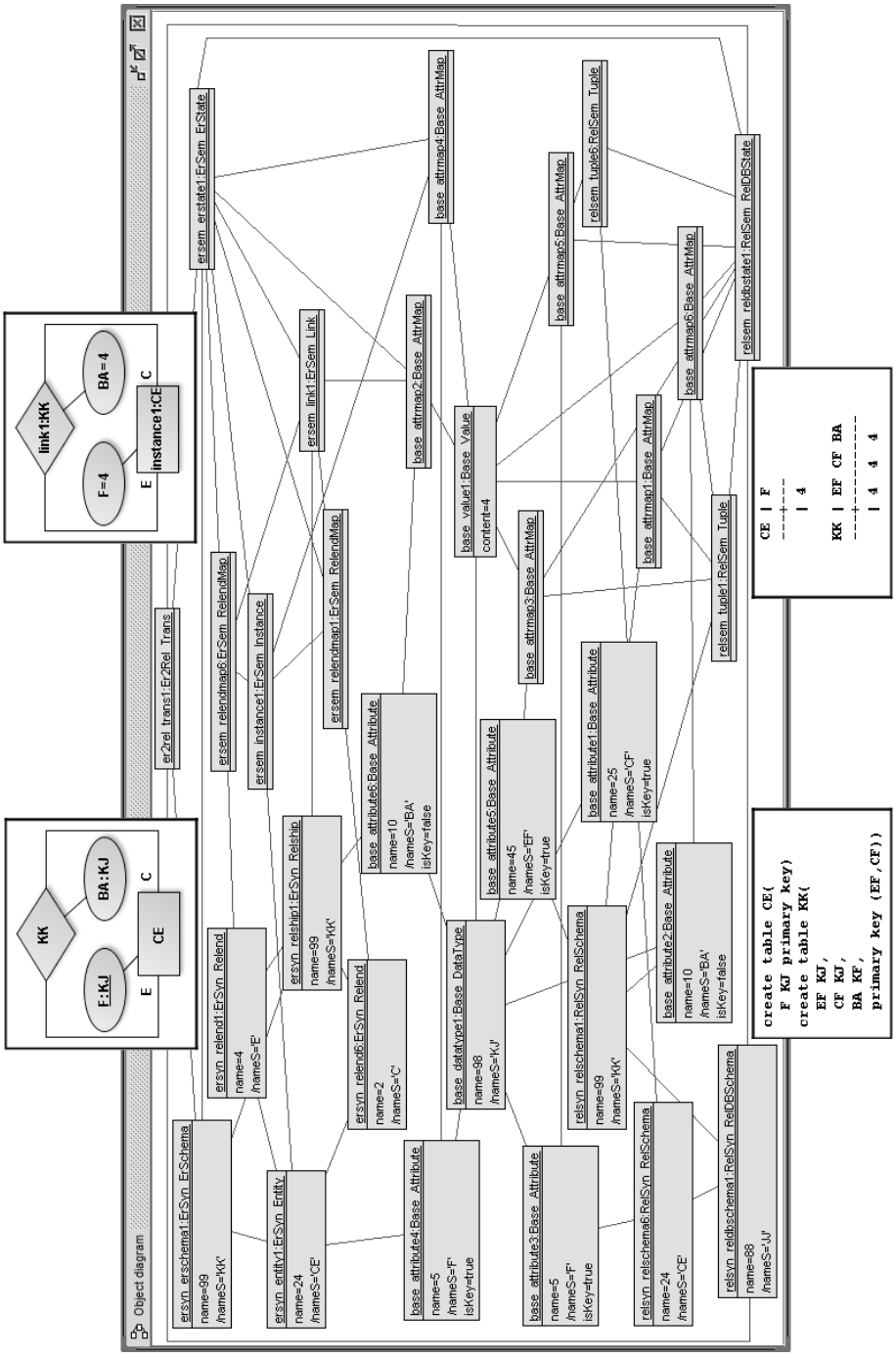


Fig. 1. Generated instantiation (object model) in UML and domain-specific notation.

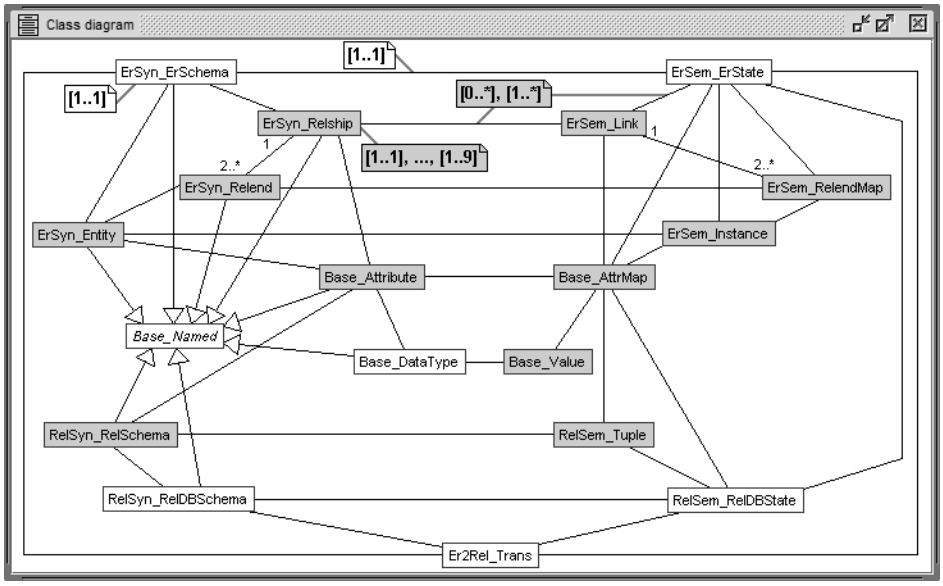


Fig. 2. Case study class model with model validator configurations indicated.

Num Objects	Num Links	USE Response	Times [in milliseconds]		
			Translation	Translation	Solving
1..1	0..*	trivially unsat	358 ms	202 ms	0 ms
1..2	0..*	unsat	328 ms	811 ms	31 ms
1..3	0..*	unsat	359 ms	3292 ms	827 ms
1..4	0..*	sat	359 ms	11092 ms	8205 ms
1..5	0..*	sat	327 ms	31231 ms	45022 ms
1..6	0..*	sat	328 ms	73445 ms	8533 ms
1..7	0..*	sat	327 ms	158839 ms	231053 ms
1..8	0..*	sat	343 ms	301907 ms	149480 ms
1..9	0..*	sat	343 ms	557427 ms	459233 ms
1..1	1..*	trivially unsat	312 ms	203 ms	0 ms
1..2	1..*	unsat	328 ms	827 ms	16 ms
1..3	1..*	unsat	343 ms	3338 ms	78 ms
1..4	1..*	unsat	340 ms	10951 ms	219 ms
1..5	1..*	unsat	343 ms	30857 ms	3572 ms
1..6	1..*	sat	375 ms	74412 ms	134878 ms
1..7	1..*	sat	343 ms	157264 ms	17628 ms
1..8	1..*	sat	394 ms	301315 ms	120432 ms
1..9	1..*	sat	375 ms	551758 ms	607059 ms

Fig. 3. Applied 18 model validator configurations and USE results.

The notion 'semantics' refers here to those parts (classes and associations) of the transformation model that handle the interpretation of 'syntax' classes; for example, the syntax class `Entity` is 'semantically' interpreted by the semantics class `Instance` or `RelSchema` is interpreted by `Tuple`. Regarding the syntax part, you see in Fig. 1 in the upper left a single (tiny) ER schema and in the lower left an equivalent relational database schema. Regarding the semantics part, you see in the upper right a single (tiny) ER state and in the lower right an equivalent relational database state. More details will be mentioned below, and the full UML and OCL model can be found in [4]. Later, Fig. 1 will be discussed further, as well.

2 Case Study Class Model

The case study is a transformation model. A transformation model is a descriptive model where the relationship between source and target is purely characterized by the (source,target) object model pairs determined by the transformation. A transformation model consists in our approach of a plain UML class model with restricting OCL invariants. Typically, there is an anchor class for the source model (class `ErSyn_ErSchema` in Fig. 2), an anchor class for the target model (class `RelSyn_RelDBSchema`), and a connecting class for the transformation (class `Er2Rel_Trans`). There are OCL invariants for restricting the source metamodel, for the target metamodel, and for the transformation. In this example, the transformation model consists of a single class and associations connecting the syntax and semantics parts.

We have studied parts of the case study before [6], however we have not yet handled the semantics parts from the right side of the class model with automatic techniques. In that earlier work, the class model for the syntax and the transformation parts included 10 classes, 11 associations, 22 invariants, and 6 OCL operations. Now, by including also the semantics part, we have 18 classes, 34 associations, 59 invariants, and 10 OCL operations. Thus the complexity of the case study has been more than doubled. In formal terms the OCL invariant coverage and their complexity grew from factor 247 to 775 (for details about the absolute numbers see [4], if needed; however, what is relevant here is the relationship between the numbers). The numbers indicate the coverage of the class model by the invariants and state the number of classes, attributes and associations touched by the constraints. The numbers are therefore one indication for the complexity of the invariants.

There are up to 6 nested quantifiers as, for example, in the invariant `Er2Rel_Trans::forTupleExistsOneInstanceXorLink` (see Fig.4 or [4] for more details, if needed). In the semantics part (not covered in [6]) the constraints also guarantee for the states the validity of key conditions (see for example `RelSem_Tuple::keyMapUnique` in Fig.4 or [4], if needed).

```
-- Entity-Relationship model syntax: Within one Entity, different
-- Attributes have different names

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

-- Relational model semantics: Two different Tuples of one RelSchema
-- can be distinguished in every RelDBState where both Tuples occur by
-- a key Attribute of the RelSchema

context self:RelSem_Tuple inv keyMapUnique:
  RelSem_Tuple.allInstances->forAll(self2 |
    (self<>self2 and self.relSchema=self2.relSchema) implies
    self.relDBState->intersection(self2.relDBState)->forAll(s |
      self.relSchema.key()->exists(ka |
        self.applyAttr(s,ka)<>self2.applyAttr(s,ka))))

-- Transformation: For every Tuple in a 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 forTupleExistsOneInstanceXorLink:
  self.relDBState->forAll(relSt | self.erState->one(erSt |
    relSt.tuple->forAll(t | erSt.instance->one(i |
        t.attrMap->forAll(amRel | i.attrMap->one(amEr |
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value)))
    xor
    erSt.link->one(l | t.attrMap->forAll(amRel |
      ( amRel.attribute.isKey=false implies
        l.attrMap->one(amEr |
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value) )
      and
      ( amRel.attribute.isKey=true implies
        l.relendMap->one(rm |
          rm.instance.attrMap->select(amEr | amEr.attribute.isKey)->
          one(amEr |
            amRel.attribute.name =
              plus(times10(rm.relend.name),amEr.attribute.name)
            and amRel.value=amEr.value))))))))
```

Fig. 4. Typical OCL invariants (3 from 59) in the transformation model.

3 Model Validator Configuration

A model validator configuration determines the population of classes, associations, and attributes: (a) One specifies a mandatory upper and an optional lower bound for each class determining the maximal and minimal number of objects in the expected system state, (b) an optional lower and optional upper bound for each association determining the number of links, and (c) a finite value set for each attribute; attribute values may be determined by finite datatype value sets. The purpose of a configuration is to determine a finite search space for object models (system states) matching the class model and the constraints.

For proving consistency we have fixed some classes and associations in the applied configurations: All 'white' classes in Fig. 2 are fixed to have exactly one object (lower and upper bound '1'), and the associations between these classes are required to have exactly one link. The remaining 'grey' classes (and the associations with at least one participating 'grey' class) have been checked with a fixed lower bound and a varying upper bound: for the classes lower bound '1' and upper bounds equal to a single integer from '1, 2, ..., 9' have been used; for the associations the bounds '0..*' and '1..*' have been employed. One finds the results of running the model validator with these 18 configurations in Fig. 3.

Concerning the data types and attributes, the used configurations employ the range `0..99` for the data type `Integer` and the attributes `Base_Named::name:Integer` and `Base_Value::value:Integer`. Thus a name (for example, for an entity or for an attribute) is handled by the model validator as an `Integer` literal. In order to present a name more intuitively as a `String` literal, we encode the ten digits as letters: 0↦'A', 1↦'B', 2↦'C', 3↦'D', 4↦'E', 5↦'F', 6↦'G', 7↦'H', 8↦'J', 9↦'K'. This encoding is realized in terms of an operation and the derived attribute `Base_Named::nameS:String`. For example, the `Entity` object in the top left of Fig. 1 has `name=24` and `nameS='CE'`. The literal `4` occurring as the value of the `F` attribute in the top right of the figure has been chosen by the model validator from the mentioned range `0..99`.

The solution found by the model validator when employing the `1..6/1..*` configuration from Fig. 3 is pictured as an object model in Fig. 1. It is an automatically constructed test case proving model consistency on the basis of the stated configuration. The (tiny) test case covers an ER schema, an ER state, a relational database schema, and a relational database state.

4 Tool response and translation and solving times

The table in Fig. 3 gives an overview on performed experiments, i.e., 18 configurations with which the model validator has been executed (more configurations have been tested, but they are not documented here). The first and second column determine the configuration's setting for the object number bounds (in the 'grey' classes) and the link number bounds in the connected associations. The third column shows the three different responses made by the model validator: (a) 'trivially unsat' means that the configuration was recognized to be

not instantiable by only analyzing the specified bounds; in this class model (see Fig. 2), for example, each ER relationship ('Relship') must be connected to at least 2 relationship end ('Relend') objects, and this cannot be satisfied by just allowing one 'Relend' object; (b) 'unsat' means that no instantiation can be found; and (c) 'sat' says that an instantiation has been found within the bounds and constructed as an object model. The three time specifications refer to the time needed (a) to translate the class model including the invariants into the relational logic of Kodkod, (b) to translate the relational formula and configuration into SAT (this step is performed by Kodkod), and (c) to solve the translated relational formula by the underlying SAT solver. For the experiments we have used the MiniSat solver.

The difference between the association bounds '0..*' and '1..*' concerns the instantiation. In the first case, not all associations must have links, whereas in the second case, links must be present for all associations: In the first case ER schemata without relationship attributes are allowed, whereas in the second case ER relationships must have attributes, i.e., the association between 'Relship' and 'Attribute' must be instantiated.

- In the '1..*' case, the restriction about the relationship attributes carries over to the relational data model and to the state part. In the '1..*' case, 'ErSem_Link' objects must have attribute values that are described by 'AttrMap' objects; therefore enough 'AttrMap' objects must be present (here, at least 6 'AttrMap' objects). The solution is only found after the object bounds have been set to '1..6' (see Fig. 1 showing six 'AttrMap' objects).
- In the '0..*' case, the solution is already found earlier when the object bounds are set to '1..4', because fewer 'AttrMap' objects are needed when there are no relationship attributes.

As a closing remark in the technical sections, let us consider the transformation model from a different perspective. Up to now, we have discussed the transformation model as transforming the ER model into the relational model, basically going in Fig. 2 from the upper part to the lower part. But formally, this direction is not expressed anywhere in the model in terms of the associations which can be navigated in both directions. One may look at the complete model also as a transformation from syntax to semantics, i.e., going in Fig. 2 from the left part to the right part.

5 Conclusion

We have shown that it is possible to automatically prove properties like consistency for UML and OCL models for non-trivial models in an automatic way. In the future, we want to carry over these results to other properties of UML and OCL models like invariant independence or implications from the stated constraints. The strength of the model validator has to be improved in order to be

able to handle larger solution spaces, i.e., we want to allow more flexible configuration bounds. Better and more detailed feedback from the model validator in case of unsatisfiability should be given more attention. Larger case studies, like for example the UML metamodel, should be considered in order to show that the combination of tests and proofs can be applied for complex and real-world models.

References

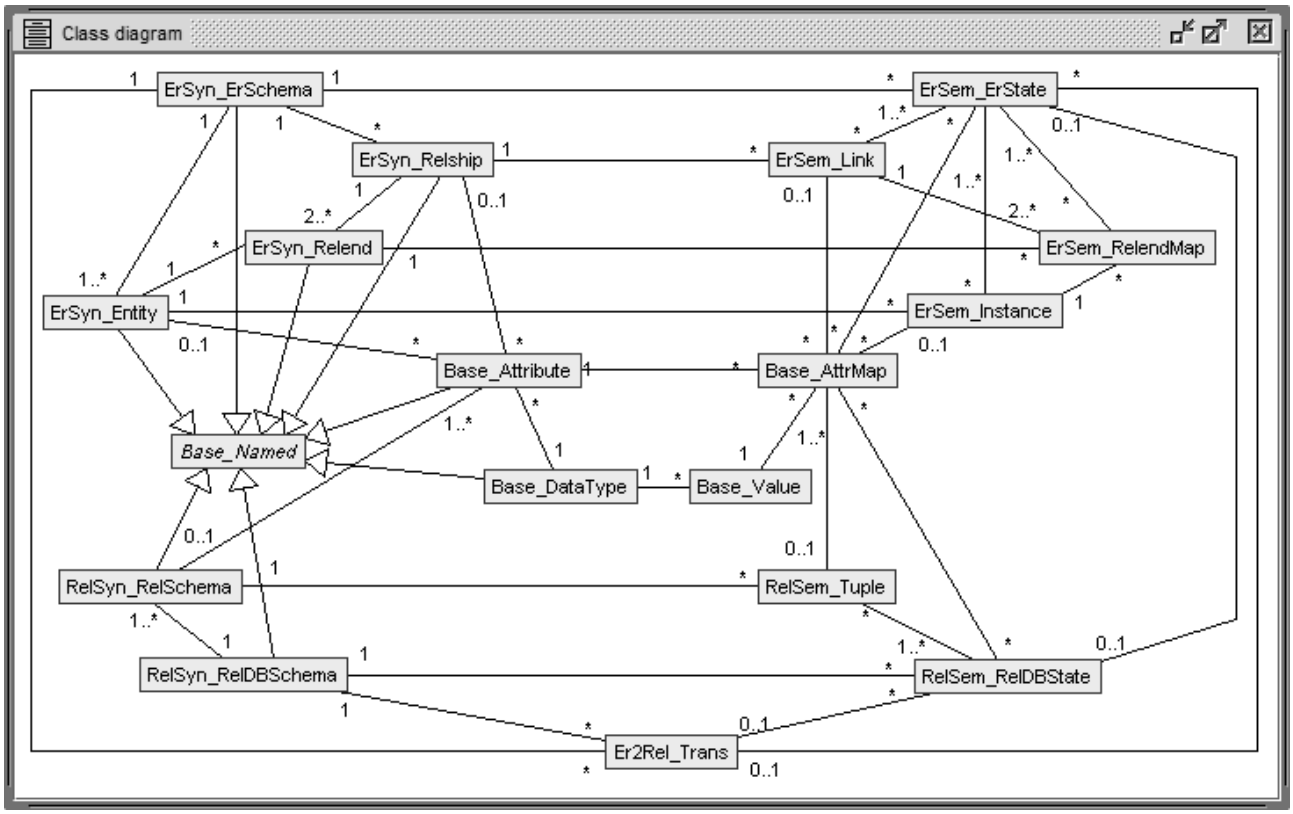
1. Bishop, P.G., Bloomfield, R.E., Cyra, L.: Combining Testing and Proof to Gain High Assurance in Software: A Case Study. In: 24th IEEE Int. Symp. Software Reliability (ISSRE), IEEE (2013) 248–257
2. Brucker, A.D., Feliachi, A., Nemouchi, Y., Wolff, B.: Test Program Generation for a Microprocessor - A Case Study. In Veanes, M., Viganò, L., eds.: Proc. 7th Int. Conf. Tests and Proofs. LNCS 7942 (2013) 76–95
3. Dierkes, M.: Combining Test and Proof in MBAT - An Aerospace Case Study. In Pires, L.F., Hammoudi, S., Filipe, J., das Neves, R.C., eds.: Proc. 2nd Int. Conf. Modelsward, SciTePress (2014) 636–644
4. Gogolla, M., Hamann, L., Hilken, F., Sedlmeier, M.: Additional Material for Checking UML and OCL Model Consistency. `http://www.db.informatik.uni-bremen.de/publications/intern/consis-casestudy-addon.pdf` (2015)
5. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69** (2007) 27–34
6. Gogolla, M., Hamann, L., Hilken, F.: Checking Transformation Model Properties with a UML and OCL Model Validator. In Amrani, M., et al., eds.: Proc. 3rd Int. VOLT Workshop, CEUR Proceedings, Vol. 1325 (2014) 16–25
7. Gogolla, M., Hamann, L., Hilken, F., Kuhlmann, M., France, R.B.: From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In Fill, H., Karagiannis, D., Reimer, U., eds.: Proc. Modellierung (MODELLIERUNG'2014), GI, LNI 225 (2014) 273–288
8. Hamann, L., Hofrichter, O., Gogolla, M.: Towards Integrated Structure and Behavior Modeling with OCL. In France, R., Kazmeier, J., Breu, R., Atkinson, C., eds.: Proc. 15th MODELS, LNCS 7590 (2012) 235–251
9. Kosmatov, N., Lemerre, M., Alec, C.: A Case Study on Verification of a Cloud Hypervisor by Proof and Structural Testing. In Seidl, M., Tillmann, N., eds.: Proc. 8th Int. Conf. Tests and Proofs. LNCS 8570 (2014) 158–164
10. 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. MODELS, LNCS 7590 (2012) 415–431
11. Ledru, Y., du Bousquet, L., Dadeau, F., Allouti, F.: A Case Study in Matching Test and Proof Coverage. ENTCS **190**(2) (2007) 73–84
12. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Proc. Int. Conf. TACAS. (2007) LNCS 4424, 632–647

Additional Material for
Checking UML and OCL Model Consistency:
An Experience Report on a Middle-Sized Case Study

Martin Gogolla, Lars Hamann, Frank Hilken, Matthias Sedlmeier
Database Systems Group, University of Bremen, Germany
{gogolla|lhamann|fhilken|ms}@informatik.uni-bremen.de

1. USE File

Class diagram with multiplicities



```
------------------------------------------------------------------------

model er2mof

------------------------------------------------------------------------
-- Order of packages:
--   Base, ErSyn, ErSem, RelSyn, RelSem, DataMods, Er2Rel
-- Dependencies between packages:
--        Er2Rel
--          |
--      DataMods
--       /    \
--    ErSem  RelSem
--      |      |
--    ErSyn  RelSyn
--       \    /
--        Base
-- Order of USE parts (because of compatibility with older USE):
--   Classes, Associations, Invariants
------------------------------------------------------------------------

------------------------------------------------------------------------
-- Base
------------------------------------------------------------------------

abstract class Base_Named
attributes
  -- assume for attribute name: 0 <= name <= 99
  name : Integer -- String
  nameS : String derived =
    let l1=(name div 10) in let l2=(name.mod(10)) in
    if name<10 then d2l(l2) else d2l(l1).concat(d2l(l2)) endif
operations
  d2l(l:Integer):String=
    if l=0 then 'A' else if l=1 then 'B' else
    if l=2 then 'C' else if l=3 then 'D' else
    if l=4 then 'E' else if l=5 then 'F' else
    if l=6 then 'G' else if l=7 then 'H' else
    if l=8 then 'J' else if l=9 then 'K' else null
    endif endif endif endif endif endif endif endif endif endif
end

class Base_DataType < Base_Named
end

class Base_Value
attributes
  content : Integer -- String
end
```

```
class Base_Attribute < Base_Named
attributes
  isKey : Boolean;
end

class Base_AttrMap
end

--------------------------------------------------------------------------
-- ErSyn
--------------------------------------------------------------------------

class ErSyn_ErSchema < Base_Named
end

class ErSyn_Entity < Base_Named
operations
  key() : Set(Base_Attribute) = self.attribute->select(a | a.isKey)
  osRelend() : Set(ErSyn_Relend) = -- other side relends
    self.relend->collect(re | re.relship.relend->excluding(re))->
      flatten->asSet
end

class ErSyn_Relship < Base_Named
end

class ErSyn_Relend < Base_Named
end

--------------------------------------------------------------------------
-- ErSem
--------------------------------------------------------------------------

class ErSem_ErState
end

class ErSem_Instance
operations
  applyAttr(aState:ErSem_ErState,anAttr:Base_Attribute):Integer=
    self.attrMap->
      select(am|am.erState->includes(aState) and am.attribute=anAttr)->
        any(true).value.content
end

class ErSem_Link
operations
  applyRelend(aState:ErSem_ErState,aRelend:ErSyn_Relend):ErSem_Instance=
    self.relendMap->
      select(rm | rm.erState->includes(aState) and rm.relend=aRelend)->
        any(true).instance
  applyAttr(aState:ErSem_ErState,anAttr:Base_Attribute):Integer=
    self.attrMap->
      select(am|am.erState->includes(aState) and am.attribute=anAttr)->
        any(true).value.content
end
```

```
class ErSem_RelendMap
end

------------------------------------------------------------------------
-- RelSyn
------------------------------------------------------------------------

class RelSyn_RelDBSchema < Base_Named
end

class RelSyn_RelSchema < Base_Named
operations
  key() : Set(Base_Attribute) = self.attribute->select(a | a.isKey)
end

------------------------------------------------------------------------
-- RelSem
------------------------------------------------------------------------

class RelSem_RelDBState
end

class RelSem_Tuple
operations
  applyAttr(aState:RelSem_RelDBState,anAttr:Base_Attribute):Integer=
    self.attrMap->
      select(am | am.relDBState->includes(aState) and
                  am.attribute=anAttr)->any(true).value.content
end

------------------------------------------------------------------------
-- Er2Rel
------------------------------------------------------------------------

class Er2Rel_Trans
operations
  times10(i:Integer):Integer=
    if i=0 then  0 else if i=1 then 10 else
    if i=2 then 20 else if i=3 then 30 else
    if i=4 then 40 else if i=5 then 50 else
    if i=6 then 60 else if i=7 then 70 else
    if i=8 then 80 else if i=9 then 90 else null
    endif endif endif endif endif endif endif endif endif endif
  plus(i:Integer,j:Integer):Integer=
    if j=0 then i   else if j=1 then i+1 else
    if j=2 then i+2 else if j=3 then i+3 else
    if j=4 then i+4 else if j=5 then i+5 else
    if j=6 then i+6 else if j=7 then i+7 else
    if j=8 then i+8 else if j=9 then i+9 else null
    endif endif endif endif endif endif endif endif endif endif
end
```

```
-------------------------------------------------------------------------
-- Base
-------------------------------------------------------------------------

association Base_ValueTyping between
  Base_Value[0..*] role value;
  Base_DataType[1] role dataType;
end

association Base_AttributeTyping between
  Base_Attribute[0..*] role attribute;
  Base_DataType[1] role dataType;
end

association Base_AttrMapTarget between
  Base_AttrMap[0..*] role attrMap;
  Base_Value[1] role value;
end

association Base_AttrMapTyping between
  Base_AttrMap[0..*] role attrMap;
  Base_Attribute[1] role attribute;
end

-------------------------------------------------------------------------
-- ErSyn
-------------------------------------------------------------------------

-- ownerships assocs

association ErSyn_OwnershipErSchemaEntity between
  ErSyn_ErSchema[1] role erSchema;
  ErSyn_Entity[1..*] role entity;
end

association ErSyn_OwnershipErSchemaRelship between
  ErSyn_ErSchema[1] role erSchema;
  ErSyn_Relship[0..*] role relship;
end

association ErSyn_OwnershipRelshipRelend between
  ErSyn_Relship[1] role relship;
  ErSyn_Relend[2..*] role relend;
end

association ErSyn_OwnershipEntityAttribute between
  ErSyn_Entity[0..1] role entity;
  Base_Attribute[0..*] role attribute;
end

association ErSyn_OwnershipRelshipAttribute between
  ErSyn_Relship[0..1] role relship;
  Base_Attribute[0..*] role attribute;
end
```

```
-- Typing associations

association ErSyn_RelendTyping between
  ErSyn_Relend[0..*] role relend;
  ErSyn_Entity[1] role entity;
end

------------------------------------------------------------------------
-- ErSem
------------------------------------------------------------------------

-- Ownership associations  - - - - - - - - - - - - - - - - - - - - - -

association ErSem_OwnershipErStateInstance between
  ErSem_ErState[1..*] role erState;
  ErSem_Instance[0..*] role instance;
end

association ErSem_OwnershipErStateLink between
  ErSem_ErState[1..*] role erState;
  ErSem_Link[0..*] role link;
end

association ErSem_OwnershipErStateAttrMap between
  ErSem_ErState[0..*] role erState;
  Base_AttrMap[0..*] role attrMap;
end

association ErSem_OwnershipErStateRelendMap between
  ErSem_ErState[1..*] role erState;
  ErSem_RelendMap[0..*] role relendMap;
end

-- Instance, AttrMap and RelendMap source associations - - - - - - - - -

association ErSem_InstanceAttrMap between
  ErSem_Instance[0..1] role instance;
  Base_AttrMap[0..*] role attrMap;
end

association ErSem_LinkAttrMap between
  ErSem_Link[0..1] role link;
  Base_AttrMap[0..*] role attrMap;
end

association ErSem_LinkRelendMap between
  ErSem_Link[1] role link;
  ErSem_RelendMap[2..*] role relendMap;
end
```

```
-- RelendMap associations  - - - - - - - - - - - - - - - - - - - - - - - - -

association ErSem_RelendMapTarget between
  ErSem_RelendMap[0..*] role relendMap;
  ErSem_Instance[1] role instance;
end

association ErSem_RelendMapTyping between
  ErSem_RelendMap[0..*] role relendMap;
  ErSyn_Relend[1] role relend;
end

-- Typing associations - - - - - - - - - - - - - - - - - - - - - - - - - -

association ErSem_ErStateTyping between
  ErSem_ErState[0..*] role erState;
  ErSyn_ErSchema[1] role erSchema;
end

association ErSem_InstanceTyping between
  ErSem_Instance[0..*] role instance;
  ErSyn_Entity[1] role entity;
end

association ErSem_LinkTyping between
  ErSem_Link[0..*] role link;
  ErSyn_Relship[1] role relship;
end

-------------------------------------------------------------------------
-- RelSyn
-------------------------------------------------------------------------

association RelSyn_OwnershipRelDBSchemaRelSchema between
  RelSyn_RelDBSchema[1] role relDBSchema;
  RelSyn_RelSchema[1..*] role relSchema;
end

association RelSyn_OwnershipRelSchemaAttribute between
  RelSyn_RelSchema[0..1] role relSchema;
  Base_Attribute[1..*] role attribute;
end

-------------------------------------------------------------------------
-- RelSem
-------------------------------------------------------------------------

association RelSem_OwnershipRelDBStateTuple between
  RelSem_RelDBState[1..*] role relDBState;
  RelSem_Tuple[0..*] role tuple;
end
```

```
association RelSem_OwnershipRelDBStateAttrMap between
  RelSem_RelDBState[0..*] role relDBState;
  Base_AttrMap[0..*] role attrMap;
end

association RelSem_TupleAttrMap between
  RelSem_Tuple[0..1] role tuple;
  Base_AttrMap[1..*] role attrMap;
end

association RelSem_RelDBStateTyping between
  RelSem_RelDBState[0..*] role relDBState;
  RelSyn_RelDBSchema[1] role relDBSchema;
end

association RelSem_TupleTyping between
  RelSem_Tuple[0..*] role tuple;
  RelSyn_RelSchema[1] role relSchema;
end

--------------------------------------------------------------------------
-- Er2Rel
--------------------------------------------------------------------------

association Er2Rel_OwnershipTransErSchema between
  Er2Rel_Trans[0..*] role trans;
  ErSyn_ErSchema[1] role erSchema;
end

association Er2Rel_OwnershipTransRelDBSchema between
  Er2Rel_Trans[0..*] role trans;
  RelSyn_RelDBSchema[1] role relDBSchema;
end

association Er2Rel_OwnershipTransErState between
  Er2Rel_Trans[0..1] role trans;
  ErSem_ErState[0..*] role erState;
end

association Er2Rel_OwnershipTransRelDBState between
  Er2Rel_Trans[0..1] role trans;
  RelSem_RelDBState[0..*] role relDBState;
end

association Er2Rel_StateTrans between
  ErSem_ErState[0..1] role erState;
  RelSem_RelDBState[0..1] role relDBState;
end
```

```
------------------------------------------------------------------------

constraints

------------------------------------------------------------------------
-- Base
------------------------------------------------------------------------

-- Distinguishability of values: Different Values have different
-- content or are linked to different DataTypes

context self:Base_Value inv differentContentOrDataType:
  Base_Value.allInstances->forAll(self2 |
    self<>self2 implies
    (self.content<>self2.content or self.dataType<>self2.dataType))

-- Commutativity restriction: The DataType of the Attribute of an
-- AttrMap is identical to the DataType of the Value of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Attribute_Value_DataType:
  self.attribute.dataType=self.value.dataType

-- Naming restriction: Different DataTypes have different names

context self:Base_DataType inv uniqueDataTypeNames:
  Base_DataType.allInstances->
    forAll(self2 | self.name=self2.name implies self=self2)

------------------------------------------------------------------------
-- ErSyn
------------------------------------------------------------------------

-- Commutativity restriction: The ErSchema of the Entity of a Relend
-- is identical to the ErSchema of the Relship of the Relend

context self:ErSyn_Relend inv c_Relend_Entity_Relship_ErSchema:
  self.entity.erSchema=self.relship.erSchema

-- Name restrictions - - - - - - - - - - - - - - - - - - - - - - - - -

-- Different ErSchemas have different names

context self:ErSyn_ErSchema inv uniqueErSchemaNames:
  ErSyn_ErSchema.allInstances->
    forAll(self2 | self.name=self2.name implies self=self2)

-- Within one ErSchema, different Entities have different names

context self:ErSyn_ErSchema inv uniqueEntityNamesWithinErSchema:
  self.entity->forAll(e1,e2 | e1.name=e2.name implies e1=e2)

-- Within one ErSchema, different Relships have different names

context self:ErSyn_ErSchema inv uniqueRelshipNamesWithinErSchema:
  self.relship->forAll(r1,r2 | r1.name=r2.name implies r1=r2)
```

```
-- Within one ErSchema, Entities and Relships have different names

context self:ErSyn_ErSchema
  inv differentEntityAndRelshipNamesWithinErSchema:
  self.entity->forAll(e | self.relship->forAll(r | e.name<>r.name))

-- Within one Relship, different Relends have different names

context self:ErSyn_Relship inv uniqueRelendNamesWithinRelship:
  self.relend->forAll(re1,re2 | re1.name=re2.name implies re1=re2)

-- Within one Entity, different Attributes have different names

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

-- Within one Relship, different Attributes have different names

context self:ErSyn_Relship inv uniqueAttributeNamesWithinRelship:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)

-- Within one Entity, opposite side Relends and Attributes have
-- different names

context self:ErSyn_Entity
  inv differentOsRelendAndAttributeNamesWithinEntity:
  self.osRelend()->forAll(re | self.attribute->forAll(a |
    re.name<>a.name))

-- Within one Relship, Relends and Attributes have different names

context self:ErSyn_Relship
  inv differentRelendAndAttributeNamesWithinRelship:
  self.relend->forAll(re | self.attribute->forAll(a | re.name<>a.name))

-- Within one Entity, different opposite side Relends have different
-- names

context self:ErSyn_Entity inv uniqueOsRelendNamesWithinEntity:
  self.osRelend()->forAll(re1,re2 | re1.name=re2.name implies re1=re2)

-- Key restrictions  - - - - - - - - - - - - - - - - - - - - - - - - - -

-- The set of key attributes of an Entity is not empty

context self:ErSyn_Entity inv entityKeyNotEmpty:
  self.key()->notEmpty

-- The set of key attributes of a Relship is empty

context self:ErSyn_Relship inv relshipKeyEmpty:
  self.attribute->select(a | a.isKey)->isEmpty
```

```
-----------------------------------------------------------------------
-- ErSem
-----------------------------------------------------------------------

-- Functional restrictions - - - - - - - - - - - - - - - - - - - - - -

-- An InstanceAttrMap, i.e., an AttrMap being linked to an Instance,
-- represents a non-redundant, functional assignment of a Value to an
-- Attribute of an Entity within an ErState for the given Instance

context self:Base_AttrMap inv instanceAttrMapIsFunction:
  Base_AttrMap.allInstances->forAll(self2 | (self<>self2 and
  self.instance->size=1 and self2.instance->size=1) implies
  ((self.attribute=self2.attribute and self.instance=self2.instance)
  implies (self.erState<>self2.erState and self.value<>self2.value)))

-- A LinkAttrMap, i.e., an AttrMap being linked to a Link, represents a
-- non-redundant, functional assignment of a Value to an Attribute of
-- a Relship within an ErState for the given Link

context self:Base_AttrMap inv linkAttrMapIsFunction:
  Base_AttrMap.allInstances->forAll(self2 |
    (self<>self2 and self.link->size=1 and self2.link->size=1)
    implies
    ((self.attribute=self2.attribute and self.link=self2.link)
     implies
     (self.erState<>self2.erState and self.value<>self.value)))

-- A RelendMap represents a non-redundant, functional assignment of an
-- Instance to a Relend within an ErState for a given Link

context self:ErSem_RelendMap inv relendMapIsFunction:
  ErSem_RelendMap.allInstances->forAll(self2 | self<>self2 implies
    ((self.relend=self2.relend and self.link=self2.link)
     implies
     (self.erState<>self2.erState and self.instance<>self.instance)))

-- Commutativity restrictions  - - - - - - - - - - - - - - - - - - - -

-- Commutativity restrictions touching ErSem and ErSyn classes - - - - -

-- The Attributes of the Entity of an Instance are identical to the
-- Attributes of the AttrMaps of the Instance; in other words, there
-- are Attribute assignments for all Attributes of an Instance

context self:ErSem_Instance inv c_Instance_Entity_AttrMap_Attribute:
  self.entity.attribute=self.attrMap.attribute->asSet

-- The Attributes of the Relship of a Link are identical to the
-- Attributes of the AttrMaps of the Link; in other words, there are
-- Attribute assignments for all Attributes of a Link

context self:ErSem_Link inv c_Link_Relship_AttrMap_Attribute:
  self.relship.attribute=self.attrMap.attribute->asSet
```

```
-- The Relends of the Relship of a Link are identical to the Relends
-- of the RelendMaps of the Link; in other words, there are Relend
-- assignments for all Relends of a Link

context self:ErSem_Link inv c_Link_Relship_RelendMap_Relend:
  self.relship.relend=self.relendMap.relend->asSet

-- The Entity of the Relend of a RelendMap is identical to the Entity
-- of the Instance of the RelendMap

context self:ErSem_RelendMap inv c_RelendMap_Relend_Instance_Entity:
  self.relend.entity=self.instance.entity

-- The Relship of the Relend of a RelendMap is identical to the
-- Relship of the Link of the RelendMap

context self:ErSem_RelendMap inv c_RelendMap_Relend_Link_Relship:
  self.relend.relship=self.link.relship

-- The ErSchema of the ErState of an Instance is identical to the
-- ErSchema of the Entity of the Instance

context self:ErSem_Instance inv c_Instance_Entity_ErState_ErSchema:
  Set{self.entity.erSchema}=self.erState.erSchema->asSet

-- The ErSchema of the ErState of a Link is identical to the ErSchema
-- of the Relship of the Link

context self:ErSem_Link inv c_Link_Relship_ErState_ErSchema:
  Set{self.relship.erSchema}=self.erState.erSchema->asSet

-- The Entity of the Instance of an AttrMap being an InstanceAttrMap
-- is identical to the Entity of the Attribute of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Attribute_Instance_Entity:
  self.attribute.entity=self.instance.entity

-- The Relship of the Link of an AttrMap being a LinkAttrMap is
-- identical to the Relship of the Attribute of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Attribute_Link_Relship:
  self.attribute.relship=self.link.relship

-- Commutativity restrictions touching only ErSem classes  - - - - - - -

-- The ErStates of the Instance of an AttrMap being an InstanceAttrMap
-- include the ErStates of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Instance_ErState:
  self.instance->size=1 implies
  self.instance.erState->includesAll(self.erState)
```

```
-- The ErStates of the Link of an AttrMap being a LinkAttrMap
-- include the ErStates of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Link_ErState:
  self.link->size=1 implies
  self.link.erState->includesAll(self.erState)

-- The ErStates of an Instance of a RelendMap include the
-- ErStates of the RelendMap

context self:ErSem_RelendMap inv c_RelendMap_Instance_ErState:
  self.instance.erState->includesAll(self.erState)

-- The ErStates of a Link of a RelendMap include the ErStates of
-- the RelendMap

context self:ErSem_RelendMap inv c_RelendMap_Link_ErState:
  self.link.erState->includesAll(self.erState)

-- The ErStates of an Instance of a RelendMap include the ErStates of
-- the Link of the Relendmap

context self:ErSem_RelendMap inv c_RelendMap_Instance_Link_ErState:
  self.instance.erState->includesAll(self.link.erState)

-- Uniqueness restrictions for keys and Relend maps  - - - - - - - - - -

-- 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:ErSem_Instance inv keyMapUnique:
  ErSem_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))))

-- Two different Links of one Relship can be distinguished in every
-- ErState where both Links occur by a Relend of the Relship

context self:ErSem_Link inv relendMapUnique:
  ErSem_Link.allInstances->forAll(self2 |
    self<>self2 and self.relship=self2.relship
    implies
    self.erState->intersection(self2.erState)->forAll(s |
      self.relship.relend->exists(re |
        self.applyRelend(s,re)<>self2.applyRelend(s,re))))
```

```
------------------------------------------------------------------------
-- RelSyn
------------------------------------------------------------------------

-- Name restrictions - - - - - - - - - - - - - - - - - - - - - - - - - -

-- Different RelDBSchemas have different names

context self:RelSyn_RelDBSchema inv uniqueRelDBSchemaNames:
  RelSyn_RelDBSchema.allInstances->forAll(self2 |
    self.name=self2.name implies self=self2)

-- Within one RelDBSchema, different RelSchemas have different names

context self:RelSyn_RelDBSchema
  inv uniqueRelSchemaNamesWithinRelDBSchema:
  self.relSchema->forAll(r1,r2 | r1.name=r2.name implies r1=r2)

-- Within one RelSchema, different Attributes have different names

context self:RelSyn_RelSchema inv uniqueAttributeNamesWithinRelSchema:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)

-- Key restriction - - - - - - - - - - - - - - - - - - - - - - - - - - -

-- The set of key Attributes of a RelSchema is not empty

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

------------------------------------------------------------------------
-- RelSem
------------------------------------------------------------------------

-- TupAttrMap is mapping - - - - - - - - - - - - - - - - - - - - - - - -

-- A TupleAttrMap, i.e., an AttrMap being linked to a Tuple,
-- represents a non-redundant, functional assignment of a Value to an
-- Attribute of a RelSchema within an ErState for the given Tuple

context self:Base_AttrMap inv tupleAttrMapIsFunction:
  Base_AttrMap.allInstances->forAll(self2 |
    (self<>self2 and self.tuple->size=1 and self2.tuple->size=1)
    implies
    ((self.attribute=self2.attribute and self.tuple=self2.tuple)
     implies
     (self.relDBState<>self2.relDBState and self.value<>self2.value)))

-- Commutativity restrictions  - - - - - - - - - - - - - - - - - - - - -

-- Commutativity restrictions touching RelSem and RelSyn classes - - - -
```

```
-- The Attributes of the RelSchema of a Tuple are identical to the
-- Attributes of the AttrMaps of the Tuple; in other words, there are
-- Attribute assignments for all Attributes of a Tuple

context self:RelSem_Tuple inv c_Tuple_RelSchema_AttrMap_Attribute:
  self.relSchema.attribute=self.attrMap.attribute->asSet

-- The RelDBSchema of the RelDBState of a Tuple is identical to the
-- RelDBSchema of the RelSchema of the Tuple

context self:RelSem_Tuple inv c_Tuple_RelSchema_RelDBState_RelDBSchema:
  Set{self.relSchema.relDBSchema}=self.relDBState.relDBSchema->asSet

-- The RelSchema of the Tuple of an AttrMap being a TupleAttrMap is
-- identical to the RelSchema of the Attribute of the Tuple

context self:Base_AttrMap inv c_AttrMap_Attribute_Tuple_RelSchema:
  self.attribute.relSchema=self.tuple.relSchema

-- Commutativity restrictions touching only RelSem classes - - - - - - -

-- The RelDBStates of the Tuple of an AttrMap being a TupleAttrMap
-- include the RelDBStates of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Tuple_RelDBState:
  self.tuple->size=1 implies
  self.tuple.relDBState->includesAll(self.relDBState)

-- Key restriction - - - - - - - - - - - - - - - - - - - - - - - - - - -

-- Two different Tuples of one RelSchema can be distinguished in every
-- RelDBState where both Tuples occur by a key Attribute of the
-- RelSchema

context self:RelSem_Tuple inv keyMapUnique:
  RelSem_Tuple.allInstances->forAll(self2 |
    self<>self2 and self.relSchema=self2.relSchema
    implies
    self.relDBState->intersection(self2.relDBState)->forAll(s |
      self.relSchema.key()->exists(ka |
        self.applyAttr(s,ka)<>self2.applyAttr(s,ka))))

-------------------------------------------------------------------------
-- DataMods - Package only with constraints; no classes; no associations
-------------------------------------------------------------------------

-- An Attribute is either an Entity Attribute or a Relship Attribute
-- or a RelSchema Attribute

context self:Base_Attribute inv linkedToOneOfEntityRelshipRelSchema:
  (self.entity->size)+(self.relship->size)+(self.relSchema->size)=1
```

```
-- An AttrMap belongs to either an Instance or a Link or a Tuple,
-- i.e., is an InstanceAttrMap or a LinkAttrMap or a TupleAttrMap

context self:Base_AttrMap inv linkedToOneOfInstanceLinkTuple:
  (self.instance->size)+(self.link->size)+(self.tuple->size)=1

-- An AttrMap lives either in a RelDBState or an ErState

context self:Base_AttrMap inv linkedToOneOfRelDBStateErState:
  self.relDBState->size>0 xor self.erState->size>0

-- An EntityAttrMap and a RelshipAttrMap live in an ErState.
-- A RelSchemaAttrMap lives in a RelDBState.

context self:Base_AttrMap inv typingAttrCharacterizesState:
  (self.attribute.entity->notEmpty or self.attribute.relship->notEmpty)=
    (self.erState->notEmpty) and
  (self.attribute.relSchema->notEmpty)=
    (self.relDBState->notEmpty)

-------------------------------------------------------------------------
-- Er2Rel
-------------------------------------------------------------------------

-- Constraints on syntax part  - - - - - - - - - - - - - - - - - - - - -

-- 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:Er2Rel_Trans inv forEntityExistsOneRelSchema:
  self.erSchema.entity->forAll(e |
    self.relDBSchema.relSchema->one(rl |
      e.name=rl.name and
      e.attribute->forAll(ea |
        rl.attribute->one(ra |
          ea.name=ra.name and ea.dataType=ra.dataType and
          ea.isKey=ra.isKey))))
```

```
-- For every Relship in the ErSchema there is a RelSchema having the
-- same name, Relends representing the arms of the relationship, and
-- Attributes with the same properties, i.e., name, DataType, and key
-- property

context self:Er2Rel_Trans inv forRelshipExistsOneRelSchema:
  self.erSchema.relship->forAll(rs |
    self.relDBSchema.relSchema->one(rl |
      rs.name=rl.name and
      rs.relend->forAll(re | re.entity.key()->forAll(rek |
        rl.attribute->one(ra |
          -- re.name.concat('_').concat(rek.name)=ra.name and
          -- re.name*10+rek.name=ra.name and
          plus(times10(re.name),rek.name)=ra.name and
          rek.dataType=ra.dataType and ra.isKey))) and
      rs.attribute->forAll(rsa |
        rl.attribute->one(ra |
          rsa.name=ra.name and rsa.dataType=ra.dataType and
          ra.isKey=false))))

-- For every RelSchema there is either an Entity or a Relship with the
-- same properties and name; if the RelSchema corresponds to an
-- Entity, both have Attributes with the same names, DataTypes, and
-- key properties; if the RelSchema corresponds to a Relship, the
-- RelSchema has Attributes corresponding to the arms of the Relship
-- and both have Attributes with the same properties

context self:Er2Rel_Trans inv forRelSchemaExistsOneEntityXorRelship:
  self.relDBSchema.relSchema->forAll(rl |
    self.erSchema.entity->one(e |
      rl.name=e.name and
      rl.attribute->forAll(ra |
        e.attribute->one(ea |
          ra.name=ea.name and ea.dataType=ra.dataType and
          ra.isKey=ea.isKey)))
    xor
    self.erSchema.relship->one(rs |
      rl.name=rs.name and
      rl.attribute->forAll(ra |
        rs.relend->one(re |
          re.entity.key()->one(rek |
            -- ra.name=re.name.concat('_').concat(rek.name) and
            -- ra.name=re.name*10+rek.name and
            ra.name=plus(times10(re.name),rek.name) and
            ra.dataType=rek.dataType and ra.isKey))
        xor
        rs.attribute->one(rsa |
          ra.name=rsa.name and ra.dataType=rsa.dataType and
          ra.isKey=false))))
```

```
-- Constraints on semantic part  - - - - - - - - - - - - - - - - - - - - -

-- For every Instance in a ErState there is exactly one Tuple in one
-- RelDBState such that for every AttrMap of the Instance there is
-- exactly one AttrMap of the Tuple having the same attribute name and
-- Value

-- forAll(erSt | one(relSt |
--   forAll(i | one(t |
--     forAll(amEr | one(amRel | equiv(amEr,amRel)))))) -- alpha

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

-- For every Link in a ErState there is exactly one Tuple in one
-- RelDBState such that (A) for every AttrMap of the Link there is
-- exactly one AttrMap of the Tuple having the same attribute name and
-- Value and (B) for every RelendMap of the Link and every AttrMap of
-- a key Attribute of the Instance referred to in the Link there is
-- exactly one AttrMap of the Tuple having a corresponding attribute
-- name and Value

-- forAll(erSt | one(relSt |
--   forAll(l | one(t |
--     forAll(amEr | one(amRel | equiv(amEr,amRel))) -- beta
--     and
--     forAll(rm | forAll(amEr | -- gamma
--       one(amRel | equiv(amEr,amRel))))))))

context self:Er2Rel_Trans inv forLinkExistsOneTuple:
  self.erState->forAll(erSt | self.relDBState->one(relSt |
    erSt.link->forAll(l | relSt.tuple->one(t |
      l.attrMap->forAll(amEr | -- beta
        t.attrMap->one(amRel |
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value))
      and
      l.relendMap->forAll(rm | -- gamma
        rm.instance.attrMap->
        select(amEr | amEr.attribute.isKey)->forAll(amEr |
          t.attrMap->select(amRel | amRel.attribute.isKey)->one(amRel |
            amRel.attribute.name =
              -- rm.relend.name.+'_'+amEr.attribute.name and
              plus(times10(rm.relend.name),amEr.attribute.name) and
              amRel.value=amEr.value))))))
```

```
-- For every Tuple in a 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

-- forAll(relSt | one(erSt |
--   forAll(t |
--     one(i | -- alpha
--       forAll(amRel | one(amEr | equiv(amRel,amEr))))
--     xor
--     one(l | forAll(amRel |
--       ( amRel.notKey implies -- beta
--         one(amEr | equiv(amRel,amEr) ) )
--       and
--       ( amRel.isKey implies -- gamma
--         one(rm | one(amEr | equiv(amRel,amEr) ) ) ))))))

context self:Er2Rel_Trans inv forTupleExistsOneInstanceXorLink:
  self.relDBState->forAll(relSt | self.erState->one(erSt |
    relSt.tuple->forAll(t |
      erSt.instance->one(i | -- alpha
        t.attrMap->forAll(amRel |
          i.attrMap->one(amEr |
            amEr.attribute.name=amRel.attribute.name and
            amEr.value=amRel.value)))
      xor
      erSt.link->one(l |
        t.attrMap->forAll(amRel |
          ( amRel.attribute.isKey=false implies -- beta
            l.attrMap->one(amEr |
              amEr.attribute.name=amRel.attribute.name and
              amEr.value=amRel.value) )
          and
          ( amRel.attribute.isKey=true implies -- gamma
            l.relendMap->one(rm |
              rm.instance.attrMap->select(amEr | amEr.attribute.isKey)->
              one(amEr |
                amRel.attribute.name =
                  -- rm.relend.name+'_'+amEr.attribute.name
                  plus(times10(rm.relend.name),amEr.attribute.name) and
                  amRel.value=amEr.value)))))))

-- In a transformation, there is exactly one RelDBState for every
-- ERState

context self:Er2Rel_Trans inv forErStateExistsOneRelDBState:
  self.erState->forAll(erSt |
    self.relDBState->one(relSt | erSt.relDBState=relSt))
-- In a transformation, there is exactly one ErState for every
```

```
-- RelDBState

context self:Er2Rel_Trans inv forRelDBStateExistsOneErState:
  self.relDBState->forAll(relSt |
    self.erState->one(erSt | relSt.erState=erSt))

-- The ErSchemas of the translated ErStates of a transformation are
-- identical to the ErSchema of the transformation

context self:Er2Rel_Trans inv c_Trans_ErState_ErSchema:
  self.erState->notEmpty implies
  self.erState.erSchema->asSet=Set{self.erSchema}

-- The RelDBSchemas of the translated RelDBStates of a transformation
-- are identical to the RelDBSchema of the transformation

context self:Er2Rel_Trans inv c_Trans_RelDBState_RelDBSchema:
  self.relDBState->notEmpty implies
  self.relDBState.relDBSchema->asSet=Set{self.relDBSchema}

-------------------------------------------------------------------------
```

2. Coverage and Complexity of Invariants

The following tables summarize the stated constraints and their complexity with respect to the class model. The number indicates the coverage of the class model by the respective invariant and specifies the number of classes, attributes and associations touched by the constraint. The number is therefore one indication for the complexity of the invariant.

Base_Value::differentContentOrDataType:	6
Base_DataType::uniqueDataTypeNames:	3
Base_Attribute::linkedToOneOfEntityRelshipRelSchema:	10
Base_AttrMap::typingAttrCharacterizesState:	19
Base_AttrMap::tupleAttrMapIsFunction:	13
Base_AttrMap::linkedToOneOfRelDBStateErState:	7
Base_AttrMap::linkedToOneOfInstanceLinkTuple:	10
Base_AttrMap::linkAttrMapIsFunction:	13
Base_AttrMap::instanceAttrMapIsFunction:	13
Base_AttrMap::c_AttrMap_Tuple_RelDBState:	9
Base_AttrMap::c_AttrMap_Link_ErState:	9
Base_AttrMap::c_AttrMap_Instance_ErState:	9
Base_AttrMap::c_AttrMap_Attribute_Value_DataType:	12
Base_AttrMap::c_AttrMap_Attribute_Tuple_RelSchema:	12
Base_AttrMap::c_AttrMap_Attribute_Link_Relship:	12
Base_AttrMap::c_AttrMap_Attribute_Instance_Entity:	12
ErSyn_Relship::uniqueRelendNamesWithinRelship:	6
ErSyn_Relship::uniqueAttributeNamesWithinRelship:	6
ErSyn_Relship::relshipKeyEmpty:	6
ErSyn_Relship::differentRelendAndAttributeNamesWithinRelship:	10
ErSyn_Relend::c_Relend_Entity_Relship_ErSchema:	12
ErSyn_ErSchema::uniqueRelshipNamesWithinErSchema:	6
ErSyn_ErSchema::uniqueErSchemaNames:	3
ErSyn_ErSchema::uniqueEntityNamesWithinErSchema:	6
ErSyn_ErSchema::differentEntityAndRelshipNamesWithinErSchema:	10
ErSyn_Entity::uniqueOsRelendNamesWithinEntity:	11
ErSyn_Entity::uniqueAttributeNamesWithinEntity:	6
ErSyn_Entity::entityKeyNotEmpty:	7
ErSyn_Entity::differentOsRelendAndAttributeNamesWithinEntity:	15

ErSem_RelendMap::relendMapIsFunction:	13
ErSem_RelendMap::c_RelendMap_Relend_Link_Relship:	12
ErSem_RelendMap::c_RelendMap_Relend_Instance_Entity:	12
ErSem_RelendMap::c_RelendMap_Link_ErState:	9
ErSem_RelendMap::c_RelendMap_Instance_Link_ErState:	12
ErSem_RelendMap::c_RelendMap_Instance_ErState:	9
ErSem_Link::relendMapUnique:	21
ErSem_Link::c_Link_Relship_RelendMap_Relend:	12
ErSem_Link::c_Link_Relship_ErState_ErSchema:	12
ErSem_Link::c_Link_Relship_AttrMap_Attribute:	12
ErSem_Instance::keyMapUnique:	26
ErSem_Instance::c_Instance_Entity_ErState_ErSchema:	12
ErSem_Instance::c_Instance_Entity_AttrMap_Attribute:	12
RelSyn_RelSchema::uniqueAttributeNamesWithinRelSchema:	6
RelSyn_RelSchema::relSchemaKeyNotEmpty:	7
RelSyn_RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema:	6
RelSyn_RelDBSchema::uniqueRelDBSchemaNames:	3
RelSem_Tuple::keyMapUnique:	26
RelSem_Tuple::c_Tuple_RelSchema_RelDBState_RelDBSchema:	12
RelSem_Tuple::c_Tuple_RelSchema_AttrMap_Attribute:	12
Er2Rel_Trans::forTupleExistsOneInstanceXorLink:	44
Er2Rel_Trans::forRelshipExistsOneRelSchema:	39
Er2Rel_Trans::forRelSchemaExistsOneEntityXorRelship:	42
Er2Rel_Trans::forRelDBStateExistsOneErState:	9
Er2Rel_Trans::forLinkExistsOneTuple:	42
Er2Rel_Trans::forInstanceExistsOneTuple:	26
Er2Rel_Trans::forErStateExistsOneRelDBState:	9
Er2Rel_Trans::forEntityExistsOneRelSchema:	27
Er2Rel_Trans::c_Trans_RelDBState_RelDBSchema:	9
Er2Rel_Trans::c_Trans_ErState_ErSchema:	9
	775

Complexity of the formerly considered invariants:

Base_DataType::uniqueDataTypeNames:	3
Base_Attribute::linkedToOneOfEntityRelshipRelSchema:	10
ErSyn_Relship::uniqueRelendNamesWithinRelship:	6
ErSyn_Relship::uniqueAttributeNamesWithinRelship:	6
ErSyn_Relship::relshipKeyEmpty:	6
ErSyn_Relship::differentRelendAndAttributeNamesWithinRelship:	10
ErSyn_Relend::c_Relend_Entity_Relship_ErSchema:	12
ErSyn_ErSchema::uniqueRelshipNamesWithinErSchema:	6
ErSyn_ErSchema::uniqueErSchemaNames:	3
ErSyn_ErSchema::uniqueEntityNamesWithinErSchema:	6
ErSyn_ErSchema::differentEntityAndRelshipNamesWithinErSchema:	10
ErSyn_Entity::uniqueOsRelendNamesWithinEntity:	11
ErSyn_Entity::uniqueAttributeNamesWithinEntity:	6
ErSyn_Entity::entityKeyNotEmpty:	7
ErSyn_Entity::differentOsRelendAndAttributeNamesWithinEntity:	15
RelSyn_RelSchema::uniqueAttributeNamesWithinRelSchema:	6
RelSyn_RelSchema::relSchemaKeyNotEmpty:	7
RelSyn_RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema:	6
RelSyn_RelDBSchema::uniqueRelDBSchemaNames:	3
Er2Rel_Trans::forRelshipExistsOneRelSchema:	39
Er2Rel_Trans::forRelSchemaExistsOneEntityXorRelship:	42
Er2Rel_Trans::forEntityExistsOneRelSchema:	27
	247

```
3. Configuration File

open er2mof.use
kodkod -config satsolver:=MiniSat
kodkod -validate consistency.properties

The 18 configurations arise by selecting
from [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ] exactly one value and
from [ 0 | 1 ] exactly one value.

-----------------------------------------------------------------------

Base_Named_name = Set{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99}

Base_DataType_min = 1
Base_DataType_max = 1

Base_Attribute_min = 1
Base_Attribute_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]
Base_Attribute_isKey = Set{false, true}

Base_Value_min = 1
Base_Value_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]
Base_Value_content = Set{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99}

Base_AttrMap_min = 1
Base_AttrMap_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Base_ValueTyping_min = [ 0 | 1 ]
Base_ValueTyping_max = *

Base_AttributeTyping_min = [ 0 | 1 ]
Base_AttributeTyping_max = *

Base_AttrMapTarget_min = [ 0 | 1 ]
Base_AttrMapTarget_max = *

Base_AttrMapTyping_min = [ 0 | 1 ]
Base_AttrMapTyping_max = *
```

```
-----------------------------------------------------------------------

ErSyn_ErSchema_min = 1
ErSyn_ErSchema_max = 1

ErSyn_Entity_min = 1
ErSyn_Entity_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

ErSyn_Relship_min = 1
ErSyn_Relship_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

ErSyn_Relend_min = 1
ErSyn_Relend_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ErSyn_OwnershipErSchemaEntity_min = [ 0 | 1 ]
ErSyn_OwnershipErSchemaEntity_max = *

ErSyn_OwnershipErSchemaRelship_min = [ 0 | 1 ]
ErSyn_OwnershipErSchemaRelship_max = *

ErSyn_OwnershipRelshipRelend_min = [ 0 | 1 ]
ErSyn_OwnershipRelshipRelend_max = *

ErSyn_RelendTyping_min = [ 0 | 1 ]
ErSyn_RelendTyping_max = *

ErSyn_OwnershipEntityAttribute_min = [ 0 | 1 ]
ErSyn_OwnershipEntityAttribute_max = *

ErSyn_OwnershipRelshipAttribute_min = [ 0 | 1 ]
ErSyn_OwnershipRelshipAttribute_max = *

-----------------------------------------------------------------------

ErSem_ErState_min = 1
ErSem_ErState_max = 1

ErSem_Instance_min = 1
ErSem_Instance_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

ErSem_Link_min = 1
ErSem_Link_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

ErSem_RelendMap_min = 1
ErSem_RelendMap_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ErSem_ErStateTyping_min = [ 0 | 1 ]
ErSem_ErStateTyping_max = *

ErSem_InstanceTyping_min = [ 0 | 1 ]
ErSem_InstanceTyping_max = *
```

```
ErSem_InstanceAttrMap_min = [ 0 | 1 ]
ErSem_InstanceAttrMap_max = *

ErSem_LinkRelendMap_min = [ 0 | 1 ]
ErSem_LinkRelendMap_max = *

ErSem_LinkTyping_min = [ 0 | 1 ]
ErSem_LinkTyping_max = *

ErSem_LinkAttrMap_min = [ 0 | 1 ]
ErSem_LinkAttrMap_max = *

ErSem_OwnershipErStateInstance_min = [ 0 | 1 ]
ErSem_OwnershipErStateInstance_max = *

ErSem_OwnershipErStateLink_min = [ 0 | 1 ]
ErSem_OwnershipErStateLink_max = *

ErSem_OwnershipErStateRelendMap_min = [ 0 | 1 ]
ErSem_OwnershipErStateRelendMap_max = *

ErSem_RelendMapTarget_min = [ 0 | 1 ]
ErSem_RelendMapTarget_max = *

ErSem_RelendMapTyping_min = [ 0 | 1 ]
ErSem_RelendMapTyping_max = *

ErSem_OwnershipErStateAttrMap_min = [ 0 | 1 ]
ErSem_OwnershipErStateAttrMap_max = *

-----------------------------------------------------------------------

RelSyn_RelDBSchema_min = 1
RelSyn_RelDBSchema_max = 1

RelSyn_RelSchema_min = 1
RelSyn_RelSchema_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RelSyn_OwnershipRelDBSchemaRelSchema_min = [ 0 | 1 ]
RelSyn_OwnershipRelDBSchemaRelSchema_max = *

RelSyn_OwnershipRelSchemaAttribute_min = [ 0 | 1 ]
RelSyn_OwnershipRelSchemaAttribute_max = *

-----------------------------------------------------------------------

RelSem_RelDBState_min = 1
RelSem_RelDBState_max = 1

RelSem_Tuple_min = 1
RelSem_Tuple_max = [ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]
```

```
-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RelSem_OwnershipRelDBStateAttrMap_min = [ 0 | 1 ]
RelSem_OwnershipRelDBStateAttrMap_max = *
RelSem_OwnershipRelDBStateTuple_min = [ 0 | 1 ]
RelSem_OwnershipRelDBStateTuple_max = *

RelSem_RelDBStateTyping_min = [ 0 | 1 ]
RelSem_RelDBStateTyping_max = *

RelSem_TupleAttrMap_min = [ 0 | 1 ]
RelSem_TupleAttrMap_max = *

RelSem_TupleTyping_min = [ 0 | 1 ]
RelSem_TupleTyping_max = *

------------------------------------------------------------------------

Er2Rel_Trans_min = 1
Er2Rel_Trans_max = 1

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Er2Rel_OwnershipTransErSchema_min = 1
Er2Rel_OwnershipTransErSchema_max = 1

Er2Rel_OwnershipTransErState_min = 1
Er2Rel_OwnershipTransErState_max = 1

Er2Rel_OwnershipTransRelDBSchema_min = 1
Er2Rel_OwnershipTransRelDBSchema_max = 1

Er2Rel_OwnershipTransRelDBState_min = 1
Er2Rel_OwnershipTransRelDBState_max = 1

Er2Rel_StateTrans_min = 1
Er2Rel_StateTrans_max = 1

------------------------------------------------------------------------

Real = Set{}

String = Set{}

Integer_min = 0
Integer_max = 99

------------------------------------------------------------------------
```