Ein formal validiertes Metamodell für die Transformation von Schemata in Informationssystemen

Arne Lindow, Martin Gogolla, Mark Richters

University of Bremen, FB 3
Postfach 330440, D-28334 Bremen, Germany
http://www.db.informatik.uni-bremen.de

Zusammenfassung

Dieses Papier stellt eine Verbindung von klassischen Datenmodellen wie dem Entity-Relationship-Modell und dem relationalen Modell mit dem Metamodellierungsansatz der Meta-Object-Facility her. Es wird eine Modellierung sowohl für das ER-Modell als auch für das relationale Modell vorgestellt. Ergänzend wird die Transformation von Entity-Relationship-Schemata in relationale Schemata diskutiert. Alle drei Ebenen werden formal spezifiziert und mit einem Werkzeug validiert.

Keywords: *Metamodellierung, Datenmodell, Transformation, ER-Modell, relationales Modell, UML, OCL*

1 Einleitung

Der Entwurf von Informationssystemen mit dem Entity-Relationship-Modell (ER-Modell) gehört zu den anerkannten Standardverfahren in der Softwaretechnik. ER-Schemata stellen eine visuelle, semi-formale Technik dar, die sich in modernen objektorientierten Ansätzen wie UML [8] unter der Bezeichnung Klassendiagramm wiederfinden. In frühen Phasen der Software-Entwicklung entworfene ER-Schemata können beispielsweise zu Zwecken der Implementierung in relationale Datenbankschemata übersetzt werden.

In der Softwaretechnik andererseits erfreuen sich Metamodellierungstechniken zunehmend größerer Beliebtheit. UML als Quasi-Standard der modernen Softwaretechnik wird durch ein Metamodell mittels Sprachmitteln der sogenannten Meta-Object-Facility (MOF) [7] beschrieben. Vereinfacht formuliert stellt MOF eine Teilmenge von UML dar. Für UML wird die abstrakte Syntax mit MOF-Klassendiagrammen festgelegt und kontextsensitive Bedingungen werden als OCL-Regeln [12] formuliert. UML und MOF wiederum setzen zur Darstellung komplexer Zusammenhänge ebenfalls visuelle Techniken ein.

Die vorliegende Arbeit verbindet nun klassische Datenbankmodelle wie das ER-Modell und das relationale Modell mit dem Metamodellierungsansatz von MOF. In dieser Arbeit wird eine MOF-Modellierung sowohl für das ER-Modell als auch für das relationale Modell vorgestellt. Dies ist im Prinzip ähnlich wie die Darstellung von UML mit MOF, indem Klassendiagramme und zugehörige Invarianten als OCL-Regeln spezifiziert werden. Zusätzlich wird in dieser Arbeit aber auch die Transformation von ER-Schemata in relationale Schemata mittels MOF-Klassendiagrammen und OCL-Regeln vorgestellt. Alle drei Ebenen, also Entity-Relationship-, relationale und Transformations-Ebene, werden mit Hilfe des USE-Systems [10] formal spezifiziert und validiert.

Transformationen zwischen Datenmodellen sind auch im Kontext von neueren Ansätzen [9, 11] zur Verbindung von Metamodellierung und Data-Warehouses von Interesse. Neben den aktuellen Arbeiten zum UML-Metamodell [2] gab es auch Ansätze, Vorgänger von UML wie OMT oder den Ansatz von Booch mit Metamodellen zu beschreiben [4, 3]. Ebenfalls durch ein Metamodell charakterisiert ist OML [5]. Diskutiert wurden auch Beschreibungen von Metamodellen ohne eine formale Validierung von Transformationen [1, 6].

Die Struktur dieser Arbeit ist wie folgt. Kapitel 2 diskutiert die Darstellung des relationalen Modells in MOF. In Kapitel 3 wird das Entity-Relationship-Modell mit MOF beschrieben. In Kapitel 4 wird schließlich die formale Transformation zwischen beiden Datenmodellen diskutiert. Die Arbeit schließt mit einem Ausblick in Kapitel 5.

2 Relationales Modell

In diesem Abschnitt wird ein MOF-Metamodell zur Beschreibung des relationalen Modells angegeben. Zusätzlich gewährleisten in OCL formulierte Regeln die Wohlgeformtheit von relationalen Schemata.

2.1 Metamodell

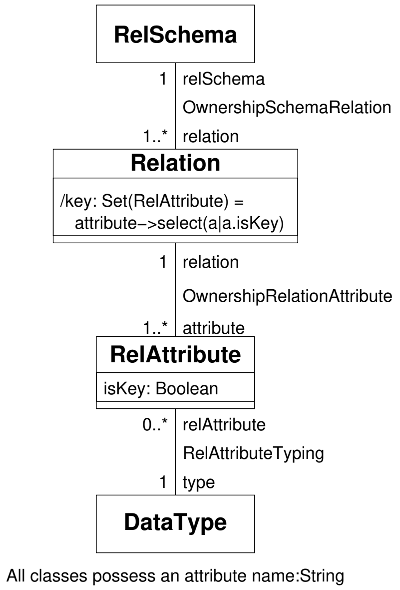

Abbildung 1: MOF-Metamodell des relationalen Modells

Abbildung 1 zeigt das MOF-Klassendiagramm des relationalen Metamodells. Dieses Metamodell ermöglicht die Modellierung von relationalen Schemata als Objektdiagramm, vergleichbar mit der Modellierung von UML-Diagrammen in MOF. Elemente, die mittels Links in der Assoziation *OwnershipSchemaRelation* einem Objekt der Klasse *RelSchema* zugeordnet werden können, sind Objekte der Klasse *Relation*. Ein *RelSchema* muss mindestens aus einer *Relation* bestehen. Diesem wiederum sind Objekte der Klasse *RelAttribute* zugeordnet (mindestens eins). Jedes Objekt der Klasse *RelAttribute* ist durch einen Link der Assoziation *RelAttributeTyping* mit einem Objekt der Klasse *DataType* verbunden. Somit weist der Link dem *RelAttribute* einen *DataType* zu.

2.2 Regeln

Die Regeln sollen sicherstellen, dass nur korrekte relationale Schemata modelliert werden können. Es gibt eine Constraints-Gruppe *Name restrictions*, die für eindeutige Bezeichnungen für die Schema-Elemente sorgen soll, und zum anderen das *Key restriction* Constraint, das garantiert, dass jeder *Relation* mindestens ein Schlüssel zugeordnet wird. Von der Gruppe der *Name restrictions* ist hier ein Constraint exemplarisch herausgegriffen worden, das die Eindeutigkeit der Relationennamen im Schema sicherstellt. Das andere Constraint der Gruppe ist *UniqueAttributesNamesWithinRelation*.

- Name restriction:
  - **uniqueRelationNamesWithinSchema**: Die Namen der *Relationen* im *Schema* müssen eindeutig sein.

```
context RelSchema
inv uniqueRelationNamesWithinSchema:
  relation->forAll(r1, r2|
    r1.name=r2.name implies r1=r2)
```

- Key restriction:
  - **relationKeyNotEmpty**: Eine *Relation* muss mindestens ein Schlüsselattribut besitzen.

```
context Relation inv relationKeyNotEmpty:
  key->notEmpty
```

2.3 Beispiel

Es wird ein relationales Schema vorgestellt, das einen Ausleihvorgang mit Benutzern und Dokumenten modelliert. Drei Relationen werden dargestellt. Der *Borrower* mit seiner *BorrowerID* als *PrimaryKey*, das *Document* mit seinem *DocID-PrimaryKey* und die Relation *Loan*.

```
Borrower(BorrowerID:int, Name:string,
         Address:string)
Document(DocID:int, Title:string, Year:int)
Loan    (BorrowerID:int, DocID:int,
         DateOfLoan:date)
```

Abbildung 2 zeigt dieses relationale Schema als Objektdiagramm des Metamodells im USE-System. Das USE-Werkzeug stellt zudem die Einhaltung der Constraints sicher.

3 ER-Modell

Der im vorigen Abschnitt gezeigte Ansatz für das relationale Modell lässt sich analog auf das ER-Modell übertragen. Der Metamodellierung des ER-Modells in MOF folgt die Beschreibung der OCL-Ausdrücke, die die Wohlgeformtheit von ER-Schemata sicherstellen.

3.1 Metamodell

Abbildung 3 zeigt das MOF-Klassendiagramm des Entity-Relationship-Metamodells. Die zentrale Klasse des Diagramms ist die Klasse *ERSchema*. Jedes *ERSchema* besitzt Beziehungen zu Objekten der Klassen *Entity* und *Relship*. Diese Objekte bilden die grundlegenden Elemente eines *ERSchemas*. Um die *Relationships* mit diesen Elementen modellieren zu können, werden sogenannte *RelshipEnds* eingeführt, die auch als Rollen bezeichnet werden. Diese Konstruktion erlaubt, *Relationships* mit mehreren gleichen beteiligten *Entities* zu mo-

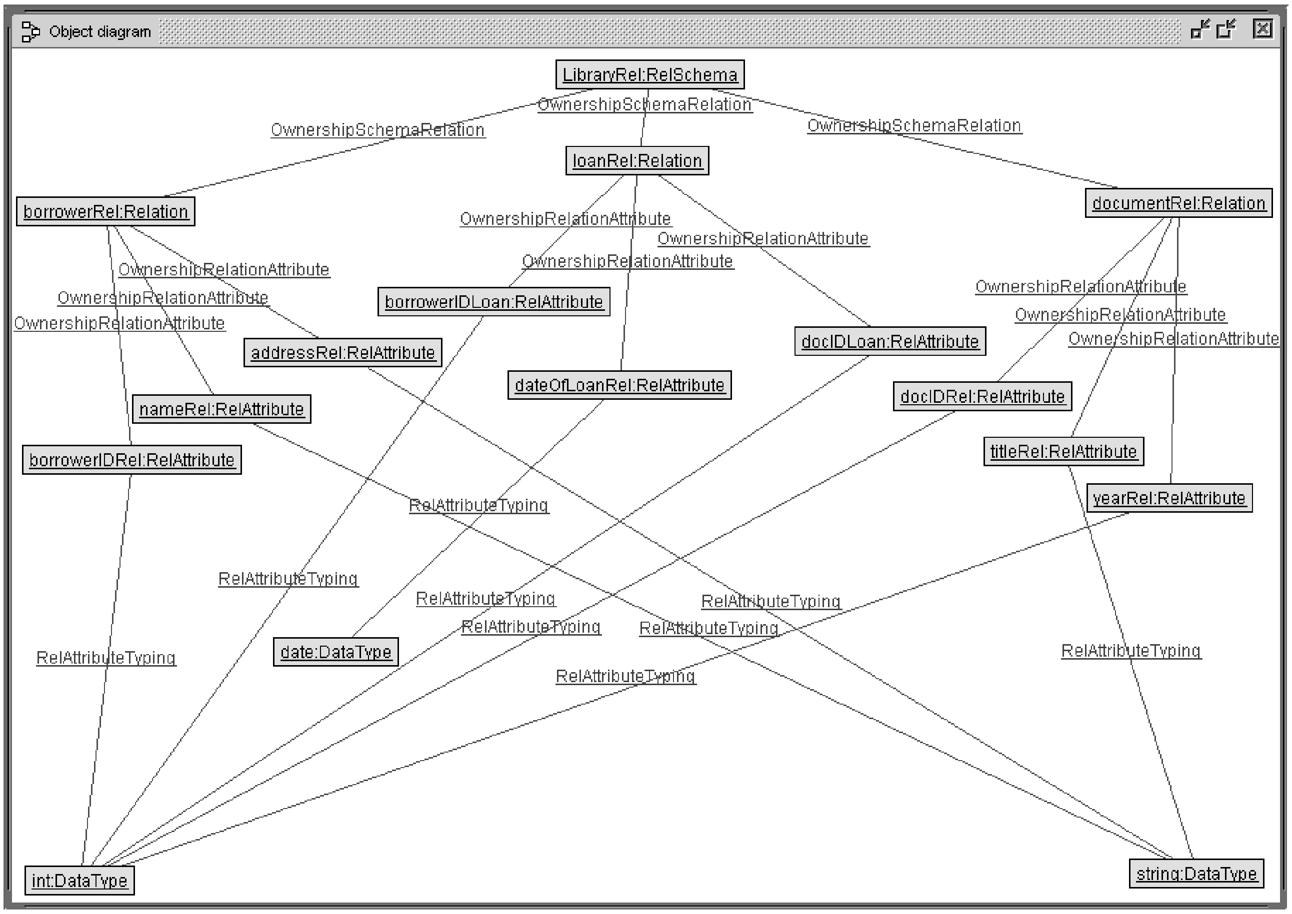

Abbildung 2: Screenshot des USE-Tools mit Beispiel eines relationalen Schemas

dellieren. *Entities* müssen mit Objekten der Klasse *ERAttribute* verbunden sein, *Relationships* können optional Links mit Objekten der Klasse *Attribute* besitzen. Jedes Attribut darf nur zu einem *Entity* oder zu einer *Relationship* gehören. Die Assoziation *ERAttributeTyping* eines *ERAttributes* zu einem *DataType* entspricht der Assoziation *RelAttributeTyping* im relationalen Modell (vgl. Abschnitt 2.1). Jedes Objekt der Klasse *ERAttribute* ist mittels eines Links der Assoziation *AttributeTyping* einem Objekt der Klasse *DataType* zugeordnet, das den Typ des *ERAttribute* angibt.

3.2 Regeln

Diese Regeln, vergleichbar mit den Wohlgeformtheitsregeln (well-formedness rules) des UML-Metamodells, sollen korrekt modellierte ER-Schemata sicherstellen. Die Regeln für das ER-Metamodell lassen sich in unterschiedliche Gruppen einteilen.

Die Gruppe der *Key restrictions* stellt sicher, dass einerseits die Menge der Schlüsselattribute für ein *Entity* nie leer ist, andererseits aber keine *Attributes* von *Relationships* als Schlüsselattribut definiert werden. Das Constraint für die *Attribute restriction* stellt sicher, dass ein *ERAttribute* entweder einem *Entity* oder einer *Relationship* zugeordnet ist. Die Gruppe *Name restrictions* soll eindeutige Namen sicherstellen, damit diese als eindeutige Bezeichner für Schema-Elemente dienen können. Die hier nicht weiter diskutierte Gruppe der *commutativity restrictions* stellt beispielsweise sicher, dass zu einem *RelshipEnd* das über *Entity* und das über *Relship* erreichbare *ERSchema* übereinstimmen.

- Key restrictions:
  - **entityKeyNotEmpty**: Jedes *Entity* muss mindestens ein *Schlüsselattribut* besitzen. Zur Überprüfung dieser Bedingung wird die Operation *key* der Klasse *Entity* verwendet.

    ```
    context Entity inv entityKeyNotEmpty:
      key->notEmpty
    ```
  - **relshipKeyEmpty**: *Attribute* von *Relationships* dürfen keine Schlüsselattribute sein.

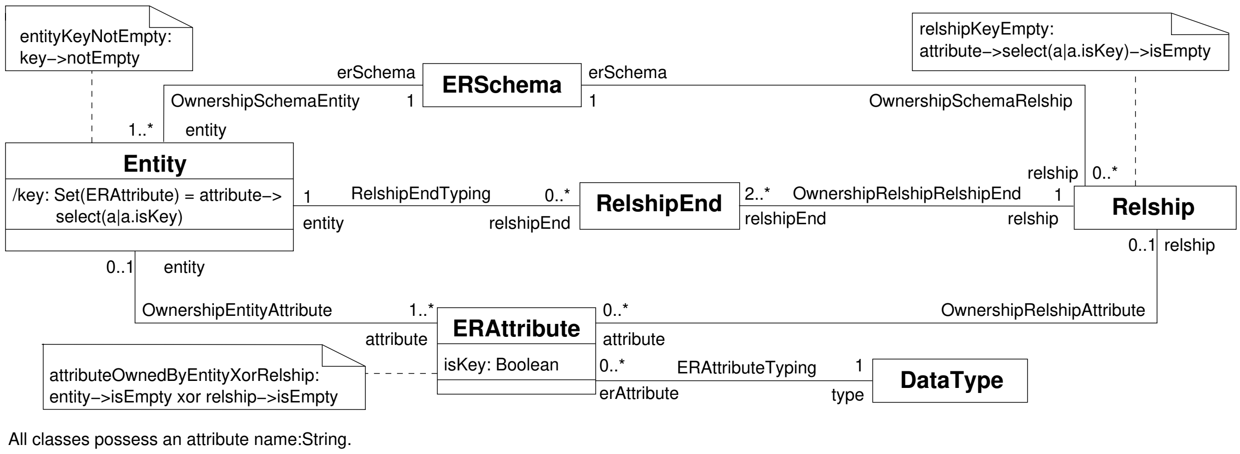

Abbildung 3: MOF-Metamodell des Entity-Relationship-Modells

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

- Attribute restriction:

  - **attributeOwnedByEntityXorRelship**: Ein *Attribute* darf entweder einem *Entity* zugeordnet sein oder einer *Relationship*.

```
context ERAttribute
inv attributeOwnedByEntityXorRelship:
  entity->isEmpty xor relship->isEmpty
```

- Name restrictions:

  - **uniqueAttributeNamesWithinEntity**: Attributnamen innerhalb eines *Entitys* müssen eindeutig sein. Um diese Bedingung zu überprüfen, werden für alle Objekte der Klasse *Entity* die Namen der mit ihr assoziierten Objekte der Klasse *ERAttribute* verglichen. Sollte es zu Namensgleichheit zweier Namens-Attribute von Objekten kommen, muss es sich dabei um das selbe Objekt handeln.

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

Nach gleichen Prinzipien, nur mit Objekten anderer Klassen, stellen folgende Constraints Namenseindeutigkeiten sicher: uniqueSchemaNamesWithinERMModel, uniqueEntityNamesWithinSchema, uniqueRelshipNamesWithinSchema, differentEntityAndRelshipNamesWithinSchema, uniqueRelshipEndNamesWithinRelship, uniqueAttributeNamesWithinRelship, differentAttributeAndRelshipEndNamesWithinEntity, differentAttributeAndRelshipEndNamesWithinRelship.

3.3 Beispiel

Abbildung 4 zeigt ein einfaches ER-Schema, welches in Abbildung 5 als Objektdiagramm des Metamodells aus Abbildung 3 mit Hilfe des USE-Systems dargestellt wird. Es umfasst Elemente vom Typ *Entity*, *Relationship* und *Attribute*, wobei beide *Entities* über Schlüsselattribute verfügen. Die Datentypen sind im ER-Diagramm der Übersichtlichkeit halber nicht explizit benannt.

4 Transformation

4.1 Zusätzliche Elemente

Das Metamodell zur Transformation der Schemata setzt sich aus den beiden oben beschriebenen Metamodellen zusammen und wird ergänzt um eine Klasse *Transformation*, die die Verbindung zwischen einem Objekt der Klasse *RelSchema* und einem Objekt der Klasse *ERSchema* herstellt, deren Äquivalenz sichergestellt werden soll. Dazu werden die beiden Assoziationen *OwnershipTransformationERSchema* und *OwnershipTransformationRelSchema* zwischen der neuen Klasse *Transformation* und den Klassen *ERSchema* bzw. *RelSchema* definiert. Abbildung 6 zeigt den entsprechenden Ausschnitt aus dem Klassendiagramm.

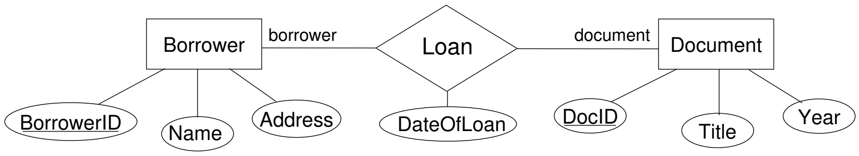

Abbildung 4: Beispiel eines ER-Schemas

4.2 Regeln

Nachdem die Metamodelle für die einzelnen Schemata spezifiziert und im vorigen Abschnitt die Ergänzungen für das Metamodell der Transformation dargestellt wurden, folgen nun die Transformationsregeln.

Exemplarisch stellen wir an dieser Stelle die Transformation von *Relationships* heraus. Zur Transformation einer *Relationship* aus einem ER-Schema in eine *Relation* in einem relationalen Schema müssen die beteiligten *Entities* und ihre Schlüsselattribute berücksichtigt werden. Letztlich müssen im relationalen Modell die Schlüsselattribute der beteiligten *Entities* in einer *Relation* zusammengefasst sein. Für das Metamodell bedeutet dies, dass die entsprechenden *RelAttribute*-Objekte mit einer *Relation*, die die *Relationship* repräsentiert, verlinkt sind.

Letztendlich lassen sich alle Anforderungen an die Transformation in zwei OCL-Ausdrücken zusammenfassen. Man beachte, dass die OCL-Ausdrücke über Quantoren nur die Objekte und ihre Attribute prüfen, aber durch die Navigation implizit die korrekten Links zwischen den Objekten sicherstellen.

- **RelationForEachEntityToTransform**: Zu jedem *Entity* im *ERSchema* muss es eine *Relation* im relationalen Schema geben, die die gleiche Anzahl an Attributen mit den gleichen Attributtypen besitzt. Das Constraint ist in Abbildung 7 dargestellt.

- **RelationForEachRelshipToTransform**: Bei einer *Relationship* im *ERSchema* ist zu beachten, dass zum einen die *Attributes*, die zu dieser *Relationship* gehören, in der zugehörigen *Relation* im relationalen Schema enthalten sind, zum anderen aber auch die Schlüsselattribute der *Entities*, die mittels *RelationshipEnds* mit dieser *Relationship* verbunden sind. Das Constraint ist in Abbildung 8 dargestellt.

4.3 Beispiel

Die Abbildung 9 zeigt einen Ausschnitt des Transformationsmetamodells und die erfüllten Invarianten. Das USE-System stellt somit zum einen die Wohlgeformheit des ER- und des relationalen Schemas sicher, zum anderen zeigt es aber auch, dass die Transformation der Schemata korrekt ist. Unter dem abgebildeten Ausschnitt des Objektdiagramms schließen sich die Beispiele aus den Abschnitten 2.3 und 3.3 an.

5 Ausblick

In dieser Arbeit wurde eine MOF-Modellierung sowohl für das ER-Modell als auch für das relationale Modell vorgestellt und zusätzlich die Transformation von ER-Schemata in relationale Schemata formal beschrieben. Alle Schritte wurden mit Hilfe des USE-Systems formal spezifiziert und validiert.

Die Behandlung weiterer Sprachmittel des ER-Modells wie beispielsweise Generalisierung oder die Ausnutzung von weiteren Möglichkeiten im relationalen Modell wie beispielsweise Fremdschlüssel bedürfen weiterer Überlegung. Die Einbeziehung dieser Elemente erscheint jedoch aufgrund der flexiblen und ausdrucksstarken Möglichkeiten von MOF und OCL unproblematisch. Ein weiteres Augenmerk in zukünftigen Arbeiten wird auf anderen Datenmodellen wie z.B. objektorientierten und zugehörigen Transformationen liegen. Weiter werden auch zugehörige Schemazustände mit in die Betrachtung einfließen.

Literatur

[1] P. Atzeni and R. Torlone. A metamodel approach for the management of multiple models and translation of schemes. *Information Systems*, 18(6):1993, 1993.

[2] T. Clark, A. Evans, and S. Kent. The Metamodelling Language Calculus: Foundation Semantics for UML. In H. Hussmann, editor, *Proc. Int. Conf. FASE*, pages 17–31. Springer, LNCS, 2001.

[3] J. Ebert and R. Süttenbach. A Booch Metamodel. Fachberichte Informatik 5–97, Universität Koblenz-Landau, 1997.

[4] J. Ebert and R. Süttenbach. An OMT Metamodel. Fachberichte Informatik 13–97, Universität Koblenz-Landau, 1997.

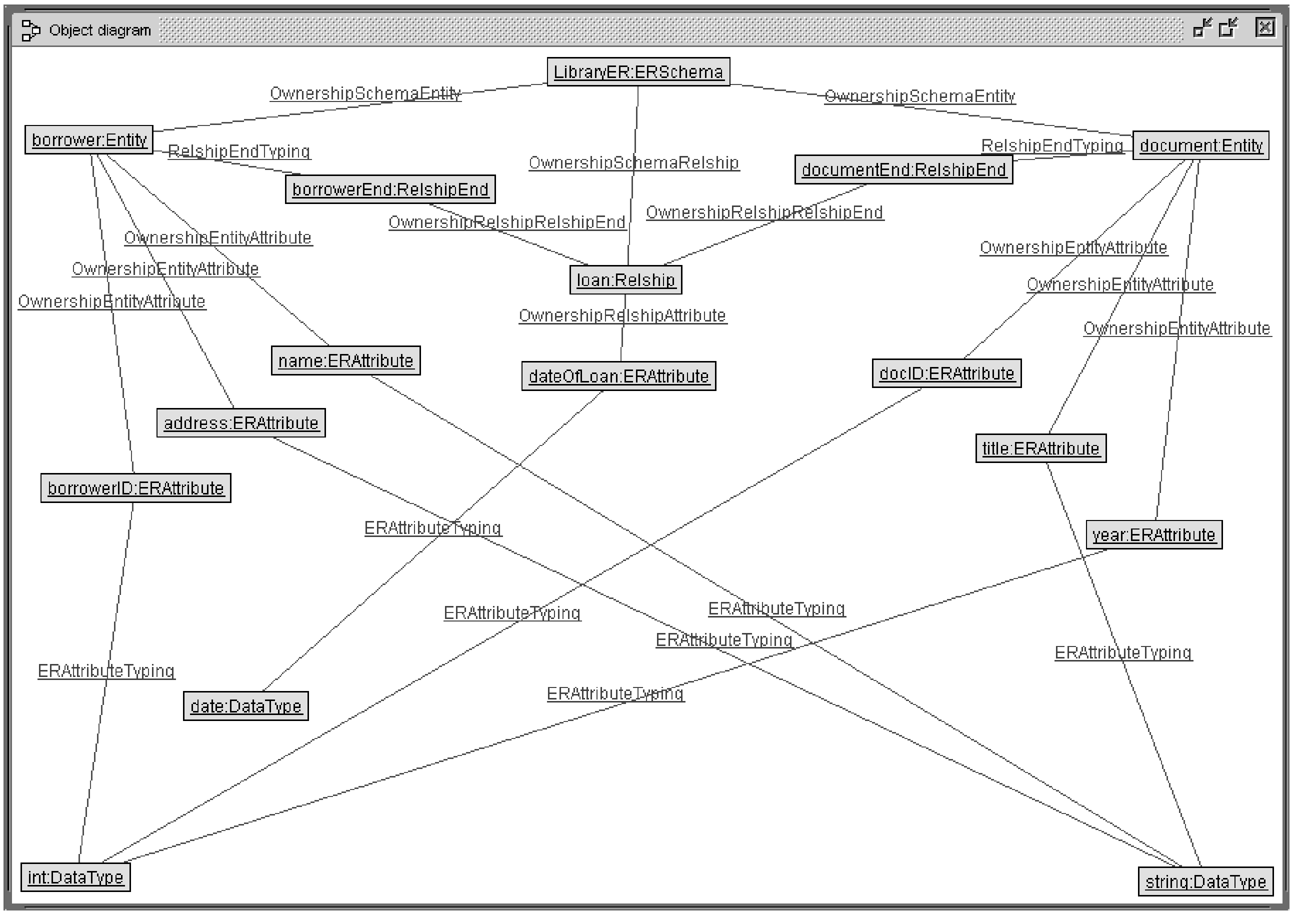

Abbildung 5: Screenshot des USE-Tools mit Beispiel eines ER-Schemas

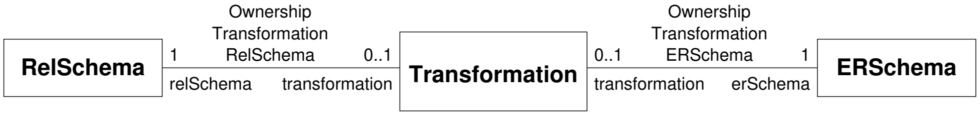

Abbildung 6: Ausschnitt aus dem Metamodell zur Transformation

[5] B. Henderson-Sellers, D. Firesmith, and I. M. Graham. OML Metamodel: Relationships and State Modeling. *Journal Of Object-Oriented Programming*, 10(1):47–51, 1997.

[6] V. B. Misic and S. Moser. A Formal Approach to Metamodeling: A Generic Object-Oriented Perspective. In *Int. Conf. Entity-Relationship Approach (ER'97)*, pages 243–256, 1997.

[7] OMG, editor. *Meta-Object Facility (MOF), Version 1.3*. Object Management Group, Inc., Framingham, Mass., `http://www.omg.org`, 1999.

[8] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3, June 1999*. Object Management Group, Inc., Framingham, Mass., `http://www.omg.org`, 1999.

[9] OMG, editor. *The CWM Specification*. Object Management Group, Inc., Framingham, Mass., `http://www.omg.org`, 2000.

[10] M. Richters and M. Gogolla. Validating UML Models and OCL Constraints. In A. Evans and S. Kent, editors, *Proc. 3rd Int. Conf. Unified Modeling Language (UML'2000)*, pages 265–277. Springer, Berlin, LNCS 1939, 2000.

[11] T. Vetterli, A. Vaduva, and M. Staudt. Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. *SIGMOD Record*, 29(3):68–75, 2000.

[12] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modelling with UML*. Object Technology Series. Addison-Wesley, Reading/MA, 1999.

```
context Transformation inv RelationForEachEntityToTransform:
  self.erSchema.entity->forAll(ent: Entity| self.relSchema.relation->exists(rel:Relation|
    ent.name=rel.name and ent.attribute->forAll(ea|
      rel.attribute->exists(ra| ea.name=ra.name and ea.type=ra.type))
    and ent.key()->forAll(ek|
      rel.key()->exists(rk| ek.name=rk.name and ek.type=rk.type))))
```

Abbildung 7: RelationForEachEntityToTransform-Constraint

```
context Transformation inv RelationForEachRelshipToTransform:
   self.erSchema.relship->forAll(relship:Relship|
       self.relSchema.relation->exists(rel:Relation|
          relship.name=rel.name and relship.relshipEnd->
            forAll(re:RelshipEnd|re.entity.key()->forAll(ek:ERAttribute|
              rel.key()->exists(rk:RelAttribute| re.name.concat(ek.name)=rk.name and ek.type=rk.type))
            and relship.attribute->forAll(rsa:ERAttribute| rel.attribute->exists(ra:RelAttribute|
              rsa.name=ra.name and rsa.type=ra.type and not rsa.isKey))))
```

Abbildung 8: RelationForEachRelshipToTransform-Constraint

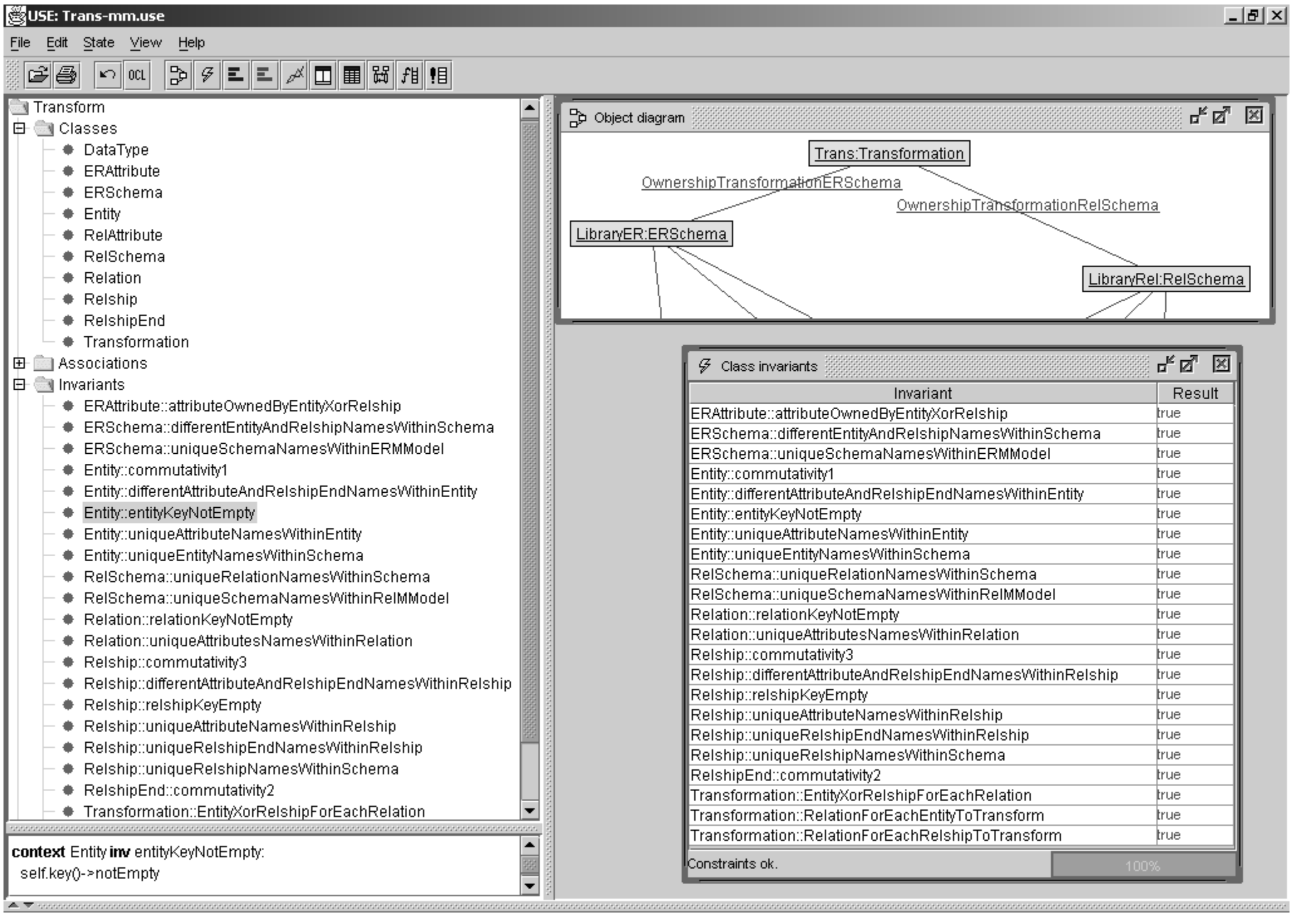

Abbildung 9: Screenshot des USE-Tools mit einem Transformationsbeispiel