Application examples of the object-oriented based translation approach to fragments of the UML, ER and Relational metaschemas

(March 2008)

Ruth Raventós

raventos@lsi.upc.edu

Departament Llenguatges i Sistemes Informàtics

Universitat Politècnica de Catalunya

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

Schema translation is an important problem in the fields of databases and information systems engineering. In current practice, schemas translation problems have been often tackled by means of ad-hoc solutions. However, ad-hoc solutions are very heavy and hard to maintain, and there is still a compelling need for a general solution able to cope, in a uniform way, the large diversity of the various formats and type of information available (Atzeni 2007, Bernstein 2003, Bernstein et al. 2000, Bernstein, Melnik 2007).

The schema translation problem can be simply stated as follows: Given a (source) metaschema MS_1, a (source) schema S_1 (instance of MS_1) and a (target) metaschema MS_2, obtain a schema S_2 instance of MS_2 that suitably corresponds to S_1.

This report illustrates three examples of the application of the object-oriented based translation approach to fragments of different instances of MOF metaschemas.

The first example describes the translation between instances of a fragment of the ER metaschema and a fragment of the Relational metaschema. The example is based on the Gogolla's report described in (Gogolla 2005).

The second example describes the translation between instances of the Simple UML metaschema and the Simple RDBMS metaschema. The example is based on the Annex A of the MOF QVT Final Adopted Specification (Object Management Group 2007) (also included at the end of this report (Annex) that describes the Simple UML to Simple RDBMS Mapping in the QVT Relations Language.

The first example is the application of the translation, described in the first example, from the osCommerce System specified as an instantiation of the Relational metaschema to the osCommerce System specified as an instantiation of the ER metaschema. The example is based on the osCommerce Conceptual Schema described in (Tort and Olivé 2007).

2. Example 1: Translation between ER and Relational Metaschema

This example shows the translation between instances of a fragment of ER metaschema and instances of a fragment of Relational metaschema. The metaschemas described here are very similar to the ones proposed by Gogolla in (Gogolla 2005) with some minor changes: the distinction between data type for the ER metaschema and relational data type for the Relational metaschema; the concept of foreign key is included in the Relational metaschema; and the columns of a table have an order within the table.

2.1 Definition of Metaschemas

ER Metaschema

Figure 1 shows the fragment of the ER metaschema and Figure 2 shows an example of one of its instances.

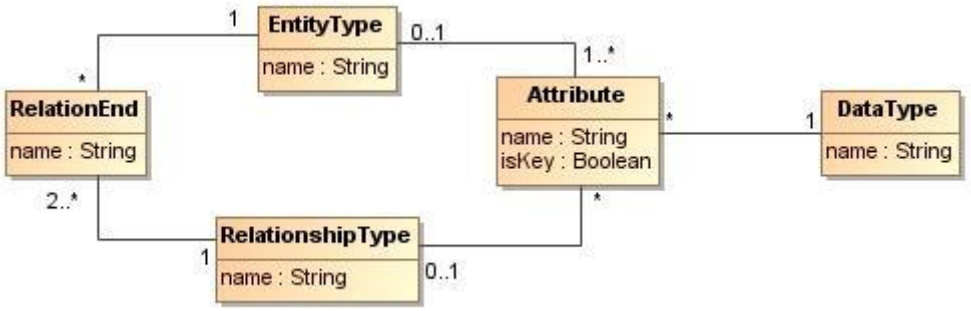

Fig. 1. Fragment of the ER metaschema

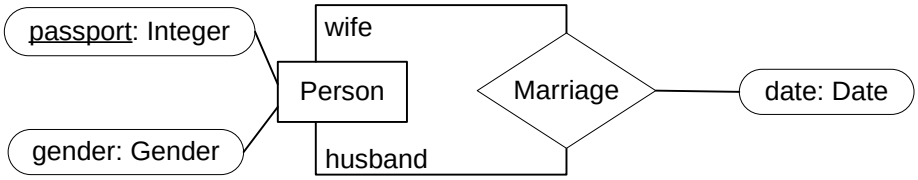

Fig. 2. Example of an instance of the fragment of the ER metaschema

The ER metaschema also includes the following constraints that, formally in OCL, are specified as follows:

Names are defined, have a non-zero length, and consist of letters, digits, parenthesis and underscore:

```
context ErElement inv nameOk:
 let small:Set(String) =
   Set{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
   'r','s','t','u','v','w','x','y','z'} in
 let capital:Set(String) =
   Set{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
   'R','S','T','U','V','W','X','Y','Z'} in
 let digit:Set(String) =
  Set{'0','1','2','3','4','5','6','7','8','9'} in
      self.name.isDefined and self.name.size>0 and
      Set{1..self.name.size}->forAll(i |Set{'_','(',')'}->union(small)->
      union(capital)->union(digit)->includes(self.name.substring(i,i)))
```

Naming restriction: Different Data Types have different names:

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

Entities and Relationships have different names:

```
context self:ErElement inv differentEntityTypeAndRelationshipTypeNames:
  ErElement.allInstances()->forAll(e1, e2 | ((e1.oclIsTypeOf(EntityType) or
    e1.oclIsTypeOf(RelationshipType)) and
   (e2.oclIsTypeOf(EntityType) or e2.oclIsTypeOf(RelationshipType)) and
    e1.name = e2.name) implies e1 = e2)
```

Within one RelationshipType, different relationEnds have different names:

```
context self:RelationshipType inv uniquerelationEndNamesWithinRelationship:
 self.relationEnd->forAll(re1,re2 | re1.name = re2.name implies re1 = re2)
```

Within one EntityType, different Attributes have different names:

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

Within one RelationshipType, different Attributes have different names:

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

The set of key attributes of an EntityType is not empty:

```
context self:EntityType inv EntityTypeKeyNotEmpty: self.key()->notEmpty()
```

The set of key attributes of a RelationshipType is empty:

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

XOR Constraint of Attribute:

```
context self:Attribute inv linkedToEntityTypeOrToRelationshipType:
 (self.entityType->size())+(self.relationshipType->size()) = 1
```

Relational Metaschema

Figure 3 shows the fragment of the Relational metaschema and Figure 4 shows an example of one of its instances.

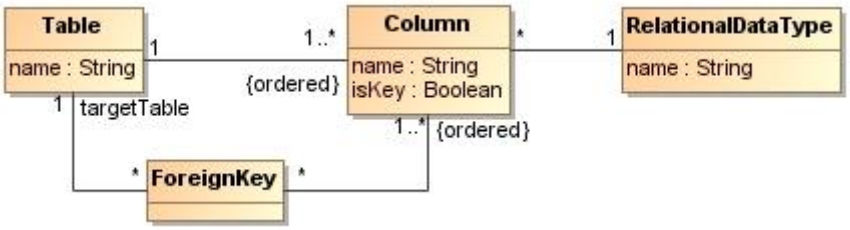

Fig. 3. Fragment of the Relational metaschema.

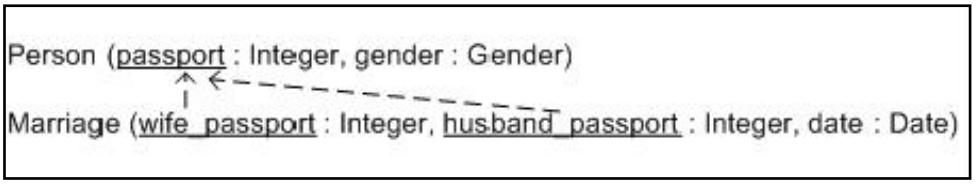

Fig. 4. Example of an instance of the fragment of the Relational metaschema.

The Relational metaschema also includes the following constraints that, formally in OCL, are specified as follows:

Names are defined, have a non-zero length, and consist of letters, digits, parenthesis and underscore:

```
context RelationalElement inv nameOk:
  let small:Set(String) =
    Set{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
    'r','s','t','u','v','w','x','y','z'} in
 let capital:Set(String) =
    Set{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
    'R','S','T','U','V','W','X','Y','Z'} in
 let digit:Set(String) =
  Set{'0','1','2','3','4','5','6','7','8','9'} in
  ifself.oclIsTypeOf(Table) then
    self.oclAsType(Table).name.isDefined and
    self.oclAsType(Table).name.size>0 and
    Set{1..self.oclAsType(Table).name.size}->forAll(i |Set{'_','(',')}->
       union(small)->union(capital)->union(digit)->
       includes(self.oclAsType(Table).name.substring(i,i)))
  else
```

```
     ifself.oclIsTypeOf(Column) then
       self.oclAsType(Column).name.isDefined and
       self.oclAsType(Column).name.size>0 and
       Set{1..self.oclAsType(Column).name.size}->forAll(i |
       Set{'_','(',')}->union(small)->union(capital)->union(digit)->
       includes(self.oclAsType(Column).name.substring(i,i)))
     else
     if self.oclIsTypeOf(RelationalDataType) then
       self.oclAsType(RelationalDataType).name.isDefined and
       self.oclAsType(RelationalDataType).name.size>0 and
       Set{1..self.oclAsType(RelationalDataType).name.size}->forAll(i |
          Set{'_'}->union(small)->union(capital)->union(digit)->
          includes(self.oclAsType(RelationalDataType).name.substring(i,i)))
     else true
     endif
    endif
  endif
```

Naming restriction: Different relational data types have different names:

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

Different Tables have different names:

```
context self:RelationalElement inv uniqueTableNames:
 RelationalElement.allInstances()->forAll(r1,r2 | (r1.oclIsTypeOf(Table) and
 r2.oclIsTypeOf(Table) and r1.oclAsType(Table).name =
 r2.oclAsType(Table).name) implies r1 = r2)
```

Within one Table, different Columns have different names:

```
context self:Table inv uniqueColNamesWithinTable:
 self.column->forAll(a1,a2 | a1.name = a2.name implies a1 = a2)
```

The set of key columns of a Table is not empty:

```
context self:Table inv TableKeyNotEmpty: self.key()->notEmpty()
```

All columns of a foreign key belong to the same table:

```
context ForeignKey inv allColumnsOfForeignKeyHaveSameTable:
 column.table->size() = 1
```

Note than the columns have an order within each table, which is necessary for a correct representation of foreing keys. However, since such order is only necessary for the key columns, the order, will not be taken into account for the columns which are not keys, in the rest of the paper. An alternative representation would have been to have two different associations between Table and Column, one for the column keys and another for the columns that are not keys.

2.1.1 Definition of Schema Units

Schema units are the knowledge components of the schemas. A schema consists of a set $S = \{u_1,...,u_n\}$ of schema units u_i, such that the knowledge expressed by S is the set of knowledge components expressed by its schema units $u_1,...,u_n$.

Syntactically, a schema unit u is a set of schema elements such that:

- they can be added to a schema S when some conditions are satisfied, and
- $S \cup \{u\}$ is a valid instance of *MS*.

The rationale behind this definition is that the knowledge expressed by a schema $S = \{u_1,...,u_i\}$ can be extended by a new schema unit u_{i+1} obtaining $S' = \{u_1,...,u_i,u_{i+1}\}$.

ER Schema Units

In an ER schema, the schema units are entity types, relationship types, attributes and data types. Figure 5 shows the definition of the root entity type, called *ErElement*. The representation and the schema elements of the schema units are defined as follows:

- Each entity type is represented by an instance of *EntityType*. The schema elements of an entity type named *e* are: (1) an instance α of *EntityType;* (2) the instance of attribute *name* of α with value *e*; (3) the (one or more) instances of *Attribute* related to α whose *isKey* attribute has the value *True*; (4) for each of these attributes: the instances of its attributes *name* and *isKey*, the instance of its relationship with α, and the instance of its relationship with the corresponding data type. If, for example, an entity type *e* has only one key attribute, then the schema elements of *e* is a set of seven instances. Note that we group an entity type and all its key attributes into a single schema unit.
- Each relationship type is represented by an instance of *RelationshipType*. The schema elements of a relationship type named *r* are: (1) an instance β of *RelationshipType*; (2) the instance of attribute *name* of β with value *r*; (3) the (two or more) instances of *RelationEnd* related to β; (4) for each of these relation ends: the instances of its attribute *name*, the instance of its relationship with β and the instance of its relationship with the corresponding entity type. Note that we do not take into account the cardinalities of the participants: we assume that they are unconstrained.
- Each data type is represented by an instance of *DataType*. The schema elements of a data type named *d* are: (1) an instance λ of *DataType*; and (2) the instance of attribute *name* of λ with value *d.*
- Each ER attribute that is not a key of an entity type is represented by an instance of A*ttribute* whose *isKey* attribute has the value *False*. The schema elements of an attribute named *a* are: (1) an instance μ of *Attribute*; (2) the instance of attribute *name* of μ with value *a* and *isKey* with value *False*, (3) the instance of its relationship with an instance of *EntityType* or *RelationshipType*, and (4) the instance of its relationship with the corresponding data type.

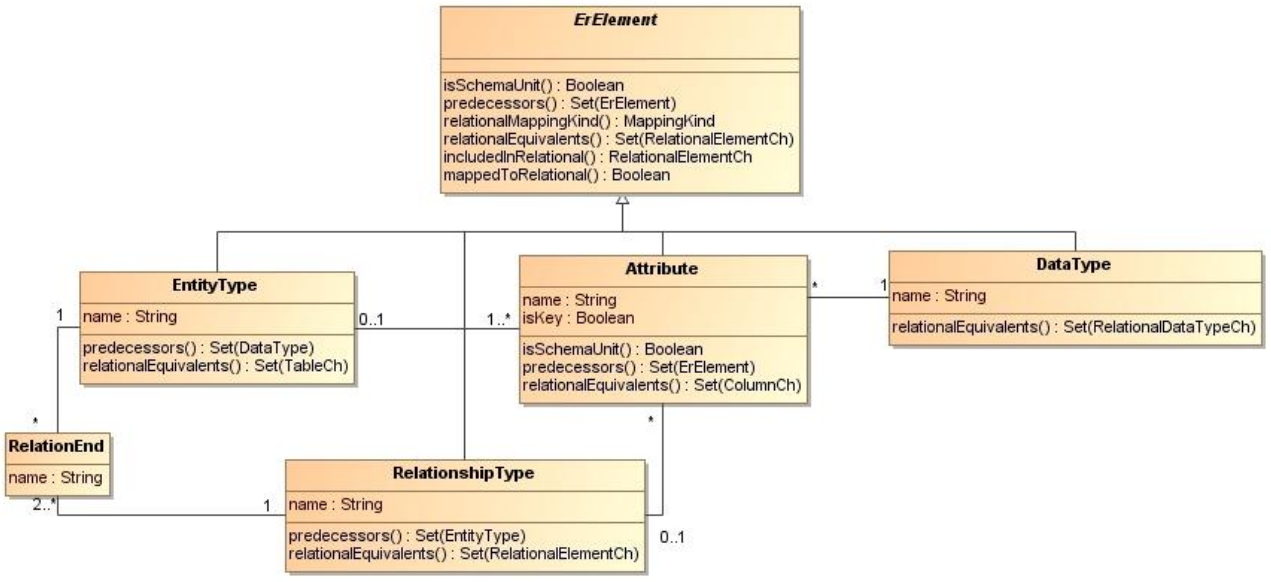

Fig. 5. Definition of ErElement.

In the ER schema example shown in Figure 2 there are seven schema units: three instances of *DataType*, one instance of *EntityType*, one instance of *RelationshipType*, and two instances of *Attribute*.

For the ErElement metaschema, the query operation *isSchemaUnit()* is defined, formally, as:

```
context ErElement::isSchemaUnit():Boolean
 body: True
```

meaning that by default all (direct or indirect) instances of *ErElement* are schema units. There is an exception: not all instances of *Attribute* are schema units, but only those that are not keys. Therefore, the above operation is redefined as follows:

```
context Attribute::isSchemaUnit():Boolean
 body: not isKey
```

Relational Schema Units

In the Relational metaschema of Fig. 3, the schema units of a relational schema are tables, columns, foreign keys and data types. The representation and the schema elements of the schema units are defined as follows:

- Each relational table is represented by an instance of *Table*. The schema elements of a table named *t* are: (1) an instance τ of *Table;* (2) the instance of attribute *name* of τ with value *t*; (3) the (one or more) instances of *Column* related to τ whose *isKey* attribute has the value *True*; (4) for each of these columns: the instances of its attributes *name* and *isKey*, its relationship with τ, and its relationship with the corresponding data type. Note that we group a table and all its key columns into a single schema unit.
- Each relational data type is represented by an instance of *RelationalDataType*. The schema elements of a data type named *d* are: (1) an instance λ of *RelationalDataType*; and (2) the instance of attribute *name* of λ with value *d.*
- Each relational column that is not a key of a table is represented by an instance of *Column* whose *isKey* attribute has the value *False*. The schema elements of a column named *c* are: (1) an instance μ of *Column*; (2) the instance of attribute *name* of μ with value *c* and *isKey* with value *False*; (3) its relationship with an instance of *Table*; and (4) its relationship with the corresponding instance of *RelationalDataType*.
- Each foreign key is represented by an instance of *ForeignKey*. The schema elements of a foreign key *fk* are: (1) an instance of *ForeignKey*; (2) the relationships of *fk* with *Column* that give the columns that comprise *fk*, and (3) the relationship of *fk* with the table referenced by the columns of *fk*.

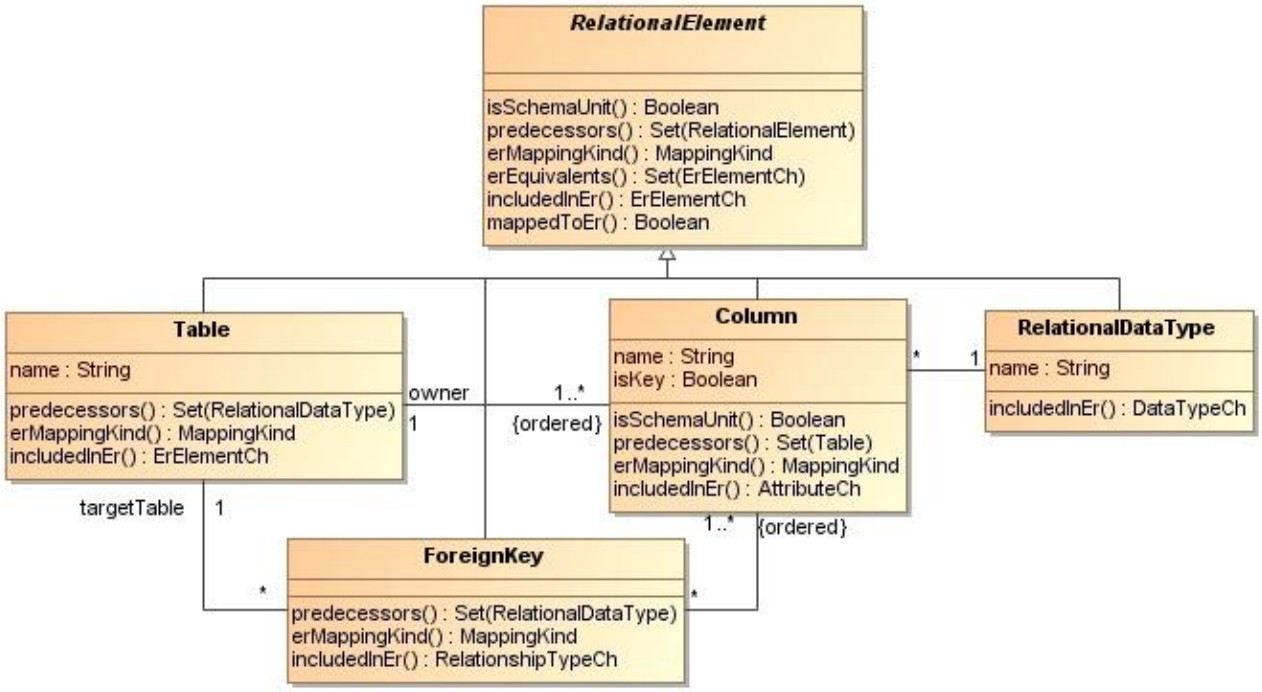

Fig. 6. Definition of RelationalElement.

In the relational example shown in Fig.4 there are nine schema units: three instances of *RelationalDataType*, two instances of *Table*, two instances of *Column*, and two instances of *ForeignKey*.

For the RelationalElement metaschema, the query operation *isSchemaUnit()* is defined, formally, as:

```
context RelationalElement::isSchemaUnit():Boolean
 body: True
```

meaning that by default all (direct or indirect) instances of *RelationalElement* are schema units. There is an exception similar to the previous one: not all instances of *Column* are schema units, but only those that are not keys. Therefore, the above operation is redefined as follows:

```
context Column::isSchemaUnit():Boolean
 body: not isKey
```

2.1.2 Predecessors

A schema consists of a set $S = \{u_1,...,u_n\}$ of schema units u_i, but in general there are precedence relationships between them. The *predecessor* units of a schema unit u_i are those schema units that are direct predecessors of u_i. A schema unit may not be a direct or indirect predecessor to itself.

ER Schema Units Predecessors

In the ER metaschema of Figure 5, the *predecessors* operation is specified as:

```
context ErElement::predecessors():Set(ErElement)
 pre: isSchemaUnit()
 body: Set{}
```

meaning that by default all schema units do not have predecessors. This is the case of *DataType* and therefore there is no need of redefining the operation for it. For *EntityType*:

```
context EntityType::predecessors():Set(DataType)
 body: attribute->select(isKey).dataType
```

meaning that the predecessors of an entity type are the data types of its key attributes. The predecessors of a relationship type are its participant entity types:

```
context RelationshipType::predecessors():Set(ErElement)
 body: relationEnd.entityType
```

Finally, the predecessors of a non-key attribute are its entity or relationship type and the data type:

```
context Attribute::predecessors():Set(ErElement)
 body: let type:ErElement =
         if entityType -> notEmpty() then entityType
         else relationshipType
         endif
       in Set{type,dataType}
```

Relational Schema Units Predecessors

In the Relational metaschema of Figure 6, the *predecessors* operation is specified as:

```
context RelationalElement::predecessors():Set(RelationalElement)
 pre: isSchemaUnit()
 body: Set{}
```

meaning that by default all schema units do not have predecessors. This is the case of *RelationalDataType* and therefore there is no need of redefining the operation for it. For *Table* we have:

```
context Table::predecessors():Set(RelationalDataType)
 body: column->select(isKey).relationalDataType
```

meaning that the predecessors of table are the relational data types of its key columns. The predecessor of a non-key column is its table and its relational data type:

```
context Column::predecessors():Set(Relationalelement)
 body: Set{table,relationalDataType}
```

Finally, the predecessors of a foreign key are its non-key columns and the source and target tables:

```
context ForeignKey::predecessors():Set(RelationalElement)
  body: column->select(not isKey)->asSet()->union(column.table->asSet())
   ->including(targetable)
```

Note that, we have not included the key-columns as predecessors because they are already included in the source table schema unit.

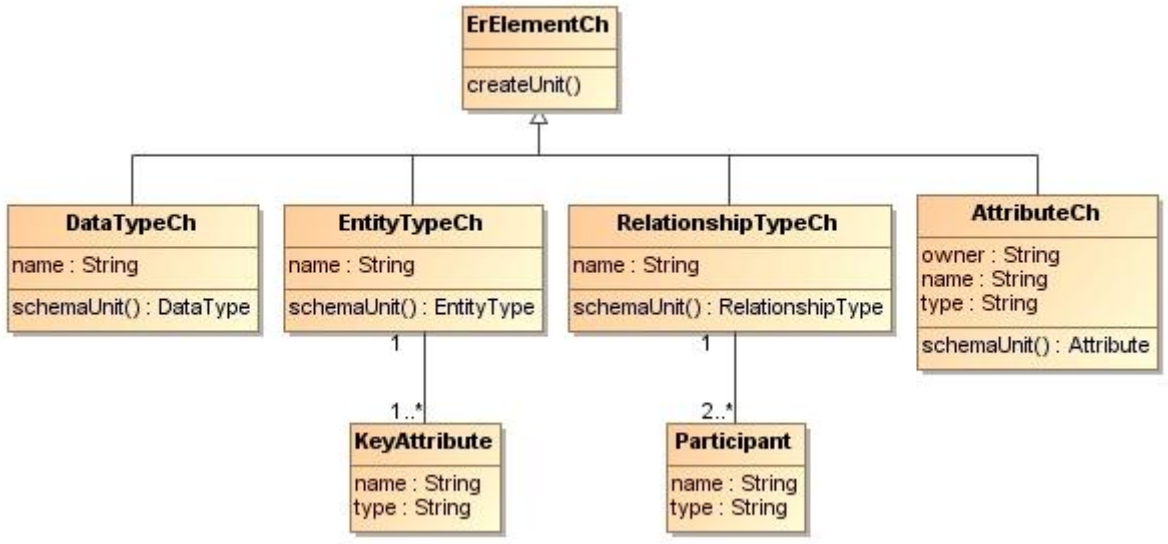

Fig. 7. Characterization Objects for the ER metaschema.

2.1.3 Characterization Objects

Each schema unit is characterized by an object (called *characterization* object). Characterization objects roughly correspond to value or domain value objects in the object-oriented design patterns field. In a metaschema, there is a characterization object type for each subtype *ST* of S_i*Element* such that some or all of its instances represent schema units. Each characterization object type includes a set of attributes that characterize the schema unit and two operations: *createUnit*() and *schemaUnit*(). The first operation creates a schema unit from its characterization object, and the second checks that the schema unit corresponding to the characterization object does indeed exist.

ER Metaschema Characterization Objects

The characterization object types of the ER metaschema are shown in Figure 7.

The specification of the *createUnit()* operation is the same for all characterization object types, and therefore is specified in the *ErElementCh* as:

```
context ErElementCh::createUnit()
 post: schemaUnit()
```

The *schemaUnit* operation is redefined in each subtype of ErElement. The simplest is *DataTypeCh*, for which the above mentioned operation could be specified as:

```
context DataTypeCh::schemaUnit():DataType
 body: DataType.allInstances() -> any(d:DataType| d.name = self.name)
```

The *schemaUnit* operation is a query that gives the schema a data type whose name is the one given in the attribute *name* of *DataTypeCh*.

EntityTypeCh requires an auxiliary object type (named *KeyAttribute*) to indicate the key attributes. We specify the operation as follows:

```
context EntityTypeCh::schemaUnit():EntityType
 body: EntityType.allInstances()->any(e:EntityType| e.name = self.name and
       self.keyAttribute->collect(k:KeyAttribute|
         Tuple{n:k.name,t:k.type})->asSet() =
         e.attribute->select(isKey)->collect(ka:Attribute|
           Tuple{n:ka.name,t:ka.dataType.name})->asSet())
```

RelationshipTypeCh also requires an auxiliary object type (named *Participant*) to indicate the participants. We specify the operation as follows:

```
context RelationshipTypeCh::schemaUnit():RelationshipType
 body: RelationshipType.allInstances()->any(r:RelationshipType|
       r.name = self.name and
       self.participant->collect(p:Participant|
         Tuple{n:p.name,t:p.type})->asSet() =
       r.relationEnd->collect(re:RelationEnd|
         Tuple{n:re.name,t:re.entityType.name})->asSet())
```

Finally, for *AttributeCh* the operation can be specified as:

```
context AttributeCh::schemaUnit():Attribute
 body: Attribute.allInstances()->any(a:Attribute|a.name = self.name and
      not a.isKey and a.dataType.name = self.type and
  (a.entityType.name = self.owner or a.relationshipType.name = owner))
```

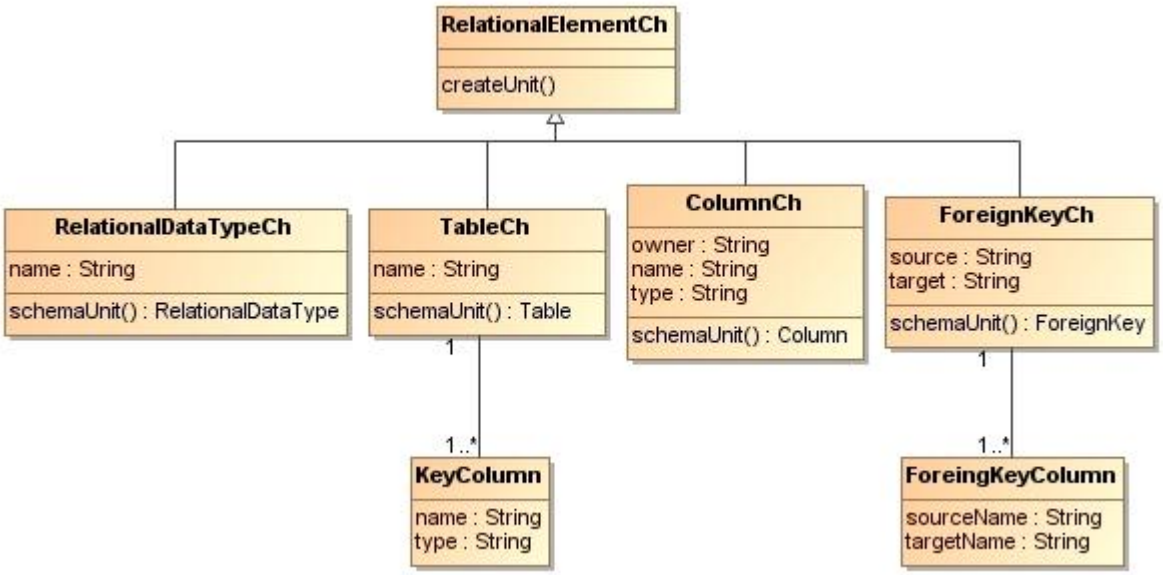

Fig. 8. Characterization Objects for the Relational metaschema

Relational Metaschema Characterization Objects

The characterization object types of the Relational metaschema are shown in Figure 8.

The specification of the *createUnit()* operation is the same for all characterization object types, and therefore is specified in the *RelationalElementCh* as:

```
context RelationalElementCh::createUnit()
post: schemaUnit()
```

The formal specification of the *schemaUnit* operation for those types is:

```
context RelationalDataTypeCh::schemaUnit():RelationalDataType
 body: RelationalDataType.allInstances()->any(d:RelationalDataType|d.name =
    self.name)

context TableCh::schemaUnit():Table
 body: Table.allInstances()->any(t:Table| t.name = self.name and
        self.keyColumn->forAll(kc:KeyColumn|
          t.column->select(isKey)->exists(c:Column|
            c.name = kc.name and c.relationalDataType.name = kc.type)) and
          t.column->select(isKey)->forAll(co:Column|
            self.keyColumn->exists(kco:KeyColumn|
              co.name = kco.name and
              co.relationalDataType.name = kco.type)))

context ColumnCh::schemaUnit():Column
  body: Column.allInstances()->any(c:Column|
          c.name = self.name and not c.isKey and
          c.relationalDataType.name = self.type and
          c.table.name = self.owner)

context ForeignKeyCh::schemaUnit():ForeignKey
 body: ForeignKey.allInstances()->any(f:ForeignKey|
          f.column->any(true).table.name = self.source and
```

```
    f.targetTable.name = self.target and
  f.column->any(true).table.name = self.source and
    f.targetTable.name = self.target and
    f.column->forAll(co:Column|
    self.foreignKeyColumn->exists(fkc|
         co.name = fkc.sourceName and
         f.targetTable.column->select(isKey)->collect(name)->
           includes(fkc.targetName))))
```

2.2 Translation mapping expressions

This section describes the operations needed to specify the translation mapping constraints between ER metaschemas and Relational metaschemas.

2.2.1 s_jMappingKind

The query operation *s_jMappingKind*():*MappingKind* indicates how a schema unit of S_i is translated into S_j. The value of this operation is mapping-dependent. *MappingKind* is an enumeration data type whose values are:

- *HasEquivalents.* A schema unit of S_i has this mapping kind when it is completely equivalent to a set $\{u_{j,1},...,u_{j,k}\}$ of one or more schema units of S_j. The mapping kind of $u_{j,1},...,u_{j,k}$ must be *IsIncluded*.
- *IsIncluded.* A schema unit of S_i has this mapping kind when it is included in a schema unit $u_{j,k}$ of S_j. The mapping kind of $u_{j,k}$ must be *HasEquivalents*.
- *Untranslatable.* A schema unit of S_i has this mapping kind when it cannot be translated into S_j. If a schema S_i contains one or more untranslatable schema units then its translation into S_j can only be partial.

In the context of *S_iElement* the operation *s_jMappingKind*() can only give a default value, and each subtype *ST* of *S_iElement* such that some or all of its instances represent schema units, redefines it (if needed) to give the correct value. The value of the operation for the instances of *ST* that are not a schema unit is undefined. This is enforced by means of the *mapping kind definition constraint*, which is specified as:

```
context ErElement::relationalMappingKind():MappingKind
 body: ifisSchemaUnit() then MappingKind::HasEquivalents else Set{} endif

context RelationalElement::erMappingKind():MappingKind
 body: if isSchemaUnit() then MappingKind::IsIncluded else Set{} endif
```

2.2.2 s_jEquivalents and IncludedIns_j

The evaluation of the *s_jEquivalents*() operation on a schema unit of S_i whose mapping kind is *HasEquivalents* gives the set of characterization objects of the schema units of S_j that are equivalent to it.

RelationalEquivalents of ErElements

The signature of the precondition and postconditions of *relationalEquivalents()* on an ErElement is defined as follows:

```
context ErElement::relationalEquivalents():Set(RelationalElementCh)
 pre hasEquivalents: relationalMappingKind() = MappingKind::HasEquivalents
 post atLeastOneCharacterizationObjectCreate:
        (RelationalElementCh.allInstances –
          RelationalElementCh.allInstances@pre)->notEmpty()
 post definingTheResult:
```

```
result = RelationalElementCh.allInstances –
         RelationalElementCh.allInstances@pre
```

The effect of the operation, defined declaratively by postconditions in its subtypes, is specified as:

A data type maps to a characterization object of a relational data type with the same name:

```
context DataType::relationalEquivalents():Set(RelationalElementCh)
post RelationalDataTypeChCreated:
      (RelationalDataTypeCh.allInstances –
        RelationalDataTypeCh.allInstances@pre)->
          select(rdt:RelationalDataTypeCh| rdt.oclIsNew() and
            rdt.name = self.name)->size() = 1
```

An entity type maps to a characterization object of a table with the same name. For each attribute that is key, there is an instance of *KeyColumn* with the attributes *name* and *type* the value of which is the name of the attribute and the name of the data type of the attribute, respectively:

```
context EntityType::relationalEquivalents():Set(RelationalElementCh)
post TableChCreated:
      (TableCh.allInstances - TableCh.allInstances@pre)->
        select(t:TableCh| t.oclIsNew() and t.name = self.name and
        self.attribute->select(isKey)->forAll(att|
          (KeyColumn.allInstances - KeyColumn.allInstances@pre)->
            select(kc:KeyColumn| kc.oclIsNew() and kc.name = att.name and
              kc.type = att.dataType.name and kc.tableCh = t)->
              size() = 1))->size() = 1
```

A relationship type maps to a characterization object of a table with the same name. For each key attribute of each relation end, there is an instance of *KeyColumn* with the attribute *name* whose value is the concatenation of the name of the relation end and the name of the key attribute of the entity type of the relation end. The value of the attribute *type* of the KeyColumn is the name of the data type of the attribute the relation end. The relational type also maps, for each relation end, to a characterization object of foreign key whose *source* attribute is the name of the relationship type; the *target* is the name of the entity type of the relationend; and for each key attribute of the entity type of the relation end, there is a *ForeignKeyColumn* whose *sourceName* attribute is the name of the relation end concatenated to the name of the key attribute of the entity type of the relation end:

```
context RelationalType::relationalEquivalents():Set(RelationalElementCh)
 post TableChAndForeignKeyChCreated:
      (TableCh.allInstances - TableCh.allInstances@pre)->select(t:TableCh|
        t.oclIsNew() and t.name = self.name and
        self.relationEnd->forAll(re|
          re.entityType.attribute->select(isKey)->forAll(attkey|
            (KeyColumn.allInstances - KeyColumn.allInstances@pre)->
            select(kc:KeyColumn| kc.oclIsNew() and
              kc.name = re.name.concat('_').concat(attkey.name) and
              kc.type = attkey.dataType.name and
              kc.tableCh = t)->size() = 1)) and
        self.relationEnd->forAll(rel|
          (ForeignKeyCh.allInstances - ForeignKeyCh.allInstances@pre)->
          select(fk:ForeignKeyCh|fk.oclIsNew() and fk.source = t.name and
            fk.target = rel.entityType.name and rel.entityType.attribute
            ->select(isKey)->forAll(attk:Attribute|
              (ForeignKeyColumn.allInstances()-
                ForeignKeyColumn.allInstances@pre)->
                select(fkc:ForeignKeyColumn| fkc.oclIsNew() and
                fkc.sourceName = rel.name.concat('_').concat(attk.name)
```

```
                    and fkc.targetName = attk.name and
                    fkc.foreignKeyCh = fk) ->size() = 1))->size() =1))->
                      size() = 1
```

An attribute maps to a characterization object of a column with the same name. The type attribute has the value the name of the data type of the attribute:

```
 context Attribute::relationalEquivalents():Set(RelationalElementCh)
  post ColumnChCreated:
       (ColumnCh.allInstances – ColumnCh.allInstances@pre)->
         select(c:ColumnCh| c.oclIsNew() and c.oclIsTypeOf(ColumnCh) and
         c.name = self.name and c.type = self.dataType.name and
           c.owner = if self.entityType->notEmpty
                 then self.entityType.name
                 else self.relationshipType.name
                 endif)->size() = 1
```

The method of the relationalEquivalenst() has been specified in OCL executable as:

```
procedure relationalEquivalentsOfERSchema()
 var  dch:RelationalDataTypeCh, tch:TableCh, kc:KeyColumn, cch:ColumnCh,
      fch:ForeignKeyCh, fkc:ForeignKeyColumn;
 begin
 for e:ErElement in [ErElement.allInstances()->select(isSchemaUnit() and
  relationalEquivalents_m->isEmpty)->asSequence]
 begin

   if [e.oclIsTypeOf(DataType)] then
   begin
     dch := Create( RelationalDataTypeCh );
     [dch].name := [e.oclAsType(DataType).name];
     Insert( RelationalEquivalents, [e], [dch] );
   end;

   if [e.oclIsTypeOf(EntityType)] then
   begin
    tch := Create( TableCh );
    Insert( RelationalEquivalents, [e], [tch] );
    [tch].name := [e.oclAsType(EntityType).name];
    for a:Attribute in [e.oclAsType(EntityType).attribute->select(isKey)->
     asSequence]
   begin
     kc := Create (KeyColumn);
     Insert (TableCh_KeyColumn, [tch], [kc]);
     [kc].name := [a.name];
     [kc].type := [a.dataType.name];
   end;
 end;

 if [e.oclIsTypeOf(Attribute)] then
   begin
     cch := Create( ColumnCh );
     Insert( RelationalEquivalents, [e], [cch] );
     [cch].name := [e.oclAsType(Attribute).name];
     [cch].type := [e.oclAsType(Attribute).dataType.name];
     [cch].owner := [if e.oclAsType(Attribute).entityType->notEmpty then
       e.oclAsType(Attribute).entityType.name else
       e.oclAsType(Attribute).relationshipType.name endif];
   end;

 if [e.oclIsTypeOf(RelationshipType)] then
```

```
  begin
    tch := Create( TableCh );
    Insert( RelationalEquivalents, [e], [tch] );
    [tch].name := [e.oclAsType(RelationshipType).name];
    for re:RelationEnd in [e.oclAsType(RelationshipType).relationEnd->
      asSequence]
     begin
      fch := Create( ForeignKeyCh );
      Insert( RelationalEquivalents, [e], [fch] );
      [fch].source := [e.oclAsType(RelationshipType).name];
      [fch].target := [re.entityType.name];
      for a:Attribute in [re.entityType.attribute->select(isKey)->
            asSequence]
        begin
          kc := Create (KeyColumn);
          Insert (TableCh_KeyColumn, [tch], [kc]);
          [kc].name := [re.name.concat('_').concat(a.name)];
          [kc].type := [a.dataType.name];
          fkc := Create(ForeignKeyColumn);
          [fkc].sourceName := [re.name.concat('_').concat(a.name)];
          [fkc].targetName := [a.name];
          Insert (ForeignKeyCh_ForeignKeyColumn, [fch], [fkc]);
        end;
      end;
    end;
  end;
end;
```

Note that, in the method, relationalEquivalents_m is the materialization of the relationalEquivalents() operation.

IncludedInEr of RelationalElements

The evaluation of the *IncludedIns*$_j$() operation on a schema unit of S_i whose mapping kind is *IsIncluded* gives the characterization object of the schema units of S_j that includes itself in the mapping.

The signature of the precondition and postconditions of *includedInEr()* on a RelationalElement is defined as follows:

```
context RelationalElement::includedInEr():ErElementCh
  pre: erMappingKind() = #IsIncluded
  post onlyOneCharacterizationObjectCreated:
       (ErElementCh.allInstances – ErElementCh.allInstances@pre)->size()= 1
  post definingTheResult:
       result = (ErElementCh.allInstances – ErElementCh.allInstances@pre)
                -> any(true)
```

The effect of the operation, defined declaratively by postconditions in its subtypes, is specified as:

A relational data type is mapped to a characterization object of data type whose name attribute is the same as the relational data type:

```
context RelationalDataType::includedInEr():ErElementCh
 post DataTypeChCreated:
       (DataTypeCh.allInstances - DataTypeCh.allInstances@pre)->
         select(d:DataTypeCh| d.oclIsNew() and d.name = self.name)->
                size() = 1
```

A table which its key columns do not participate in any foreign key is mapped to a characterization object of an entity type whose *name* attribute is the same as the table. For each key column there is a key attribute whose *name* and *type* are the equivalents to said key column. A table which its key columns participate in a foreign key is mapped to a characterization object of a relationship type whose *name* attribute is the name as the table. For each key column there is a participant whose *name* attribute is the substring of the column previous to the '_' and the *type* attribute is the name of the relational data type of the column:

```
context Table::includedInEr():ErElementCh
 post EntityTypeChXorRelationshipTypeChCreated:
       if self.column->select(isKey)->forAll(c| c.foreignKey->isEmpty)
       then
          (EntityTypeCh.allInstances() - EntityTypeCh.allInstances()@pre)->
          select(e:EntityTypeCh| e.oclIsNew() and e.name = self.name and
               self.column->select(isKey)->forAll(co:Column|
                 (KeyAttribute.allInstances() -
               KeyAttribute.allInstances()@pre) ->
                  select(ka:KeyAttribute| ka.name = co.name and
                    ka.type = co.relationalDataType.name and
                    ka.entityTypeCh = e)->size()= 1))->size()= 1
       else
          (RelationshipTypeCh.allInstances -
            RelationshipTypeCh.allInstances@pre)->
            select(r:RelationshipTypeCh| r.oclIsNew() and
              r.name = self.name and self.column->select(isKey)->
              forAll(co:Column| (Participant.allInstances() -
                Participant.allInstances@pre)->select(p:Participant|
                p.oclIsNew() and
                p.name = co.name.substring(1,Set{1..co.name.size}->
                select(pos:Integer| co.name.substring(1,pos+1) =
                co.name.substring(1,pos).concat('_'))->any(true)) and
                p.type = co.relationalDataType.name)->size() = 1))
                ->size() = 1
       endif
```

A column is mapped to a characterization object of attribute whose *name* and *type* attributes correspond to the name of the column and the name of the relational data type of the column, respectively:

```
context Column::includedInEr():ErElementCh
 post AttributeChCreated:
      (AttributeCh.allInstances - AttributeCh.allInstances@pre)->
      select(a:AttributeCh| a.oclIsNew() and a.name = self.name and
        a.type = self.relationalDataType.name and
        a.owner = self.table.name)->size() = 1
```

A foreign key is mapped to a characterization object of relationship type whose *name* is the name of the table of the columns that form the foreign key. For each key column of said table, there is a participant whose *name* is the substring of the name of the key column after the '_':

```
context ForeignKey::includedInEr():ErElementCh
 post RelationshipChCreated:
      (RelationshipTypeCh.allInstances -
           RelationshipTypeCh.allInstances@pre)->select(r:RelationshipTypeCh|
             r.oclIsNew() and r.name = self.column.table.name->asSet
             ->any(true) and self.column.table->asSet->any(true).column->
               select(isKey)->forAll(co:Column|
                 (Participant.allInstances()-Participant.allInstances@pre)->
                 select(p:Participant| p.oclIsNew() and
                   p.name = co.name.substring(1,Set{1..co.name.size}
                   ->select(pos:Integer| co.name.substring(1,pos+1) =
                     co.name.substring(1,pos).concat('_'))
```

```
                        ->any(true)) and
                        p.type = co.relationalDataType.name)->
                        size() = 1))->size() = 1
```

The method of the includedInEr() has been specified in OCL executable as:

```
procedure includedInEROfRelationalSchema()
 var  dch:DataTypeCh, ech:EntityTypeCh, ka:KeyAttribute,
      rch:RelationshipTypeCh,p:Participant, ach:AttributeCh,
      rch2:RelationshipTypeCh;

begin
 for r:RelationalElement in [RelationalElement.allInstances()->
   select(isSchemaUnit() and includedInEr_m->isEmpty)->asSequence]
   begin
    if [r.oclIsTypeOf(RelationalDataType)] then
    begin
      dch := Create( DataTypeCh );
      Insert( IncludedInEr, [r], [dch] );
      [dch].name := [r.oclAsType(RelationalDataType).name];
   end;

   if [r.oclIsTypeOf(Table)] then
    begin
      if [r.oclAsType(Table).column->select(isKey)->forAll(c:Column|
           c.foreignKey->isEmpty)] then
      begin
        ech := Create( EntityTypeCh );
        Insert( IncludedInEr, [r], [ech] );
        [ech].name := [r.oclAsType(Table).name];
        for c:Column in [r.oclAsType(Table).column->select(isKey)]
        begin
         ka := Create (KeyAttribute);
         Insert (EntityTypeCh_KeyAttribute, [ech], [ka]);
         [ka].name := [c.name];
         [ka].type := [c.relationalDataType.name];
        end;
      end;
      if [r.oclAsType(Table).column->select(isKey)->
        exists(c:Column|c.foreignKey->notEmpty)] then
       begin
        rch := Create (RelationshipTypeCh);
        Insert( IncludedInEr, [r], [rch] );
        [rch].name := [r.oclAsType(Table).name];
        for co:Column in [r.oclAsType(Table).column->select(isKey)]
        begin
          p := Create (Participant);
          Insert (RelationshipTypeCh_Participant, [rch], [p]);
          [p].type := [co.foreignKey.targetTable->any(true).name];
          for i:Integer in [Sequence{1..co.name.size}]
          begin
              if [co.name.substring(1,i).concat('_')
                     = co.name.substring(1,i+1)] then
               begin [p].name := [co.name.substring(1,i)]; end;
          end;
        end;
       end;
   end;

   if [r.oclIsTypeOf(Column)] then
```

```
      begin
        ach := Create( AttributeCh );
        Insert( IncludedInEr, [r], [ach] );
        [ach].name := [r.oclAsType(Column).name];
        [ach].type := [r.oclAsType(Column).relationalDataType.name];
        [ach].owner := [r.oclAsType(Column).table.name];
      end;

      if [r.oclIsTypeOf(ForeignKey)] then
      begin
        rch := Create (RelationshipTypeCh);
        Insert( IncludedInEr, [r], [rch] );
        [rch].name := [r.oclAsType(ForeignKey).column->any(true).table.name];
        for co:Column in [r.oclAsType(ForeignKey).column->
          any(true).table.column->select(isKey)]
         begin
          p := Create (Participant);
          Insert (RelationshipTypeCh_Participant, [rch], [p]);
          [p].type := [r.oclAsType(ForeignKey).targetTable.name];
          for i:Integer in [Sequence{1..co.name.size}]
          begin
            if [co.name.substring(1,i).concat('_') = co.name.substring(1,i+1)]
            then
            begin [p].name := [co.name.substring(1,i)]; end;
          end;
         end;
      end;
    end;
  end;
```

2.2.3 MappedToS$_j$

A schema unit is translated correctly if the results of the previous operations are consistent. This is defined in OCL in two operations as follows:

```
    context ErElement::mappedToRelational():Boolean
     body: if relationalMappingKind() = MappingKind::HasEquivalents
          then
            self.relationalEquivalents()->forAll(re:RelationalElementCh|
              re.schemaUnit()->notEmpty() and
              re.schemaUnit().umlMappingKind() = MappingKind::IsIncluded and
              re.schemaUnit().includedInUml().schemaUnit() = self)
          else
            if relationalMappingKind() = MappingKind::IsIncluded
            then
              self.includedInRelational().schemaUnit()->notEmpty() and
              self.includedInRelational().schemaUnit().erMappingKind() =
              MappingKind::HasEquivalents and
              self.includedInRelational().schemaUnit().erEquivalents().
                schemaUnit()->includes(self)
            else
              false
            endif
          endif

    context RelationalElement::mappedToEr():Boolean
     body: if erMappingKind() = MappingKind::HasEquivalents
          then
            self.erEquivalents()->forAll(er:ErElementCh|
```

```
      er.schemaUnit()->notEmpty()and
      er.schemaUnit().relationalMappingKind() = MappingKind::IsIncluded
      and er.schemaUnit().includedInRelational().schemaUnit() = self)
    else
      if erMappingKind() = MappingKind::IsIncluded
      then
        self.includedInEr().schemaUnit()->notEmpty() and
        self.includedInEr().schemaUnit().relationalMappingKind() =
          MappingKind::HasEquivalents and
        self.includedInEr().schemaUnit().relationalEquivalents().
          schemaUnit()->includes(self)
      else
        false
      endif
    endif
```

2.2.4 Translation mapping constraints

Let $M = (MS_1, MS_2, \Sigma)$ be a translation mapping where MS_1 and MS_2 are instances of MOF. The translation mapping constraints Σ consists of exactly two constraints that are defined formally by the following OCL invariants:

```
context ErElement inv completeMappingToRelational:
 isSchemaUnit() and
 (relationalMappingKind() = MappingKind::HasEquivalents or
   relationalMappingKind() = MappindgKind::IsIncluded)
 implies mappedToRelational()

context RelationalElement inv completeMappingToEr:
 isSchemaUnit() and
 (erMappingKind() = MappingKind::HasEquivalents or
   erMappingKind() = MappingKind::IsIncluded)
 implies mappedToEr()
```

2.2.5 Validating the model

In order to validate the model, the USE tool allows to validate pre- and postconditions by simulating operation calls.

In the following, we show an example consisting on a sequence of steps that simulates different states of the model and the expected success or failure of constraints. It creates the instances of ER metaschema and Relational metaschema shown in Figures 2 and 4.

The following scripts, simulates the creation of three instances of primitive data type (Integer, Date and Gender):

```
!create IntegerEr:DataType
!set IntegerEr.name:= 'Integer'
!create DateEr:DataType
!set DateEr.name:= 'Date'
!create GenderEr:DataType
!set GenderEr.name:= 'Gender'
```

The simulation of execution of the two operations calls *relationalEquivalents()* and *includedInEr()* explained above and the validation of the complete mapping is the following:

```
gen start Er2RelationalEquivalents.assl relationalEquivalents()
gen start Er2RelationalIncludedIn.assl includedInEr()
gen result accept
gen load C:\...\CompleteMapping.invs
```

```
check -d RelationalElement::completeMappingToEr
check -d ErElement::completeMappingToRelational
```

The screenshot below (Figure 9) shows the object diagram created and the result of the evaluation of both invariants. Three instances of *RelationalDataTypeCh* have been created. The first invariant *RelationalElement::completeMappingToEr* has succeded since there are no instances on the relational schema; and the *ErModelElement::completeMappingToRelational* has failed since the three instances have no equivalent elements in the Relational schema.

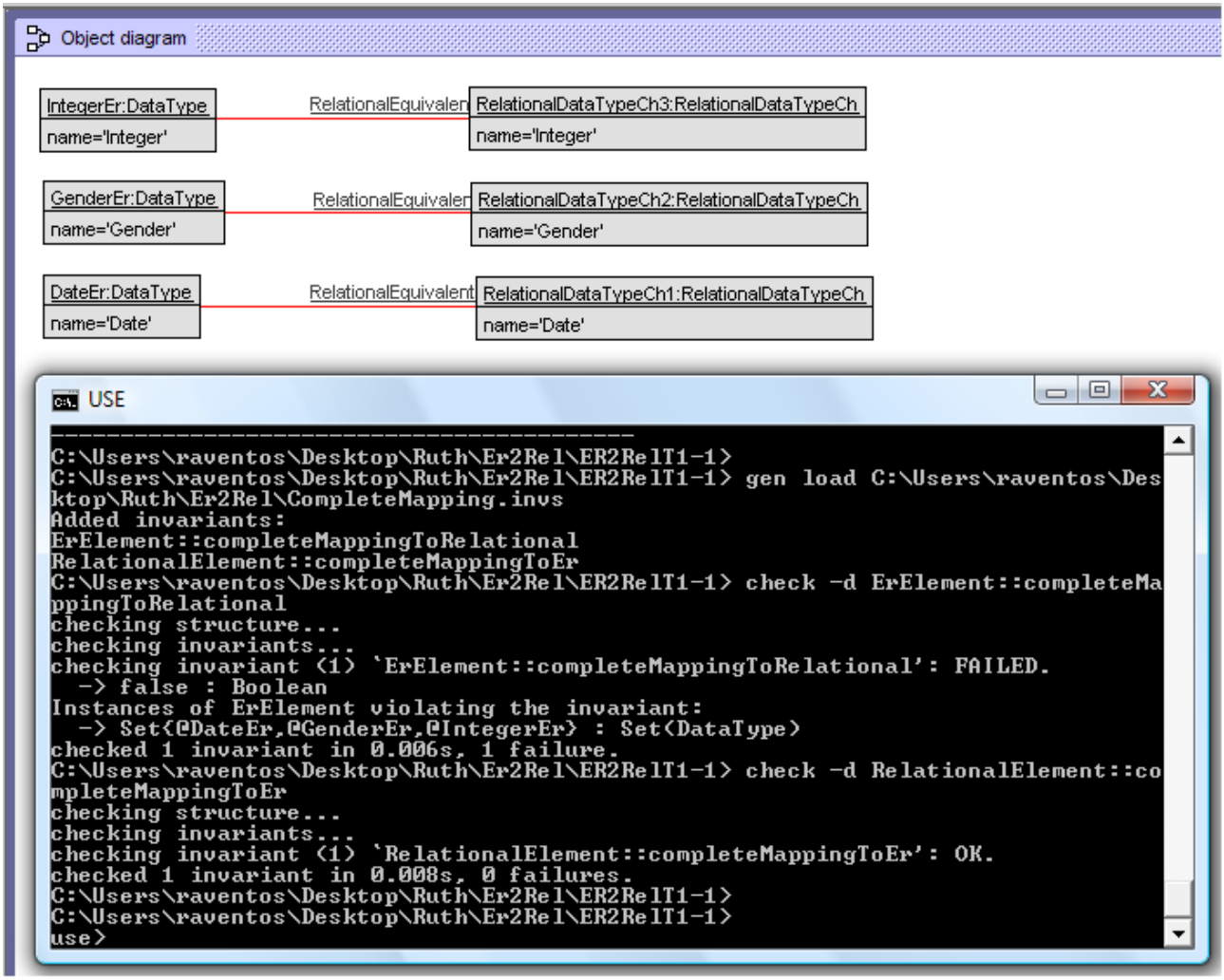

Fig. 9. Screenshot of the example (1)

After the simulation of creation of three instances of relational data type (Integer, Date and Gender):

```
!create IntegerRel:RelationalDataType
!set IntegerRel.name:= 'Integer'
!create DateRel:DataType
!set DateRel.name:= 'Date'
!create GenderRel:DataType
!set GenderRel.name:= 'Gender'
```

The screenshot below (Figure 10) shows the result after the execution of the two operations calls *relationalEquivalents()* and *includedInEr()* and the validation of the complete mapping invariants. Now, both invariants have succeeded.

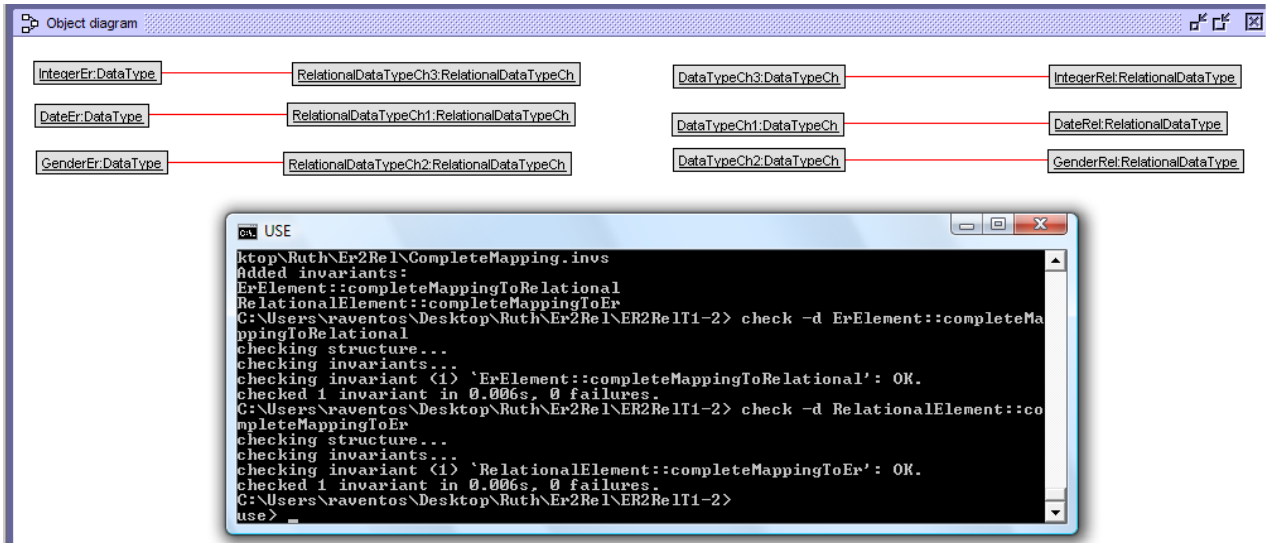

Fig. 10. Screenshot of the example (2)

Now, we simulate the creation of an instance of entity type with its key column (Person with passport):

```
!create PersonEr:EntityType
 !set PersonEr.name:= 'Person'
 !create passportEr:Attribute
 !set passportEr.name:= 'passport'
 !insert (passportEr,IntegerEr) into AttributeTyping
 !set passportEr.isKey:= true
 !insert (PersonEr,passportEr) into EntityType_Attribute
```

The screenshot below (Figure 11) shows the result and now, the *ERElement::completeMappingToRelational* invariant have failed since there is no table in the Relational schema equivalent to the entity type created.

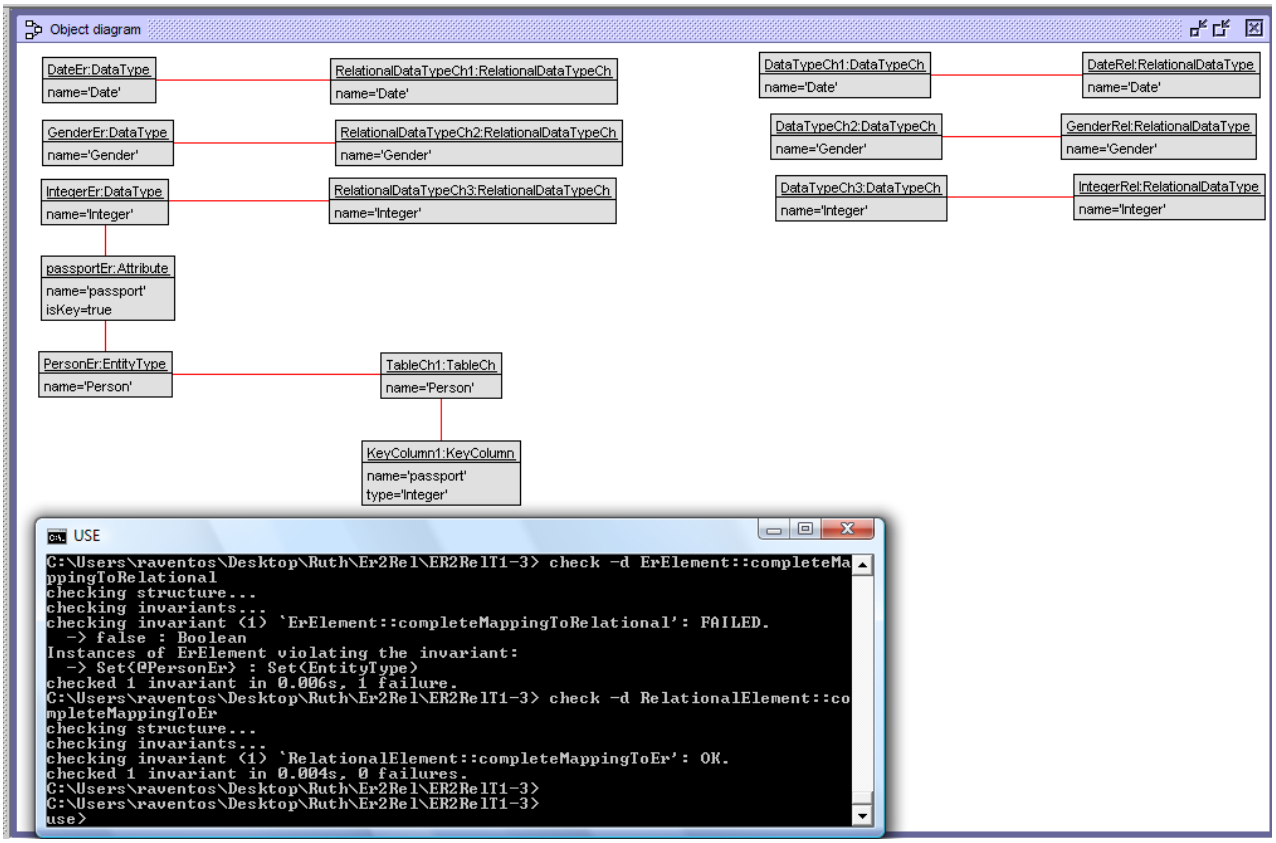

Fig. 11. Screenshot of the example (3)

Now, we simulate the creation of the instance of table equivalent to the entity type Person:

```
!create PersonRel:Table
!set PersonRel.name:= 'Person'
!create passportRel:Column
!set passportRel.name:= 'passport'
!set passportRel.isKey:= true
!insert (PersonRel,passportRel) into Table_Column
!insert (passportRel,IntegerRel) into ColumnTyping
```

The screenshot below (Figure 12) shows that the evaluation of both complete mapping invariants has succeded.

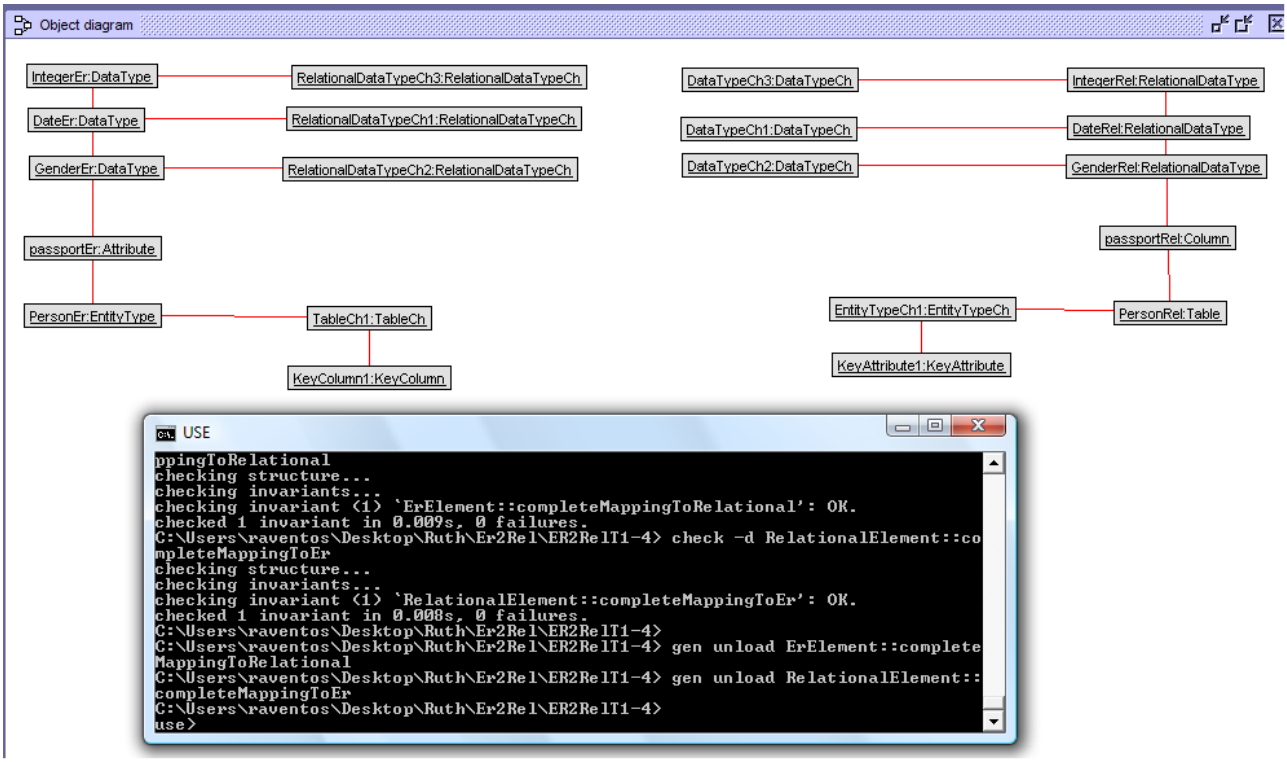

Fig. 12. Screenshot of the example (4)

Now, we simulate the creation of the instance of the attribute and column *gender* and the *marriage* relationship type with its relation ends and the date attribute:

```
!create genderEr:Attribute
!set genderEr.name:= 'gender'
!set genderEr.isKey:= false
!insert (PersonEr,genderEr) into EntityType_Attribute
!insert (genderEr,GenderEr) into AttributeTyping

!create genderRel:Column
!set genderRel.name:= 'gender'
!set genderRel.isKey:= false
!insert (PersonRel,genderRel) into Table_Column
!insert (genderRel,GenderRel) into ColumnTyping

!create MarriageEr:RelationshipType
!set MarriageEr.name:= 'Marriage'
!create husbandEr:RelationEnd
!set husbandEr.name:= 'husband'
!create wifeEr:RelationEnd
```

```
!set wifeEr.name:= 'wife'
!create dateEr:Attribute
!set dateEr.name:= 'date'
!set dateEr.isKey:= false
!insert (husbandEr,PersonEr) into RelationEndTyping
!insert (wifeEr,PersonEr) into RelationEndTyping
!insert (MarriageEr,husbandEr) into RelationshipType_RelationEnd
!insert (MarriageEr,wifeEr) into RelationshipType_RelationEnd
!insert (MarriageEr,dateEr) into RelationshipType_Attribute
!insert (dateEr,DateEr) into AttributeTyping
```

The screenshot below (Figure 13) shows that the evaluation of the completeMappingToRelationa has failed because there are no equivalents elements in the Relational metaschema of the relationship type and of its attribute.

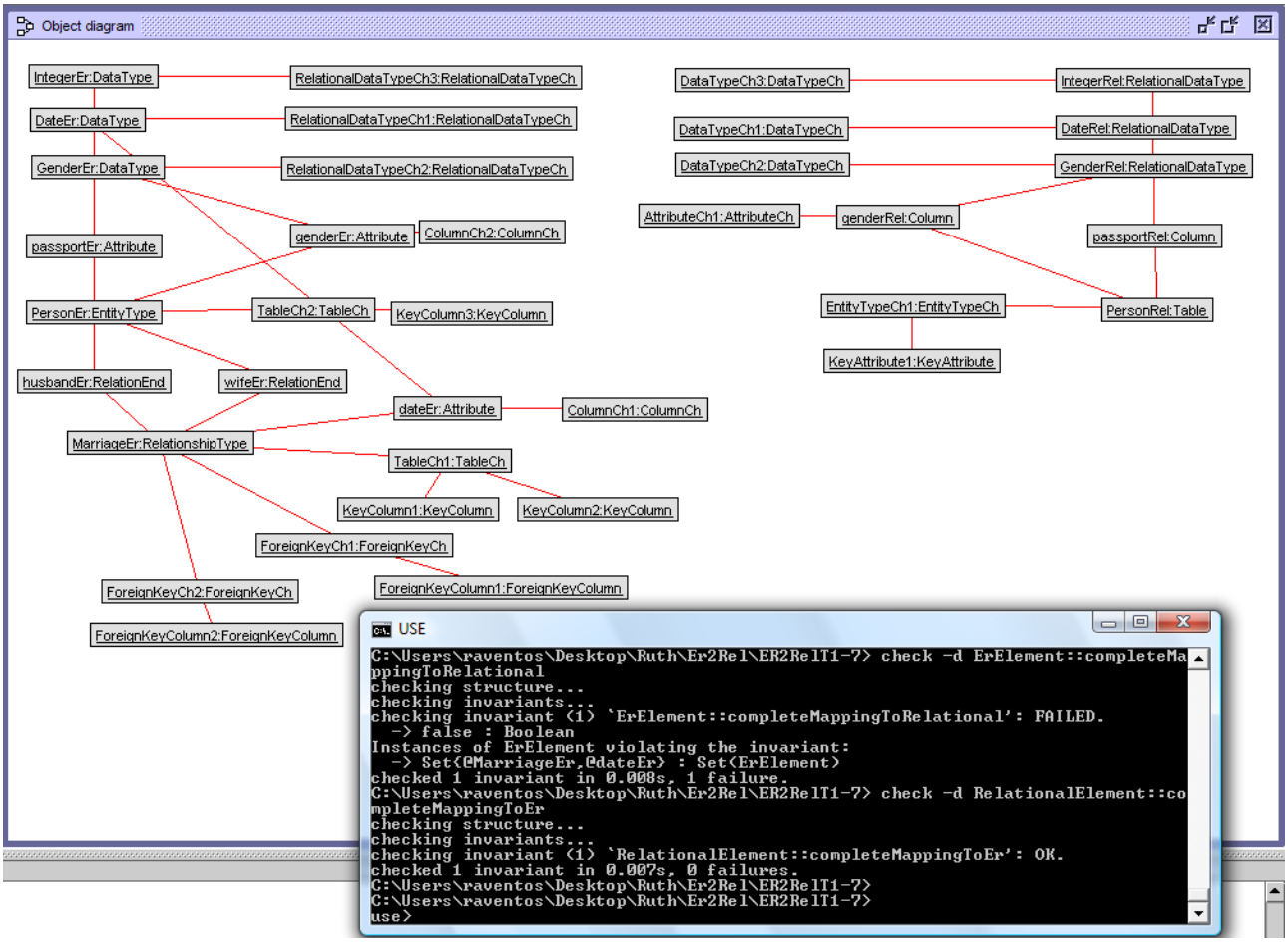

Fig. 13. Screenshot of the example (5)

Finally, after the simulation of the instances of the table and foreign keys of marriage with its column *date*:

```
!create MarriageRel:Table
!set MarriageRel.name:= 'Marriage'
!create husbandRel:Column
!set husbandRel.name:= 'husband_passport'
!set husbandRel.isKey:= true
!create wifeRel:Column
!set wifeRel.name:= 'wife_passport'
!set wifeRel.isKey:= true
!create dateRel:Column
!set dateRel.name:= 'date'
!set dateRel.isKey:= false
```

```
!insert (husbandRel,IntegerRel) into ColumnTyping
!insert (wifeRel,IntegerRel) into ColumnTyping
!insert (MarriageRel,husbandRel) into Table_Column
!insert (MarriageRel,wifeRel) into Table_Column
!insert (MarriageRel,dateRel) into Table_Column
!insert (dateRel,DateRel) into ColumnTyping
!create ForeignKey1 : ForeignKey
!create ForeignKey2 : ForeignKey
!insert (PersonRel,ForeignKey1) into Table_ForeignKey
!insert (PersonRel,ForeignKey2) into Table_ForeignKey
!insert (ForeignKey1,husbandRel) into ForeignKey_Column
!insert (ForeignKey2,wifeRel) into ForeignKey_Column
```

The evaluation of both invariants has succeeded as shown in Figure 14.

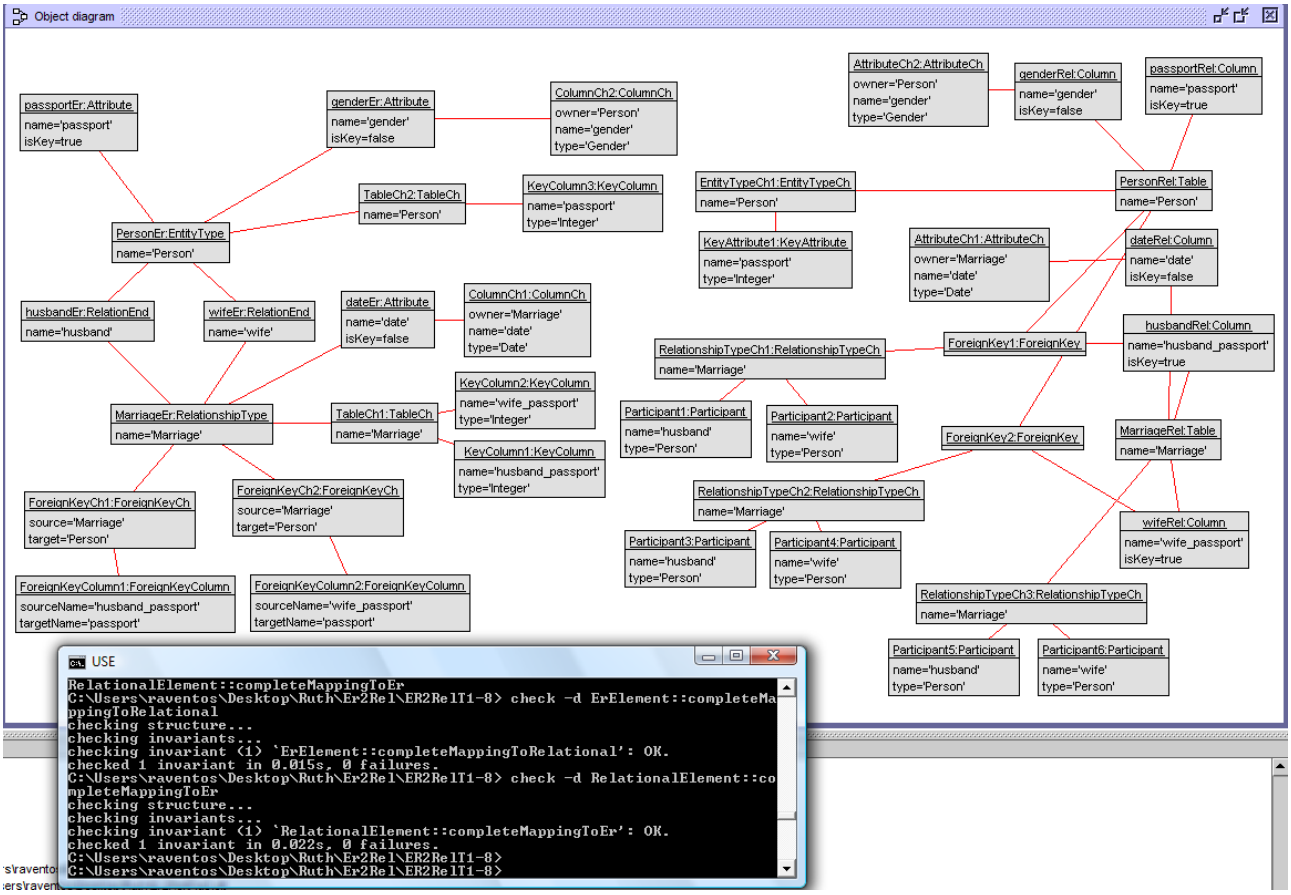

Fig. 14. Screenshot of the example (6)

2.3 Translating schemas

The operations defined in the previous sections may be use for the translation of schemas. Let $M = (MS_1,MS_2,\Sigma)$ be a mapping, and $S_1 = \{u_{1,1},...,u_{1,n}\}$ an instance of MS_1. The translation of S_1 into MS_2 is a schema $S_2 = \{u_{2,1},...,u_{2,m}\}$ such that $<S_1,S_2>$ is an instance of M. The translation of S_2 into MS_1 is defined similarly. The approach to the translation of a schema $S_1 = (u_{1,1},...,u_{1,n})$ consists in translating each of its schema units $u_{1,j}$ following the order given by the operation *predecessors*, starting with the units that have no predecessors.

The translation is done by applying the operation called $translateToS_j()$ to the schema units.

2.3.1 The operation translateToS$_j$

An instance $u_{i,k}$ of *S$_i$Element* can be translated into S_j if it represents a schema unit whose mapping kind is *HasEquivalents* or *IsIncluded*. The effect of the operation must be that $u_{i,k}$ is mapped to S_j. This is captured by the simple following formal specification:

```
context ErElement::translateToRelational()
  pre: isSchemaUnit() and (relationalMappingKind() =
        MappingKind::HasEquivalents or
        relationalMappingKind() = MappingKind::IsIncluded)
  post: mappedToSj()

context RelationalElement::translateToEr()
  pre: isSchemaUnit() and (erMappingKind() = MappingKind::HasEquivalents or
      erMappingKind() = MappingKind::IsIncluded)
  post: mappedToSj()
```

There is no need to refine the specification of this operation in the subtypes of *S$_i$Element*. Concerning its implementation, the specification of *mappedToS$_j$* suggests a straightforward implementation using the methods of the operations *s$_j$Equivalents* and *isIncludedInS$_j$*, that have been described before (See Section 2.2.2) and the methods of the operation *createUnit* whose formal definition for each subtype of characterzation object is the following:

Create units of characterization objects of characterization objects of ErElements

```
procedure CreateUnitOfDataTypeCh()
  var   d:DataType;
  begin
      for el:DataTypeCh in [DataTypeCh.allInstances()->asSequence]
       begin
         d := Create( DataType);
         [d].name := [el.name];
       end;
   end;

procedure CreateUnitOfEntityTypeCh()
  var   en:EntityType, a:Attribute, d:DataType, el:EntityTypeCh;
  begin
      for name:String in [EntityTypeCh.allInstances()->select(e:EntityTypeCh|
        EntityType.allInstances()->collect(name)->excludes(e.name))->
        collect(ech:EntityTypeCh|ech.name)->asSet->asSequence]
      begin
        el := Any([EntityTypeCh.allInstances()->select(e:EntityTypeCh|
          e.name = name)->asSequence]);
        en := Create(EntityType);
        [en].name := [el.name];
        for ka:KeyAttribute in [el.keyAttribute->asSequence]
          begin
            a:= Create(Attribute);
            [a].name := [ka.name];
            [a].isKey := [true];
            Insert(EntityType_Attribute, [en],[a]);
            d := Any([DataType.allInstances()->select(e:DataType|
              e.name = ka.type)->asSequence]);
            Insert(AttributeTyping, [a],[d]);
          end;
      end;
```

```
 end;

procedure CreateUnitOfAttributeCh()
 var  a:Attribute, d:DataType, en:EntityType, re:RelationshipType;
 begin
  for el:AttributeCh in [AttributeCh.allInstances()->asSequence]
   begin
     a:= Create(Attribute);
     [a].name := [el.name];
     [a].isKey := [false];
     If [EntityType.allInstances()->select(e:EntityType|e.name = el.owner)
          ->notEmpty] then
       begin
         en := Any([EntityType.allInstances()->select(e:EntityType|
           e.name = el.owner)->asSequence]);
         Insert(EntityType_Attribute, [en],[a]);
       end;
     if [RelationshipType.allInstances()->select(r:RelationshipType|
       r.name = el.owner)->notEmpty] then
       begin
         re := Any([RelationshipType.allInstances()->
         select(r:RelationshipType|r.name = el.owner)->asSequence]);
         Insert(RelationshipType_Attribute, [re],[a]);
       end;
     d := Any([DataType.allInstances()->select(e:DataType|
     e.name = el.type)->asSequence]);
     Insert(AttributeTyping, [a],[d]);
    end;
 end;

procedure CreateUnitOfRelationshipTypeCh()
 var  en:EntityType, a:Attribute, r:RelationshipType, re:RelationEnd,
      el:RelationshipTypeCh;
 begin
   for name:String in [RelationshipTypeCh.allInstances()->
     select(rech:RelationshipTypeCh| RelationshipType.allInstances()->
     collect(name)->excludes(rech.name))->collect(rch:RelationshipTypeCh|
       rch.name)->asSet->asSequence]
   begin
     el := Any([RelationshipTypeCh.allInstances()->select(
       rl:RelationshipTypeCh| rl.name = name)->asSequence]);
     r := Create(RelationshipType);
     [r].name := [el.name];
     for p:Participant in [el.participant->asSequence]
     begin
       re := Create(RelationEnd);
       [re].name := [p.name];
       en := Any([EntityType.allInstances()->select(e:EntityType|
       e.name = p.type)->asSequence]);
       Insert(RelationEndTyping, [re], [en]);
       Insert(RelationshipType_RelationEnd, [r], [re]);
     end;
   end;
 end;
```

Create units of characterization objects of characterization objects of RelationalElements

```
procedure CreateUnitOfRelationalDataTypeCh()
 var  d:RelationalDataType;
```

```
 begin
  for el:RelationalDataTypeCh in [RelationalDataTypeCh.allInstances()->
   asSequence]
   begin
    d := Create( RelationalDataType);
    [d].name := [el.name];
   end;
 end;

procedure CreateUnitOfTableCh()
 var  t:Table,c:Column, d:RelationalDataType;
 begin
 for el:TableCh in [TableCh.allInstances()->asSequence]
   begin
      t := Create(Table);
      [t].name := [el.name];
      for kc:KeyColumn in [el.keyColumn]
      begin
         c:= Create(Column);
         [c].name := [kc.name];
         [c].isKey := [true];
         Insert(Table_Column, [t],[c]);
         d := Any([RelationalDataType.allInstances()->
           select(e:RelationalDataType| e.name = kc.type)->asSequence]);
         Insert(ColumnTyping, [c],[d]);
      end;
   end;
 end;

procedure CreateUnitOfColumnCh()
 var  t:Table,c:Column, d:RelationalDataType;
  begin
  for el:ColumnCh in [ColumnCh.allInstances()->asSequence]
   begin
      c:= Create(Column);
      [c].name := [el.name];
      [c].isKey := [false];
      t := Any([Table.allInstances()->select(e:Table|e.name = el.owner)->
        asSequence]);
      Insert(Table_Column, [t],[c]);
      d := Any([RelationalDataType.allInstances()->
        select(e:RelationalDataType| e.name = el.type)->asSequence]);
      Insert(ColumnTyping, [c],[d]);
   end;
end;

procedure CreateUnitOfForeignKeyCh()
 var  t:Table,c:Column, f:ForeignKey;
 begin
   for el:ForeignKeyCh in [ForeignKeyCh.allInstances()->asSequence]
   begin
      f := Create(ForeignKey);
      t:= Any([Table.allInstances()->select(e:Table| e.name = el.target)->
        asSequence]);
      Insert(Table_ForeignKey, [t], [f]);
      for fkc:ForeignKeyColumn in [el.foreignKeyColumn->asSequence]
      begin
         c := Any ([Column.allInstances()->select(co:Column|
         co.table.name = el.source and co.isKey and co.name =
```

```
            fkc.sourceName)->asSequence]);
          Insert(ForeignKey_Column, [f], [c]);
        end;
    end;
end;
```

3. Example 2: Translation between Simple UML and Simple RDBMS Metaschemas

The second example is based on the Annex A of the MOF QVT Final Adopted Specification (Object Management Group 2007) (also included at the end of this report (Annex) that describes the UML to RDBMS Mapping in the QVT Relations Language.

3.1 Definition of Metaschemas

Simple UML Metaschema

Figure 15 shows the Simple UML metaschema used as example of instance of MOF. Note that there are some differences from the example presented in the Annex A of the QVT document: in this paper, the type of an attribute may be only a primitive data type and the concept of generalization has been represented as a metaclass.

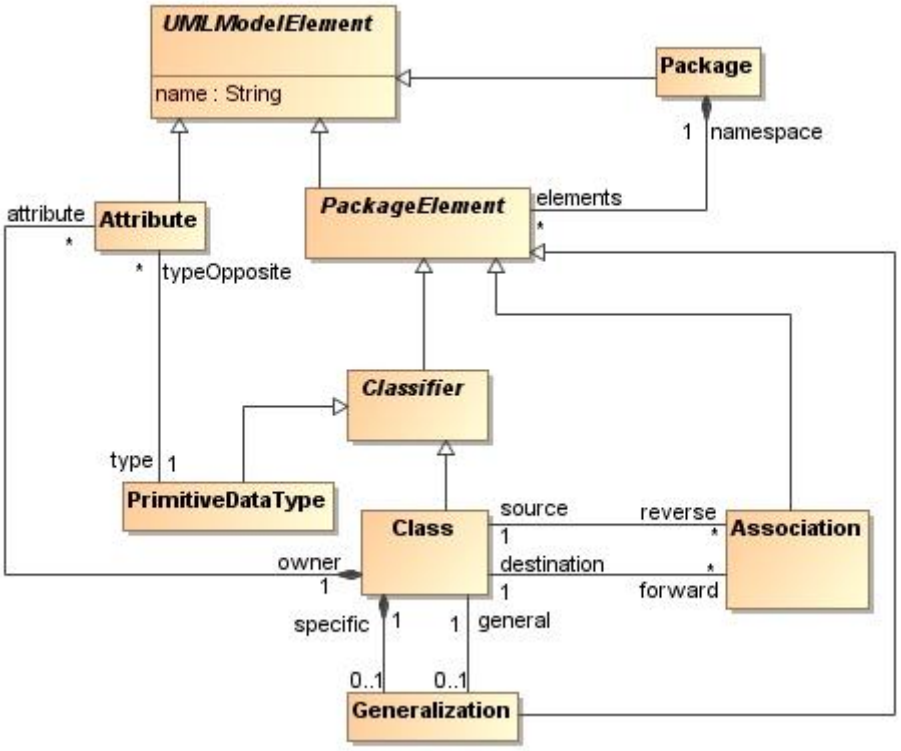

Fig. 15. Simple UML Metaschema

The Simple UML Metaschema also includes the following constraints that are specified as follows:

Names are defined, have anon-zero length and consists of letters and digits:

```
context UMLModelElement inv nameOk:
   let small:Set(String) =
     Set{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p',
     'q', 'r','s','t','u','v','w','x','y','z'} in
   let capital:Set(String) =
     Set{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P',
     'Q', 'R','S','T','U','V','W','X','Y','Z'} in
   let digit:Set(String) =
```

```
    Set{'0','1','2','3','4','5','6','7','8','9'} in
  if self.oclIsTypeOf(Class) then
    self.oclAsType(Class).name.isDefined and
    self.oclAsType(Class).name.size() > 0 and
    Set{1..self.oclAsType(Class).name.size()} -> forAll(i |
      small->union(capital)->union(digit)->
        includes(self.oclAsType(Class).name.substring(i,i)))
  else
    if self.oclIsTypeOf(Attribute) then
      self.oclAsType(Attribute).name.isDefined and
      self.oclAsType(Attribute).name.size > 0 and
      Set{1..self.oclAsType(Attribute).name.size}->forAll(i |
        Set{'_'}->union(small)->union(capital)->union(digit)->
          includes(self.oclAsType(Attribute).name.substring(i,i)))
    else true
    endif
  endif
```

Naming restriction: The names of primitive types are String, Integer and Boolean:

```
context self:PrimitiveDataType inv namesOfPrimitiveDataTypes:
 self.name = 'String' or self.name = 'Boolean' or self.name = 'Integer'
```

Different packages have different names:

```
context self:UMLModelElement inv uniquePackageNames:
 UMLModelElement.allInstances()->forAll(r1,r2 | (r1.oclIsTypeOf(Package)
 and r2.oclIsTypeOf(Package) and r1.oclAsType(Package).name =
 r2.oclAsType(Package).name) implies r1 = r2)
```

Within one package, different classes have have different names:

```
context self:Package inv uniqueClassNamesWithinPackage:
 self.class->forAll(a1,a2 | a1.name = a2.name implies a1 = a2)
```

Within one package, different associations have have different names:

```
context self:Package inv uniqueAssociationNamesWithinPackage:
 self.association->forAll(a1,a2 | a1.name = a2.name implies a1 = a2)
```

Within one Class, different attributes have different names

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

Simple RDBMS Metaschema

Figure 16 shows the Simple RDBMS metaschemas. Note that there are two differences from the QVT example: the types of columns are represented as Sqltypes and some multiplicities have been added in the metaschema.

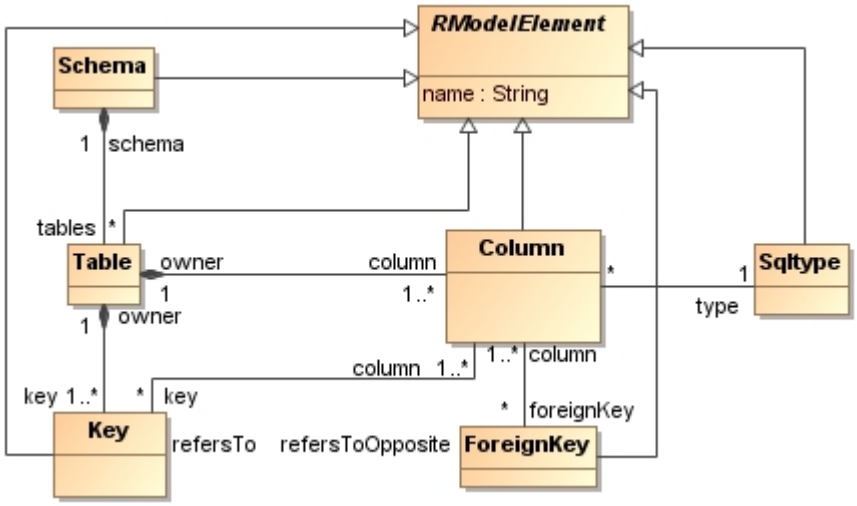

Fig. 16. Simple RDBMS Metaschema

The RDBMS metaschema also includes the following constraints that, formally in OCL, are specified as follows:

Names are defined, have a non-zero length, and consist of letters, digits, parenthesis and underscore:

```
context RModelElement inv nameOk:
 let small:Set(String) =
   Set{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
   'r','s','t','u','v','w','x','y','z'} in
 let capital:Set(String) =
   Set{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',
   'R','S','T','U','V','W','X','Y','Z'} in
 let digit:Set(String) =
  Set{'0','1','2','3','4','5','6','7','8','9'} in
 if self.oclIsTypeOf(Table)
 then
   self.oclAsType(Table).name.isDefined and
   self.oclAsType(Table).name.size > 0 and
   Set{1..self.oclAsType(Table).name.size} -> forAll(i |Set{'_','(',')} ->
     union(small) -> union(capital) -> union(digit)->
     includes(self.oclAsType(Table).name.substring(i,i)))
 else
   if self.oclIsTypeOf(Column)
   then
     self.oclAsType(Column).name.isDefined and
     self.oclAsType(Column).name.size>0 and
     Set{1..self.oclAsType(Column).name.size} -> forAll(i |
       Set{'_','(',')} -> union(small) -> union(capital) -> union(digit) ->
       includes(self.oclAsType(Column).name.substring(i,i)))
   else
     if self.oclIsTypeOf(Sqltype)
     then
       self.oclAsType(Sqltype).name.isDefined and
       self.oclAsType(Sqltype).name.size > 0 and
       Set{1..self.oclAsType(Sqltype).name.size} -> forAll(i |
       Set{'_'}->union(small) -> union(capital) -> union(digit) ->
       includes(self.oclAsType(Sqltype ype).name.substring(i,i)))
     else
       true
     endif
```

```
    endif
  endif
```

Naming restriction: Different sqltypes have different names:

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

Different Tables have different names:

```
context self:RelationalElement inv uniqueTableNames:
   RModelElement.allInstances() -> forAll(r1,r2 |
      (r1.oclIsTypeOf(Table) and r2.oclIsTypeOf(Table) and
      r1.oclAsType(Table).name = r2.oclAsType(Table).name)
      implies r1 = r2)
```

Within one Table, different Columns have different names:

```
context self:Table inv uniqueColNamesWithinTable:
   self.column -> forAll(a1,a2 | a1.name = a2.name implies a1 = a2)
```

All columns of a foreign key belong to the same table:

```
context ForeignKey inv allColumnsOfForeignKeyHaveSameTable:
   column.table -> size() = 1
```

3.1.1 Definition of Schema Units

The definition of mapping expressions is based on the concept of schema units. Schema units are the knowledge components of the schemas. A schema consists of a set $S = \{u_1,...,u_n\}$ of schema units u_i, such that the knowledge expressed by S is the set of knowledge components expressed by its schema units $u_1,...,u_n$.

Syntactically, a schema unit u is a set of schema elements such that:

- they can be added to a schema S when some conditions are satisfied, and
- $S \cup \{u\}$ is a valid instance of *MS*.

The rationale behind this definition is that the knowledge expressed by a schema $S = \{u_1,...,u_i\}$ can be extended by a new schema unit u_{i+1} obtaining $S' = \{u_1,...,u_i,u_{i+1}\}$.

Simple UML Schema Units

In a SimpleUML schema, the schema units are packages, classes, primitive data types, attributes, associations and generalizations. The representation and the schema elements of the schema units is the following (see Figure 17):

- Each package is represented by an instance of *Package*. The schema elements of a package named *p* are: (1) the instance α of *Package*; and (2) the instance of attribute *name* of α with value *p*.
- Each class is represented by an instance of *Class*. The schema elements of a class named *c* are: (1) the instance β of *Class*; (2) the instance of attribute *name* of β with value *c*; and (3) the instance of its relationship with an instance of *Package*.
- Each primitive data type is represented by an instance of *PrimitiveDataType*. The schema elements of a primitive type named *pr* are: (1) the instance δ of *PrimitiveType*; (2) the instance of attribute *name* of δ with value *pr*; and (3) the instance of its relationship with an instance of *Package*.

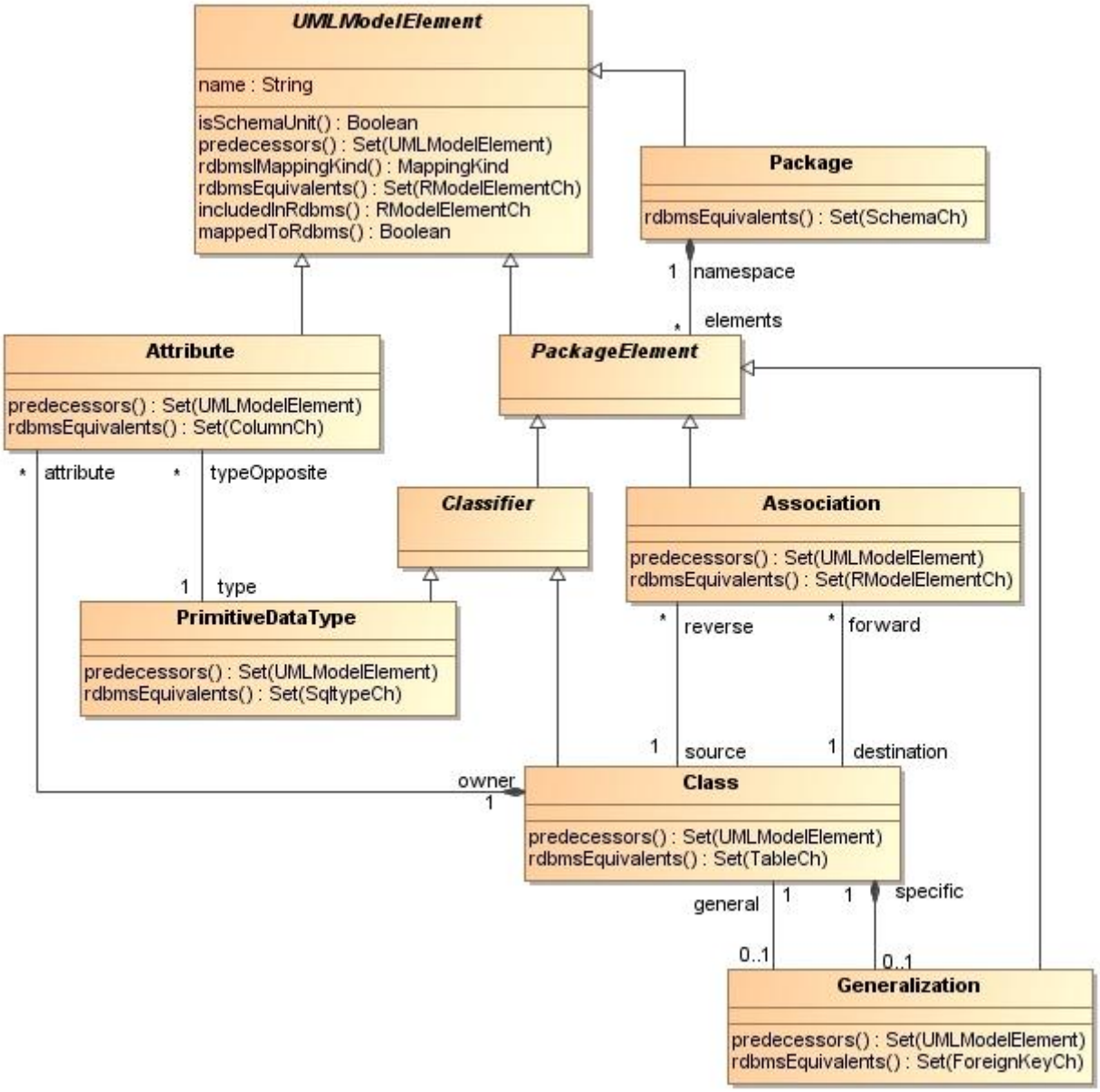

Fig. 17. Simple UML Metaschema Schema Units

- Each attribute is represented by an instance of Attribute. The schema elements of an attribute named at are: (1) the instance π of Attribute; (2) the instance of attribute name of π with value at; (3) the instance of its relationship with an instance of Class (the owner); (4) the instance of its relationship with an instance of PrimitiveDataType;
- Each association is represented by an instance of *Association*. The schema elements of an association named *as* are: (1) the instance ρ of *Association*; (2) the instance of attribute *name* of ρ with value *as*; (3) the instance of its relationship with an instance of *Package*; (4) the instance of its relationship with an instance of *Class* (the source); and (5) the instance of its relationship with an instance of *Class* (the target);
- Each generalization is represented by an instance of *Generalization*. The schema elements of a generalization are: (1) the instance ψ of *Generalization*; (2) the instance of its relationship with an

instance of *Package*; (3) the instance of its relationship with an instance of *Class* (general); and (4) the instance of its relationship with an instance of *Class* (specific);

Therefore, for the SimpleUML metaschemas, the schema units are defined, formally, as:

```
context UMLModelElement::isSchemaUnit():Boolean
   body: True
```

meaning that by default all (direct or indirect) instances of UMLModelElement are schema units.

Simple RDBMS Schema Units

In the Simple RDBMS metaschema of Fig. 16, the schema units of a relational schema are schemas, tables, columns, foreign keys and sqltype. The representation and the schema elements of the schema units, as shown in Figure 18, is the following:

- Each schema is represented by an instance of *Schema*. The schema elements of a schema named *s* are: (1) the instance ω of *Schema*; and (2) the instance of attribute *name* of ω with value *s*.
- Each table is represented by an instance of *Table*. The schema elements of a table named *t* are: (1) an instance σ of *Table*; (2) the instance of attribute *name* of σ with value *t*; (3) the instance of *Key* related to σ; and (4) for the key: the instance of its relationship with σ, the instance of its attribute *name*; (5) the (one or more) instances of *Column* related to σ that comprise the key; (6) for each one of these columns: the instance of its relationship with σ, the instances of its relationships with the instances of *Key*, the instances of its attributes *name* and *type*. Note that we group a table and all its key columns into a single schema unit.
- Each column of a table that is not part of a key of such a table is represented by an instance of *Column* that is not associated to any instance of *Key*. The schema elements of a column named *c* are: (1) an instance γ of Column; (2) the instance of attribute *name* of γ with value *c*; (3) the instance of its attribute *type* of γ; and (4) its relationship with an instance of *Table*.
- Each foreign key is represented by an instance of *ForeignKey*. The schema elements of a foreign key named *fk* are: (1) an instance η of *ForeignKey*; (2) the instance of attribute *name* of η with value *fk* (3) the relationships of η with *Column* that give the columns that comprise η; (4) the relationship of η with the table that owns the η; and (5) the relationship of η with the instance of *Key* referenced by η.
- Each sqltype is represented by an instance of *Sqltype*. The schema elements of a sqltype named *sq* is: (1) the instance ζ of *Sqltype*; and (2) the instance of attribute *name* of ζ with value *sq*.

Therefore, in the Simple RDBMS metaschema of Figure 16 the schema units (shown in Figure 18) are defined as follows:

```
context RModelElement::isSchemaUnit():Boolean
 body: True
```

meaning that by default all (direct or indirect) instances of *RModelElement* are schema units. There are two exceptions: (1) the instances of Key are not schema units, and (2) not all instances of *Column* are schema units, but only those that do not comprise keys. Therefore, the above operation is redefined as follows:

```
context Key::isSchemaUnit():Boolean
 body: false


context Column::isSchemaUnit():Boolean
 body: self.key->isEmpty()
```

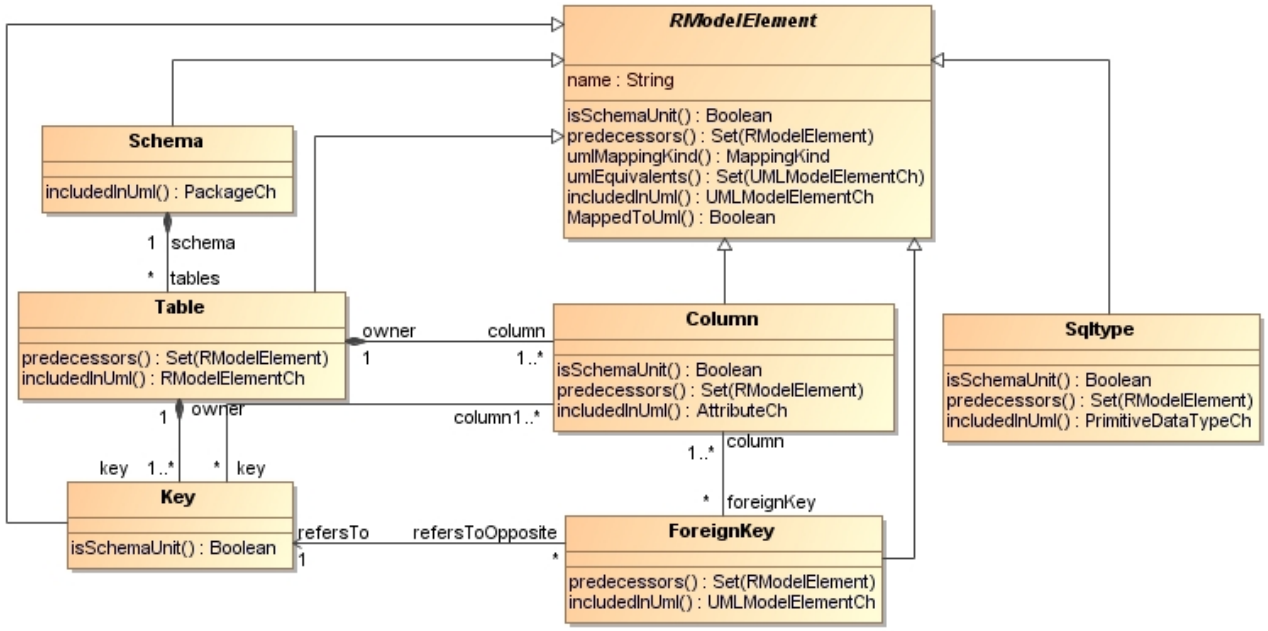

Fig. 18. Simple RDBMS Metaschema Schema Units

3.1.2 Predecessors

A schema consists of a set $S = \{u_1,...,u_n\}$ of schema units u_i, but in general there are precedence relationships between them. The *predecessor* units of a schema unit u_i are those schema units that are direct predecessors of u_i. A schema unit may not be a direct or indirect predecessor to itself.

Simple UML Schema Units Predecessors

In the Simple UML metaschema of Figure 17, the *predecessors* operation is specified as:

```
context UMLModelElement::predecessors():Set(UMLModelElement)
 pre: isSchemaUnit()
 body: Set{}
```

meaning that by default all schema units do not have predecessors. This is the case of *Package* and therefore there is no need of redefining the operation for it.

For *PrimitiveDataType* , the *predecessors* operation is redefined as:

```
context PrimitiveDataType::predecessors():Package
 body: self.namespace
```

meaning that the predecessors of an primitive data type is its owning namespace.

The *predecessors* operation of *Class* is specified as:

```
context Class::predecessors():Set(UMLModelElement)
 body: UMLModelElement.allInstances()->select(el:UMLModelElement |
      (el.oclIsTypeOf(Package) and el.oclAsType(Package) = self.namespace)
      or (el.oclIsTypeOf(Class) and self.general->
            includes(el.oclAsType(Class))))
```

meaning that the predecessors of a class are its owning namespace and its general classes.

The *predecessors* operation of *Attribute* is specified as:

```
context Attribute::predecessors():Set(UMLModelElement)
 body: UMLModelElement.allInstances()->select(el:UMLModelElement |
      (el.oclIsTypeOf(Class) and el.oclAsType(Class) = self.owner) or
      (el.oclIsTypeOf(PrimitiveDataType) and
      el.oclAsType(PrimitiveDataType) = self.type))
```

meaning that the predecessors of an attribute are its owning class and its type.

The *predecessors* operation of *Association* is specified as:

```
context Association::predecessors():Set(UMLModelElement)
 body: UMLModelElement.allInstances()->select(el:UMLModelElement |
      (el.oclIsTypeOf(Package) and el.oclAsType(Package) = self.namespace)
      or (el.oclIsTypeOf(Class) and (el.oclAsType(Class) = self.source or
        el.oclAsType(Class) = self.destination)))
```

meaning that the predecessors of an *association* are its namespace and the two participants in the association.

The *predecessors* operation of *Generalization* is specified as:

```
context Generalization::predecessors():Set(UMLModelElement)
 body: UMLModelElement.allInstances()->select(el:UMLModelElement |
      (el.oclIsTypeOf(Package) and el.oclAsType(Package) = self.namespace)
      or (el.oclIsTypeOf(Class) and (el.oclAsType(Class) = self.general or
      el.oclAsType(Class) = self.specific)))
```

meaning that the predecessors of a *generalization* are its namespace and the two participants in the generalization.

Simple RDBMS Schema Units Predecessors

In the Simple RDBMS metaschema of Fig. 18 the *predecessors* operation of an element of the metaschemas is specified as:

```
context RModelElement::predecessors():Set(RelationalElement)
 pre: isSchemaUnit()
 body: Set{}
```

meaning that by default all schema units do not have predecessors. This is the case of *Schema* and therefore there is no need of redefining the operation for it.

The *predecessors* operation of *Table* is specified as:

```
context Table::predecessors():Set(RelationalDataType)
 body: RModelElement.allInstances()->select(el:RModelElement |
      (el.oclIsTypeOf(Schema) and el.oclAsType(Schema) = self.schema) or
      (el.oclIsTypeOf(Sqltype) and self.key->collect(column.type)->flatten->
        asSet()->includes(el.oclAsType(Sqltype))))
```

meaning that the predecessor of table is its schema and the types of its key columns.

The predecesssor of a non-key column is its table and its type schema units:

```
context Column::predecessors():Set(Table)
 body: RModelElement.allInstances()-> select(el:RModelElement |
      (el.oclIsTypeOf(Table) and el.oclAsType(Table) = self.owner) or
      (el.oclIsTypeOf(Sqltype) and self.type = el.oclAsType(Sqltype)))
```

Finally, the predecessors of a foreign key are the source columns and the referenced table:

```
context ForeignKey::predecessors():Set(RelationalElement)
 body: RModelElement.allInstances()->select(el:RModelElement |
      (el.oclIsTypeOf(Table) and el.oclAsType(Table) = self.refersTo.owner)
      or (el.oclIsTypeOf(Column) and
           self.column->includes(el.oclAsType(Column)))
      or (el.oclIsTypeOf(Sqltype) and
```

```
self.column.type->includes(el.oclAsType(Sqltype))))
```

3.1.3 Characterization Objects

Each schema unit is characterized by an object (called *characterization* object). Characterization objects roughly correspond to value or domain value objects in the object-oriented design patterns field. In a metaschema, there is a characterization object type for each subtype *ST* of S_i*Element* such that some or all of its instances represent schema units. Each characterization object type includes a set of attributes that characterize the schema unit and two operations: *createUnit*() and *schemaUnit*(). The first operation creates a schema unit from its characterization object, and the second checks that the schema unit corresponding to the characterization object does indeed exist.

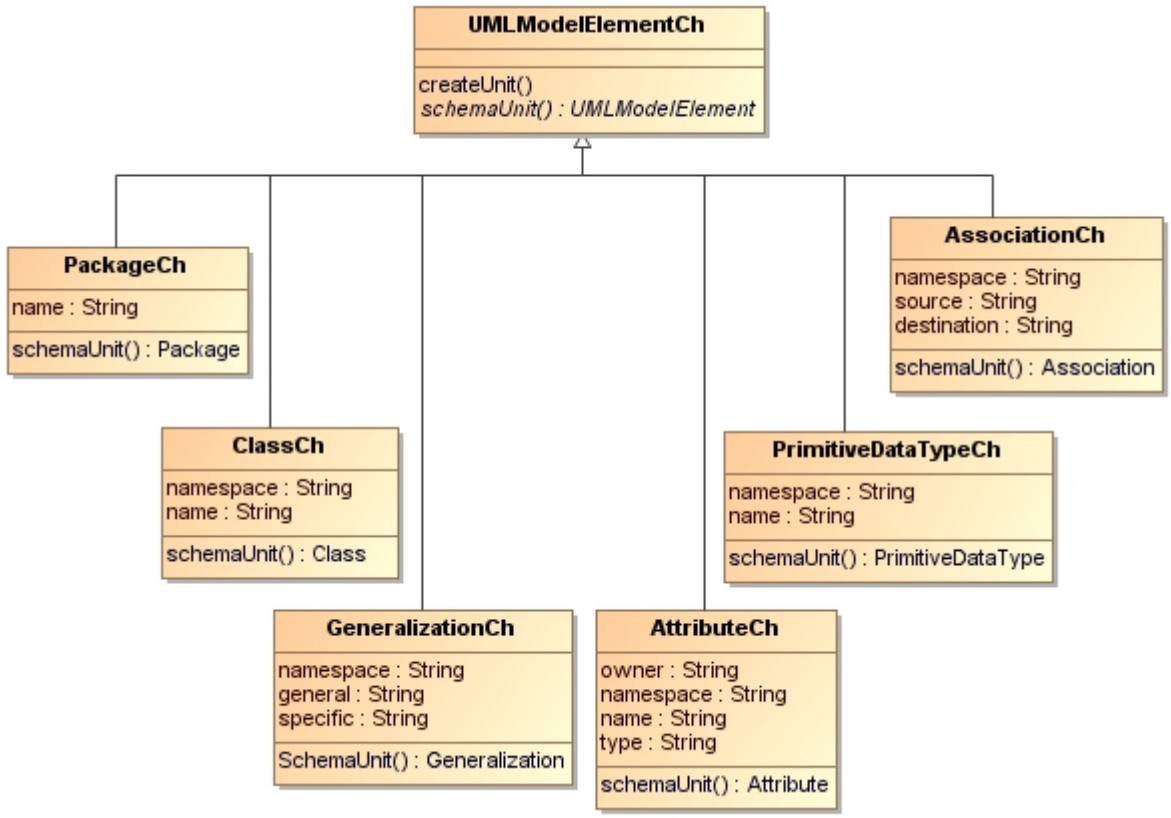

Fig. 19. Simple UML Characterization Objects Types

Simple UML Metaschema Characterization Objects

The characterization object types of the Simple UML metaschema are shown in Figure 19.

The specification of the *createUnit()* operation is the same for all characterization object types, and therefore is specified in the *UMLModelElementCh* as:

```
context UMLModelElementCh::createUnit()
 post: schemaUnit()->notEmpty()
```

The specification of the *schemaUnit()* operation is defined in each subtype of *UMLModelElementCh* as follows:

The simplest is *PackageCh* for which the above mentioned operation is specified as:

```
context PackageCh::schemaUnit():Package
 body: Package.allInstances() -> select(p:Package|p.name = self.name)->
      any(true)
```

The *schemaUnit* operation is a query that gives the package whose name is the one given in the attribute *name* of *PackageCh*.

The *schemaUnit* operation *ClassCh* is specified as:

```
context ClassCh::schemaUnit ():Class
 body: Class.allInstances() -> select(c:Class|c.name = self.name and
        c.namespace.name = self.namespace) -> any(true)
```

The *schemaUnit* operation of *ClassCh* gives the class whose name is the one given in the attribute *name* and which is included in the namespace whose name is the one given in the attribute *namespace* of *ClassCh*.

The *schemaUnit* operation of *Attribute* is specified as:

```
context AttributeCh::schemaUnit():Attribute
 body: Attribute.allInstances() -> select(a:Attribute|
      a.name = self.name and a.owner.name = self.owner and
      a.owner.namespace.name = self.namespace and
      a.type.name = self.type and a.type.namespace.name = self.namespace)
        ->any(true)
```

The *schemaUnit* operation gives the attribute whose name is the one given in the attribute *name*; the name and namespace of its owner is the one given by the attributes *owner* and *namespace*; and the name of its type is given by the attribute *type* of *AttributeCh*.

The *schemaUnit* of *PrimitiveDataTypeCh* is as follows:

```
context PrimitiveDataType::schemaUnit():PrimitiveDataType
 body: PrimitiveDatatype.allInstances() -> select(p:PrimitiveDataType|
      p.name = self.name and p.namespace.name = self.namespace) -> any(true)
```

The *schemaUnit* operation gives the primitive data type whose name is the one given in the attribute *name* of *PrimitiveDataTypeCh* and which is owned by the package whose name is the attribute given in *namespace*.

The *schemaUnit* of *AssociationCh* is specified as follows:

```
context AssociationCh::schemaUnit():Association
 body: Association.allInstances()->select(a:Association|
      a.source.name = self.source and
      a.source.namespace.name = self.namespace and
      a.destination.name = self.destination and
      a.destination.namespace.name = self.namespace and
      a.namespace.name = self.namespace) -> any(true)
```

The *schemaUnit* operation gives the association whose source name and destination name are the ones given in the attributes *source* and *destination*, and all of them are included in the same namespace whose name is the attribute given in *namespace*.

The *schemaUnit* of *GeneralizationCh* is specified as follows:

```
context GeneralizationCh::schemaUnit():Association
body: Generalization.allInstances() -> select(g:Generalization|
      g.general.name = self.general and
      g.general.namespace.name = self.namespace and
      g.specific.name = self.specific and
      g.specific.namespace.name = self.namespace and
      g.namespace.name = self.namespace) -> any(true)
```

The *schemaUnit* operation gives the generalization whose general name and specific name of classes are the ones given in the attributes *general* and *specific*, and all of them are included in the same namespace whose name is the attribute given in *namespace*.

Simple RDBMS Metaschema Characterization Objects

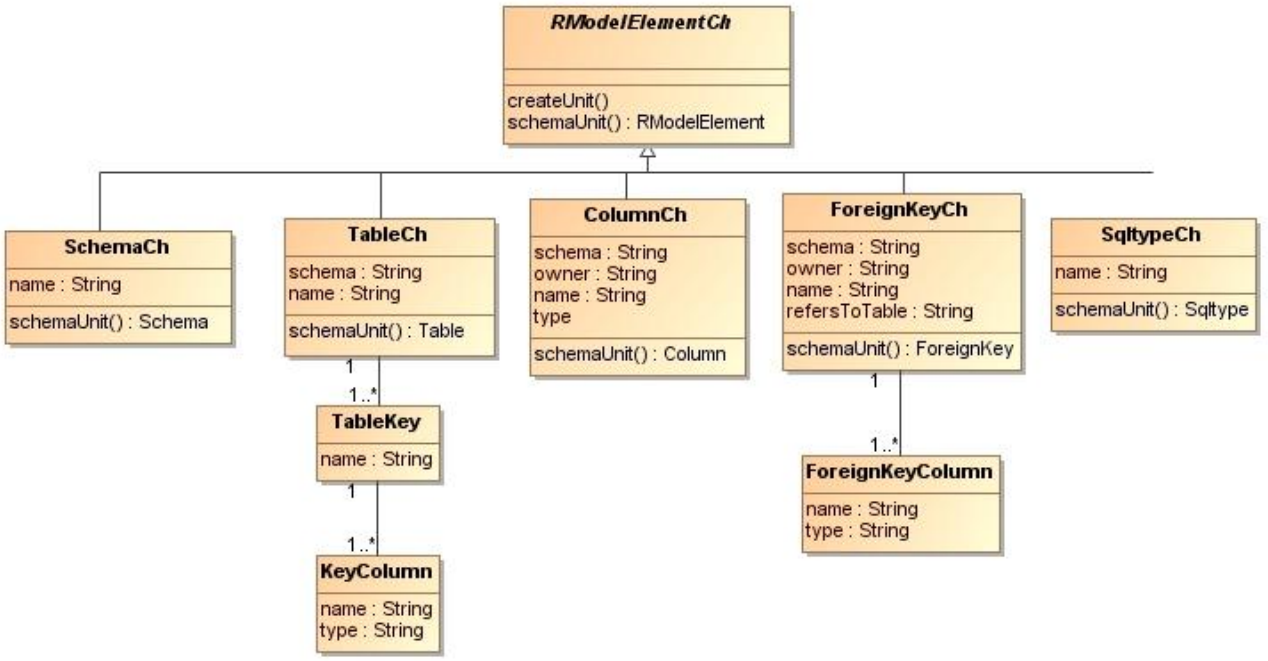

Fig. 20. Simple RDBMS Characterization Objects Types

The characterization object types of the Simple RDBMS metaschema are shown in Figure 20.

The specification of the *createUnit()* operation is the same for all characterization object types, and therefore is specified in the *RModelElementCh* as:

```
context RModelElementCh::createUnit()
 post: schemaUnit()
```

The specification of the *schemaUnit()* operation is defined in each subtype of *RModelElementCh* as follows:

The simplest is *SchemaCh*, for which the above mentioned operation is specified as:

```
context SchemaCh::schemaUnit():Schema
 body: Schema.allInstances() -> select(s:Schema|
         s.name = self.name) -> any(true)
```

The *schemaUnit* operation is a query that gives the "schema" whose name is the one given in the attribute *name* of *SchemaCh*.

The *schemaUnit* operation *TableCh* is specified as:

```
context TableCh::schemaUnit ():Table
 body: Table.allInstances() -> select(t:Table|t.name = self.name and
       t.schema.name = self.schema and
       t.key.name->asSet() = self.tableKey.name -> asSet() and
       t.key.column->asSet() -> collect(k:Column|
          Tuple{kcn:k.name,kt:k.type.name}) -> asSet() =
       self.tableKey.keyColumn -> asSet() -> collect(c:KeyColumn|
          Tuple{kcn:c.name, kt:c.type}) -> asSet()) -> any(true)
```

The *schemaUnit* operation is a query that gives the table whose name is the one given in the attribute *name*, which is included in the schema whose name is the one given in the attribute *schema* and whose keys with their columns are given in the attribute *name* of the *TableKey* and the attributes *name* and *type* of *KeyColumn* auxiliary objects of *ClassCh*.

The *schemaUnit* operation *ColumnCh* is specified as:

```
context ColumnCh::schemaUnit():Column
 body: Column.allInstances() -> select(c:Column|
      c.name = self.name and c.owner.name = self.owner and
      c.owner.schema.name = self.schema and c.type.name = self.type)
        ->any(true)
```

The *schemaUnit* operation is a query that gives the column whose name is the one given in the attribute *name*, the name and schema of its owner is the one given by the attributes *owner* and *schema* and the name of its type is given by the attribute *type* of *ColumnCh*.

The *schemaUnit* operation *SqltypeCh* is specified as:

```
context SqltypeCh::schemaUnit():Sqltype
 body: Sqltype.allInstances() -> select(s:Sqltype|
      s.name = self.name) -> any(true)
```

The *schemaUnit* operation is a query that gives the sqltype whose name is the one given in the attribute *name* of *SqltypeCh*.

The *schemaUnit* of *ForeignKeyCh* is as follows:

```
context ForeignKeyCh::schemaUnit():ForeignKey
 body: ForeignKey.allInstances() -> select(fk:ForeignKey|
      fk.name = self.name and fk.column -> collect(c:Column|
        Tuple{o:c.owner.name, s:c.owner.schema.name, n:c.name,
              t:c.type.name}) -> asSet() =
        self.foreignKeyColumn -> collect(kc:ForeignKeyColumn|
          Tuple{o:self.owner, s:self.schema, n:kc.name, t:kc.type})->asSet()
      and fk.refersTo.owner.schema.name = self.schema and
      fk.refersTo.owner.name = self.refersToTable) -> any(true)
```

The *schemaUnit* operation gives the foreign key whose name is the one given by the attribute *name*, whose columns correspond to the ones given by the attributes *schema*, *owner*, *name* and *type* and which refers to a key given by the attributes *schema*, *owner* and *refersToKey* of *ForeignKeyCh*.

3.2 Translation mapping expressions

This section describes the five operations needed to specify the translation mapping constraints between the Simple UML metaschemas and the Simple Rdbms metaschemas.

The translation proposed here is in both directions, from Simple UML to Simple Rdbms and viceversa.

In the Annex A of the QVT document, a class maps to a table, a primary key and an identifying column. Attributes of classes maps to columns of tables: an attribute of a primitive data type maps to a single column; and attributes inherited from the class hierarchy are also mapped to the columns of the table. An association between two classes maps to a foreign key relationship between the corresponding tables.

This works differs from the QVT example in that the generalization maps to a foreign key and the columns that correspond to the attributes of the superclass are not mapped as columns of the table that represents the subclass. On the other hand, an association maps to a table with two additional foreign key, one for each participant of the association. The mapping of association and generalization has been taken from (Muller 1999, Teorey, Lightstone & Nadeau 2006).

3.2.1 s_jMappingKind

The query operation s_j*MappingKind*():*MappingKind* indicates how a schema unit of S_i is translated into S_j. The value of this operation is mapping-dependent. *MappingKind* is an enumeration data type whose values are:

- *HasEquivalents*. A schema unit of S_i has this mapping kind when it is completely equivalent to a set $\{u_{j,1},...,u_{j,k}\}$ of one or more schema units of S_j. The mapping kind of $u_{j,1},...,u_{j,k}$ must be *IsIncluded*.
- *IsIncluded*. A schema unit of S_i has this mapping kind when it is included in a schema unit $u_{j,k}$ of S_j. The mapping kind of $u_{j,k}$ must be *HasEquivalents*.
- *Untranslatable*. A schema unit of S_i has this mapping kind when it cannot be translated into S_j. If a schema S_i contains one or more untranslatable schema units then its translation into S_j can only be partial.

In the context of S_i*Element* the operation s_j*MappingKind*() can only give a default value, and each subtype *ST* of S_i*Element* such that some or all of its instances represent schema units, redefines it (if needed) to give the correct value. The value of the operation for the instances of *ST* that are not a schema unit is undefined. This is enforced by means of the *mapping kind definition constraint*, which is specified as:

```
context UMLModelElement::rdbmsMappingKind():MappingKind
 body:  if isSchemaUnit()
        then MappingKind::HasEquivalents
        else Set{}
        endif

context RModelElement::umlMappingKind():MappingKind
 body:  if isSchemaUnit()
        then MappingKind::IsIncluded
        else Set{}
        endif
```

3.2.2 s_jEquivalents and IncludedIns_j

The evaluation of the s_j*Equivalents*() operation on a schema unit of S_i whose mapping kind is *HasEquivalents* gives the set of characterization objects of the schema units of S_j that are equivalent to it.

RdbmsEquivalents of Simple UML Elements

The signature of the precondition and postconditions of *rdbmEquivalents()* on an UMLModelElement is defined as follows:

```
context UMLModelElement::rdbmsEquivalents():Set(RModelElementCh)
 pre hasEquivalents: rdbmsMappingKind() = MappingKind::HasEquivalents
 post atLeastOneCharacterizationObjectCreate:
      (RModelElementCh.allInstances() - RModelElementCh.allInstances@pre())
        -> notEmpty()
 post definingTheResult:
      result = RModelElementCh.allInstances()-
                  RModelElementCh.allInstances()@pre
```

The effect of the operation, defined declaratively by postconditions in its subtypes, is specified as:

A package maps to a characterization object of a schema with the same name:

```
context Package::rdbmsEquivalents():Set(SchemaCh)
 post SchemaChCreated:
      SchemaCh.allInstances() - SchemaCh.allInstances@pre())->
        select(s:SchemaCh| s.oclIsNew() and name = self.name) -> size() = 1
```

A class maps to a characterization object of a table with the same name and associated to an instance of TableKey with the attribute *name* as the name of the class followed by '_pk'; and the tableKey is associated to an instance of KeyColumn with the attribute *name* with value as the name of the class followed by '_tid':

```
context Class::rdbmsEquivalents():Set(TableCh)
 post TableChCreated:
```

```
(TableCh.allInstances - TableCh.allInstances@pre)->
  select(t:TableCh| t.oclIsNew() and t.name = self.name and
    t.schema = self.namespace.name and
(TableKey.allInstances - TableKey.allInstances@pre) ->
  select(tk:TableKey| tk.oclIsNew() and tk.tableCh = t and tk.name =
  self.name.concat('_pk') and
(KeyColumn.allInstances - KeyColumn.allInstances@pre)->
  select(c:KeyColumn| c.oclIsNew() and c.tableKey = tk and
    c.name = self.name.concat('_tid') and c.type = 'NUMBER')
    -> size() = 1)-> size() = 1) -> size() = 1
```

An attribute maps to a characterization object of a column with the same name, with the name of the schema and owner as the name of the namespace and owner of the attribute respectively and with the primitive data type converted to the sqltype:

```
context Attribute::rdbmsEquivalents():Set(ColumnCh)
 post ColumnChCreated:
      (ColumnCh.allInstances() – ColumnCh.allInstances@pre()) ->
      select(c:ColumnCh| c.oclIsNew() and c.name = self.name and
        c.schema = self.owner.namespace.name and c.owner = self.owner.name
        and c.type = (if self.type.name = 'String' then 'VARCHAR'
                      else if self.type.name = 'Boolean' then 'BOOLEAN'
                           else 'NUMBER' endif endif))->size() = 1
```

A primitive data type maps to a characterization object of a sqltype with the name of the type converted to its corresponding type:

```
context PrimitiveDataType::rdbmsEquivalents():Set(SqltypeCh)
post SqltypeChCreated:
      (SqltypeCh.allInstances() - SqltypeCh.allInstances@pre()) ->
      select(s:SqltypeCh| s.oclIsNew() and s.schema = self.namespace.name
        and s.name = (if self.name = 'String' then 'VARCHAR'
                      else if self.name = 'Boolean' then 'BOOLEAN'
                           else 'NUMBER' endif endif)) -> size() = 1
```

An association type maps to a characterization object of a table with the name of the association, with two columns named as the columns of the primary keys of the tables that corresponds to the participants of the associations; a primary key formed by these two columns and two foreign keys that relates each column to its refered table:

```
context Association::rdbmsEquivalents():Set(RModelElementCh)
post TableChCreated:
      (TableCh.allInstances() - TableCh.allInstances@pre()) ->
      select(t:TableCh| t.oclIsNew() and t.name = self.name and
        t.schema = self.namespace.name and
        (TableKey.allInstances() - TableKey.allInstances@pre()) ->
        select(tk:TableKey| tk.oclIsNew() and
          tk.name = self.name.concat('_pk') and tk.tableCh = t and
          (KeyColumn.allInstances() - KeyColumn.allInstances@pre()) ->
          select(kc:KeyColumn| kc.oclIsNew() and
            kc.name = self.source.name.concat('_tid') and
            kc.type = 'NUMBER' and kc.tableKey = tk and
            (ForeignKeyCh.allInstances() - ForeignKeyCh.allInstances@pre())
            ->select(fk:ForeignKeyCh| fk.oclIsNew() and
              fk.schema = self.namespace.name and fk.owner = t.name and
              fk.name = kc.name.concat('_fk') and
              fk.refersToTable = self.source.name and
              (ForeignKeyColumn.allInstances() –
                ForeignKeyColumn.allInstances@pre()) ->
                select(fkc:ForeignKeyColumn| fkc.oclIsNew() and
                  fkc.name=kc.name and
```

```
                    fkc.type='NUMBER' and fkc.foreignKeyCh=fk) ->size() = 1)
                      ->size() = 1) -> size() = 1 and
          (KeyColumn.allInstances() - KeyColumn.allInstances@pre()) ->
          select(kc:KeyColumn| kc.oclIsNew() and
            kc.name = self.destination.name.concat('_tid') and
            kc.tableKey=tk and kc.type = 'NUMBER' and
            (ForeignKeyCh.allInstances() - ForeignKeyCh.allInstances@pre()) ->
            select(fk:ForeignKeyCh| fk.oclIsNew() and
              fk.schema = self.namespace.name and fk.owner = t.name and
              fk.name = kc.name.concat('_fk') and
              fk.refersToTable = self.destination.name and
              (ForeignKeyColumn.allInstances() –
                ForeignKeyColumn.allInstances@pre()) ->
                select(fkc:ForeignKeyColumn| fkc.oclIsNew() and
                  fkc.name = kc.name and fkc.type = 'NUMBER' and
                  fkc.foreignKeyCh = fk) -> size() = 1) -> size() = 1) ->
                    size() = 1) -> size() = 1) -> size() = 1
```

A generalizaton type maps to a characterization object of foreign key with the owner the name of the specific class and the refersToTable the name of the general class:

```
context Generalization::rdbmsEquivalents():Set(ForeignKeyCh)
 post ForeignKeyChCreated:
      (ForeignKeyCh.allInstances() – ForeignKeyCh.allInstances@pre) ->
      select(f:ForeignKeyCh| f.oclIsNew() and
        f.name = self.specific.name.concat('_'.concat(self.name.concat(
        self.general.name))) and f.schema = self.namespace.name and
        f.owner = self.specific.name and f.refersToTable = self.general.name
        and (ForeignKeyColumn.allInstances() –
              ForeignKeyColumn.allInstances@pre) ->
          select(fkc:ForeignKeyColumn| fkc.oclIsNew() and
            fkc.name = f.name.concat('_tid') and
            fkc.type = 'NUMBER' and fkc.foreignKeyCh = f) -> size() = 1)
              ->size() = 1
```

The method of the rdbmsEquivalenst() has been specified in OCL executable as:

```
procedure rdbmsEquivalents()
 var  sch:SchemaCh, tch:TableCh, tk:TableKey, kc:KeyColumn, kc1:KeyColumn,
      kc2:KeyColumn, cch:ColumnCh, fchg:ForeignKeyCh,
      fch:ForeignKeyCh, fch1:ForeignKeyCh, fch2:ForeignKeyCh,
      fkc:ForeignKeyColumn, fkc1:ForeignKeyColumn, fkc2:ForeignKeyColumn,
      sqch:SqltypeCh;
begin
for e:UMLModelElement in [UMLModelElement.allInstances()->
  select(isSchemaUnit() and rdbmsEquivalents_m->isEmpty)->asSequence]
begin
    if [e.oclIsTypeOf(Package)] then
   begin
    sch := Create( SchemaCh );
  [sch].name := [e.oclAsType(Package).name];
  Insert( RdbmsEquivalents, [e], [sch] );
 end;

   if [e.oclIsTypeOf(PrimitiveDataType)] then
   begin
    sqch := Create( SqltypeCh );
    [sqch].schema := [e.oclAsType(PrimitiveDataType).namespace.name];
   [sqch].name := [if e.oclAsType(PrimitiveDataType).name = 'String' then
        'VARCHAR' else ife.oclAsType(PrimitiveDataType).name = 'Boolean'
        then 'BOOLEAN' else 'NUMBER' endif endif];
```

```
   Insert( RdbmsEquivalents, [e], [sqch] );
end;

if[e.oclIsTypeOf(Class)] then
  begin
   tch := Create( TableCh );
   Insert( RdbmsEquivalents, [e], [tch] );
   [tch].name := [e.oclAsType(Class).name];
   [tch].schema := [e.oclAsType(Class).namespace.name];
   tk := Create (TableKey);
   [tk].name :=  [e.oclAsType(Class).name.concat('_pk')];
   Insert (TableCh_TableKey, [tch], [tk]);
   kc := Create (KeyColumn);
   [kc].name :=  [e.oclAsType(Class).name.concat('_tid')];
   [kc].type :=  ['NUMBER'];
   Insert (TableKey_KeyColumn, [tk], [kc]);
  end;

  if [e.oclIsTypeOf(Attribute)] then
  begin
   cch := Create( ColumnCh );
   [cch].name := [e.oclAsType(Attribute).name];
   [cch].schema := [e.oclAsType(Attribute).owner.namespace.name];
   [cch].owner := [e.oclAsType(Attribute).owner.name];
   [cch].type := [if e.oclAsType(Attribute).type.name = 'String' then
       VARCHAR' else ife.oclAsType(Attribute).type.name = 'Boolean' then
       BOOLEAN' else 'NUMBER' endif endif];
 Insert( RdbmsEquivalents, [e], [cch] );
end;

  if [e.oclIsTypeOf(Association)] then
  begin
   tch := Create(TableCh);
   [tch].name := [e.oclAsType(Association).name];
   [tch].schema :=  [e.oclAsType(Association).namespace.name];
   tk := Create (TableKey);
   [tk].name := [e.oclAsType(Association).name.concat('_pk')];
   Insert (TableCh_TableKey, [tch], [tk]);
   kc1 := Create (KeyColumn);
   [kc1].name :=  e.oclAsType(Association).source.name.concat('_tid')];
 [kc1].type :=  ['NUMBER'];
 Insert (TableKey_KeyColumn, [tk], [kc1]);
 fch1 := Create (ForeignKeyCh);
 [fch1].schema := [e.oclAsType(Association).namespace.name];
 [fch1].owner := [e.oclAsType(Association).name];
 [fch1].name := [kc1.name.concat('_fk')];
 [fch1].refersToTable := [e.oclAsType(Association).source.name];
 fkc1 := Create (ForeignKeyColumn);
 [fkc1].name := [kc1.name];
 [fkc1].type := ['NUMBER'];
 Insert (ForeignKeyCh_ForeignKeyColumn, [fch1], [fkc1]);
 kc2 := Create (KeyColumn);
 [kc2].name :=
      [e.oclAsType(Association).destination.name.concat('_tid')];
 [kc2].type :=  ['NUMBER'];
 Insert (TableKey_KeyColumn, [tk], [kc2]);
 fch2 := Create (ForeignKeyCh);
 [fch2].schema := [e.oclAsType(Association).namespace.name];
 [fch2].owner := [e.oclAsType(Association).name];
```

```
      [fch2].name := [kc2.name.concat('_fk')];
        [fch2].refersToTable :=
      [e.oclAsType(Association).destination.name];
      fkc2 := Create (ForeignKeyColumn);
      [fkc2].name := [kc2.name];
      [fkc2].type := ['NUMBER'];
      Insert (ForeignKeyCh_ForeignKeyColumn, [fch2], [fkc2]);
      Insert( RdbmsEquivalents, [e], [tch] );
      Insert( RdbmsEquivalents, [e], [fch1] );
      Insert( RdbmsEquivalents, [e], [fch2] );
    end;

    if[e.oclIsTypeOf(Generalization)] then
    begin
     fchg := Create (ForeignKeyCh);
       [fchg].schema := [e.oclAsType(Generalization).namespace.name];
     [fchg].owner := [e.oclAsType(Generalization).specific.name];
     [fchg].name :=
           [e.oclAsType(Generalization).general.name.concat('_tid_fk')];
     [fchg].refersToTable := [e.oclAsType(Generalization).general.name];
     fkc := Create (ForeignKeyColumn);
     [fkc].name :=
     [e.oclAsType(Generalization).specific.name.concat('_tid')];
      [fkc].type := ['NUMBER'];
      Insert (ForeignKeyCh_ForeignKeyColumn, [fchg], [fkc]);
    Insert( RdbmsEquivalents, [e], [fchg] );
    end;
  end;
 end;
```

UMLEquivalents of Simple RDBMS Elements

The signature of the precondition and postconditions of *umlEquivalents()* on a UMLModelElement is defined as follows:

```
context RModelElement::umlEquivalents():Set(UMLModelElementCh)
 pre: umlMappingKind() = MappingKind::HasEquivalents
 post definingTheResult: result =
    (UMLModelElementCh.allInstances - UMLModelElementCh.allInstances@pre)
```

The effect of this operation is not redefined in the subtypes of *RModelElement* because there are not schema units with *HasEquivalents* umlMapping.

IncludedInRdbms of Simple UML Elements

The evaluation of the $IncludedIns_j()$ operation on a schema unit of S_i whose mapping kind is *IsIncluded* gives the characterization object of the schema units of S_j that includes itself in the mapping

The signature of the precondition and postconditions of *includedInRdbms ()* on a UMLModelElement is defined as follows:

```
context UMLModelElement::includedInRdbms ():RModelElementCh
 pre isIncluded: rdbmsMappingKind() = MappingKind::IsIncluded
 post onlyOneCharacterizationObjectCreate:
    (RModelElementCh.allInstances()-RModelElementCh.allInstances@pre())->
    size() = 1
```

The effect of this operation is not redefined in the subtypes of *UMLModelElement* because there are not schema units with *IsIncluded* umlMapping.

IncludedInUml of Simple Rdbms Elements

The signature of the precondition and postconditions of *includedInUml ()* on a RModelElement is defined as follows:

```
context RModelElement::includedInUml ():UMLModelElementCh
 pre isIncluded: umlMappingKind() = MappingKind::IsIncluded
 post onlyOneCharacterizationObjectCreate:
    (UMLModelElementCh.allInstances()-UMLModelElementCh.allInstances@pre)->
    size() = 1
 post definingTheResult: result = (UMLModelElementCh.allInstances() -
    UMLModelElementCh.allInstances@pre())->any(true)
```

The effect of the operation, defined declaratively by postconditions in its subtypes, is specified as:

A schema maps to a characterization object of a package with the same name:

```
context Schema::includedInUml ():PackageCh
 post PackageChCreated:
  (PackageCh.allInstances - PackageCh.allInstances@pre)->
  select(p:PackageCh|p.oclIsNew() and p.name = self.name)->size() = 1
```

A table with no foreign key or a foreign key is mapped to a class with the same name and a table with two foreign key is map to an association with the same name:

```
context Table::includedInUml():RModelElementCh
 post ClassChOrAssociationCreated:
    if self.foreignKey->size()<>2 then
       (ClassCh.allInstances - ClassCh.allInstances@pre)->select(c:ClassCh|
       c.oclIsNew() and c.name = self.name and c.namespace =
self.schema.name)->
       size() = 1
    else
       (AssociationCh.allInstances - AssociationCh.allInstances@pre)->
       select(a:AssociationCh| a.oclIsNew() and a.name = self.name and
       a.namespace = self.schema.name and self.foreignKey.refersTo
       ->select(k:Key| a.source = k.owner.name)->size = 1 and
       self.foreignKey.refersTo->select(k:Key| a.destination = k.owner.name)-
>
       size = 1)->size() = 1
  endif
```

A non key column is mapped to a class of a table that has the same name as the owner of the column and the type of the attribute is the primitive data type that corresponds to the sqltype of the column:

```
context Column::includedInUml():AttributeCh
post AttributeChCreated: (AttributeCh.allInstances() -
 AttributeCh.allInstances@pre())->
 select(a:AttributeCh| a.oclIsNew() and a.name = self.name and
    a.namespace = self.owner.schema.name and a.owner = self.owner.name and
     a.type = (if self.type.name = 'VARCHAR' then 'String' else if
     self.type.name = 'BOOLEAN' then 'Boolean' else 'Integer' endif endif))->
     size() = 1
```

A sqltype is mapped to a primitive data type:

```
context Sqltype::includedInUml():PrimitiveDataTypeCh
 post PrimitiveDataTypeChCreated: (PrimitiveDataTypeCh.allInstances() -
    PrimitiveDataTypeCh.allInstances@pre())->
    select(p:PrimitiveDataTypeCh| p.oclIsNew() and p.namespace =
    self.schema.name and p.name = (if self.name = 'VARCHAR' then 'String'
    else ifself.name = 'BOOLEAN' then 'Boolean' else 'Integer' endif endif))-
    >size() = 1
```

A foreign key that is owned by a table that does not have any other foreign key, is mapped to a generalization and a foreign key that is owned by a table that has two foreign keys, is mapped to an association:

```
context ForeignKey::includedInUML():UMLModelElementCh
 post AssociationOrGeneralizationChCreated:
 ifself.owner.foreignKey->size() = 1 then
    (GeneralizationCh.allInstances() - GeneralizationCh.allInstances@pre())->
    select(g:GeneralizationCh| g.oclIsNew() and
    g.namespace = self.owner.schema.name and g.general =
self.refersTo.owner.name
    and g.specific = self.owner.name)->size() = 1
 else
    (AssociationCh.allInstances() - AssociationCh.allInstances@pre())->
     select(a:AssociationCh| a.oclIsNew() and
     a.namespace = self.owner.schema.name and a.source = self.owner.name and
     a.destination = self.refersTo.owner.name)->size() = 1
 endif
```

The method of the includedInUML() has been specified in OCL executable as:

```
procedure includedInUml()
 var  pch:PackageCh, dch:PrimitiveDataTypeCh, cch:ClassCh,
      ach:AssociationCh, gch:GeneralizationCh, atch:AttributeCh,
      fk:ForeignKey;

begin
for r:RModelElement in [RModelElement.allInstances()->select(isSchemaUnit()
      and includedInUml_m->isEmpty)->asSequence]
 begin
   if [r.oclIsTypeOf(Schema)] then
     begin
       pch := Create(PackageCh);
       [pch].name := [r.oclAsType(Schema).name];
       Insert( IncludedInUml, [r], [pch] );
     end;

   if [r.oclIsTypeOf(Sqltype)] then
     begin
       dch := Create( PrimitiveDataTypeCh );
       [dch].namespace := [r.oclAsType(Sqltype).schema.name];
       [dch].name := [if r.oclAsType(Sqltype).name = 'VARCHAR' then 'String'
                     else if r.oclAsType(Sqltype).name = 'BOOLEAN' then
                     'Boolean' else 'Integer' endif endif];
       Insert( IncludedInUml, [r], [dch] );
     end;

  if [r.oclIsTypeOf(Table)] then
     begin
       if [r.oclAsType(Table).foreignKey -> size() < 2] then
       begin
          cch := Create( ClassCh );
     I    Insert( IncludedInUml, [r], [cch] );
          [cch].name := [r.oclAsType(Table).name];
          [cch].namespace := [r.oclAsType(Table).schema.name];
       end;

       if [r.oclAsType(Table).foreignKey -> size() = 2] then
       begin
          ach := Create(AssociationCh);
```

```
        Insert( IncludedInUml, [r], [ach] );
        [ach].name := [r.oclAsType(Table).name];
        [ach].namespace := [r.oclAsType(Table).schema.name];
        [ach].source := [r.oclAsType(Table).foreignKey.refersTo ->
          asSequence -> first.owner.name];
        [ach].destination := [r.oclAsType(Table).foreignKey.refersTo->
          asSequence->last.owner.name];
      end;
    end;

  if [r.oclIsTypeOf(Column) and r.oclAsType(Column).key->isEmpty] then
    begin
      atch := Create( AttributeCh );
      Insert( IncludedInUml, [r], [atch] );
      [atch].name := [r.oclAsType(Column).name];
      [atch].namespace := [r.oclAsType(Column).owner.schema.name];
      [atch].type := [if r.oclAsType(Column).type.name = 'VARCHAR' then
                      'String' else
                        if r.oclAsType(Column).type.name = 'BOOLEAN' then
                         'Boolean' else 'Integer' endif
                      endif];
      [atch].owner := [r.oclAsType(Column).owner.name];
    end;

  if [r.oclIsTypeOf(ForeignKey)] then
    begin
      if [r.oclAsType(ForeignKey).owner.foreignKey->size() = 1] then
        begin
          gch:= Create(GeneralizationCh);
          [gch].namespace := [r.oclAsType(ForeignKey).owner.schema.name];
          [gch].general := [r.oclAsType(ForeignKey).refersTo.owner.name];
          [gch].specific := [r.oclAsType(ForeignKey).owner.name];
          Insert( IncludedInUml, [r], [gch] );
        end;
      if [r.oclAsType(ForeignKey).owner.foreignKey->size() <> 1] then
        begin
          ach:= Create(AssociationCh);
          [ach].name := [r.oclAsType(ForeignKey).owner.name];
          [ach].namespace := [r.oclAsType(ForeignKey).owner.schema.name];
          [ach].source :=
               [r.oclAsType(ForeignKey).owner.foreignKey.refersTo->
               asSequence->first.owner.name];
          [ach].destination :=
               [r.oclAsType(ForeignKey).owner.foreignKey.refersTo->
               asSequence->last.owner.name];
          Insert( IncludedInUml, [r], [ach] );
        end;
    end;
  end;
 end;
```

3.2.3 MappedToS$_j$

A schema unit is translated correctly if the results of the previous operations are consistent. This is defined in OCL in two operations as follows:

```
context UMLModelElement::mappedToRdbms():Boolean
body:
 if rdbmsMappingKind() = MappingKind::HasEquivalents then
```

```
  self.rdbmsEquivalents()->forAll(re:RModelElementCh|
   re.schemaUnit()->notEmpty() and re.schemaUnit().umlMappingKind() =
   MappingKind::IsIncluded and
  re.schemaUnit().includedInUml().schemaUnit() = self)
 else
  ifrdbmsMappingKind() = MappingKind::IsIncluded then
   self.includedInRdbms().schemaUnit()->notEmpty() and
   self.includedInRdbms().schemaUnit().umlMappingKind() =
   MappingKind::HasEquivalents and
   self.includedInRdbms().schemaUnit().umlEquivalents().schemaUnit()
   ->includes(self)
  else
      false
  endif
endif

 context RModelElement::mappedToUml():Boolean
 body:
  if umlMappingKind() = MappingKind::HasEquivalents then
   self.umlEquivalents()->forAll(er:UMLModelElementCh|
    er.schemaUnit()->notEmpty() and
    er.schemaUnit().rdbmsMappingKind() = MappingKind::IsIncluded and
    er.schemaUnit().includedInRdbms().schemaUnit() = self)
  else
    if umlMappingKind() = MappingKind::IsIncluded then
      self.includedInUml().schemaUnit()->notEmpty() and
      self.includedInUml().schemaUnit().rdbmsMappingKind() =
      MappingKind::HasEquivalents and
      self.includedInUml().schemaUnit().rdbmsEquivalents().schemaUnit()->
        includes(self)
    else
      false
    endif
  endif
```

3.2.4 Translation mapping constraints

Let $M = (MS_1, MS_2, \Sigma)$ be a translation mapping where MS_1 and MS_2 are instances of MOF. The translation mapping constraints Σ consists of exactly two constraints that are defined formally by the following OCL invariants:

```
context UMLModelElement inv completeMappingToRdbms:
  isSchemaUnit() and
  (rdbmsMappingKind() = MappingKind::HasEquivalents or
   rdbmsMappingKind() = MappingKind::IsIncluded) implies mappedToRdbms()

context RModelElement inv completeMappingToUml:
  isSchemaUnit() and
  (umlMappingKind() = MappingKind::HasEquivalents or
   umlMappingKind() = MappingKind::IsIncluded) implies mappedToUml()
```

3.2.5 Validating the model

The example consists on a sequence of steps that simulates different states of the model and the expected success or failure of constraints. It will create the instances of Simple UML metaschemas and Simple Rdbms metaschemas of Figure 21:

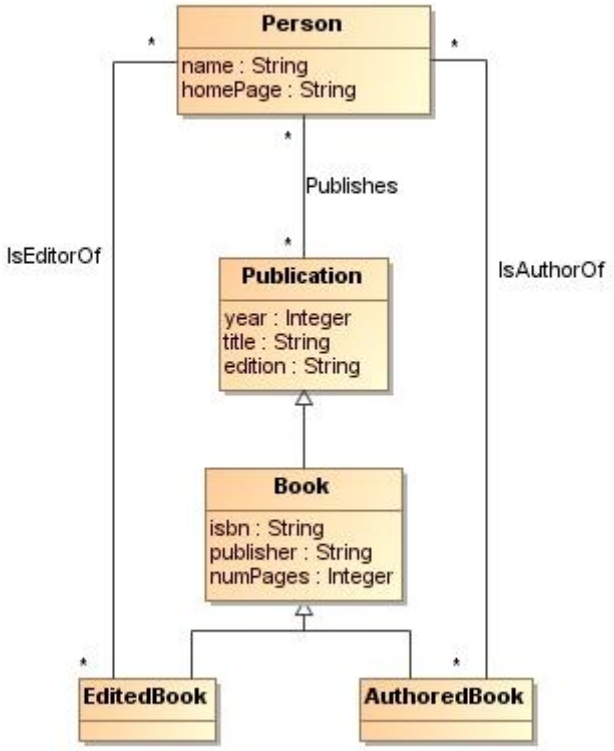

Fig. 21. Example of Instance of the Simple UML Metaschema

Fig 22 shows the same example shown in Fig.2, as instance of the RDBMS metaschema, represented in SQL.

```
  CREATE TABLE Person
   (Person_tid   NUMBER,
   name      VARCHAR,
    homepage   VARCHAR,
    CONSTRAINT Person_pk PRIMARY KEY (Person_tid));

  CREATE TABLE Publication
   (Publication_tid NUMBER,
   year      NUMBER,
    title      VARCHAR,
    edition      VARCHAR,
    CONSTRAIN Publication_pk PRIMARY KEY (Publication_tid));

  CREATE TABLE Publishes
   (Person_tid   NUMBER,
    Publication_tid   NUMBER,
    CONSTRAINT Publishes_pk PRIMARY KEY (Person_tid, Publication_tid),
    CONSTRAINT Person_tid_fk FOREIGN KEY (Person_tid) REFERENCES Person
    CONSTRAINT Publication_tid_fk FOREIGN KEY (Publication_tid) REFERENCES
Publication);

  CREATE TABLE Book
   (Book_tid   NUMBER,
      isbn       VARCHAR,
    publisher    VARCHAR,
    numpages   NUMBER,
```

```
    CONSTRAINT Book_pk PRIMARY KEY (Book_tid)
   CONSTRAINT Publication_tid_fk FOREING KEY (Book_tid) REFERENCES
Publication);

   CREATE TABLE EditedBook
   (EditedBook_tid  NUMBER,
   CONSTRAINT EditedBook_pk PRIMARY KEY (EditedBook_tid)
   CONSTRAINT Book_tid_fk FOREING KEY (EditedBook_tid) REFERENCES Book);

  CREATE TABLE AuthoredBook
   (AuthoredBook_tid  NUMBER,
   CONSTRAINT AuthoredBook_pk PRIMARY KEY (AuthoredBook_tid)
   CONSTRAINT Book_tid_fk FOREING KEY (AuthoredBook_tid) REFERENCES Book);

  CREATE TABLE IsEditorOf
   (Person_tid   NUMBER,
    EditedBook_tid  NUMBER,
    CONSTRAINT IsEditorOf_pk PRIMARY KEY (Person_tid, EditedBook_tid),
    CONSTRAINT Person_tid_fk FOREIGN KEY (Person_tid) REFERENCES Person
    CONSTRAINT EditedBook_tid_fk FOREIGN KEY (EditedBook_tid)
  REFERENCES EditedBook);

  CREATE TABLE IsAuthorOf
   (Person_tid   NUMBER,
    AuthoredBook_tid  NUMBER,
    CONSTRAINT IsAuthorOf_pk PRIMARY KEY (Person_tid, AuthoredBook_tid),
    CONSTRAINT Person_tid_fk FOREIGN KEY (Person_tid) REFERENCES Person
    CONSTRAINT AuthoredBook_tid_fk FOREIGN KEY (AuthoredBook_tid)
  REFERENCES AuthoredBook);
```

Fig. 22. Example of Instance of the Simple RDBMS Metaschema

The following scripts, simulates the creation of a package called 'DBLPFragment' with three instances of primitive data type (Integer, String and Boolean):

```
!create PackageUml:Package
!set PackageUml.name:= 'DBLPFragment'
!create IntegerUml:PrimitiveDataType
!set IntegerUml.name:= 'Integer'
!insert(PackageUml, IntegerUml) into Namespace_Elements
!create StringUml:PrimitiveDataType
!set StringUml.name:= 'String'
!insert(PackageUml, StringUml) into Namespace_Elements
!create BooleanUml:PrimitiveDataType
!set BooleanUml.name:= 'Boolean'
!insert(PackageUml, BooleanUml) into Namespace_Elements
```

The simulation of execution of the two operations calls rdbmsEquivalents() and includedInUml() explained above and the validation of the complete mapping is the following:

```
gen start UML2RDBMSEquivalents.assl rdbmsEquivalents()
gen start UML2RDBMSIncludedIn.assl includedInUml()
gen result accept
gen load C:\...\CompleteMapping.invs
check -d RModelElement::completeMappingToUml
check -d UMLModelElement::completeMappingToRdbms
```

The screenshot below (Figure 23) shows the object diagram created and the result of the evaluation of both invariants. An instance of SchemaCh and three instances of SqltypeCh have been created. The first invariant RModelElement::completeMappingToUML has succeed since there are no instances of RModelElement and

UMLModelElement::completeMappingToRdbms has failed since the four instances have no equivalent elements in the Rdbms schema.

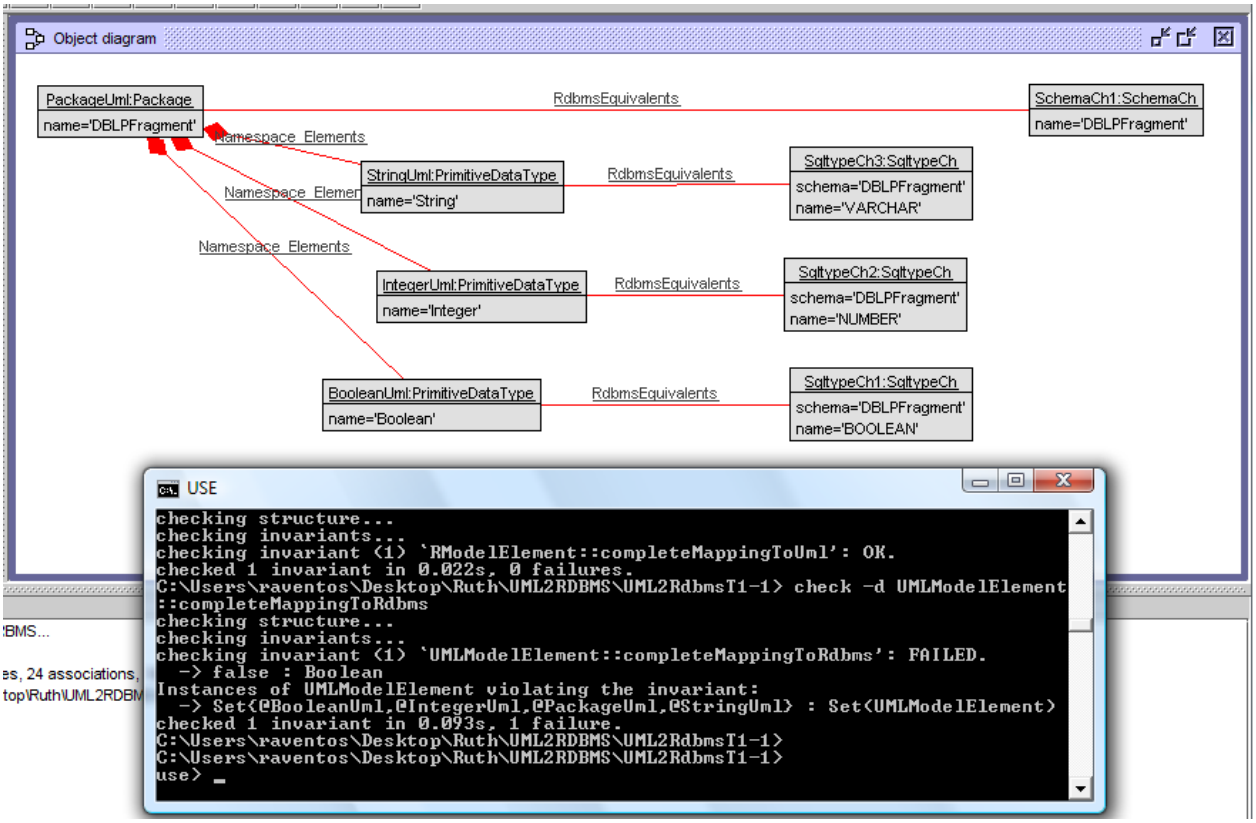

Fig. 23. Screenshot of the example (1).

Now, we simulate the creation of a schema called 'DBLPFragment' with three instances of sqltype (Varchar, Number and Boolean:

```
!create PackageUml:Package
!set PackageUml.name:= 'DBLPFragment'
!create IntegerUml:PrimitiveDataType
!set IntegerUml.name:= 'Integer'
!insert(PackageUml, IntegerUml) into Namespace_Elements
!create StringUml:PrimitiveDataType
!set StringUml.name:= 'String'
!insert(PackageUml, StringUml) into Namespace_Elements
!create BooleanUml:PrimitiveDataType
!set BooleanUml.name:= 'Boolean'
!insert(PackageUml, BooleanUml) into Namespace_Elements
```

The screenshot below (Figure 24) shows the result after the simulation of execution of the two operations calls rdbmsEquivalents() and includedInUml() and the validation of the complete mapping invariants. Now, an additional instance of PackageCh and three instances of PrimitiveDataType have been created. Both invariants have succeeded.

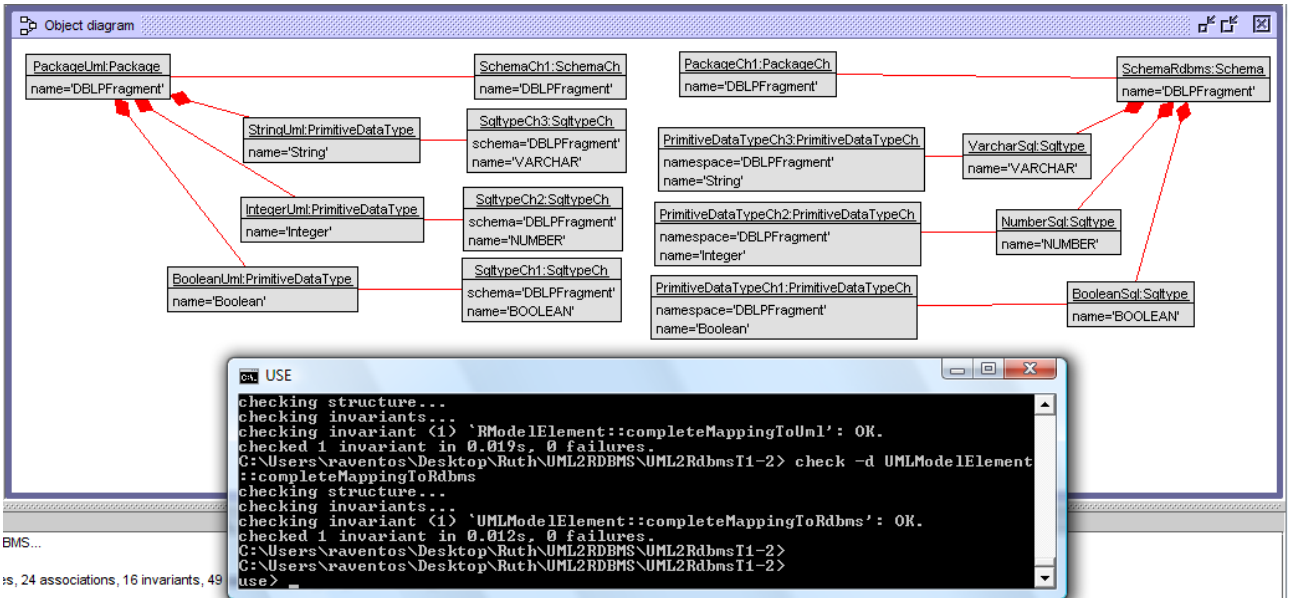

Fig. 24. Screenshot of the example (2).

Now, after simulating the creation of the five classes shown in figure with their attributes:

```
!create Class1:Class
!set Class1.name = 'Person'
!insert(PackageUml, Class1) into Namespace_Elements

!create Attribute1:Attribute
!set Attribute1.name = 'name'
!insert (Class1, Attribute1) into ClassAttribute
!insert(Attribute1, StringUml) into Attribute_Type

!create Attribute2:Attribute
!set Attribute2.name = 'homePage'
!insert (Class1, Attribute2) into ClassAttribute
!insert(Attribute2, StringUml) into Attribute_Type

!create Class2:Class
!set Class2.name = 'Publication'
!insert(PackageUml, Class2) into Namespace_Elements

!create Attribute3:Attribute
!set Attribute3.name = 'year'
!insert (Class2, Attribute3) into ClassAttribute
!insert(Attribute3, IntegerUml) into Attribute_Type

!create Attribute4:Attribute
!set Attribute4.name = 'title'
!insert (Class2, Attribute4) into ClassAttribute
!insert(Attribute4, StringUml) into Attribute_Type

!create Attribute5:Attribute
!set Attribute5.name = 'edition'
!insert (Class2, Attribute5) into ClassAttribute
!insert(Attribute5, StringUml) into Attribute_Type

!create Class3:Class
!set Class3.name = 'Book'
!insert(PackageUml, Class3) into Namespace_Elements
```

```
!create Attribute6:Attribute
!set Attribute6.name = 'isbn'
!insert (Class3, Attribute6) into ClassAttribute
!insert(Attribute6, StringUml) into Attribute_Type

!create Attribute7:Attribute
!set Attribute7.name = 'publisher'
!insert (Class3, Attribute7) into ClassAttribute
!insert(Attribute7, StringUml) into Attribute_Type

!create Attribute8:Attribute
!set Attribute8.name = 'numPages'
!insert (Class3, Attribute8) into ClassAttribute
!insert(Attribute8, IntegerUml) into Attribute_Type

!create Class4:Class
!set Class4.name = 'EditedBook'
!insert(PackageUml, Class4) into Namespace_Elements

!create Class5:Class
!set Class5.name = 'AuthoredBook'
!insert(PackageUml, Class5) into Namespace_Elements
```

The screenshot below (Figure 25) shows the object diagram created and the result of the evaluation of both invariants. Five instances of TableCh with their table key and key column have been created. For each attribute, an instance of ColumnCh has been also created. The first invariant RModelElement::completeMappingToUML has succeed since all instances of RModelElement are still mapped to UML but the UMLModelElement::completeMappingToRdbms has failed since all these new instance of UML have no equivalent elements in the Rdbms schema. Note that the instances of package, primitive data types, schema and sql types have been hidden in the object diagram.

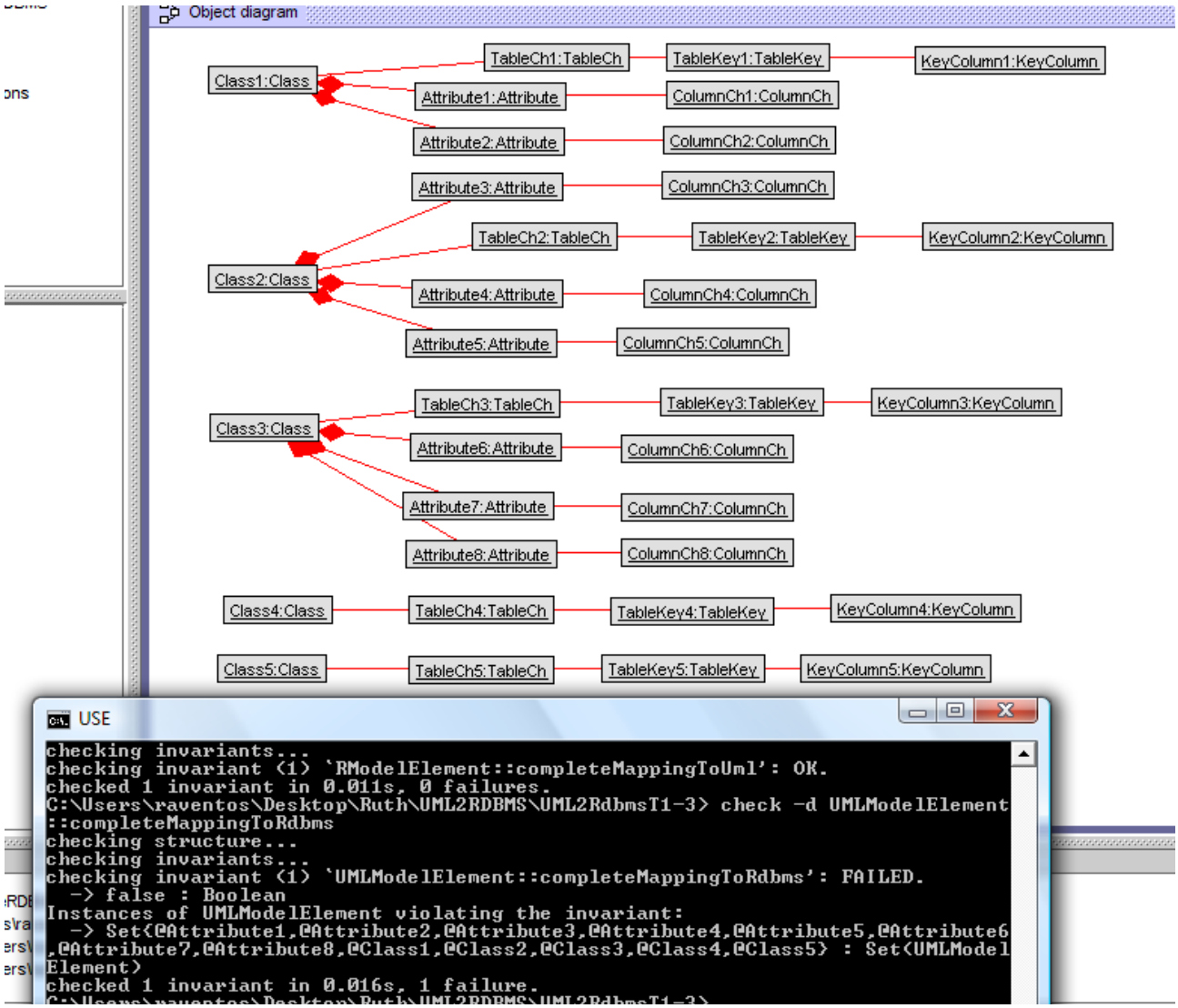

Fig. 25. Screenshot of the example (3).

After simulating the creation of the five tables with their columns shown in figure:

```
!create Table1:Table
!set Table1.name:= 'Person'
!insert(SchemaRdbms, Table1) into Schema_table
!create Table1_pk:Key
!set Table1_pk.name:= 'Person_pk'
!insert(Table1, Table1_pk) into Table_Key
!create IdTable1:Column
!set IdTable1.name:= 'Person_tid'
!insert(Table1_pk, IdTable1) into Key_Column
!insert(IdTable1, NumberSql) into Column_Type
!insert (Table1, IdTable1) into Table_Column

!create Column1:Column
!set Column1.name:= 'name'
!insert (Table1, Column1) into Table_Column
!insert(Column1, VarcharSql) into Column_Type

!create Column2:Column
!set Column2.name:= 'homePage'
!insert (Table1, Column2) into Table_Column
```

```
!insert(Column2, VarcharSql) into Column_Type

!create Table2:Table
!set Table2.name:= 'Publication'
!insert(SchemaRdbms, Table2) into Schema_table
!create Table2_pk:Key
!set Table2_pk.name:= 'Publication_pk'
!insert(Table2, Table2_pk) into Table_Key
!create IdTable2:Column
!set IdTable2.name:= 'Publication_tid'
!insert(Table2_pk, IdTable2) into Key_Column
!insert(IdTable2, NumberSql) into Column_Type
!insert (Table2, IdTable2) into Table_Column

!create Column3:Column
!set Column3.name:= 'year'
!insert (Table2, Column3) into Table_Column
!insert(Column3, NumberSql) into Column_Type

!create Column4:Column
!set Column4.name:= 'title'
!insert (Table2, Column4) into Table_Column
!insert(Column4, VarcharSql) into Column_Type

!create Column5:Column
!set Column5.name:= 'edition'
!insert (Table2, Column5) into Table_Column
!insert(Column5, VarcharSql) into Column_Type

!create Table3:Table
!set Table3.name:= 'Book'
!insert(SchemaRdbms, Table3) into Schema_table
!create Table3_pk:Key
!set Table3_pk.name:= 'Book_pk'
!insert(Table3, Table3_pk) into Table_Key
!create IdTable3:Column
!set IdTable3.name:= 'Book_tid'
!insert(Table3_pk, IdTable3) into Key_Column
!insert(IdTable3, NumberSql) into Column_Type
!insert (Table3, IdTable3) into Table_Column

!create Column6:Column
!set Column6.name:= 'isbn'
!insert (Table3, Column6) into Table_Column
!insert(Column6, VarcharSql) into Column_Type

!create Column7:Column
!set Column7.name:= 'publisher'
!insert (Table3, Column7) into Table_Column
!insert(Column7, VarcharSql) into Column_Type

!create Column8:Column
!set Column8.name:= 'numPages'
!insert (Table3, Column8) into Table_Column
!insert(Column8, NumberSql) into Column_Type

!create Table4:Table
!set Table4.name:= 'EditedBook'
!insert(SchemaRdbms, Table4) into Schema_table
```

```
!create Table4_pk:Key
!set Table4_pk.name:= 'EditedBook_pk'
!insert(Table4, Table4_pk) into Table_Key
!create IdTable4:Column
!set IdTable4.name:= 'EditedBook_tid'
!insert(Table4_pk, IdTable4) into Key_Column
!insert(IdTable4, NumberSql) into Column_Type
!insert (Table4, IdTable4) into Table_Column

!create Table5:Table
!set Table5.name:= 'AuthoredBook'
!insert(SchemaRdbms, Table5) into Schema_table
!create Table5_pk:Key
!set Table5_pk.name:= 'AuthoredBook_pk'
!insert(Table5, Table5_pk) into Table_Key
!create IdTable5:Column
!set IdTable5.name:= 'AuthoredBook_tid'
!insert(Table5_pk, IdTable5) into Key_Column
!insert(IdTable5, NumberSql) into Column_Type
!insert (Table5, IdTable5) into Table_Column
```

The screenshot below (Figure 26) shows the object diagram created and the result of the evaluation of both invariants. Five instances of ClassCh have been created. For each column that does not belong to a key, an instance of AttributeCh has been also created. Both invariants RModelElement::completeMappingToUML and UMLModelElement::completeMappingToRdbms has succeeded. Note that the instances of package, primitive data type, schema and sql type have been hidden in the object diagram.

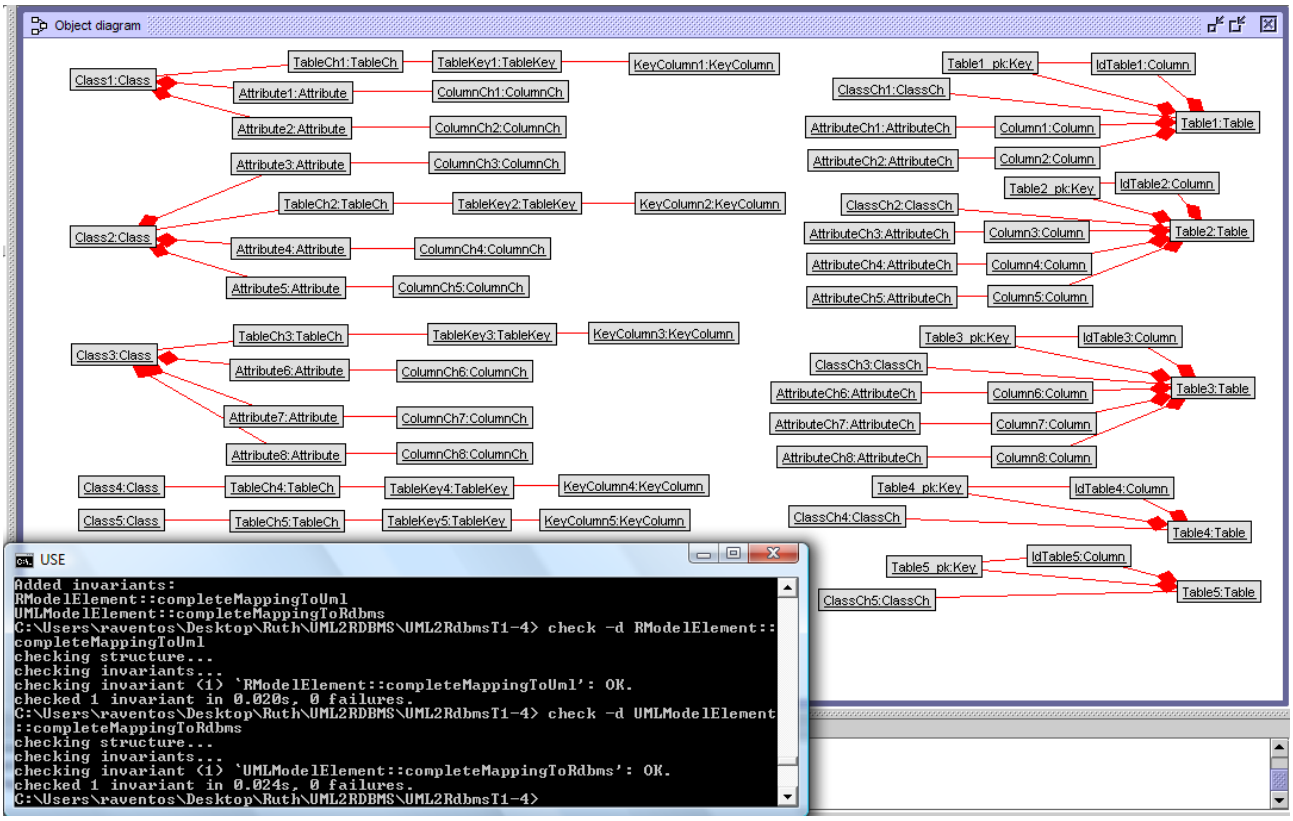

Fig. 26. Screenshot of the example (4).

Now, after the creation of the three instances of association and the three instances of generalization:

```
!create Association1:Association
!set Association1.name:= 'Publishes'
!insert(Class1, Association1) into Source_Reverse
!insert(Class2, Association1) into Destination_Forward
!insert(PackageUml, Association1) into Namespace_Elements

!create Association2:Association
!set Association2.name:= 'IsEditorOf'
!insert(Class1, Association2) into Source_Reverse
!insert(Class4, Association2) into Destination_Forward
!insert(PackageUml, Association2) into Namespace_Elements

!create Association3:Association
!set Association3.name:= 'IsAuthorOf'
!insert(Class1, Association3) into Source_Reverse
!insert(Class5, Association3) into Destination_Forward
!insert(PackageUml, Association3) into Namespace_Elements

!create Generalization1:Generalization
!insert(Class3, Generalization1) into Specific_GeneralizationOfSpecific
!insert(Class2, Generalization1) into General_Generalization
!insert(PackageUml, Generalization1) into Namespace_Elements

!create Generalization2:Generalization
!insert(Class4, Generalization2) into Specific_GeneralizationOfSpecific
!insert(Class3, Generalization2) into General_Generalization
!insert(PackageUml, Generalization2) into Namespace_Elements

!create Generalization3:Generalization
!insert(Class5, Generalization3) into Specific_GeneralizationOfSpecific
!insert(Class3, Generalization3) into General_Generalization
!insert(PackageUml, Generalization3) into Namespace_Elements
```

Now, the invariant UMLModelElement::completeMappingToRdbms has failed because the instances of association and generalization have no equivalents in the Rdbms schema (See Figure 27).

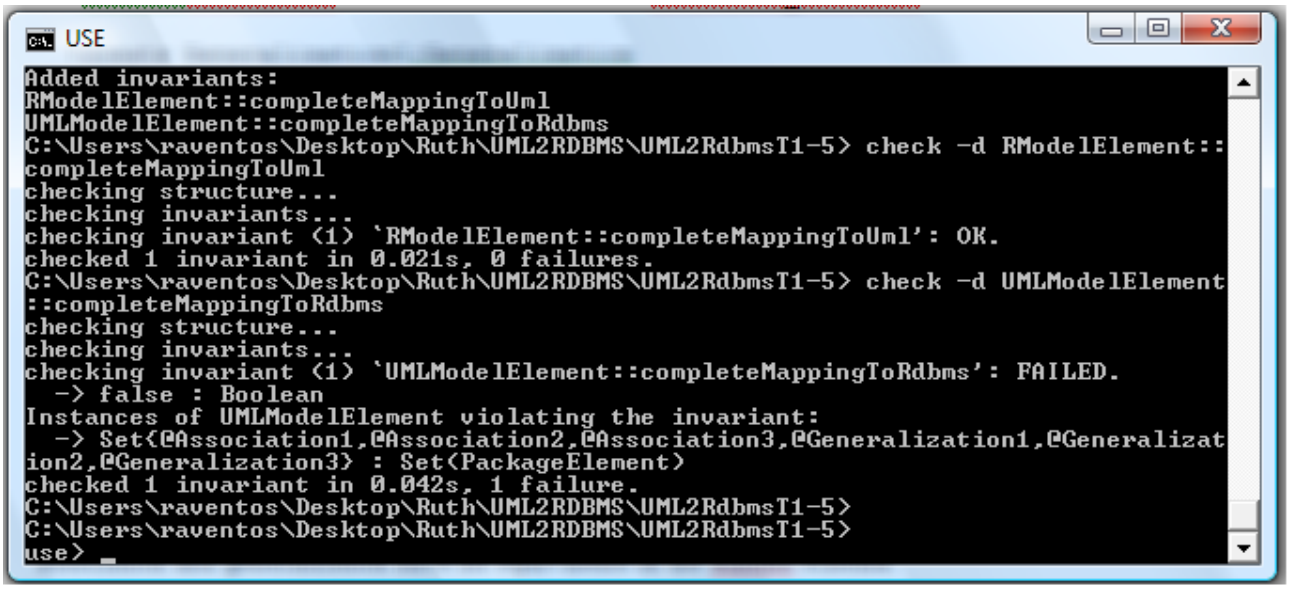

Fig. 27. Screenshot of the example (5).

Finally, after the simulation of creation of tables and foreign keys of Rdbms equivalents to the associations and generalizations:

```
!create Table6:Table
!set Table6.name:= 'Publishes'
!insert(SchemaRdbms, Table6) into Schema_table
!create Table6_pk:Key
!set Table6_pk.name:= 'Publishes_pk'
!insert(Table6, Table6_pk) into Table_Key
!create Id1Table6:Column
!set Id1Table6.name:= 'Person_tid'
!insert(Table6_pk, Id1Table6) into Key_Column
!insert(Id1Table6, NumberSql) into Column_Type
!insert (Table6, Id1Table6) into Table_Column
!create Id2Table6:Column
!set Id2Table6.name:= 'Publication_tid'
!insert(Table6_pk, Id2Table6) into Key_Column
!insert(Id2Table6, NumberSql) into Column_Type
!insert (Table6, Id2Table6) into Table_Column

!create ForeignKey1:ForeignKey
!set ForeignKey1.name:= 'Person_tid_fk'
!insert(Table6, ForeignKey1) into Table_ForeignKey
!insert(ForeignKey1,Table1_pk) into ForeignKey_Key
!insert(ForeignKey1,Id1Table6) into ForeignKey_Column

!create ForeignKey2:ForeignKey
!set ForeignKey2.name:= 'Publication_tid_fk'
!insert(Table6, ForeignKey2) into Table_ForeignKey
!insert(ForeignKey2,Table2_pk) into ForeignKey_Key
!insert(ForeignKey2,Id2Table6) into ForeignKey_Column

!create Table7:Table
!set Table7.name:= 'IsEditorOf'
!insert(SchemaRdbms, Table7) into Schema_table
!create Table7_pk:Key
!set Table7_pk.name:= 'IsEditorOf_pk'
!insert(Table7, Table7_pk) into Table_Key
!create Id1Table7:Column
!set Id1Table7.name:= 'Person_tid'
!insert(Table7_pk, Id1Table7) into Key_Column
!insert(Id1Table7, NumberSql) into Column_Type
!insert (Table7, Id1Table7) into Table_Column
!create Id2Table7:Column
!set Id2Table7.name:= 'EditedBook_tid'
!insert(Table7_pk, Id2Table7) into Key_Column
!insert(Id2Table7, NumberSql) into Column_Type
!insert (Table7, Id2Table7) into Table_Column

!create ForeignKey3:ForeignKey
!set ForeignKey3.name:= 'Person_tid_fk'
!insert(Table7, ForeignKey3) into Table_ForeignKey
!insert(ForeignKey3,Table1_pk) into ForeignKey_Key
!insert(ForeignKey3,Id1Table7) into ForeignKey_Column

!create ForeignKey4:ForeignKey
!set ForeignKey4.name:= 'EditedBook_tid_fk'
!insert(Table7, ForeignKey4) into Table_ForeignKey
!insert(ForeignKey4,Table4_pk) into ForeignKey_Key
```

```
!insert(ForeignKey4,Id2Table7) into ForeignKey_Column

!create Table8:Table
!set Table8.name:= 'IsAuthorOf'
!insert(SchemaRdbms, Table8) into Schema_table
!create Table8_pk:Key
!set Table8_pk.name:= 'IsAuthorOf_pk'
!insert(Table8, Table8_pk) into Table_Key
!create Id1Table8:Column
!set Id1Table8.name:= 'Person_tid'
!insert(Table8_pk, Id1Table8) into Key_Column
!insert(Id1Table8, NumberSql) into Column_Type
!insert (Table8, Id1Table8) into Table_Column
!create Id2Table8:Column
!set Id2Table8.name:= 'AuthoredBook_tid'
!insert(Table8_pk, Id2Table8) into Key_Column
!insert(Id2Table8, NumberSql) into Column_Type
!insert (Table8, Id2Table8) into Table_Column

!create ForeignKey5:ForeignKey
!set ForeignKey5.name:= 'Person_tid_fk'
!insert(Table8, ForeignKey5) into Table_ForeignKey
!insert(ForeignKey5,Table1_pk) into ForeignKey_Key
!insert(ForeignKey5,Id1Table8) into ForeignKey_Column

!create ForeignKey6:ForeignKey
!set ForeignKey6.name:= 'AuthoredBook_tid_fk'
!insert(Table8, ForeignKey6) into Table_ForeignKey
!insert(ForeignKey6,Table5_pk) into ForeignKey_Key
!insert(ForeignKey6,Id2Table8) into ForeignKey_Column

!create ForeignKey7:ForeignKey
!set ForeignKey7.name:= 'Publication_tid_fk'
!insert(Table3, ForeignKey7) into Table_ForeignKey
!insert(ForeignKey7,Table2_pk) into ForeignKey_Key
!insert(ForeignKey7,IdTable3) into ForeignKey_Column

!create ForeignKey8:ForeignKey
!set ForeignKey8.name:= 'Book_tid_fk'
!insert(Table4, ForeignKey8) into Table_ForeignKey
!insert(ForeignKey8,Table3_pk) into ForeignKey_Key
!insert(ForeignKey8,IdTable4) into ForeignKey_Column

!create ForeignKey9:ForeignKey
!set ForeignKey9.name:= 'Book_tid_fk'
!insert(Table5, ForeignKey9) into Table_ForeignKey
!insert(ForeignKey9,Table3_pk) into ForeignKey_Key
!insert(ForeignKey9,IdTable5) into ForeignKey_Column
```

The result of the evaluation of both mappings is OK as shown in the following snapshot (see Figure 28):

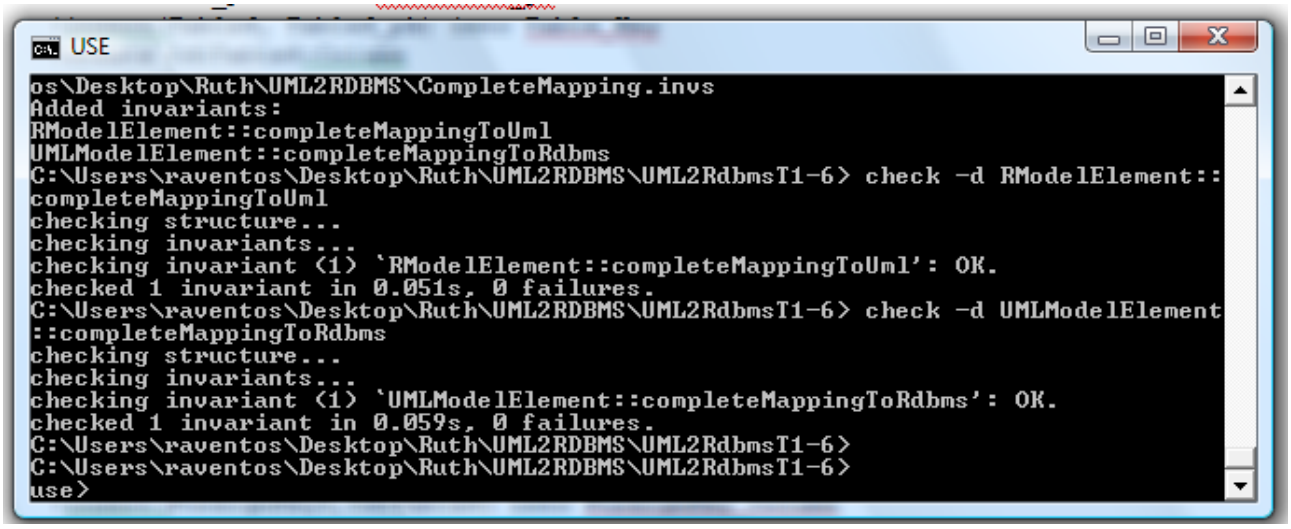

Fig. 28. Screenshot of the example (6).

3.3 Translating schemas

The operations defined in the previous sections may be use for the translation of schemas. Let $M = (MS_1,MS_2,\Sigma)$ be a mapping, and $S_1 = \{u_{1,1},...,u_{1,n}\}$ an instance of MS_1. The translation of S_1 into MS_2 is a schema $S_2 = \{u_{2,1},...,u_{2,m}\}$ such that $<S_1,S_2>$ is an instance of M. The translation of S_2 into MS_1 is defined similarly. The approach to the translation of a schema $S_1 = (u_{1,1},...,u_{1,n})$ consists in translating each of its schema units $u_{1,j}$ following the order given by the operation *predecessors*, starting with the units that have no predecessors.

The translation is done by applying the operation called *translateToS$_j$*() to the schema units.

3.3.1 The operation translateToS$_j$

An instance $u_{i,k}$ of *S$_i$Element* can be translated into S_j if it represents a schema unit whose mapping kind is *HasEquivalents* or *IsIncluded*. The effect of the operation must be that $u_{i,k}$ is mapped to S_j. This is captured by the simple following formal specification:

```
context UMLModelElement::translateToSimpleRDBMS()
  pre: isSchemaUnit() and
       (rdbmsMappingKind() = MappingKind::HasEquivalents or
         rdbmsMappingKind() = MappingKind::IsIncluded)
  post: mappedToSj()

context RModelElement::translateToSimpleUML()
  pre: isSchemaUnit() and (
       umlMappingKind() = MappingKind::HasEquivalents or
         umlMappingKind() = MappingKind::IsIncluded)
  post: mappedToSj()
```

There is no need to refine the specification of this operation in the subtypes of *S$_i$Element*. Concerning its implementation, the specification of *mappedToS$_j$* suggests a straightforward implementation using the methods of the operations *s$_j$Equivalents* and *isIncludedInS$_j$* defined in Section 3.3, and the methods of the operation *createUnit* defined below:

Create units of characterization object of Simple UML Elements

```
procedure CreateUnitOfPackageCh()
var s:Package;
 begin
  for el:PackageCh in [PackageCh.allInstances()->asSequence]
   begin
      s := Create( Package);
     [s].name := [el.name];
   end;
 end;

procedure CreateUnitOfPrimitiveDataTypeCh()
var d:PrimitiveDataType, s:Package;
 begin
 for el:PrimitiveDataTypeCh in [PrimitiveDataTypeCh.allInstances()->
       asSequence]
  begin
     d := Create(PrimitiveDataType);
     [d].name := [el.name];
     s := Any([Package.allInstances()->select(e:Package| e.name =
          el.namespace) -> asSequence]);
     Insert(Namespace_Elements, [s],[d]);
  end;
 end;

procedure CreateUnitOfClassCh()
var c:Class, s:Package;
begin
 for el:ClassCh in [ClassCh.allInstances()->asSequence]
  begin
     c := Create(Class);
     [c].name := [el.name];
     s := Any([Package.allInstances() -> select(e:Package|
          e.name = el.namespace) ->asSequence]);
     Insert(Namespace_Elements, [s],[c]);
   end;
 end;

procedure CreateUnitOfGeneralizationCh()
var g:Generalization, s:Package, c1:Class, c2:Class;
begin
 for el:GeneralizationCh in [GeneralizationCh.allInstances()->asSequence]
  begin
     g := Create(Generalization);
     s := Any([Package.allInstances()->select(e:Package|
          e.name = el.namespace) -> asSequence]);
     c1 := Any([Class.allInstances()->select(c:Class| c.name = el.general and
          c.namespace.name = el.namespace)->asSequence]);
     c2 := Any([Class.allInstances()->select(c:Class| c.name = el.specific and
          c.namespace.name = el.namespace)->asSequence]);
     Insert(Namespace_Elements, [s],[g]);
     Insert(General_Generalization, [c1],[g]);
     Insert(Specific_GeneralizationOfSpecific, [c2],[g]);
  end;
 end;

procedure CreateUnitOfAssociationCh()
var a:Association, s:Package, c1:Class, c2:Class;
```

```
begin
  for el:AssociationCh in [AssociationCh.allInstances()->asSequence]
  begin
      a := Create(Association);
      s := Any([Package.allInstances()->select(e:Package|
           e.name = el.namespace) -> asSequence]);
      c1 := Any([Class.allInstances()->select(c:Class| c.name = el.source and
           c.namespace.name = el.namespace)->asSequence]);
      c2 := Any([Class.allInstances()->select(c:Class| c.name = el.destination
           and c.namespace.name = el.namespace)->asSequence]);
      Insert(Namespace_Elements, [s],[a]);
      Insert(Source_Reverse, [c1],[a]);
      Insert(Destination_Forward, [c2],[a]);
   end;
  end;

procedure CreateUnitOfAttributeCh()
var c:Class, s:Package, a:Attribute, d:PrimitiveDataType;
begin
  for el:AttributeCh in [AttributeCh.allInstances()->asSequence]
   begin
      a:= Create(Attribute);
      [a].name := [el.name];
      c := Any([Class.allInstances()->select(e:Class|e.name = el.owner and
           e.namespace.name = el.namespace)->asSequence]);
      Insert(Class_Attribute, [c],[a]);
      d := Any([PrimitiveDataType.allInstances()->select(e:PrimitiveDataType|
           e.name = el.type and e.namespace.name = el.namespace)->asSequence]);
      Insert(Attribute_Type, [a],[d]);
    end;
  end;
```

Create units of characterization object of Simple RDBMS Elements

```
procedure CreateUnitOfSchemaCh()
var s:Schema;
begin
   for el:SchemaCh in [SchemaCh.allInstances()->asSequence]
   begin
      s := Create( Schema);
      [s].name := [el.name];
   end;
  end;

procedure CreateUnitOfSqltypeCh()
var d:Sqltype, s:Schema;
begin
  for el:SqltypeCh in [SqltypeCh.allInstances()->asSequence]
  begin
      d := Create( Sqltype);
      [d].name := [el.name];
      s := Any([Schema.allInstances()->select(e:Schema| e.name = el.schema)->
           asSequence]);
      Insert(Schema_sqltype, [s],[d]);
  end;
end;

procedure CreateUnitOfTableCh()
var t:Table,c:Column, d:Sqltype, s:Schema, k:Key;
```

```
begin
for el:TableCh in [TableCh.allInstances()->asSequence]
  begin
    t := Create(Table);
    [t].name := [el.name];
    s := Any([Schema.allInstances()->select(e:Schema| e.name = el.schema)->
        asSequence]);
    Insert(Schema_table, [s],[t]);
    for tk:TableKey in [el.tableKey->asSequence]
    begin
      k := Create(Key);
      [k].name := [tk.name];
      Insert(Table_Key, [t],[k]);
      for kc:KeyColumn in [tk.keyColumn->asSequence]
      begin
        c:= Create(Column);
        [c].name := [kc.name];
        Insert(Table_Column, [t],[c]);
        Insert(Key_Column, [k],[c]);
        d := Any([Sqltype.allInstances()->select(e:Sqltype|
              e.name = kc.type)->asSequence]);
        Insert(Column_Type, [c],[d]);
      end;
    end;
  end;
 end;

procedure CreateUnitOfColumnCh()
var t:Table,c:Column, d:Sqltype;
begin
  for el:ColumnCh in [ColumnCh.allInstances()->asSequence]
    begin
      c:= Create(Column);
      [c].name := [el.name];
      t := Any([Table.allInstances()->select(e:Table|e.name = el.owner)->
            asSequence]);
      Insert(Table_Column, [t],[c]);
      d := Any([Sqltype.allInstances()->select(e:Sqltype|
            e.name = el.type)->asSequence]);
      Insert(Column_Type, [c],[d]);
    end;
end;

procedure CreateUnitOfForeignKeyCh()
var t:Table, c:Column, f:ForeignKey, k:Key;
begin
 for el:ForeignKeyCh in [ForeignKeyCh.allInstances()->asSequence]
  begin
    f := Create(ForeignKey);
    [f].name := [el.name];
    t := Any([Table.allInstances()->select(e:Table| e.name = el.owner and
          e.schema.name = el.schema)->asSequence]);
    k := Any([Key.allInstances()->select(ke:Key|
          ke.owner.name = el.refersToTable)->asSequence]);
    Insert(Table_ForeignKey, [t], [f]);
    Insert(ForeignKey_Key,[f],[k]);
    for fkc:ForeignKeyColumn in [el.foreignKeyColumn->asSequence]
    begin
      c := Any([Column.allInstances()->select(co:Column|
```

```
            co.owner.name = el.owner and co.owner.schema.name = el.schema
            and co.name = fkc.name and co.type.name = fkc.type)
              ->asSequence]);
      Insert(ForeignKey_Column,[f],[c]);
    end;
  end;
end;
```

4. Example 3: Translation from the osCommerce System Relational schema to the osCommerce System ER schema.

The third example is the application of translation between ER metaschema and Relational metaschema, described in the first example, to a larger system. In particular, the osCommerce System specified as an instance of the Relational hasb been translated to the osCommerce System specified as an instance of the ER metaschema.

osCommerce is an e-commerce solution available as free software under the GNU General Public License. *osCommerce* project was started in March 2000 in Germany and since then, it has become the base of thousands of online stores around the world.

The instantiation of the osCommerce is based on the conceptual schema of the osCommerce system described in (Tort and Olivé 2007).

Following there is the Relational schema of the osCommerce database:

```
-------------------------------------------
-- Relational schema OsCommerce database
-------------------------------------------
--Data types:

!create Integer:RelationalDataType
!set Integer.name:='Integer'

!create Datetime:RelationalDataType
!set Datetime.name:='Datetime'

!create Date:RelationalDataType
!set Date.name:='Date'

!create Month:RelationalDataType
!set Month.name:='Month'

!create Float:RelationalDataType
!set Float.name:='Float'

!create Char1:RelationalDataType
!set Char1.name:='Char(1)'

!create Char2:RelationalDataType
!set Char2.name:='Char(2)'

!create Char3:RelationalDataType
!set Char3.name:='Char(3)'

!create Varchar10:RelationalDataType
!set Varchar10.name:='Varchar(10)'

!create Varchar12:RelationalDataType
!set Varchar12.name:='Varchar(12)'

!create Varchar32:RelationalDataType
!set Varchar32.name:='Varchar(32)'

!create Varchar40:RelationalDataType
!set Varchar40.name:='Varchar(40)'
```

```
!create Varchar48:RelationalDataType
!set Varchar48.name:='Varchar(48)'

!create Varchar64:RelationalDataType
!set Varchar64.name:='Varchar(64)'

!create Varchar96:RelationalDataType
!set Varchar96.name:='Varchar(96)'

!create Varchar128:RelationalDataType
!set Varchar128.name:='Varchar(128)'

!create Varchar255:RelationalDataType
!set Varchar255.name:='Varchar(255)'

!create Text:RelationalDataType
!set Text.name:='Text'


-----------------------------------------------
--TABLE: Administrators
-----------------------------------------------
!create Administrators:Table
!set Administrators.name:='Administrators'
---
!create administratorId:Column
!set administratorId.name:='administratorId'
!set administratorId.isKey:=true
!insert (Administrators,administratorId) into Table_Column
!insert (administratorId,Integer) into ColumnTyping
---
!create userName:Column
!set userName.name:='user_name'
!set userName.isKey:=false
!insert (Administrators,userName) into Table_Column
!insert (userName,Varchar32) into ColumnTyping
---
!create userPassword:Column
!set userPassword.name:='userPassword'
!set userPassword.isKey:=false
!insert (Administrators,userPassword) into Table_Column
!insert (userPassword,Varchar40) into ColumnTyping
-----------------------------------------------
--TABLE: Address_format
-----------------------------------------------
!create Address_format:Table
!set Address_format.name:='Address_format'
---
!create addressFormatId:Column
!set addressFormatId.name:='addressFormatId'
!set addressFormatId.isKey:=true
!insert (Address_format,addressFormatId) into Table_Column
!insert (addressFormatId,Integer) into ColumnTyping
---
!create address_format_name:Column
!set address_format_name.name:='address_format'
!set address_format_name.isKey:=false
!insert (Address_format,address_format_name) into Table_Column
```

```
!insert (address_format_name,Varchar128) into ColumnTyping
---
!create address_summary:Column
!set address_summary.name:='address_summary'
!set address_summary.isKey:=false
!insert (Address_format,address_summary) into Table_Column
!insert (address_summary,Varchar48) into ColumnTyping
------------------------------------------------
--TABLE: Address_Book
------------------------------------------------

!create Address_Book:Table
!set Address_Book.name:='Address_Book'

!create addressBookId:Column
!set addressBookId.name:='addressBookId'
!set addressBookId.isKey:=true
!insert (Address_Book,addressBookId) into Table_Column
!insert (addressBookId,Integer) into ColumnTyping
---
!create entry_gender:Column
!set entry_gender.name:='entry_gender'
!set entry_gender.isKey:=false
!insert (Address_Book,entry_gender) into Table_Column
!insert (entry_gender,Char1) into ColumnTyping
---
!create entry_company:Column
!set entry_company.name:='entry_company'
!set entry_company.isKey:=false
!insert (Address_Book,entry_company) into Table_Column
!insert (entry_company,Varchar32) into ColumnTyping
---
!create entry_firstname:Column
!set entry_firstname.name:='entry_firstname'
!set entry_firstname.isKey:=false
!insert (Address_Book,entry_firstname) into Table_Column
!insert (entry_firstname,Varchar32) into ColumnTyping
---
!create entry_lastname:Column
!set entry_lastname.name:='entry_lastname'
!set entry_lastname.isKey:=false
!insert (Address_Book,entry_lastname) into Table_Column
!insert (entry_lastname,Varchar32) into ColumnTyping
---
!create entry_street_address:Column
!set entry_street_address.name:='entry_street_address'
!set entry_street_address.isKey:=false
!insert (Address_Book,entry_street_address) into Table_Column
!insert (entry_street_address,Varchar64) into ColumnTyping
---
!create entry_suburb:Column
!set entry_suburb.name:='entry_suburb'
!set entry_suburb.isKey:=false
!insert (Address_Book,entry_suburb) into Table_Column
!insert (entry_suburb,Varchar32) into ColumnTyping
---
!create entry_postcode:Column
!set entry_postcode.name:='entry_postcode'
!set entry_postcode.isKey:=false
```

```
!insert (Address_Book,entry_postcode) into Table_Column
!insert (entry_postcode,Varchar10) into ColumnTyping
---
!create entry_city:Column
!set entry_city.name:='entry_city'
!set entry_city.isKey:=false
!insert (Address_Book,entry_city) into Table_Column
!insert (entry_city,Varchar32) into ColumnTyping
---
!create entry_state:Column
!set entry_state.name:='entry_state'
!set entry_state.isKey:=false
!insert (Address_Book,entry_state) into Table_Column
!insert (entry_state,Varchar32) into ColumnTyping
---
------------------------------------------------
--TABLE:Customers
------------------------------------------------
!create Customers:Table
!set Customers.name:='Customers'

!create customerId:Column
!set customerId.name:='customerId'
!set customerId.isKey:=true
!insert (Customers,customerId) into Table_Column
!insert (customerId,Integer) into ColumnTyping
---
!create customerGender:Column
!set customerGender.name:='customerGender'
!set customerGender.isKey:=false
!insert (Customers,customerGender) into Table_Column
!insert (customerGender,Char1) into ColumnTyping
---
!create customerFirstName:Column
!set customerFirstName.name:='customerFirstName'
!set customerFirstName.isKey:=false
!insert (Customers,customerFirstName) into Table_Column
!insert (customerFirstName,Varchar32) into ColumnTyping
---
!create customerLastName:Column
!set customerLastName.name:='customerLastName'
!set customerLastName.isKey:=false
!insert (Customers,customerLastName) into Table_Column
!insert (customerLastName,Varchar32) into ColumnTyping
---
!create customerDob:Column
!set customerDob.name:='DateOfBirth'
!set customerDob.isKey:=false
!insert (Customers,customerDob) into Table_Column
!insert (customerDob,Datetime) into ColumnTyping
---
!create customerEMail:Column
!set customerEMail.name:='EMail'
!set customerEMail.isKey:=false
!insert (Customers,customerEMail) into Table_Column
!insert (customerEMail,Varchar96) into ColumnTyping
---
!create customerTelephone:Column
!set customerTelephone.name:='Telephone'
```

```
!set customerTelephone.isKey:=false
!insert (Customers,customerTelephone) into Table_Column
!insert (customerTelephone,Varchar32) into ColumnTyping
---
!create customerFax:Column
!set customerFax.name:='Fax'
!set customerFax.isKey:=false
!insert (Customers,customerFax) into Table_Column
!insert (customerFax,Varchar32) into ColumnTyping
---
!create customerPassword:Column
!set customerPassword.name:='Password'
!set customerPassword.isKey:=false
!insert (Customers,customerPassword) into Table_Column
!insert (customerPassword,Varchar40) into ColumnTyping
---
!create customerNewsLetter:Column
!set customerNewsLetter.name:='NewsLetter'
!set customerNewsLetter.isKey:=false
!insert (Customers,customerNewsLetter) into Table_Column
!insert (customerNewsLetter,Char1) into ColumnTyping
-----------------------------------------------
--TABLE:Customers_info
-----------------------------------------------
!create CustomersInfo:Table
!set CustomersInfo.name:='CustomersInfo'

!create customerInfoId:Column
!set customerInfoId.name:='customerInfoId'
!set customerInfoId.isKey:=true
!insert (CustomersInfo,customerInfoId) into Table_Column
!insert (customerInfoId,Integer) into ColumnTyping
---
!create customerDateLastLogon:Column
!set customerDateLastLogon.name:='customerDateLastLogon'
!set customerDateLastLogon.isKey:=false
!insert (CustomersInfo,customerDateLastLogon) into Table_Column
!insert (customerDateLastLogon,Datetime) into ColumnTyping
---
!create customerDateAccountCreated:Column
!set customerDateAccountCreated.name:='customerDateAccountCreated'
!set customerDateAccountCreated.isKey:=false
!insert (CustomersInfo,customerDateAccountCreated) into Table_Column
!insert (customerDateAccountCreated,Datetime) into ColumnTyping
---
!create customerDateAccountModified:Column
!set customerDateAccountModified.name:='customerDateAccountModified'
!set customerDateAccountModified.isKey:=false
!insert (CustomersInfo,customerDateAccountModified) into Table_Column
!insert (customerDateAccountModified,Datetime) into ColumnTyping
---
!create numberOfLogons:Column
!set numberOfLogons.name:='numberOfLogons'
!set numberOfLogons.isKey:=false
!insert (CustomersInfo,numberOfLogons) into Table_Column
!insert (numberOfLogons,Integer) into ColumnTyping
---
!create globalProductNotification:Column
!set globalProductNotification.name:='globalProductNotification'
```

```
!set globalProductNotification.isKey:=false
!insert (CustomersInfo,globalProductNotification) into Table_Column
!insert (globalProductNotification,Char1) into ColumnTyping
---
-----------------------------------------------
---Association CustomerInfo (0..1) - Customer (1)
-----------------------------------------------

!create CustomerInfoToCustomer:Table
!set CustomerInfoToCustomer.name:='CustomerInfoToCustomer'
---
!create citc1:Column
!set citc1.name:='info_customerInfoId'
!set citc1.isKey:=true
!insert (CustomerInfoToCustomer,citc1) into Table_Column
!insert (citc1,Integer) into ColumnTyping
---
!create citc2:Column
!set citc2.name:='customer_customerId'
!set citc2.isKey:=true
!insert (CustomerInfoToCustomer,citc2) into Table_Column
!insert (citc2,Integer) into ColumnTyping
---
!create ForeignKeyctc1 : ForeignKey
!insert (CustomersInfo,ForeignKeyctc1) into Table_ForeignKey
!insert (ForeignKeyctc1,citc1) into ForeignKey_Column
---
!create ForeignKeyctc2 : ForeignKey
!insert (Customers,ForeignKeyctc2) into Table_ForeignKey
!insert (ForeignKeyctc2,citc2) into ForeignKey_Column
-----------------------------------------------
--TABLE:Countries
-----------------------------------------------
!create Countries:Table
!set Countries.name:='Countries'

!create countryId:Column
!set countryId.name:='countryId'
!set countryId.isKey:=true
!insert (Countries,countryId) into Table_Column
!insert (countryId,Integer) into ColumnTyping
---
!create countries_name:Column
!set countries_name.name:='countries_name'
!set countries_name.isKey:=false
!insert (Countries,countries_name) into Table_Column
!insert (countries_name,Varchar64) into ColumnTyping
---
!create countries_iso_code_2:Column
!set countries_iso_code_2.name:='countries_iso_code_2'
!set countries_iso_code_2.isKey:=false
!insert (Countries,countries_iso_code_2) into Table_Column
!insert (countries_iso_code_2,Char2) into ColumnTyping
---
!create countries_iso_code_3:Column
!set countries_iso_code_3.name:='countries_iso_code_3'
!set countries_iso_code_3.isKey:=false
!insert (Countries,countries_iso_code_3) into Table_Column
!insert (countries_iso_code_3,Char3) into ColumnTyping
```

```
-----------------------------------------------
 ---Association Country (*) - Address_format (1)
-----------------------------------------------

!create CountryToAddressFormat:Table
!set CountryToAddressFormat.name:='CountryToAddressFormat'
---
!create cta1:Column
!set cta1.name:='country_countryId'
!set cta1.isKey:=true
!insert (CountryToAddressFormat,cta1) into Table_Column
!insert (cta1,Integer) into ColumnTyping
---
!create cta2:Column
!set cta2.name:='addressFormat_addressFormatId'
!set cta2.isKey:=true
!insert (CountryToAddressFormat,cta2) into Table_Column
!insert (cta2,Integer) into ColumnTyping
---
!create ForeignKey1 : ForeignKey
!insert (Countries,ForeignKey1) into Table_ForeignKey
!insert (ForeignKey1,cta1) into ForeignKey_Column
---
!create ForeignKey2 : ForeignKey
!insert (Address_format,ForeignKey2) into Table_ForeignKey
!insert (ForeignKey2,cta2) into ForeignKey_Column

-----------------------------------------------
--Association Address_book (*) - Country (1)
-----------------------------------------------

!create AddressBookToCountry:Table
!set AddressBookToCountry.name:='AddressBookToCountry'
---
!create atc1:Column
!set atc1.name:='addBook_addressBookId'
!set atc1.isKey:=true
!insert (AddressBookToCountry,atc1) into Table_Column
!insert (atc1,Integer) into ColumnTyping
---
!create atc2:Column
!set atc2.name:='country_countryId'
!set atc2.isKey:=true
!insert (AddressBookToCountry,atc2) into Table_Column
!insert (atc2,Integer) into ColumnTyping
---
!create ForeignKey3:ForeignKey
!insert (Address_Book,ForeignKey3) into Table_ForeignKey
!insert (ForeignKey3,atc1) into ForeignKey_Column
---
!create ForeignKey4: ForeignKey
!insert (Countries,ForeignKey4) into Table_ForeignKey
!insert (ForeignKey4,atc2) into ForeignKey_Column

-----------------------------------------------
---Association Address_book (*) - Customer (1)
-----------------------------------------------

!create AddressBookToCustomer:Table
```

```
!set AddressBookToCustomer.name:='AddressBookToCustomer'

!create atcu1:Column
!set atcu1.name:='addBook_addressBookId'
!set atcu1.isKey:=true
!insert (AddressBookToCustomer,atcu1) into Table_Column
!insert (atcu1,Integer) into ColumnTyping

!create atcu2:Column
!set atcu2.name:='customer_customerId'
!set atcu2.isKey:=true
!insert (AddressBookToCustomer,atcu2) into Table_Column
!insert (atcu2,Integer) into ColumnTyping

!create ForeignKeyatcu1:ForeignKey
!insert (Address_Book,ForeignKeyatcu1) into Table_ForeignKey
!insert (ForeignKeyatcu1,atcu1) into ForeignKey_Column

!create ForeignKeyatcu2: ForeignKey
!insert (Customers,ForeignKeyatcu2) into Table_ForeignKey
!insert (ForeignKeyatcu2,atcu2) into ForeignKey_Column
-------------------------------------------------------------------------
------------
---Association Address_book (defaultAddress,1) - Customer
(customerOfDefaultAddress,0..1)
-------------------------------------------------------------------------
-------------

!create DefaultAddressBookToCustomer:Table
!set DefaultAddressBookToCustomer.name:='DefaultAddressBookToCustomer'

!create datc1:Column
!set datc1.name:='defAdd_addressBookId'
!set datc1.isKey:=true
!insert (DefaultAddressBookToCustomer,datc1) into Table_Column
!insert (datc1,Integer) into ColumnTyping

!create datc2:Column
!set datc2.name:='custOfDefAdd_customerId'
!set datc2.isKey:=true
!insert (DefaultAddressBookToCustomer,datc2) into Table_Column
!insert (datc2,Integer) into ColumnTyping

!create ForeignKeydatc1:ForeignKey
!insert (Address_Book,ForeignKeydatc1) into Table_ForeignKey
!insert (ForeignKeydatc1,datc1) into ForeignKey_Column

!create ForeignKeydatc2:ForeignKey
!insert (Customers,ForeignKeydatc2) into Table_ForeignKey
!insert (ForeignKeydatc2,datc2) into ForeignKey_Column
------------------------------
--TABLE: Zones
------------------------------
!create Zones:Table
!set Zones.name:='Zones'
---
!create zoneId:Column
!set zoneId.name:='zoneId'
!set zoneId.isKey:=true
```

```
!insert (Zones,zoneId) into Table_Column
!insert (zoneId,Integer) into ColumnTyping
---
!create zone_name:Column
!set zone_name.name:='zone_name'
!set zone_name.isKey:=false
!insert (Zones,zone_name) into Table_Column
!insert (zone_name,Varchar32) into ColumnTyping
---
!create zone_code:Column
!set zone_code.name:='zone_code'
!set zone_code.isKey:=false
!insert (Zones,zone_code) into Table_Column
!insert (zone_code,Varchar32) into ColumnTyping
-----------------------------------------
--Association Zone (*) - Country (1)
-----------------------------------------
!create ZoneToCountry:Table
!set ZoneToCountry.name:='ZoneToCountry'
---
!create ztc1:Column
!set ztc1.name:='zone_zoneId'
!set ztc1.isKey:=true
!insert (ZoneToCountry,ztc1) into Table_Column
!insert (ztc1,Integer) into ColumnTyping
---
!create ztc2:Column
!set ztc2.name:='country_countryId'
!set ztc2.isKey:=true
!insert (ZoneToCountry,ztc2) into Table_Column
!insert (ztc2,Integer) into ColumnTyping
---
!create ForeignKeyztc1:ForeignKey
!insert (Zones,ForeignKeyztc1) into Table_ForeignKey
!insert (ForeignKeyztc1,ztc1) into ForeignKey_Column
---
!create ForeignKeyztc2: ForeignKey
!insert (Countries,ForeignKeyztc2) into Table_ForeignKey
!insert (ForeignKeyztc2,ztc2) into ForeignKey_Column
-----------------------------------------
--Association Address_book (*) - Zone (1)
-----------------------------------------
!create AddressBookToZone:Table
!set AddressBookToZone.name:='AddressBookToZone'
---
!create atz1:Column
!set atz1.name:='addBook_addressBookId'
!set atz1.isKey:=true
!insert (AddressBookToZone,atz1) into Table_Column
!insert (atz1,Integer) into ColumnTyping
---
!create atz2:Column
!set atz2.name:='zone_zoneId'
!set atz2.isKey:=true
!insert (AddressBookToZone,atz2) into Table_Column
!insert (atz2,Integer) into ColumnTyping
---
!create ForeignKeyatz1:ForeignKey
!insert (Address_Book,ForeignKeyatz1) into Table_ForeignKey
```

```
!insert (ForeignKeyatz1,atz1) into ForeignKey_Column
---
!create ForeignKeyatz2: ForeignKey
!insert (Zones,ForeignKeyatz2) into Table_ForeignKey
!insert (ForeignKeyatz2,atz2) into ForeignKey_Column
---
------------------------------------------------
--TABLE:Currencies
------------------------------------------------
!create Currencies:Table
!set Currencies.name:='Currencies'

!create currencyId:Column
!set currencyId.name:='currencyId'
!set currencyId.isKey:=true
!insert (Currencies,currencyId) into Table_Column
!insert (currencyId,Integer) into ColumnTyping
---
!create currencyTitle:Column
!set currencyTitle.name:='currencyTitle'
!set currencyTitle.isKey:=false
!insert (Currencies,currencyTitle) into Table_Column
!insert (currencyTitle,Varchar32) into ColumnTyping
---
!create currencyCode:Column
!set currencyCode.name:='currencyCode'
!set currencyCode.isKey:=false
!insert (Currencies,currencyCode) into Table_Column
!insert (currencyCode,Char3) into ColumnTyping
---
!create currencySymbolLeft:Column
!set currencySymbolLeft.name:='currencySymbolLeft'
!set currencySymbolLeft.isKey:=false
!insert (Currencies,currencySymbolLeft) into Table_Column
!insert (currencySymbolLeft,Varchar12) into ColumnTyping
---
!create currencySymbolRight:Column
!set currencySymbolRight.name:='currencySymbolRight'
!set currencySymbolRight.isKey:=false
!insert (Currencies,currencySymbolRight) into Table_Column
!insert (currencySymbolRight,Varchar12) into ColumnTyping
---
!create decimalPoint:Column
!set decimalPoint.name:='decimalPoint'
!set decimalPoint.isKey:=false
!insert (Currencies,decimalPoint) into Table_Column
!insert (decimalPoint,Char1) into ColumnTyping
---
!create thousandsPoint:Column
!set thousandsPoint.name:='thousandsPoint'
!set thousandsPoint.isKey:=false
!insert (Currencies,thousandsPoint) into Table_Column
!insert (thousandsPoint,Char1) into ColumnTyping
---
!create decimalPlaces:Column
!set decimalPlaces.name:='decimalPlaces'
!set decimalPlaces.isKey:=false
!insert (Currencies,decimalPlaces) into Table_Column
!insert (decimalPlaces,Char1) into ColumnTyping
```

```
---
!create value:Column
!set value.name:='value'
!set value.isKey:=false
!insert (Currencies,value) into Table_Column
!insert (value,Float) into ColumnTyping
---
!create lastUpdated:Column
!set lastUpdated.name:='lastUpdated'
!set lastUpdated.isKey:=false
!insert (Currencies,lastUpdated) into Table_Column
!insert (lastUpdated,Datetime) into ColumnTyping
---
------------------------------------------------
--TABLE:Banners
------------------------------------------------
!create Banners:Table
!set Banners.name:='Banners'

!create bannerId:Column
!set bannerId.name:='bannerId'
!set bannerId.isKey:=true
!insert (Banners,bannerId) into Table_Column
!insert (bannerId,Integer) into ColumnTyping
---
!create bannerTitle:Column
!set bannerTitle.name:='bannerTitle'
!set bannerTitle.isKey:=false
!insert (Banners,bannerTitle) into Table_Column
!insert (bannerTitle,Varchar64) into ColumnTyping
---
!create bannerUrl:Column
!set bannerUrl.name:='bannerUrl'
!set bannerUrl.isKey:=false
!insert (Banners,bannerUrl) into Table_Column
!insert (bannerUrl,Varchar255) into ColumnTyping
---
!create bannerImage:Column
!set bannerImage.name:='bannerImage'
!set bannerImage.isKey:=false
!insert (Banners,bannerImage) into Table_Column
!insert (bannerImage,Varchar64) into ColumnTyping
---
!create bannerGroup:Column
!set bannerGroup.name:='bannerGroup'
!set bannerGroup.isKey:=false
!insert (Banners,bannerGroup) into Table_Column
!insert (bannerGroup,Varchar10) into ColumnTyping
---
!create bannerHTML:Column
!set bannerHTML.name:='bannerHTML'
!set bannerHTML.isKey:=false
!insert (Banners,bannerHTML) into Table_Column
!insert (bannerHTML,Text) into ColumnTyping
---
!create expiresImpressions:Column
!set expiresImpressions.name:='ExpiresImpressions'
!set expiresImpressions.isKey:=false
!insert (Banners,expiresImpressions) into Table_Column
```

```
!insert (expiresImpressions,Integer) into ColumnTyping
---
!create dateScheduled:Column
!set dateScheduled.name:='DateScheduled'
!set dateScheduled.isKey:=false
!insert (Banners,dateScheduled) into Table_Column
!insert (dateScheduled,Datetime) into ColumnTyping
---
!create bannerDateAdded:Column
!set bannerDateAdded.name:='BannerDateAdded'
!set bannerDateAdded.isKey:=false
!insert (Banners,bannerDateAdded) into Table_Column
!insert (bannerDateAdded,Datetime) into ColumnTyping
---
!create bannerDateStatusChange:Column
!set bannerDateStatusChange.name:='BannerDateStatusChange'
!set bannerDateStatusChange.isKey:=false
!insert (Banners,bannerDateStatusChange) into Table_Column
!insert (bannerDateStatusChange,Datetime) into ColumnTyping
---
!create bannerStatus:Column
!set bannerStatus.name:='BannerStatus'
!set bannerStatus.isKey:=false
!insert (Banners,bannerStatus) into Table_Column
!insert (bannerStatus,Integer) into ColumnTyping
-----------------------------------------------
--TABLE:BannersHistory
-----------------------------------------------
!create BannersHistory:Table
!set BannersHistory.name:='BannersHistory'

!create bannerHistoryId:Column
!set bannerHistoryId.name:='bannerHistoryId'
!set bannerHistoryId.isKey:=true
!insert (BannersHistory,bannerHistoryId) into Table_Column
!insert (bannerHistoryId,Integer) into ColumnTyping
---
!create bannerShown:Column
!set bannerShown.name:='BannerShown'
!set bannerShown.isKey:=false
!insert (BannersHistory,bannerShown) into Table_Column
!insert (bannerShown,Integer) into ColumnTyping
---
!create bannerClicked:Column
!set bannerClicked.name:='BannerClicked'
!set bannerClicked.isKey:=false
!insert (BannersHistory,bannerClicked) into Table_Column
!insert (bannerClicked,Integer) into ColumnTyping
---
!create bannerHistoryDate:Column
!set bannerHistoryDate.name:='BannerHistoryDate'
!set bannerHistoryDate.isKey:=false
!insert (BannersHistory,bannerHistoryDate) into Table_Column
!insert (bannerHistoryDate,Integer) into ColumnTyping
-----------------------------------------
--Association BannersHistory (*) - Banners (1)
-----------------------------------------
!create BannerHistoryToBanner:Table
!set BannerHistoryToBanner.name:='BannerHistoryToBanner'
```

```
---
!create bhtb1:Column
!set bhtb1.name:='bannerHistory_bannerHistoryId'
!set bhtb1.isKey:=true
!insert (BannerHistoryToBanner,bhtb1) into Table_Column
!insert (bhtb1,Integer) into ColumnTyping
---
!create bhtb2:Column
!set bhtb2.name:='banner_bannerId'
!set bhtb2.isKey:=true
!insert (BannerHistoryToBanner,bhtb2) into Table_Column
!insert (bhtb2,Integer) into ColumnTyping
---
!create ForeignKeybhtb1:ForeignKey
!insert (BannersHistory,ForeignKeybhtb1) into Table_ForeignKey
!insert (ForeignKeybhtb1,bhtb1) into ForeignKey_Column
---
!create ForeignKeybhtb2:ForeignKey
!insert (Banners,ForeignKeybhtb2) into Table_ForeignKey
!insert (ForeignKeybhtb2,bhtb2) into ForeignKey_Column
---
------------------------------------------------
--TABLE:Languages
------------------------------------------------
!create Languages:Table
!set Languages.name:='Languages'

!create languageId:Column
!set languageId.name:='languageId'
!set languageId.isKey:=true
!insert (Languages,languageId) into Table_Column
!insert (languageId,Integer) into ColumnTyping
---
!create languageName:Column
!set languageName.name:='LanguageName'
!set languageName.isKey:=false
!insert (Languages,languageName) into Table_Column
!insert (languageName,Varchar32) into ColumnTyping
---
!create languageCode:Column
!set languageCode.name:='languageCode'
!set languageCode.isKey:=false
!insert (Languages,languageCode) into Table_Column
!insert (languageCode,Char2) into ColumnTyping
---
!create languageImage:Column
!set languageImage.name:='languageImage'
!set languageImage.isKey:=false
!insert (Languages,languageImage) into Table_Column
!insert (languageImage,Varchar64) into ColumnTyping
---
!create languageDirectory:Column
!set languageDirectory.name:='languageDirectory'
!set languageDirectory.isKey:=false
!insert (Languages,languageDirectory) into Table_Column
!insert (languageDirectory,Varchar32) into ColumnTyping
---
!create languageSortOrder:Column
!set languageSortOrder.name:='LanguageSortOrder'
```

```
!set languageSortOrder.isKey:=false
!insert (Languages,languageSortOrder) into Table_Column
!insert (languageSortOrder,Integer) into ColumnTyping
---

------------------------------------------------
--TABLE:Counter
------------------------------------------------
!create Counter:Table
!set Counter.name:='Counter'

!create startDate:Column
!set startDate.name:='startDate'
!set startDate.isKey:=true
!insert (Counter,startDate) into Table_Column
!insert (startDate,Date) into ColumnTyping
---
!create counterValue:Column
!set counterValue.name:='counterValue'
!set counterValue.isKey:=false
!insert (Counter,counterValue) into Table_Column
!insert (counterValue,Integer) into ColumnTyping
---
------------------------------------------------
--TABLE:CounterHistory
------------------------------------------------
!create CounterHistory:Table
!set CounterHistory.name:='CounterHistory'

!create monthHistory:Column
!set monthHistory.name:='startDate'
!set monthHistory.isKey:=true
!insert (CounterHistory,monthHistory) into Table_Column
!insert (monthHistory,Month) into ColumnTyping
---
!create counterInMonth:Column
!set counterInMonth.name:='counterInMonth'
!set counterInMonth.isKey:=false
!insert (CounterHistory,counterInMonth) into Table_Column
!insert (counterInMonth,Integer) into ColumnTyping
---
------------------------------------------------
--TABLE:ConfigurationGroup
------------------------------------------------
!create ConfigurationGroup:Table
!set ConfigurationGroup.name:='ConfigurationGroup'

!create configurationGroupId:Column
!set configurationGroupId.name:='configurationGroupId'
!set configurationGroupId.isKey:=true
!insert (ConfigurationGroup,configurationGroupId) into Table_Column
!insert (configurationGroupId,Integer) into ColumnTyping
---
!create configurationGroupTitle:Column
!set configurationGroupTitle.name:='configurationGroupTitle'
!set configurationGroupTitle.isKey:=false
!insert (ConfigurationGroup,configurationGroupTitle) into Table_Column
!insert (configurationGroupTitle,Varchar64) into ColumnTyping
---
```

```
!create configurationGroupDescription:Column
!set configurationGroupDescription.name:='configurationGroupDescription'
!set configurationGroupDescription.isKey:=false
!insert (ConfigurationGroup,configurationGroupDescription) into Table_Column
!insert (configurationGroupDescription,Varchar255) into ColumnTyping
---
!create configurationGroupSortOrder:Column
!set configurationGroupSortOrder.name:='configurationGroupSortOrder'
!set configurationGroupSortOrder.isKey:=false
!insert (ConfigurationGroup,configurationGroupSortOrder) into Table_Column
!insert (configurationGroupSortOrder,Integer) into ColumnTyping
---
!create configurationVisible:Column
!set configurationVisible.name:='configurationVisible'
!set configurationVisible.isKey:=false
!insert (ConfigurationGroup,configurationVisible) into Table_Column
!insert (configurationVisible,Integer) into ColumnTyping
---

------------------------------------------------
--TABLE:Configuration
------------------------------------------------
!create Configuration:Table
!set Configuration.name:='Configuration'

!create configurationId:Column
!set configurationId.name:='configurationId'
!set configurationId.isKey:=true
!insert (Configuration,configurationId) into Table_Column
!insert (configurationId,Integer) into ColumnTyping
---
!create configurationTitle:Column
!set configurationTitle.name:='configurationTitle'
!set configurationTitle.isKey:=false
!insert (Configuration,configurationTitle) into Table_Column
!insert (configurationTitle,Varchar255) into ColumnTyping
---
!create configurationKey:Column
!set configurationKey.name:='configurationKey'
!set configurationKey.isKey:=false
!insert (Configuration,configurationKey) into Table_Column
!insert (configurationKey,Varchar255) into ColumnTyping
---
!create configurationValue:Column
!set configurationValue.name:='configurationValue'
!set configurationValue.isKey:=false
!insert (Configuration,configurationValue) into Table_Column
!insert (configurationValue,Varchar255) into ColumnTyping
---
!create configurationDescription:Column
!set configurationDescription.name:='configurationDescription'
!set configurationDescription.isKey:=false
!insert (Configuration,configurationDescription) into Table_Column
!insert (configurationDescription,Varchar255) into ColumnTyping
---
!create useFunction:Column
!set useFunction.name:='useFunction'
!set useFunction.isKey:=false
!insert (Configuration,useFunction) into Table_Column
```

```
!insert (useFunction,Varchar255) into ColumnTyping
---
!create setFunction:Column
!set setFunction.name:='setFunction'
!set setFunction.isKey:=false
!insert (Configuration,setFunction) into Table_Column
!insert (setFunction,Varchar255) into ColumnTyping
---
!create confSortOrder:Column
!set confSortOrder.name:='confSortOrder'
!set confSortOrder.isKey:=false
!insert (Configuration,confSortOrder) into Table_Column
!insert (confSortOrder,Varchar255) into ColumnTyping
---
!create confDateAdded:Column
!set confDateAdded.name:='confDateAdded'
!set confDateAdded.isKey:=false
!insert (Configuration,confDateAdded) into Table_Column
!insert (confDateAdded,Datetime) into ColumnTyping
---
!create confLastModified:Column
!set confLastModified.name:='confLastModified'
!set confLastModified.isKey:=false
!insert (Configuration,confLastModified) into Table_Column
!insert (confLastModified,Datetime) into ColumnTyping
---
-----------------------------------------
--Association Configuration(*) - ConfigurationGroup(1)
-----------------------------------------
!create ConfToConfGroup:Table
!set ConfToConfGroup.name:='ConfToConfGroup'
---
!create ctcg1:Column
!set ctcg1.name:='configuration_configurationId'
!set ctcg1.isKey:=true
!insert (ConfToConfGroup,ctcg1) into Table_Column
!insert (ctcg1,Integer) into ColumnTyping
---
!create ctcg2:Column
!set ctcg2.name:='configurationGroup_configurationGroupId'
!set ctcg2.isKey:=true
!insert (ConfToConfGroup,ctcg2) into Table_Column
!insert (ctcg2,Integer) into ColumnTyping
---
!create ForeignKeyctcg1:ForeignKey
!insert (Configuration,ForeignKeyctcg1) into Table_ForeignKey
!insert (ForeignKeyctcg1,ctcg1) into ForeignKey_Column
---
!create ForeignKeyctcg2:ForeignKey
!insert (ConfigurationGroup,ForeignKeyctcg2) into Table_ForeignKey
!insert (ForeignKeyctcg2,ctcg2) into ForeignKey_Column
---
-----------------------------------------------
--TABLE:Categories
-----------------------------------------------
!create Categories:Table
!set Categories.name:='Categories'

!create categoryId:Column
```

```
!set categoryId.name:='categoryId'
!set categoryId.isKey:=true
!insert (Categories,categoryId) into Table_Column
!insert (categoryId,Integer) into ColumnTyping
!create categoryImage:Column
!set categoryImage.name:='categoryImage'
!set categoryImage.isKey:=false
!insert (Categories,categoryImage) into Table_Column
!insert (categoryImage,Varchar64) into ColumnTyping
!create categorySortOrder:Column
!set categorySortOrder.name:='categorySortOrder'
!set categorySortOrder.isKey:=false
!insert (Categories,categorySortOrder) into Table_Column
!insert (categorySortOrder,Integer) into ColumnTyping
!create categoryDateAdded:Column
!set categoryDateAdded.name:='categoryDateAdded'
!set categoryDateAdded.isKey:=false
!insert (Categories,categoryDateAdded) into Table_Column
!insert (categoryDateAdded,Datetime) into ColumnTyping
!create categoryLastModified:Column
!set categoryLastModified.name:='categoryLastModified'
!set categoryLastModified.isKey:=false
!insert (Categories,categoryLastModified) into Table_Column
!insert (categoryLastModified,Datetime) into ColumnTyping
-----------------------------------------
--Association Categories, child(*) - Categories, parent(0..1)
-----------------------------------------
!create CatchildToCatParent:Table
!set CatchildToCatParent.name:='CatchildToCatParent'
---
!create ctcp1:Column
!set ctcp1.name:='child_categoryId'
!set ctcp1.isKey:=true
!insert (CatchildToCatParent,ctcp1) into Table_Column
!insert (ctcp1,Integer) into ColumnTyping
---
!create ctcp2:Column
!set ctcp2.name:='parent_categoryId'
!set ctcp2.isKey:=true
!insert (CatchildToCatParent,ctcp2) into Table_Column
!insert (ctcp2,Integer) into ColumnTyping
---
---
!create ForeignKeyctcp1:ForeignKey
!insert (Categories,ForeignKeyctcp1) into Table_ForeignKey
!insert (ForeignKeyctcp1,ctcp1) into ForeignKey_Column
---
!create ForeignKeyctcp2:ForeignKey
!insert (Categories,ForeignKeyctcp2) into Table_ForeignKey
!insert (ForeignKeyctcp2,ctcp2) into ForeignKey_Column
---
-----------------------------------------------
--TABLE:CategoriesDescription
-----------------------------------------------
!create CategoriesDescription:Table
!set CategoriesDescription.name:='CategoriesDescription'

!create categoryDescId:Column
!set categoryDescId.name:='category_categoryId'
```

```
!set categoryDescId.isKey:=true
!insert (CategoriesDescription,categoryDescId) into Table_Column
!insert (categoryDescId,Integer) into ColumnTyping


!create languageDescId:Column
!set languageDescId.name:='language_languageId'
!set languageDescId.isKey:=true
!insert (CategoriesDescription,languageDescId) into Table_Column
!insert (languageDescId,Integer) into ColumnTyping


!create categoryName:Column
!set categoryName.name:='categoryName'
!set categoryName.isKey:=false
!insert (CategoriesDescription,categoryName) into Table_Column
!insert (categoryName,Varchar32) into ColumnTyping
---
!create ForeignKeycdtc1:ForeignKey

!insert (Categories,ForeignKeycdtc1) into Table_ForeignKey
!insert (ForeignKeycdtc1,categoryDescId) into ForeignKey_Column
---
!create ForeignKeycdtc2:ForeignKey
!insert (Languages,ForeignKeycdtc2) into Table_ForeignKey
!insert (ForeignKeycdtc2,languageDescId) into ForeignKey_Column
---
------------------------------------------------
--TABLE:Manufacturers
------------------------------------------------
!create Manufacturers:Table
!set Manufacturers.name:='Manufacturers'

!create manufacturerId:Column
!set manufacturerId.name:='manufacturerId'
!set manufacturerId.isKey:=true
!insert (Manufacturers,manufacturerId) into Table_Column
!insert (manufacturerId,Integer) into ColumnTyping

!create manufacturerName:Column
!set manufacturerName.name:='manufacturerName'
!set manufacturerName.isKey:=false
!insert (Manufacturers,manufacturerName) into Table_Column
!insert (manufacturerName,Varchar32) into ColumnTyping

!create manufacturerImage:Column
!set manufacturerImage.name:='manufacturerImage'
!set manufacturerImage.isKey:=false
!insert (Manufacturers,manufacturerImage) into Table_Column
!insert (manufacturerImage,Varchar64) into ColumnTyping

!create manufacturerDateAdded:Column
!set manufacturerDateAdded.name:='manufacturerDateAdded'
!set manufacturerDateAdded.isKey:=false
!insert (Manufacturers,manufacturerDateAdded) into Table_Column
!insert (manufacturerDateAdded,Datetime) into ColumnTyping
!create manufacturerLastModified:Column
!set manufacturerLastModified.name:='manufacturerLastModified'
!set manufacturerLastModified.isKey:=false
```

```
!insert (Manufacturers,manufacturerLastModified) into Table_Column
!insert (manufacturerLastModified,Datetime) into ColumnTyping
------------------------------------------------
--TABLE:ManufacturersDescription
------------------------------------------------
!create ManufacturersDescription:Table
!set ManufacturersDescription.name:='ManufacturersDescription'

!create manufacturerDescId:Column
!set manufacturerDescId.name:='manufacturer_manufacturerId'
!set manufacturerDescId.isKey:=true
!insert (ManufacturersDescription,manufacturerDescId) into Table_Column
!insert (manufacturerDescId,Integer) into ColumnTyping

!create languageMaDescId:Column
!set languageMaDescId.name:='language_languageId'
!set languageMaDescId.isKey:=true
!insert (ManufacturersDescription,languageMaDescId) into Table_Column
!insert (languageMaDescId,Integer) into ColumnTyping

!create manufacturerURL:Column
!set manufacturerURL.name:='manufacturerURL'
!set manufacturerURL.isKey:=false
!insert (ManufacturersDescription,manufacturerURL) into Table_Column
!insert (manufacturerURL,Varchar255) into ColumnTyping
---
!create manufacturerURLclicked:Column
!set manufacturerURLclicked.name:='manufacturerURLclicked'
!set manufacturerURLclicked.isKey:=false
!insert (ManufacturersDescription,manufacturerURLclicked) into Table_Column
!insert (manufacturerURLclicked,Integer) into ColumnTyping
---
!create manufacturerDateLastClick:Column
!set manufacturerDateLastClick.name:='manufacturerDateLastClick'
!set manufacturerDateLastClick.isKey:=false
!insert (ManufacturersDescription,manufacturerDateLastClick) into
Table_Column
!insert (manufacturerDateLastClick,Datetime) into ColumnTyping
---
!create ForeignKeymdtm1:ForeignKey
!insert (Manufacturers,ForeignKeymdtm1) into Table_ForeignKey
!insert (ForeignKeymdtm1,manufacturerDescId) into ForeignKey_Column
---
!create ForeignKeymdtm2:ForeignKey
!insert (Languages,ForeignKeymdtm2) into Table_ForeignKey
!insert (ForeignKeymdtm2,languageMaDescId) into ForeignKey_Column
---
```

…

To be completed

Following there is the ER schema of the osCommerce system, that results of the translation between ER and Relational metaschema described in the first example:

```
----------------------------------
-- OSCommerce ER created
----------------------------------

!create DataType1 : DataType
!set @DataType1.name := 'Char(1)'

!create DataType2 : DataType
!set @DataType2.name := 'Varchar(10)'

!create DataType3 : DataType
!set @DataType3.name := 'Varchar(12)'

!create DataType4 : DataType
!set @DataType4.name := 'Varchar(128)'

!create DataType5 : DataType
!set @DataType5.name := 'Varchar(255)'

!create DataType6 : DataType
!set @DataType6.name := 'Varchar(32)'

!create DataType7 : DataType
!set @DataType7.name := 'Varchar(40)'

!create DataType8 : DataType
!set @DataType8.name := 'Varchar(48)'

!create DataType9 : DataType
!set @DataType9.name := 'Varchar(64)'

!create DataType10 : DataType
!set @DataType10.name := 'Varchar(96)'

!create DataType11 : DataType
!set @DataType11.name := 'Char(2)'

!create DataType12 : DataType
!set @DataType12.name := 'Char(3)'

!create DataType13 : DataType
!set @DataType13.name := 'Date'

!create DataType14 : DataType
!set @DataType14.name := 'Datetime'

!create DataType15 : DataType
!set @DataType15.name := 'Float'

!create DataType16 : DataType
!set @DataType16.name := 'Integer'

!create DataType17 : DataType
!set @DataType17.name := 'Month'

!create DataType18 : DataType
```

```
!set @DataType18.name := 'Text'

!create EntityType1 : EntityType
!set @EntityType1.name := 'Address_Book'

!create Attribute1 : Attribute
!set @Attribute1.name := 'addressBookId'
!set @Attribute1.isKey := true
!insert (EntityType1,Attribute1) into EntityType_Attribute
!insert (Attribute1,DataType16) into AttributeTyping

!create EntityType2 : EntityType
!set @EntityType2.name := 'Address_format'

!create Attribute2 : Attribute
!set @Attribute2.name := 'addressFormatId'
!set @Attribute2.isKey := true
!insert (EntityType2,Attribute2) into EntityType_Attribute
!insert (Attribute2,DataType16) into AttributeTyping

!create EntityType3 : EntityType
!set @EntityType3.name := 'Administrators'

!create Attribute3 : Attribute
!set @Attribute3.name := 'administratorId'
!set @Attribute3.isKey := true
!insert (EntityType3,Attribute3) into EntityType_Attribute
!insert (Attribute3,DataType16) into AttributeTyping

!create EntityType4 : EntityType
!set @EntityType4.name := 'Banners'

!create Attribute4 : Attribute
!set @Attribute4.name := 'bannerId'
!set @Attribute4.isKey := true
!insert (EntityType4,Attribute4) into EntityType_Attribute
!insert (Attribute4,DataType16) into AttributeTyping

!create EntityType5 : EntityType
!set @EntityType5.name := 'BannersHistory'

!create Attribute5 : Attribute
!set @Attribute5.name := 'bannerHistoryId'
!set @Attribute5.isKey := true
!insert (EntityType5,Attribute5) into EntityType_Attribute
!insert (Attribute5,DataType16) into AttributeTyping

!create EntityType6 : EntityType
!set @EntityType6.name := 'Categories'

!create Attribute6 : Attribute
!set @Attribute6.name := 'categoryId'
!set @Attribute6.isKey := true
!insert (EntityType6,Attribute6) into EntityType_Attribute
!insert (Attribute6,DataType16) into AttributeTyping

!create EntityType7 : EntityType
!set @EntityType7.name := 'Configuration'
```

```
!create Attribute7 : Attribute
!set @Attribute7.name := 'configurationId'
!set @Attribute7.isKey := true
!insert (EntityType7,Attribute7) into EntityType_Attribute
!insert (Attribute7,DataType16) into AttributeTyping

!create EntityType8 : EntityType
!set @EntityType8.name := 'ConfigurationGroup'

!create Attribute8 : Attribute
!set @Attribute8.name := 'configurationGroupId'
!set @Attribute8.isKey := true
!insert (EntityType8,Attribute8) into EntityType_Attribute
!insert (Attribute8,DataType16) into AttributeTyping

!create EntityType9 : EntityType
!set @EntityType9.name := 'Counter'

!create Attribute9 : Attribute
!set @Attribute9.name := 'startDate'
!set @Attribute9.isKey := true
!insert (EntityType9,Attribute9) into EntityType_Attribute
!insert (Attribute9,DataType13) into AttributeTyping

!create EntityType10 : EntityType
!set @EntityType10.name := 'CounterHistory'

!create Attribute10 : Attribute
!set @Attribute10.name := 'startDate'
!set @Attribute10.isKey := true
!insert (EntityType10,Attribute10) into EntityType_Attribute
!insert (Attribute10,DataType17) into AttributeTyping

!create EntityType11 : EntityType
!set @EntityType11.name := 'Countries'

!create Attribute11 : Attribute
!set @Attribute11.name := 'countryId'
!set @Attribute11.isKey := true
!insert (EntityType11,Attribute11) into EntityType_Attribute
!insert (Attribute11,DataType16) into AttributeTyping

!create EntityType12 : EntityType
!set @EntityType12.name := 'Currencies'

!create Attribute12 : Attribute
!set @Attribute12.name := 'currencyId'
!set @Attribute12.isKey := true
!insert (EntityType12,Attribute12) into EntityType_Attribute
!insert (Attribute12,DataType16) into AttributeTyping

!create EntityType13 : EntityType
!set @EntityType13.name := 'Customers'

!create Attribute13 : Attribute
!set @Attribute13.name := 'customerId'
!set @Attribute13.isKey := true
!insert (EntityType13,Attribute13) into EntityType_Attribute
!insert (Attribute13,DataType16) into AttributeTyping
```

```
!create EntityType14 : EntityType
!set @EntityType14.name := 'CustomersInfo'

!create Attribute14 : Attribute
!set @Attribute14.name := 'customerInfoId'
!set @Attribute14.isKey := true
!insert (EntityType14,Attribute14) into EntityType_Attribute
!insert (Attribute14,DataType16) into AttributeTyping

!create EntityType15 : EntityType
!set @EntityType15.name := 'Languages'

!create Attribute15 : Attribute
!set @Attribute15.name := 'languageId'
!set @Attribute15.isKey := true
!insert (EntityType15,Attribute15) into EntityType_Attribute
!insert (Attribute15,DataType16) into AttributeTyping

!create EntityType16 : EntityType
!set @EntityType16.name := 'Manufacturers'

!create Attribute16 : Attribute
!set @Attribute16.name := 'manufacturerId'
!set @Attribute16.isKey := true
!insert (EntityType16,Attribute16) into EntityType_Attribute
!insert (Attribute16,DataType16) into AttributeTyping

!create EntityType17 : EntityType
!set @EntityType17.name := 'Zones'

!create Attribute17 : Attribute
!set @Attribute17.name := 'zoneId'
!set @Attribute17.isKey := true
!insert (EntityType17,Attribute17) into EntityType_Attribute
!insert (Attribute17,DataType16) into AttributeTyping

!create RelationshipType1 : RelationshipType
!set @RelationshipType1.name := 'AddressBookToCountry'

!create RelationEnd1 : RelationEnd
!set @RelationEnd1.name := 'country'
!insert (RelationEnd1,EntityType11) into RelationEndTyping
!insert (RelationshipType1,RelationEnd1) into RelationshipType_RelationEnd

!create RelationEnd2 : RelationEnd
!set @RelationEnd2.name := 'addBook'
!insert (RelationEnd2,EntityType1) into RelationEndTyping
!insert (RelationshipType1,RelationEnd2) into RelationshipType_RelationEnd

!create RelationshipType2 : RelationshipType
!set @RelationshipType2.name := 'AddressBookToCustomer'

!create RelationEnd3 : RelationEnd
!set @RelationEnd3.name := 'customer'
!insert (RelationEnd3,EntityType13) into RelationEndTyping
!insert (RelationshipType2,RelationEnd3) into RelationshipType_RelationEnd

!create RelationEnd4 : RelationEnd
```

```
!set @RelationEnd4.name := 'addBook'
!insert (RelationEnd4,EntityType1) into RelationEndTyping
!insert (RelationshipType2,RelationEnd4) into RelationshipType_RelationEnd

!create RelationshipType3 : RelationshipType
!set @RelationshipType3.name := 'AddressBookToZone'

!create RelationEnd5 : RelationEnd
!set @RelationEnd5.name := 'zone'
!insert (RelationEnd5,EntityType17) into RelationEndTyping
!insert (RelationshipType3,RelationEnd5) into RelationshipType_RelationEnd

!create RelationEnd6 : RelationEnd
!set @RelationEnd6.name := 'addBook'
!insert (RelationEnd6,EntityType1) into RelationEndTyping
!insert (RelationshipType3,RelationEnd6) into RelationshipType_RelationEnd

!create RelationshipType4 : RelationshipType
!set @RelationshipType4.name := 'BannerHistoryToBanner'

!create RelationEnd7 : RelationEnd
!set @RelationEnd7.name := 'bannerHistory'
!insert (RelationEnd7,EntityType5) into RelationEndTyping
!insert (RelationshipType4,RelationEnd7) into RelationshipType_RelationEnd

!create RelationEnd8 : RelationEnd
!set @RelationEnd8.name := 'banner'
!insert (RelationEnd8,EntityType4) into RelationEndTyping
!insert (RelationshipType4,RelationEnd8) into RelationshipType_RelationEnd

!create RelationshipType5 : RelationshipType
!set @RelationshipType5.name := 'CatchildToCatParent'

!create RelationEnd9 : RelationEnd
!set @RelationEnd9.name := 'parent'
!insert (RelationEnd9,EntityType6) into RelationEndTyping
!insert (RelationshipType5,RelationEnd9) into RelationshipType_RelationEnd

!create RelationEnd10 : RelationEnd
!set @RelationEnd10.name := 'child'
!insert (RelationEnd10,EntityType6) into RelationEndTyping
!insert (RelationshipType5,RelationEnd10) into RelationshipType_RelationEnd

!create RelationshipType6 : RelationshipType
!set @RelationshipType6.name := 'CategoriesDescription'

!create RelationEnd11 : RelationEnd
!set @RelationEnd11.name := 'language'
!insert (RelationEnd11,EntityType15) into RelationEndTyping
!insert (RelationshipType6,RelationEnd11) into RelationshipType_RelationEnd

!create RelationEnd12 : RelationEnd
!set @RelationEnd12.name := 'category'
!insert (RelationEnd12,EntityType6) into RelationEndTyping
!insert (RelationshipType6,RelationEnd12) into RelationshipType_RelationEnd

!create RelationshipType7 : RelationshipType
!set @RelationshipType7.name := 'ConfToConfGroup'
```

```
!create RelationEnd13 : RelationEnd
!set @RelationEnd13.name := 'configurationGroup'
!insert (RelationEnd13,EntityType8) into RelationEndTyping
!insert (RelationshipType7,RelationEnd13) into RelationshipType_RelationEnd

!create RelationEnd14 : RelationEnd
!set @RelationEnd14.name := 'configuration'
!insert (RelationEnd14,EntityType7) into RelationEndTyping
!insert (RelationshipType7,RelationEnd14) into RelationshipType_RelationEnd

!create RelationshipType8 : RelationshipType
!set @RelationshipType8.name := 'CountryToAddressFormat'

!create RelationEnd15 : RelationEnd
!set @RelationEnd15.name := 'addressFormat'
!insert (RelationEnd15,EntityType2) into RelationEndTyping
!insert (RelationshipType8,RelationEnd15) into RelationshipType_RelationEnd

!create RelationEnd16 : RelationEnd
!set @RelationEnd16.name := 'country'
!insert (RelationEnd16,EntityType11) into RelationEndTyping
!insert (RelationshipType8,RelationEnd16) into RelationshipType_RelationEnd

!create RelationshipType9 : RelationshipType
!set @RelationshipType9.name := 'CustomerInfoToCustomer'

!create RelationEnd17 : RelationEnd
!set @RelationEnd17.name := 'info'
!insert (RelationEnd17,EntityType14) into RelationEndTyping
!insert (RelationshipType9,RelationEnd17) into RelationshipType_RelationEnd

!create RelationEnd18 : RelationEnd
!set @RelationEnd18.name := 'customer'
!insert (RelationEnd18,EntityType13) into RelationEndTyping
!insert (RelationshipType9,RelationEnd18) into RelationshipType_RelationEnd

!create RelationshipType10 : RelationshipType
!set @RelationshipType10.name := 'DefaultAddressBookToCustomer'

!create RelationEnd19 : RelationEnd
!set @RelationEnd19.name := 'defAdd'
!insert (RelationEnd19,EntityType1) into RelationEndTyping
!insert (RelationshipType10,RelationEnd19) into RelationshipType_RelationEnd

!create RelationEnd20 : RelationEnd
!set @RelationEnd20.name := 'custOfDefAdd'
!insert (RelationEnd20,EntityType13) into RelationEndTyping
!insert (RelationshipType10,RelationEnd20) into RelationshipType_RelationEnd

!create RelationshipType11 : RelationshipType
!set @RelationshipType11.name := 'ManufacturersDescription'

!create RelationEnd21 : RelationEnd
!set @RelationEnd21.name := 'manufacturer'
!insert (RelationEnd21,EntityType16) into RelationEndTyping
!insert (RelationshipType11,RelationEnd21) into RelationshipType_RelationEnd

!create RelationEnd22 : RelationEnd
!set @RelationEnd22.name := 'language'
```

```
!insert (RelationEnd22,EntityType15) into RelationEndTyping
!insert (RelationshipType11,RelationEnd22) into RelationshipType_RelationEnd

!create RelationshipType12 : RelationshipType
!set @RelationshipType12.name := 'ZoneToCountry'

!create RelationEnd23 : RelationEnd
!set @RelationEnd23.name := 'zone'
!insert (RelationEnd23,EntityType17) into RelationEndTyping
!insert (RelationshipType12,RelationEnd23) into RelationshipType_RelationEnd

!create RelationEnd24 : RelationEnd
!set @RelationEnd24.name := 'country'
!insert (RelationEnd24,EntityType11) into RelationEndTyping
!insert (RelationshipType12,RelationEnd24) into RelationshipType_RelationEnd

!create Attribute18 : Attribute
!set @Attribute18.name := 'address_format'
!set @Attribute18.isKey := false
!insert (EntityType2,Attribute18) into EntityType_Attribute
!insert (Attribute18,DataType4) into AttributeTyping

!create Attribute19 : Attribute
!set @Attribute19.name := 'BannerShown'
!set @Attribute19.isKey := false
!insert (EntityType5,Attribute19) into EntityType_Attribute
!insert (Attribute19,DataType16) into AttributeTyping

!create Attribute20 : Attribute
!set @Attribute20.name := 'BannerStatus'
!set @Attribute20.isKey := false
!insert (EntityType4,Attribute20) into EntityType_Attribute
!insert (Attribute20,DataType16) into AttributeTyping

!create Attribute21 : Attribute
!set @Attribute21.name := 'bannerTitle'
!set @Attribute21.isKey := false
!insert (EntityType4,Attribute21) into EntityType_Attribute
!insert (Attribute21,DataType9) into AttributeTyping

!create Attribute22 : Attribute
!set @Attribute22.name := 'bannerUrl'
!set @Attribute22.isKey := false
!insert (EntityType4,Attribute22) into EntityType_Attribute
!insert (Attribute22,DataType5) into AttributeTyping

!create Attribute23 : Attribute
!set @Attribute23.name := 'categoryDateAdded'
!set @Attribute23.isKey := false
!insert (EntityType6,Attribute23) into EntityType_Attribute
!insert (Attribute23,DataType14) into AttributeTyping

!create Attribute24 : Attribute
!set @Attribute24.name := 'categoryImage'
!set @Attribute24.isKey := false
!insert (EntityType6,Attribute24) into EntityType_Attribute
!insert (Attribute24,DataType9) into AttributeTyping

!create Attribute25 : Attribute
```

```
!set @Attribute25.name := 'categoryLastModified'
!set @Attribute25.isKey := false
!insert (EntityType6,Attribute25) into EntityType_Attribute
!insert (Attribute25,DataType14) into AttributeTyping

!create Attribute26 : Attribute
!set @Attribute26.name := 'categoryName'
!set @Attribute26.isKey := false
!insert (RelationshipType6,Attribute26) into RelationshipType_Attribute
!insert (Attribute26,DataType6) into AttributeTyping

!create Attribute27 : Attribute
!set @Attribute27.name := 'categorySortOrder'
!set @Attribute27.isKey := false
!insert (EntityType6,Attribute27) into EntityType_Attribute
!insert (Attribute27,DataType16) into AttributeTyping

!create Attribute28 : Attribute
!set @Attribute28.name := 'confDateAdded'
!set @Attribute28.isKey := false
!insert (EntityType7,Attribute28) into EntityType_Attribute
!insert (Attribute28,DataType14) into AttributeTyping

!create Attribute29 : Attribute
!set @Attribute29.name := 'address_summary'
!set @Attribute29.isKey := false
!insert (EntityType2,Attribute29) into EntityType_Attribute
!insert (Attribute29,DataType8) into AttributeTyping

!create Attribute30 : Attribute
!set @Attribute30.name := 'confLastModified'
!set @Attribute30.isKey := false
!insert (EntityType7,Attribute30) into EntityType_Attribute
!insert (Attribute30,DataType14) into AttributeTyping

!create Attribute31 : Attribute
!set @Attribute31.name := 'confSortOrder'
!set @Attribute31.isKey := false
!insert (EntityType7,Attribute31) into EntityType_Attribute
!insert (Attribute31,DataType5) into AttributeTyping

!create Attribute32 : Attribute
!set @Attribute32.name := 'configurationDescription'
!set @Attribute32.isKey := false
!insert (EntityType7,Attribute32) into EntityType_Attribute
!insert (Attribute32,DataType5) into AttributeTyping

!create Attribute33 : Attribute
!set @Attribute33.name := 'configurationGroupDescription'
!set @Attribute33.isKey := false
!insert (EntityType8,Attribute33) into EntityType_Attribute
!insert (Attribute33,DataType5) into AttributeTyping

!create Attribute34 : Attribute
!set @Attribute34.name := 'configurationGroupSortOrder'
!set @Attribute34.isKey := false
!insert (EntityType8,Attribute34) into EntityType_Attribute
!insert (Attribute34,DataType16) into AttributeTyping
```

```
!create Attribute35 : Attribute
!set @Attribute35.name := 'configurationGroupTitle'
!set @Attribute35.isKey := false
!insert (EntityType8,Attribute35) into EntityType_Attribute
!insert (Attribute35,DataType9) into AttributeTyping

!create Attribute36 : Attribute
!set @Attribute36.name := 'configurationKey'
!set @Attribute36.isKey := false
!insert (EntityType7,Attribute36) into EntityType_Attribute
!insert (Attribute36,DataType5) into AttributeTyping

!create Attribute37 : Attribute
!set @Attribute37.name := 'configurationTitle'
!set @Attribute37.isKey := false
!insert (EntityType7,Attribute37) into EntityType_Attribute
!insert (Attribute37,DataType5) into AttributeTyping

!create Attribute38 : Attribute
!set @Attribute38.name := 'configurationValue'
!set @Attribute38.isKey := false
!insert (EntityType7,Attribute38) into EntityType_Attribute
!insert (Attribute38,DataType5) into AttributeTyping

!create Attribute39 : Attribute
!set @Attribute39.name := 'configurationVisible'
!set @Attribute39.isKey := false
!insert (EntityType8,Attribute39) into EntityType_Attribute
!insert (Attribute39,DataType16) into AttributeTyping

!create Attribute40 : Attribute
!set @Attribute40.name := 'BannerClicked'
!set @Attribute40.isKey := false
!insert (EntityType5,Attribute40) into EntityType_Attribute
!insert (Attribute40,DataType16) into AttributeTyping

!create Attribute41 : Attribute
!set @Attribute41.name := 'counterInMonth'
!set @Attribute41.isKey := false
!insert (EntityType10,Attribute41) into EntityType_Attribute
!insert (Attribute41,DataType16) into AttributeTyping

!create Attribute42 : Attribute
!set @Attribute42.name := 'counterValue'
!set @Attribute42.isKey := false
!insert (EntityType9,Attribute42) into EntityType_Attribute
!insert (Attribute42,DataType16) into AttributeTyping

!create Attribute43 : Attribute
!set @Attribute43.name := 'countries_iso_code_2'
!set @Attribute43.isKey := false
!insert (EntityType11,Attribute43) into EntityType_Attribute
!insert (Attribute43,DataType11) into AttributeTyping

!create Attribute44 : Attribute
!set @Attribute44.name := 'countries_iso_code_3'
!set @Attribute44.isKey := false
!insert (EntityType11,Attribute44) into EntityType_Attribute
!insert (Attribute44,DataType12) into AttributeTyping
```

```
!create Attribute45 : Attribute
!set @Attribute45.name := 'countries_name'
!set @Attribute45.isKey := false
!insert (EntityType11,Attribute45) into EntityType_Attribute
!insert (Attribute45,DataType9) into AttributeTyping

!create Attribute46 : Attribute
!set @Attribute46.name := 'currencyCode'
!set @Attribute46.isKey := false
!insert (EntityType12,Attribute46) into EntityType_Attribute
!insert (Attribute46,DataType12) into AttributeTyping

!create Attribute47 : Attribute
!set @Attribute47.name := 'currencySymbolLeft'
!set @Attribute47.isKey := false
!insert (EntityType12,Attribute47) into EntityType_Attribute
!insert (Attribute47,DataType3) into AttributeTyping

!create Attribute48 : Attribute
!set @Attribute48.name := 'currencySymbolRight'
!set @Attribute48.isKey := false
!insert (EntityType12,Attribute48) into EntityType_Attribute
!insert (Attribute48,DataType3) into AttributeTyping

!create Attribute49 : Attribute
!set @Attribute49.name := 'currencyTitle'
!set @Attribute49.isKey := false
!insert (EntityType12,Attribute49) into EntityType_Attribute
!insert (Attribute49,DataType6) into AttributeTyping

!create Attribute50 : Attribute
!set @Attribute50.name := 'customerDateAccountCreated'
!set @Attribute50.isKey := false
!insert (EntityType14,Attribute50) into EntityType_Attribute
!insert (Attribute50,DataType14) into AttributeTyping

!create Attribute51 : Attribute
!set @Attribute51.name := 'BannerDateAdded'
!set @Attribute51.isKey := false
!insert (EntityType4,Attribute51) into EntityType_Attribute
!insert (Attribute51,DataType14) into AttributeTyping

!create Attribute52 : Attribute
!set @Attribute52.name := 'customerDateAccountModified'
!set @Attribute52.isKey := false
!insert (EntityType14,Attribute52) into EntityType_Attribute
!insert (Attribute52,DataType14) into AttributeTyping

!create Attribute53 : Attribute
!set @Attribute53.name := 'customerDateLastLogon'
!set @Attribute53.isKey := false
!insert (EntityType14,Attribute53) into EntityType_Attribute
!insert (Attribute53,DataType14) into AttributeTyping

!create Attribute54 : Attribute
!set @Attribute54.name := 'DateOfBirth'
!set @Attribute54.isKey := false
!insert (EntityType13,Attribute54) into EntityType_Attribute
```

```
!insert (Attribute54,DataType14) into AttributeTyping

!create Attribute55 : Attribute
!set @Attribute55.name := 'EMail'
!set @Attribute55.isKey := false
!insert (EntityType13,Attribute55) into EntityType_Attribute
!insert (Attribute55,DataType10) into AttributeTyping

!create Attribute56 : Attribute
!set @Attribute56.name := 'Fax'
!set @Attribute56.isKey := false
!insert (EntityType13,Attribute56) into EntityType_Attribute
!insert (Attribute56,DataType6) into AttributeTyping

!create Attribute57 : Attribute
!set @Attribute57.name := 'customerFirstName'
!set @Attribute57.isKey := false
!insert (EntityType13,Attribute57) into EntityType_Attribute
!insert (Attribute57,DataType6) into AttributeTyping

!create Attribute58 : Attribute
!set @Attribute58.name := 'customerGender'
!set @Attribute58.isKey := false
!insert (EntityType13,Attribute58) into EntityType_Attribute
!insert (Attribute58,DataType1) into AttributeTyping

!create Attribute59 : Attribute
!set @Attribute59.name := 'customerLastName'
!set @Attribute59.isKey := false
!insert (EntityType13,Attribute59) into EntityType_Attribute
!insert (Attribute59,DataType6) into AttributeTyping

!create Attribute60 : Attribute
!set @Attribute60.name := 'NewsLetter'
!set @Attribute60.isKey := false
!insert (EntityType13,Attribute60) into EntityType_Attribute
!insert (Attribute60,DataType1) into AttributeTyping

!create Attribute61 : Attribute
!set @Attribute61.name := 'Password'
!set @Attribute61.isKey := false
!insert (EntityType13,Attribute61) into EntityType_Attribute
!insert (Attribute61,DataType7) into AttributeTyping

!create Attribute62 : Attribute
!set @Attribute62.name := 'BannerDateStatusChange'
!set @Attribute62.isKey := false
!insert (EntityType4,Attribute62) into EntityType_Attribute
!insert (Attribute62,DataType14) into AttributeTyping

!create Attribute63 : Attribute
!set @Attribute63.name := 'Telephone'
!set @Attribute63.isKey := false
!insert (EntityType13,Attribute63) into EntityType_Attribute
!insert (Attribute63,DataType6) into AttributeTyping

!create Attribute64 : Attribute
!set @Attribute64.name := 'DateScheduled'
!set @Attribute64.isKey := false
```

```
!insert (EntityType4,Attribute64) into EntityType_Attribute
!insert (Attribute64,DataType14) into AttributeTyping

!create Attribute65 : Attribute
!set @Attribute65.name := 'decimalPlaces'
!set @Attribute65.isKey := false
!insert (EntityType12,Attribute65) into EntityType_Attribute
!insert (Attribute65,DataType1) into AttributeTyping

!create Attribute66 : Attribute
!set @Attribute66.name := 'decimalPoint'
!set @Attribute66.isKey := false
!insert (EntityType12,Attribute66) into EntityType_Attribute
!insert (Attribute66,DataType1) into AttributeTyping

!create Attribute67 : Attribute
!set @Attribute67.name := 'entry_city'
!set @Attribute67.isKey := false
!insert (EntityType1,Attribute67) into EntityType_Attribute
!insert (Attribute67,DataType6) into AttributeTyping

!create Attribute68 : Attribute
!set @Attribute68.name := 'entry_company'
!set @Attribute68.isKey := false
!insert (EntityType1,Attribute68) into EntityType_Attribute
!insert (Attribute68,DataType6) into AttributeTyping

!create Attribute69 : Attribute
!set @Attribute69.name := 'entry_firstname'
!set @Attribute69.isKey := false
!insert (EntityType1,Attribute69) into EntityType_Attribute
!insert (Attribute69,DataType6) into AttributeTyping

!create Attribute70 : Attribute
!set @Attribute70.name := 'entry_gender'
!set @Attribute70.isKey := false
!insert (EntityType1,Attribute70) into EntityType_Attribute
!insert (Attribute70,DataType1) into AttributeTyping

!create Attribute71 : Attribute
!set @Attribute71.name := 'entry_lastname'
!set @Attribute71.isKey := false
!insert (EntityType1,Attribute71) into EntityType_Attribute
!insert (Attribute71,DataType6) into AttributeTyping

!create Attribute72 : Attribute
!set @Attribute72.name := 'entry_postcode'
!set @Attribute72.isKey := false
!insert (EntityType1,Attribute72) into EntityType_Attribute
!insert (Attribute72,DataType2) into AttributeTyping

!create Attribute73 : Attribute
!set @Attribute73.name := 'bannerGroup'
!set @Attribute73.isKey := false
!insert (EntityType4,Attribute73) into EntityType_Attribute
!insert (Attribute73,DataType2) into AttributeTyping

!create Attribute74 : Attribute
!set @Attribute74.name := 'entry_state'
```

```
!set @Attribute74.isKey := false
!insert (EntityType1,Attribute74) into EntityType_Attribute
!insert (Attribute74,DataType6) into AttributeTyping

!create Attribute75 : Attribute
!set @Attribute75.name := 'entry_street_address'
!set @Attribute75.isKey := false
!insert (EntityType1,Attribute75) into EntityType_Attribute
!insert (Attribute75,DataType9) into AttributeTyping

!create Attribute76 : Attribute
!set @Attribute76.name := 'entry_suburb'
!set @Attribute76.isKey := false
!insert (EntityType1,Attribute76) into EntityType_Attribute
!insert (Attribute76,DataType6) into AttributeTyping

!create Attribute77 : Attribute
!set @Attribute77.name := 'ExpiresImpressions'
!set @Attribute77.isKey := false
!insert (EntityType4,Attribute77) into EntityType_Attribute
!insert (Attribute77,DataType16) into AttributeTyping

!create Attribute78 : Attribute
!set @Attribute78.name := 'globalProductNotification'
!set @Attribute78.isKey := false
!insert (EntityType14,Attribute78) into EntityType_Attribute
!insert (Attribute78,DataType1) into AttributeTyping

!create Attribute79 : Attribute
!set @Attribute79.name := 'languageCode'
!set @Attribute79.isKey := false
!insert (EntityType15,Attribute79) into EntityType_Attribute
!insert (Attribute79,DataType11) into AttributeTyping

!create Attribute80 : Attribute
!set @Attribute80.name := 'languageDirectory'
!set @Attribute80.isKey := false
!insert (EntityType15,Attribute80) into EntityType_Attribute
!insert (Attribute80,DataType6) into AttributeTyping

!create Attribute81 : Attribute
!set @Attribute81.name := 'languageImage'
!set @Attribute81.isKey := false
!insert (EntityType15,Attribute81) into EntityType_Attribute
!insert (Attribute81,DataType9) into AttributeTyping

!create Attribute82 : Attribute
!set @Attribute82.name := 'LanguageName'
!set @Attribute82.isKey := false
!insert (EntityType15,Attribute82) into EntityType_Attribute
!insert (Attribute82,DataType6) into AttributeTyping

!create Attribute83 : Attribute
!set @Attribute83.name := 'LanguageSortOrder'
!set @Attribute83.isKey := false
!insert (EntityType15,Attribute83) into EntityType_Attribute
!insert (Attribute83,DataType16) into AttributeTyping

!create Attribute84 : Attribute
```

```
!set @Attribute84.name := 'bannerHTML'
!set @Attribute84.isKey := false
!insert (EntityType4,Attribute84) into EntityType_Attribute
!insert (Attribute84,DataType18) into AttributeTyping

!create Attribute85 : Attribute
!set @Attribute85.name := 'lastUpdated'
!set @Attribute85.isKey := false
!insert (EntityType12,Attribute85) into EntityType_Attribute
!insert (Attribute85,DataType14) into AttributeTyping

!create Attribute86 : Attribute
!set @Attribute86.name := 'manufacturerDateAdded'
!set @Attribute86.isKey := false
!insert (EntityType16,Attribute86) into EntityType_Attribute
!insert (Attribute86,DataType14) into AttributeTyping

!create Attribute87 : Attribute
!set @Attribute87.name := 'manufacturerDateLastClick'
!set @Attribute87.isKey := false
!insert (RelationshipType11,Attribute87) into RelationshipType_Attribute
!insert (Attribute87,DataType14) into AttributeTyping

!create Attribute88 : Attribute
!set @Attribute88.name := 'manufacturerImage'
!set @Attribute88.isKey := false
!insert (EntityType16,Attribute88) into EntityType_Attribute
!insert (Attribute88,DataType9) into AttributeTyping

!create Attribute89 : Attribute
!set @Attribute89.name := 'manufacturerLastModified'
!set @Attribute89.isKey := false
!insert (EntityType16,Attribute89) into EntityType_Attribute
!insert (Attribute89,DataType14) into AttributeTyping

!create Attribute90 : Attribute
!set @Attribute90.name := 'manufacturerName'
!set @Attribute90.isKey := false
!insert (EntityType16,Attribute90) into EntityType_Attribute
!insert (Attribute90,DataType6) into AttributeTyping

!create Attribute91 : Attribute
!set @Attribute91.name := 'manufacturerURL'
!set @Attribute91.isKey := false
!insert (RelationshipType11,Attribute91) into RelationshipType_Attribute
!insert (Attribute91,DataType5) into AttributeTyping

!create Attribute92 : Attribute
!set @Attribute92.name := 'manufacturerURLclicked'
!set @Attribute92.isKey := false
!insert (RelationshipType11,Attribute92) into RelationshipType_Attribute
!insert (Attribute92,DataType16) into AttributeTyping

!create Attribute93 : Attribute
!set @Attribute93.name := 'numberOfLogons'
!set @Attribute93.isKey := false
!insert (EntityType14,Attribute93) into EntityType_Attribute
!insert (Attribute93,DataType16) into AttributeTyping
```

```
!create Attribute94 : Attribute
!set @Attribute94.name := 'setFunction'
!set @Attribute94.isKey := false
!insert (EntityType7,Attribute94) into EntityType_Attribute
!insert (Attribute94,DataType5) into AttributeTyping

!create Attribute95 : Attribute
!set @Attribute95.name := 'BannerHistoryDate'
!set @Attribute95.isKey := false
!insert (EntityType5,Attribute95) into EntityType_Attribute
!insert (Attribute95,DataType16) into AttributeTyping

!create Attribute96 : Attribute
!set @Attribute96.name := 'thousandsPoint'
!set @Attribute96.isKey := false
!insert (EntityType12,Attribute96) into EntityType_Attribute
!insert (Attribute96,DataType1) into AttributeTyping

!create Attribute97 : Attribute
!set @Attribute97.name := 'useFunction'
!set @Attribute97.isKey := false
!insert (EntityType7,Attribute97) into EntityType_Attribute
!insert (Attribute97,DataType5) into AttributeTyping

!create Attribute98 : Attribute
!set @Attribute98.name := 'user_name'
!set @Attribute98.isKey := false
!insert (EntityType3,Attribute98) into EntityType_Attribute
!insert (Attribute98,DataType6) into AttributeTyping

!create Attribute99 : Attribute
!set @Attribute99.name := 'userPassword'
!set @Attribute99.isKey := false
!insert (EntityType3,Attribute99) into EntityType_Attribute
!insert (Attribute99,DataType7) into AttributeTyping

!create Attribute100 : Attribute
!set @Attribute100.name := 'value'
!set @Attribute100.isKey := false
!insert (EntityType12,Attribute100) into EntityType_Attribute
!insert (Attribute100,DataType15) into AttributeTyping

!create Attribute101 : Attribute
!set @Attribute101.name := 'zone_code'
!set @Attribute101.isKey := false
!insert (EntityType17,Attribute101) into EntityType_Attribute
!insert (Attribute101,DataType6) into AttributeTyping

!create Attribute102 : Attribute
!set @Attribute102.name := 'zone_name'
!set @Attribute102.isKey := false
!insert (EntityType17,Attribute102) into EntityType_Attribute
!insert (Attribute102,DataType6) into AttributeTyping

!create Attribute103 : Attribute
!set @Attribute103.name := 'bannerImage'
!set @Attribute103.isKey := false
!insert (EntityType4,Attribute103) into EntityType_Attribute
```

```
!insert (Attribute103,DataType9) into AttributeTyping!insert
(Attribute95,DataType9) into AttributeTyping
```

REFERENCES

Atzeni, P. 2007, "Schema and Data Translation: A Personal Perspective", *Advances in Databases and Information Systems*, vol. LNCS 4690, pp. 14-27.

Bernstein, P.A. 2003, "Applying model management to classical meta data problems", *CIDR 2003*, pp. 209-220.

Bernstein, P.A., Haas, L.M., Jarke, M., Rahm, E. & Wiederhold, G. 2000, "Panel: Is Generic Metadata Management Feasible?", *VLDB 2000*, pp. 660-662.

Bernstein, P.A. & Melnik, S. 2007, "Model management 2.0: manipulating richer mappings", *ACM SIGMOD 2007*, pp. 1-12.

Gogolla, M. 2005, *Tales of ER and RE Syntax and Semantics*, Dagstuhl Seminar Proceedings 05161.

Muller, R.J. 1999, *Database design for smarties: using UML for data modeling*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Object Management Group 2007, *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, OMG Final Adopted Specification (ptc/2007-07-07)*.

Teorey, T., Lightstone, S. & Nadeau, T. 2006, *Database modeling and design*, 4th edition edn, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Tort, A., Olive, A. 2007, *osCommerce Conceptual Schema*. Universitat Politècnica de Catalunya. http://guifre.lsi.upc.edu/OSCommerce.pdf

ANNEX A

A.1 Relations Examples

A.1.1 UML to RDBMS Mapping

A.1.1.1 Overview

This example maps persistent classes of a simple UML model to tables of a simple RDBMS model. A persistent class maps to a table, a primary key and an identifying column. Attributes of the persistent class map to columns of the table: an attribute of a primitive datatype maps to a single column; an attribute of a complex data type maps to a set of columns corresponding to its exploded set of primitive datatype attributes; attributes inherited from the class hierarchy are also mapped to the columns of the table. An association between two persistent classes maps to a foreign key relationship between the corresponding tables.

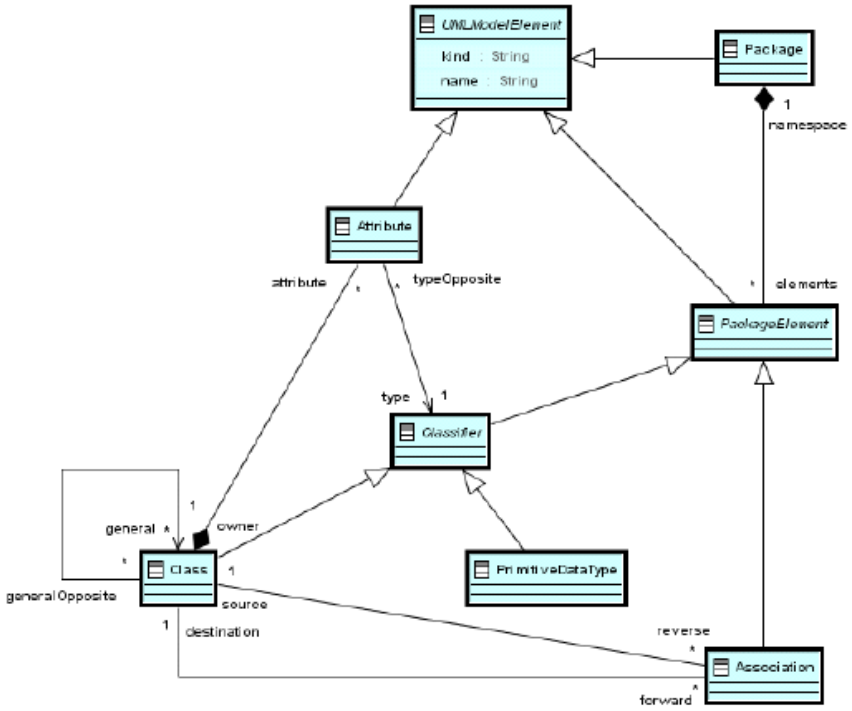

Figure A.1 - Simple UML Metamodel

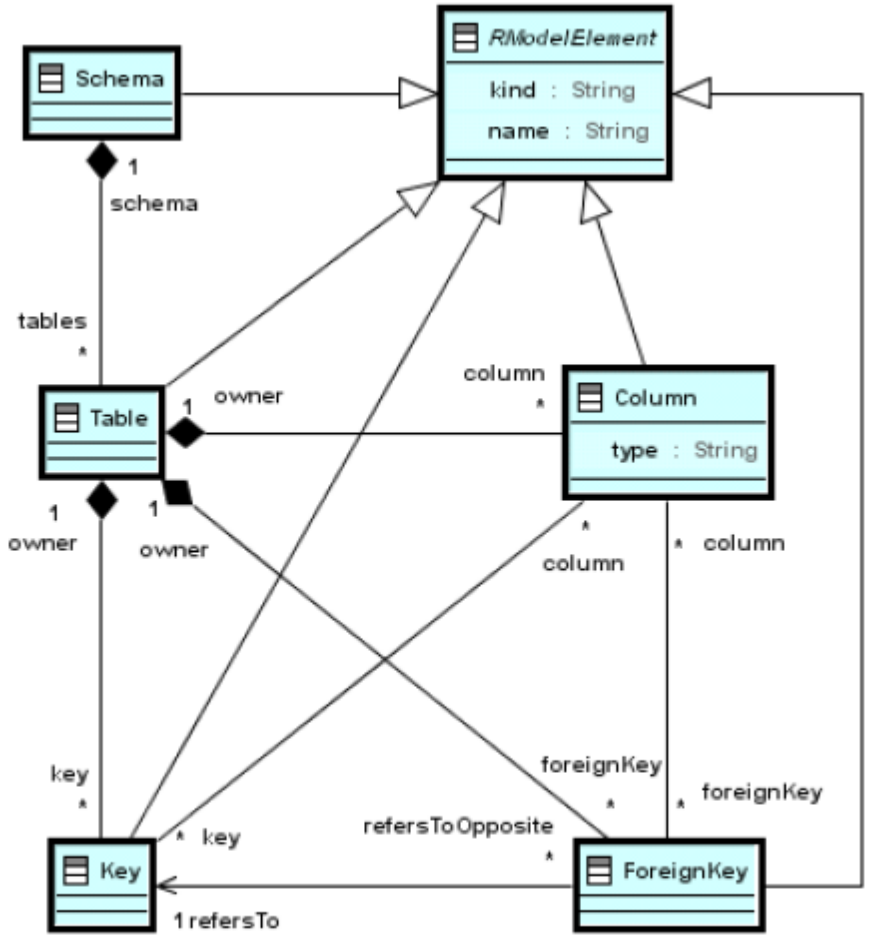

Figure A.2 - Simple RDBMS Metamodel

UML to RDBMS mapping in textual syntax

```
transformation umlToRdbms(uml:SimpleUML, rdbms:SimpleRDBMS)
{
key Table (name, schema);
key Column (name, owner);   // owner:Table opposite column:Column
key Key (name, owner); // key of class ëKeyí;
                       // owner:Table opposite key:Key

top relation PackageToSchema   // map each package to a schema
{
   pn: String;

   checkonly domain uml p:Package {name=pn};
   enforce domain rdbms s:Schema {name=pn};
}

top relation ClassToTable   // map each persistent class to a table
{
   cn, prefix: String;

   checkonly domain uml c:Class {namespace=p:Package {},
                                 kind='Persistent', name=cn};
   enforce domain rdbms t:Table {schema=s:Schema {}, name=cn,
```

```
            column=cl:Column {name=cn+'_tid', type='NUMBER'},
            key=k:Key {name=cn+'_pk', column=cl}};
   when {
      PackageToSchema(p, s);
   }
   where {
      prefix = '';
      AttributeToColumn(c, t, prefix);
   }
}

relation AttributeToColumn
{
   checkonly domain uml c:Class {};
   enforce domain rdbms t:Table {};
   primitive domain prefix:String;
   where {
      PrimitiveAttributeToColumn(c, t, prefix);
      ComplexAttributeToColumn(c, t, prefix);
      SuperAttributeToColumn(c, t, prefix);
   }
}

relation PrimitiveAttributeToColumn
{
   an, pn, cn, sqltype: String;

   checkonly domain uml c:Class {attribute=a:Attribute {name=an,
                                    type=p:PrimitiveDataType {name=pn}}};
   enforce domain rdbms t:Table {column=cl:Column {name=cn,
                                    type=sqltype}};
   primitive domain prefix:String;
   where {
      cn = if (prefix = '') then an else prefix+'_'+an endif;
      sqltype = PrimitiveTypeToSqlType(pn);
   }
}

relation ComplexAttributeToColumn
{
   an, newPrefix: String;

   checkonly domain uml c:Class {attribute=a:Attribute {name=an,
                                    type=tc:Class {}}};
   enforce domain rdbms t:Table {};
   primitive domain prefix:String;
   where {
      newPrefix = prefix+'_'+an;
      AttributeToColumn(tc, t, newPrefix);
   }
}

relation SuperAttributeToColumn
{
   checkonly domain uml c:Class {general=sc:Class {}};
   enforce domain rdbms t:Table {};
   primitive domain prefix:String;
   where {
      AttributeToColumn(sc, t, prefix);
   }
}

// map each association between persistent classes to a foreign key
top relation AssocToFKey
```

```
{
   srcTbl, destTbl: Table;
   pKey: Key;
   an, scn, dcn, fkn, fcn: String;

   checkonly domain uml a:Association {namespace=p:Package {},
            name=an,
            source=sc:Class {kind='Persistent',name=scn},
            destination=dc:Class {kind='Persistent',name=dcn}
         };
   enforce domain rdbms fk:ForeignKey {schema=s:Schema {},
            name=fkn,
            owner=srcTbl,
            column=fc:Column {name=fcn,type='NUMBER',owner=srcTbl},
            refersTo=pKey
         };
   when {   /* when refers to pre-condition */
      PackageToSchema(p, s);
      ClassToTable(sc, srcTbl);
      ClassToTable(dc, destTbl);
      pKey = destTbl.key;
   }
   where {
      fkn=scn+'_'+an+'_'+dcn;
      fcn=fkn+'_tid';
   }
}

function PrimitiveTypeToSqlType(primitiveTpe:String):String
{
   if (primitiveType='INTEGER')
   then 'NUMBER'
   else if (primitiveType='BOOLEAN')
      then 'BOOLEAN'
      else 'VARCHAR'
      endif
   endif;
}

}
```

UML to RDBMS mapping in graphical syntax

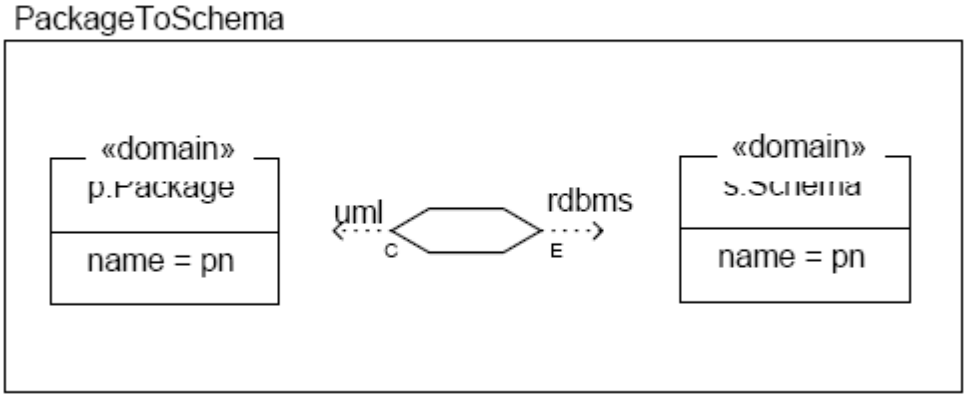

Figure A.3 - PackageToSchema relation

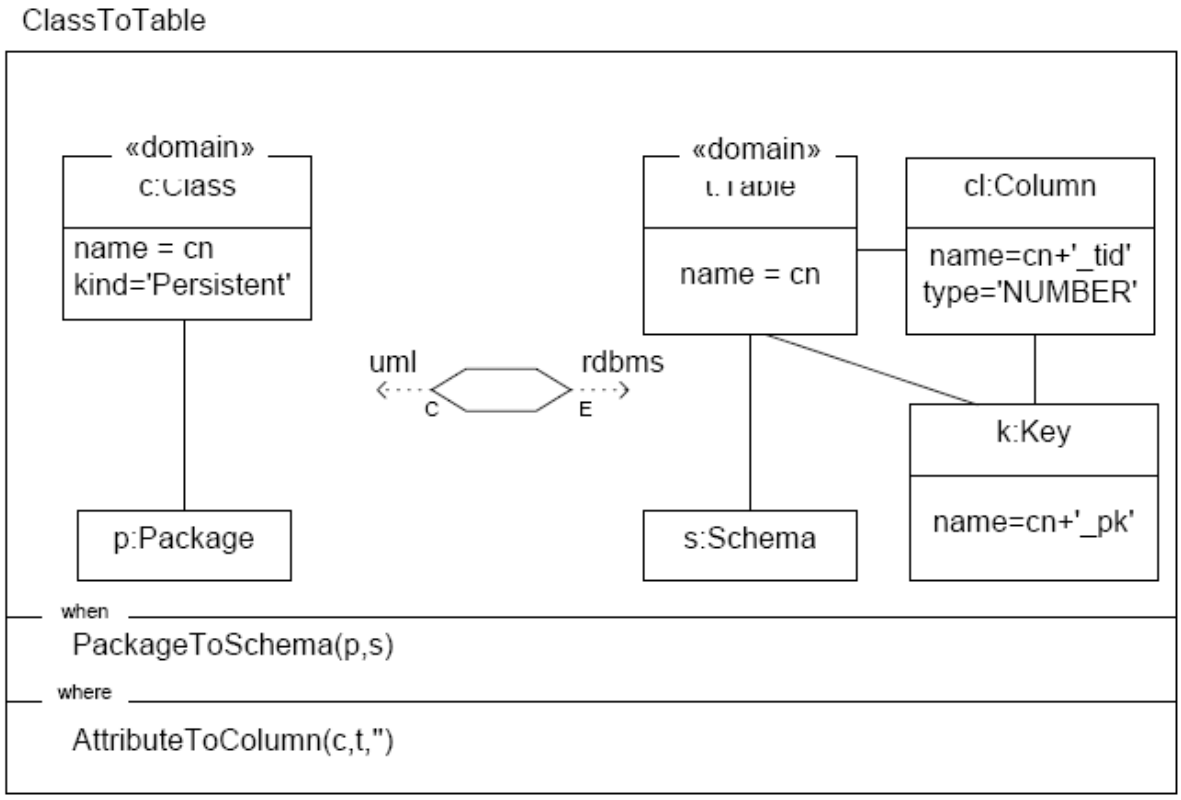

Figure A.4 - ClassToTable relation

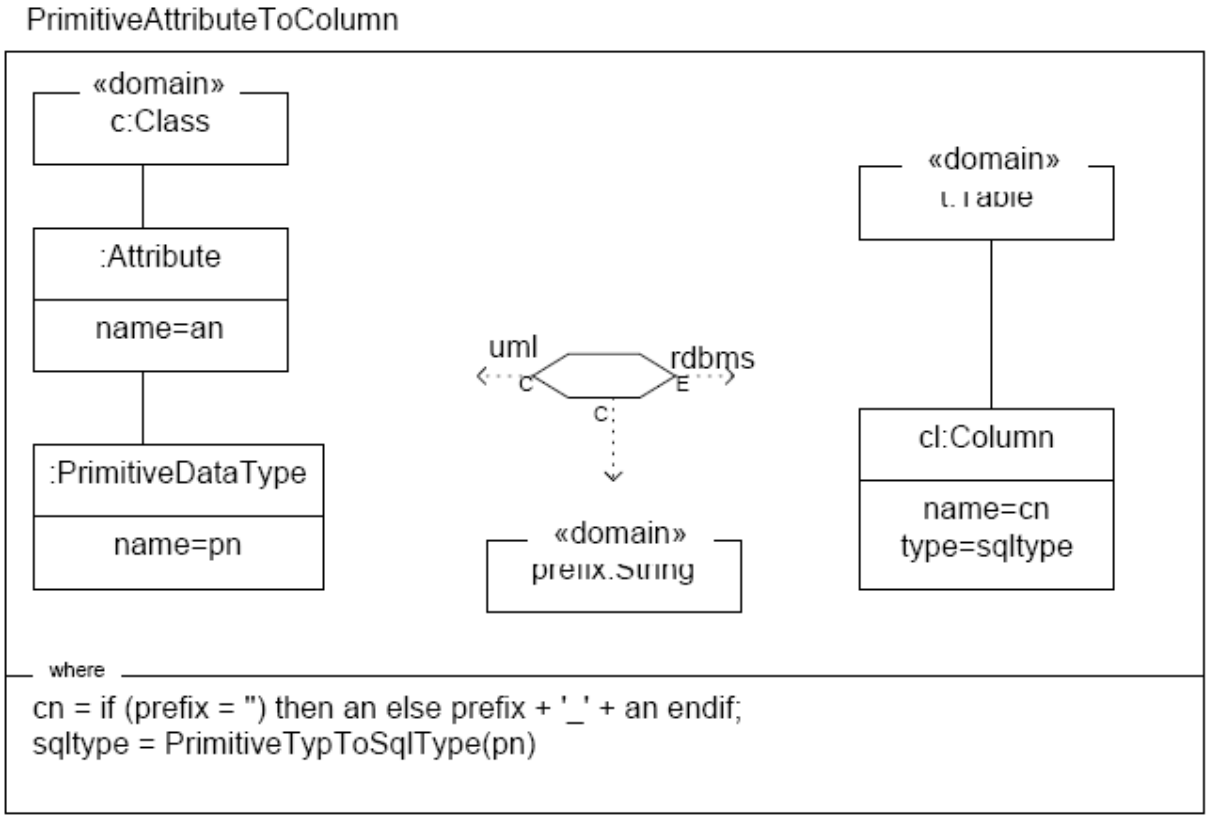

Figure A.5 - PrimitiveAttributeToColumn relation

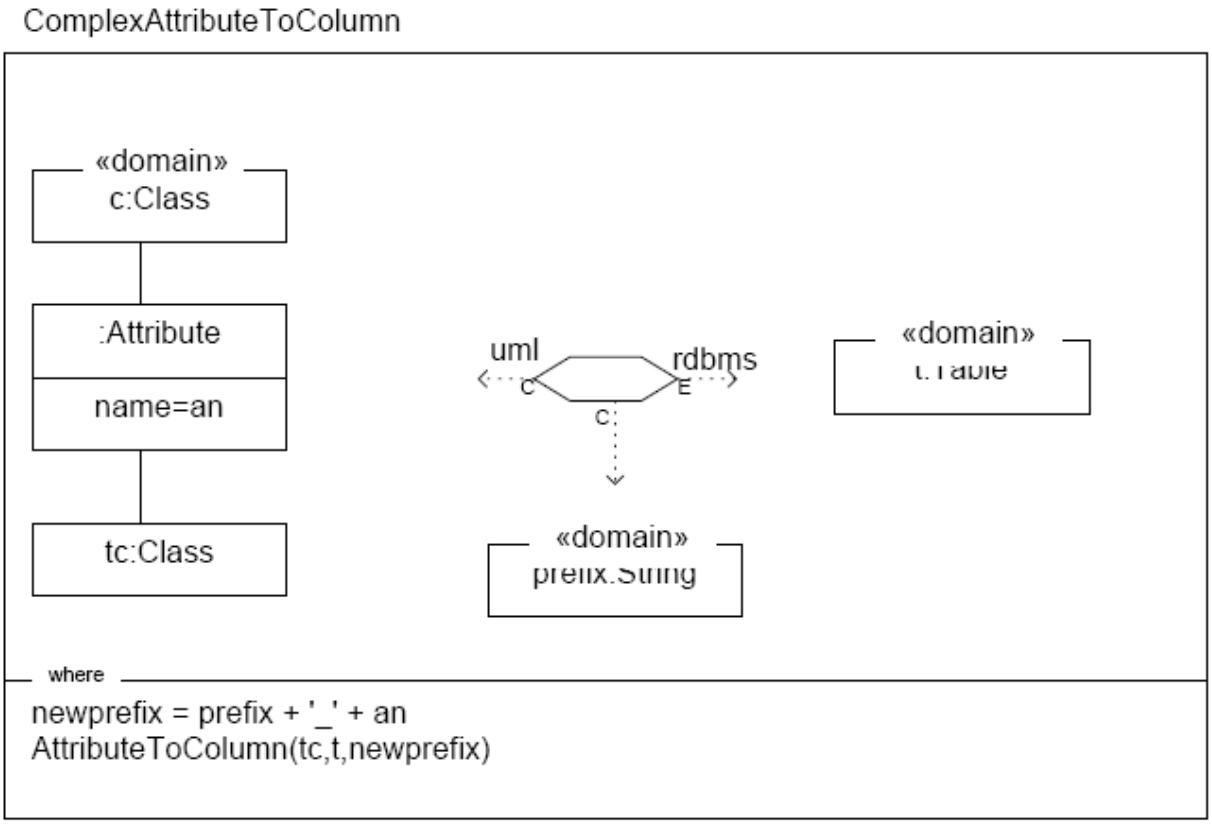

Figure A.6 - ComplexAttributeToColumn relation

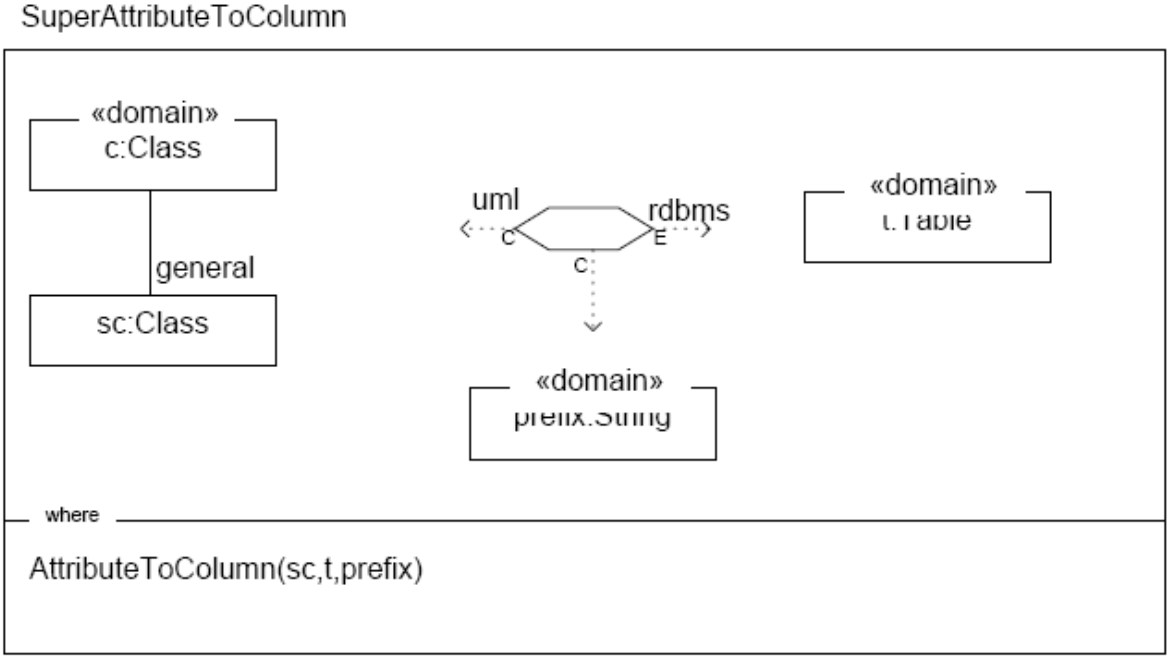

Figure A.7 - SuperAttributeToColumn relation

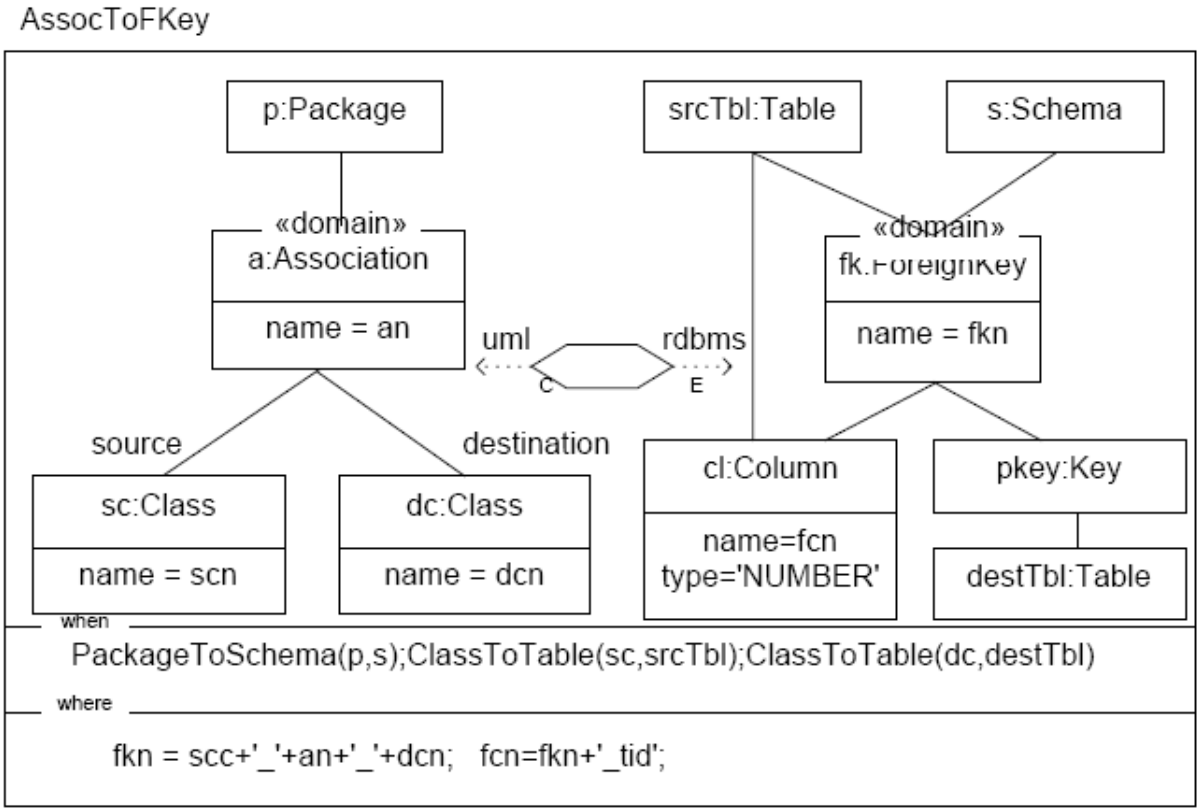

Figure A.8 - AssocToFKey relation