Algebraic Specification of a Model Transformation Engine[♦]

Artur Boronat, José Á. Carsí, Isidro Ramos

Department of Information Systems and Computation
Polytechnic University of Valencia
Camí de Vera s/n
46022 Valencia-Spain
{aboronat | pcarsi | iramos}@dsic.upv.es

Abstract. In Model-Driven Engineering, a software development process is a sequence of manipulation tasks that are applied to models, where model transformations play a relevant role. MOMENT (MOdel manageMENT) is a framework that is integrated in the Eclipse platform. MOMENT provides a collection of generic set-oriented operators to manipulate EMF models. In this paper, we present the model transformation mechanism that is embodied by the *ModelGen* operator. This operator uses the term rewriting system Maude as transformation engine and provides support for traceability. *ModelGen* has been defined in an algebraic specification so that we can use formal tools to reason about transformation features, such as termination and confluence. Furthermore, its application to EMF models shows that formal methods can be applied to industrial modeling tools in an efficient way. Finally, we indicate how the *ModelGen* operator provides support for the QVT Relations language in the MOMENT Framework.

Keywords: Model-Driven Engineering, Model Transformation, QVT, Algebraic Specifications, Traceability.

1 Introduction

Nowadays, the Model-Driven Architecture (MDA) [1] and the Software Factories [2] initiatives are leading the Model-Driven Engineering field. Both agree that any software artifact in a software development process can be dealt with as a model. Models provide a more abstract description of a software artifact than the final code of the application. Therefore, working on models increases the productivity in a software development process. It also increases the portability and the quality of the final code by applying generative techniques. In MDA, model transformations have become a relevant issue by means of the forthcoming standard Query/Views/Transformations (QVT) [3]. Since any software artifact can be viewed as a model, model transformation is the basic mechanism that permits the manipulation of software artifacts [4].

[♦] This work was supported by the Spanish Government under the National Program for Research, Development and Innovation, DYNAMICA Project TIC 2003-07804-C05-01.

Within this arena, the Model Management discipline [5] considers models as first-class citizens and provides a set of generic operators to manipulate them: *Merge*, *Diff*, *ModelGen*, etc. We have developed a framework, called MOMENT (MOdel manageMENT) [6], that is embedded in the Eclipse platform and that provides a set of generic operators to deal with models through the Eclipse Modeling Framework (EMF). An algebra of model management operators has been specified generically by using the Maude algebraic specification formalism [7] in the MOMENT framework.

In this paper, we focus on the *ModelGen* operator, the model transformation mechanism of MOMENT, which was presented as a proposal in [8]. This operator provides support for the QVT Relations language and also provides support for traceability. *ModelGen* is used by the other model management operators of the Framework when a model manipulation has to be performed. Since the *ModelGen* operator is algebraically specified in Maude, this term rewriting system is used as the underlying runtime environment for model transformations in MOMENT. This fact provides an efficient environment to execute the *ModelGen* operator and a formal environment where algebraic features can be proved, such as the confluence and the termination of a model transformation.

The structure of the paper is as follows: Section 2 presents an overview of the QVT support in the MOMENT Framework and an example; Section 3 presents the algebraic specification of the *ModelGen* operator, how it is related to a QVT transformation, the execution of a model transformation in the Framework, and the analysis of the complexity of the operator; Section 4 compares our approach with other model transformation tools. Finally, Section 5 summarizes the main contributions of this paper.

2 Model Transformations in the MOMENT Framework

The *ModelGen* operator embodies the model transformation mechanism in the MOMENT Framework. This operator has been algebraically specified, although we use it to manipulate graphical models. To deal with models from an industrial standpoint and to manipulate them from a mathematical standpoint, we use two complementary Technical Spaces. A Technical Space (TS) is a working context with a set of concepts, a body of knowledge, tools, required skills, and possibilities [9]. We use the EMF and Maude technical spaces in our Framework. The former is characterized by its interoperability with industrial tools for solving actual Software Engineering problems. The latter constitutes the formal backbone of our model management approach. To achieve the interoperability between them, two kinds of bridges are used in our Framework: two for regular metamodels and one for the QVT Relations metamodel.

For regular metamodels, two bridges are defined between the two technical spaces, at the M2-layer and at the M1-layer (using the Meta-Object Facility [10] terminology). Both of these permit the integration of MOMENT with EMF. The projection mechanism at the M2-layer automatically obtains the algebraic specification[1] that corresponds to a regular metamodel, by applying generative programming techniques. An algebraic specification that is generated in this way provides the constructors that are needed to define models of the corresponding metamodel as sets. The inverse projection mechanism that obtains an EMF metamodel from an algebraic specification is not relevant in our tool because the algebraic specification must conform to several features in order to be used by our operators and they should be automatically achieved. We also think that visual modeling environments are more suitable for defining these metamodels. At the M1-layer, we have developed a bidirectional projection mechanism that permits us to project an EMF model as a term of an algebra and to project a term of a metamodel algebra as an EMF model. In this case, bidirectionality is needed to apply an operator to an input model since the input model must be serialized as a term and the output term must be deserialized into an EMF model in order to be persisted.

Another bridge is defined for the QVT Relations metamodel. This permits the projection of a QVT transformation as an algebraic specification. This specification constitutes the axiomatic presentation of the *ModelGen* operator, when a transformation is defined among several metamodels.

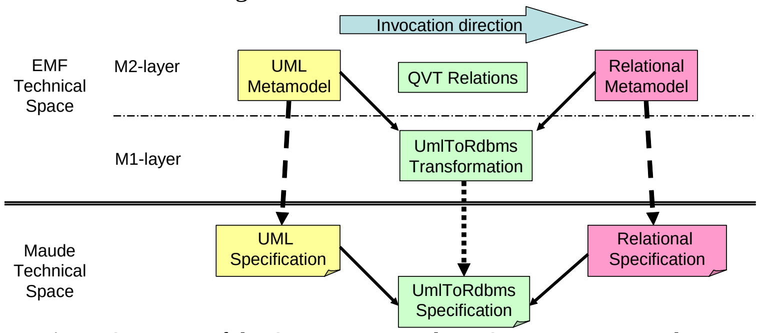


Fig. 1. Overview of the QVT support in the MOMENT Framework

In Fig. 1, the projection bridges between the two Technical Spaces (EMF and Maude) that are used in this work are shown. On the one hand, the discontinuous arrows represent the projection mechanism that obtains an algebraic specification for a regular EMF metamodel. On the other hand, the dotted arrow represents the projection mechanism that obtains the algebraic specification of a model transformation between several metamodels, when it is invoked in one direction. The importation of metamodels in an EMF QVT transformation model is also projected into the Maude TS by using the Maude module importation mechanism (shown as continuous arrows in the figure). Taking into account both kinds of projectors, we can deal with a QVT Relations model as the description of a model transformation or as a regular model. Therefore, it can be transformed by using another QVT Relations

[1] The algebraic specification that is generated for a given metamodel (defined in EMF as an Ecore model) permits the representation of models as algebraic terms. Thus, models can be manipulated by our model management operators. Algebraic specifications of this kind do not specify operational semantics for the concepts of the metamodel; they only permit the representation of information for model management issues.

model. This fact permits us to define the *ModelGen* operator as a mechanism to obtain higher order transformations through the QVT Relations language.

2.1 A Running Example

We have chosen the UmlToRdbms transformation that is presented in the QVT-Merge proposal [3] as an example to illustrate the use of the *ModelGen* operator in the MOMENT Framework. The Ecore metamodel [11] has been used as implementation of the UML metamodel. The Relational metamodel of the QVT-Merge proposal has been specified as an EMF metamodel. Using both metamodels, the UmlToRdbms transformation is applied to the source UML model in Fig. 2 to obtain the target relational schema, which is shown in the figure by using the default EMF graphical modeller.

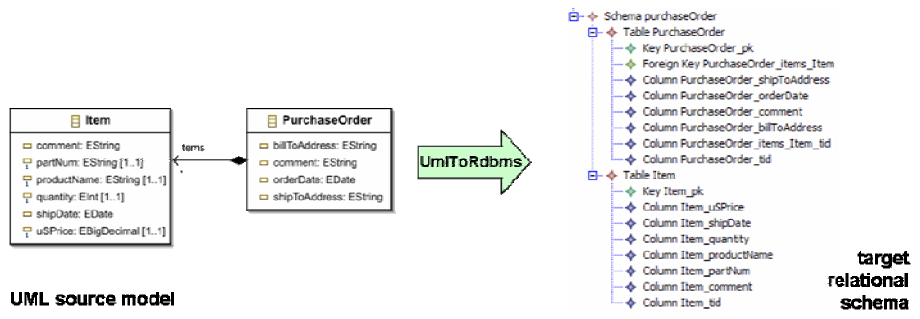


Fig. 2. Example of transformation of a UML model into a relational schema.

3 The Model Transformation Mechanism in MOMENT: the *ModelGen* Operator

In this section, an overview of the QVT Relations language support in MOMENT that is based on the *ModelGen* operator is provided. We indicate how the model transformation in Section 2.1 is specified with the *ModelGen* operator, taking the Relations language as reference point.

3.1 Overview

In MOMENT, a model transformation can be applied to several source models, which may or may not conform to the same metamodel. It generates one target model and a set of traceability models. A traceability model contains a set of traces that relate the elements of the source model to the elements of the target model, indicating which transformation rule has been applied to each source element. To apply a transformation to one or several models in MOMENT, two steps must be performed:

1. The transformation has to be defined as a model of the QVT Relations metamodel by means of a graphical interface or as a program using the Relations language. The transformation model can either be defined by the user or automatically produced by another transformation.
2. The transformation must be invoked by indicating the actual models to be transformed. In MOMENT, this second step is performed automatically. This step can be divided into three phases:
   a) The generation of the Maude modules that contain the algebraic specification of the model transformation operational semantics.
   b) The loading of the corresponding modules into Maude.
   c) The invocation of the *ModelGen* operator, indicating the input models for the transformation. To apply the operator, the input models are projected as terms of the algebra that is obtained from the metamodel of the corresponding input model. The term that is constituted by the operator call and the input terms is thenreduced by Maude, thereby performing the transformation. The resulting models are projected as EMF models. Therefore, any plug-in that provides the suitable graphical editor for these models can be used to show them.

In this paper, we focus on the second step, indicating the structure of the *ModelGen* operator and how it is related to the QVT Relations metamodel.

3.2 Relations: the *ModelGenRule* operator

A QVT relation specifies a relationship that must hold between the model elements of different candidate models. When one or more domains of the relation are enforceable, the relation describes a model transformation. In MOMENT, a QVT relation is projected into several directed transformation rules for the *ModelGen* operator. There is exactly one transformation for each direction in which the relation can be invoked.

The operation *ModelGenRule* that permits the definition of directed transformation rules is declared as follows:

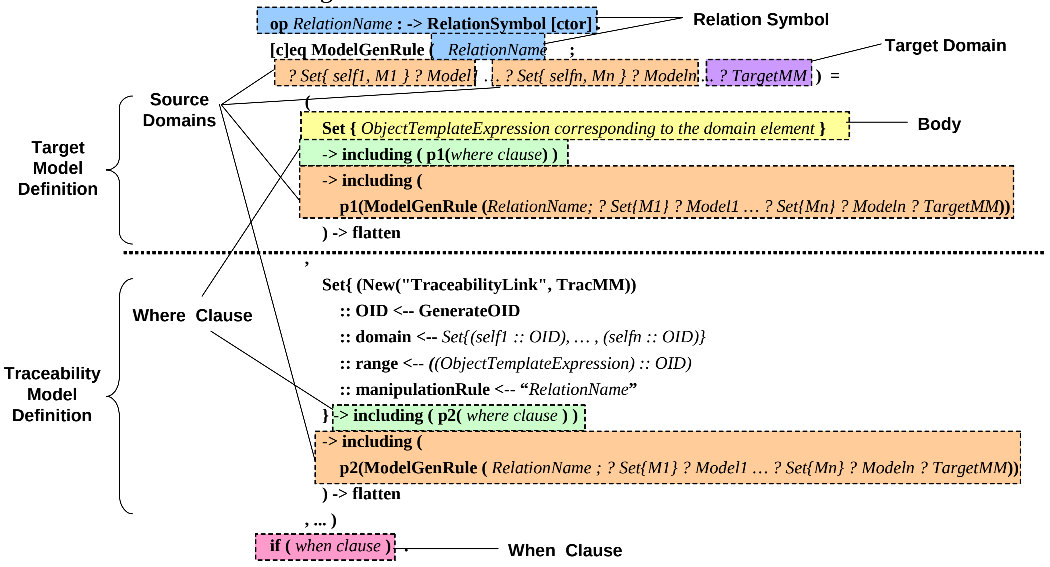
```
op ModelGenRule`(_;_`) : RelationSymbol ParameterList ->
     Tuple{ TTargetMM, TTraceabilityMetamodel,…} .
```

The *ModelGenRule* operator has two parameters: the name of the relation (a term of the sort RelationSymbol) and a polymorphic list of parameters. The membership equational logic [7] is used to define the operational semantics of this operator by means of equations. To use the Maude equational deduction as the runtime engine for our transformation rules, we need them to be confluent and terminating. The first parameter makes the confluence satisfaction easier to achieve. It permits to differentiate two transformation rules, even though they have the same list of parameters. Therefore, we avoid the situation where several equations can be applied to reduce the same term. We discuss the termination issue in more depth in Section 3.2.2. The second parameter permits the definition of a polymorphic list of parameters for the transformation rule. This means that we can define a parameter of any type (either a model or a basic type) as input for the transformation rule.

The result of a transformation rule is a tuple of several elements, where an element can be a model of any of the metamodels involved in the transformation. Among the models that are produced by a transformation rule, we distinguish the target model and several traceability models. There is one traceability model for each pair *(source model, target model).*

3.2.1 Structure of a transformation rule in MOMENT

We can only know the direction of a transformation by means of the enforceable property during the invocation process. During this process, the *ModelGenRule* axioms that specify the operational semantics of the relation are generated to be invoked later. Fig. 3 shows the two parts that constitute a *ModelGenRule* axiom: the specification of the model transformation and the specification of the traceability model definition. The Maude code that is generated for each part of a QVT relation is structured as shown in Fig. 3.



```
op RelationName : -> RelationSymbol [ctor] .
[c]eq ModelGenRule ( RelationName   ;
 ? Set{ self1, M1 } ? Model1 ... ? Set{ selfn, Mn } ? Modeln ... ? TargetMM ) =
(
  Set { ObjectTemplateExpression corresponding to the domain element }
  -> including ( p1(where clause) )
  -> including (
    p1(ModelGenRule (RelationName; ? Set{M1} ? Model1 … ? Set{Mn} ? Modeln ? TargetMM))
  ) -> flatten
,
  Set{ (New("TraceabilityLink", TracMM))
    :: OID <-- GenerateOID
    :: domain <-- Set{(self1 :: OID), … , (selfn :: OID)}
    :: range <-- ((ObjectTemplateExpression) :: OID)
    :: manipulationRule <-- "RelationName"
  } -> including ( p2( where clause ) )
  -> including (
    p2(ModelGenRule ( RelationName ; ? Set{M1} ? Model1 … ? Set{Mn} ? Modeln ? TargetMM))
  ) -> flatten
, ... )
if ( when clause ) .
```

Fig. 3. Axiom for the *ModelGenRule* operator.

- The Relation Symbol. This is the name of the transformation rule. It is used to define a constructor for the sort *RelationSymbol*, which is used in the *ModelGenRule* axioms for the sake of confluence.
- The Domains. They constitute the patterns of the transformation rules and the body. Two kinds of domains can be distinguished in the transformation invocation:
  - The domain that is selected as target when the transformation is invoked is used to generate the body of the transformation rules. In the body of the transformation, object template expressions are used to create new instances in the target model and to obtain the information from the source models. In these object template expressions, OCL is used as the query language and new functions can be declared in a transformation to manipulate data. On the one hand, a parametric algebraic specification has been defined to provide the

operational semantics for OCL 2.0 expressions[2] in MOMENT. On the other hand, Maude itself is suitable to define the operational semantics of QVT functions.

  – The source domains of the QVT relation are used to define the pattern of the transformation rule. In the pattern, each input model appears twice. This is needed to achieve two goals at the same time: to use the pattern matching mechanism that Maude provides by means of recursion and to provide support for OCL expressions. The first model is used to search the corresponding element of the pattern that is needed in the transformation rule. The second model is needed because Maude does not provide support for side effects, as in pure functional programming languages. This forces us to keep the whole model throughout the term reduction process in order to navigate it by means of OCL expressions, which can be used in the guard of the pattern. In the following section, we study the pattern matching mechanism in further detail.

– The *When clause.* This clause is used as a precondition for the transformation rule. Therefore, this guard participates in the pattern matching mechanism. It is used to obtain the condition of the equations that constitute the transformation rule. The resulting equations are applied by the conditional pattern matching mechanism of Maude.
– The *Where clause*: this clause is used as postcondition for a transformation rule. In MOMENT, it is used to generate the code that invokes QVT relations that are not defined as *top relations* in the transformation. In this clause, variables can also be initialized with new values to be used in the transformation.

In the QVT proposal, the traceability support is implicit in the QVT Relations language and is explicit in the QVT Core language. The *ModelGen* operator also generates traceability models automatically, but its definition has to be explicitly specified in the *ModelGenRule* axioms. The traceability model produced by the *ModelGen* operator conforms to the traceability metamodel that is presented in [12]. Thus, it can also be manipulated as just another model by the operator *ModelGen*.

In the second part of the axiom shown in Fig. 3, a new trace (instance of the TraceabilityLink class) is added to the traceability metamodel. Its *domain* field is filled with the identifiers (or references) of the source elements that have been matched with the pattern. Its *range* field is filled with the identifiers of all the instances that are created in the body of the transformation rule. Finally, the *manipulationRule* field indicates which transformation rule has been applied. Since the axioms for the *ModelGenRule* transformation rules are automatically generated from a QVT relations model, the traceability support in MOMENT is automatically generated and is kept hidden from the final user.

3.2.2 Pattern Matching

The pattern matching mechanism that is used to apply the transformation rules to source models is provided by Maude through its associative commutative (AC) matching algorithm [13]. To understand how the *ModelGenRule* uses it, we study the

[2] We do not provide further details on OCL 2.0 support because it is out of the scope of this paper.

parts of a transformation rule that are involved in the pattern matching: the *source domains* and the *when clause*.

```
op _,_ : Magma{X} Magma{X} -> Magma{X} [assoc comm ctor] .
op Set{_} : Magma{X} -> Set{X} [ctor] .
```

A source domain is a model used as input in a transformation. It is a set that is constituted by an associative commutative magma of elements. The generic constructors for a Set are the following:

Taking into account that the sort that represents model elements is a subsort of *Magma{X}*, a UML model that is constituted by a package and a class can be defined as a set with the following term: *Set{(UML-Package …) , (UML-Class …) }*; where *UML-Package* and *UML-Class* are the constructors that correspond to these concepts in the UML metamodel. By using a recursion mechanism we can use the Maude pattern matching mechanism. We define the following variables: the variable $self_i$ can be bound to a term that represents any element of a $Model_i$ model. The variable M_i can be bound to any magma of elements of a $Model_i$ model. For each QVT relation, we obtain two axioms for the *ModelGenRule* operator by using the following patterns:

– The base case *Set{* $self_i$ *}*: the right-hand side of this equation is constituted by the tuple of the target model and the traceability models.

eq ModelGenRule (RelationName ; ? Set{ $self_i$ *} ?* $Model_i$ *… ? TargetMM) = (Set{…}, Set{…},…) .*

– The regular case *Set{* $self_i$ *,* M_i *}*: the right-hand side of this equation is constituted by the tuple of the target model and the traceability models, but also contains the application of the recursive transformation rule *RelationName* to the models that are constituted by the residuary magmas of elements. In the first element of the tuple, we apply the recursive operation and we add the first argument of the returning tuple by means of *p1*. In the second element, we use *p2* to get the second element of the returning tuple.

eq ModelGenRule (RelationName ; ? Set{ $self_i$*,* M_i *} ? Model … ? TargetMM) =*
(Set{…}->including(p1(ModelGenRule (PackageToSchema ; ? Set{ M_i *} ?* $Model_i$ *? TargetMM)))->flatten*
,
Set{…}->including(p2(ModelGenRule(PackageToSchema ; ? Set{ M_i *} ?* $Model_i$ *? TargetMM)))->flatten) .*

The *when clause* also provides relevant information to determine whether or not an axiom can be matched to perform a transformation. When the guard described in the *when clause* does not contain an OCL query that searches elements through a model, it can be added to the axiom as a condition, turning the axiom into a conditional equation. When the guard contains an OCL query, the guard is added as an *if…then…else…fi* clause in the body of the equation. Both considerations are needed for the sake of the efficiency of the Maude AC matching algorithm.

When no axiom of the *ModelGenRule* operator can be applied, there is a general axiom for all the transformation rules that is be applied by default. This axiom guarantees the termination of a transformation and is specified as follows:

```
var TR : RelationSymbol .        var PL : ParameterList .
eq ModelGenRule (TR ; PL) = (empty-set, empty-set ) [owise] .
```

As an example of a *ModelGenRule* axiom, we show the axiom that represents the regular case pattern for the following relation *PackageToSchema of the UmlToRdbms transformation* (when the transformation is applied to a UML model):

```
QVT Relation:
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};
}

Maude Axiom:
ceq ModelGenRule (PackageToSchema ; ? Set{ self, M } ? Model ? TargetMM) =
(
     Set{ (New("Schema", TargetMM))
          :: OID <-- (self :: OID)
          :: name <-- (self :: name)
     } -> including ( p1(ModelGenRule (PackageToSchema ; ? Set{ M } ? Model ? TargetMM))) -> flatten
,
     Set{ (New("TraceabilityLink", TracMM))
          :: OID <-- generateOID
          :: domain <-- Set{ (self :: OID) }
          :: range <-- Set{ (self :: OID) }
          :: manipulationRule <-- "EPackage-to-Schema"
     } -> including( p2(ModelGenRule (PackageToSchema ; ? Set{ M } ? Model ? TargetMM)) ) -> flatten
)
if (self :: ecore-EPackage) .
```

3.3 Transformations: the *ModelGen* operator

The *ModelGen* operator provides the model transformation mechanism in the MOMENT Framework. This operator corresponds to a *QVT Transformation* when it is invoked in a specific direction. It uses the *ModelGenRule* operator, whose axioms describe the *QVT relations* associated to the transformation. The *ModelGen* operator is declared as follows:

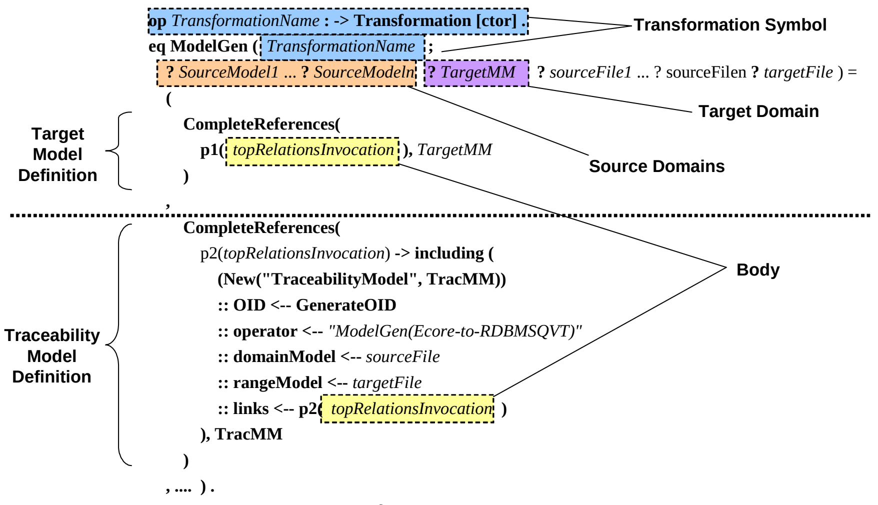
```
op ModelGen`(_;_`) : TransformationSymbol ParameterList
-> Tuple{ TTargetMM, TTraceabilityMetamodel,… } .
```

Similarly to the *ModelGenRule* operator, it has two formal parameters: the symbol that represents the name of the transformation and a polymorphic list of parameters for the transformation. The result of the operator is a tuple that is constituted by the resulting target model and by traceability models. There is one traceability model for each pair *(source model, target model).*

The code that is generated for a transformation includes the definition of the symbol of the transformation and an axiom that specifies the operational semantics of the transformation. Fig. 4 shows the structure of an axiom for the *ModelGen* operator.

With regard to the definition of a QVT transformation, we identify the following parts:

– The transformation symbol: represents the name of the transformation.
– The parameters of the transformation:
  – Source Domains. *? SourceModel1 … ? SourceModeln* constitute a list of terms that represent the source models of the transformation.
  – Target Domain. *TargetMM* is a term that represents the target metamodel. It is used to create new instances of the classes that appear in the metamodel by means of the Maude reflection mechanism.
  – List of names of the input and target models. This list is used to define traceability models. This information is needed to indicate the models that are related by means of a traceability model.
– Target model definition. This is the first argument of the resulting tuple, where the target model is generated by applying the *ModelGenRule* axioms (which describe the top QVT relations) to the elements of the source models.
– Traceability model definition. The other arguments of the resulting tuple are the traceability models that relate the elements of the target model to those of each source model. Furthermore, a new instance of the *TraceabilityModel* class is created to relate a source model to the target one.



```
op TransformationName : -> Transformation [ctor] .
eq ModelGen ( TransformationName ;
 ? SourceModel1 ... ? SourceModeln ? TargetMM  ? sourceFile1 ... ? sourceFilen ? targetFile ) =
  (
    CompleteReferences(
      p1( topRelationsInvocation ), TargetMM
    )
  ,
    CompleteReferences(
      p2(topRelationsInvocation) -> including (
        (New("TraceabilityModel", TracMM))
        :: OID <-- GenerateOID
        :: operator <-- "ModelGen(Ecore-to-RDBMSQVT)"
        :: domainModel <-- sourceFile
        :: rangeModel <-- targetFile
        :: links <-- p2( topRelationsInvocation )
      ), TracMM
    )
  , .... ) .
```

Fig. 4. Axiom for the *ModelGen* operator.

The operator *CompleteReferences* is needed to simplify the definition of transformation rules when the metamodel has opposite references. In EMF, UML associations that may appear in a MOF metamodel are defined by means of opposite references (two instances of the *EReference* class) [11]. If the metamodel has opposite references, both references must be initialized when a model is defined. This fact can make the definition of transformations more complex. Although the EMF editor solves this problem automatically, an independent solution is needed. The *CompleteReferences* operator fulfils this goal. Thus, it permits MOMENT to remain independent of any other technological space that is different from the Maude TS and to compose several transformations without loss of information. This operator has two input parameters: the model to be completed with references and the corresponding metamodel, which is expressed as a model of the Ecore metamodel. The *CompleteReferences* operator uses the Maude reflection mechanism to traverse the whole input model in order to complete the forgotten references. The output of the operator is the completed model.

Taking into account the example of the UmlToRdbms transformation that is presented in the QVT Merge proposal, we indicate the Maude code that provides the operational semantics of the transformation when it is invoked to transform a UML model into a relational schema. In the *ModelGen* axiom, we have only added the code to obtain the first element of the returning tuple, i.e. the target model.

```
QVT Transformation:
transformation umlToRdbms(uml:SimpleUML, rdbms:SimpleRDBMS)
{
      top relation ClassToTable {…}
      top relation PackageToSchema {…}
      top relation AssocToFKey {…}
      …
}
Maude code:
op umlToRdbms : -> Transformation [ctor] .
eq ModelGen (umlToRdbms; ? Model ? TargetMM ? sourceFile ? targetFile ) =
(
      CompleteReferences(
            p1(ModelGenRule (PackageToSchema ; ? Model ? Model ? TargetMM))
            -> including(
                  p1(ModelGenRule (ClassToTable ; ? Model ? Model ? TargetMM)))
            -> including(
                  p1(ModelGenRule (AssocToFKey ; ? Model ? Model ? TargetMM)))
            -> flatten
      , TargetMM)
, … ) .
```

3.4 Operator execution

In this section, we present how MOMENT executes the *ModelGen* operator, transforming the UML model of the example in Section 2 into a relational schema. Fig. 5 shows the two MOF layers involved in a model transformation: the M2-layer, where the metamodels and the model transformation are defined; and the M1-layer, where the models are defined and manipulated. The front part of the figure represents the front-end of the MOMENT framework, i.e. EMF and all the plug-ins that are built on it. The back part of the figure represents the formal back-end of the MOMENT Framework, where Maude remains. The traceability support has not been taken into account in the figure.

Fig. 5 represents the transformation of the UML model by using the *ModelGen* operator that is generated from the QVT transformation of the example. The steps that are automatically performed by the MOMENT Framework when the *ModelGen* operator is applied to the source UML model are the following:

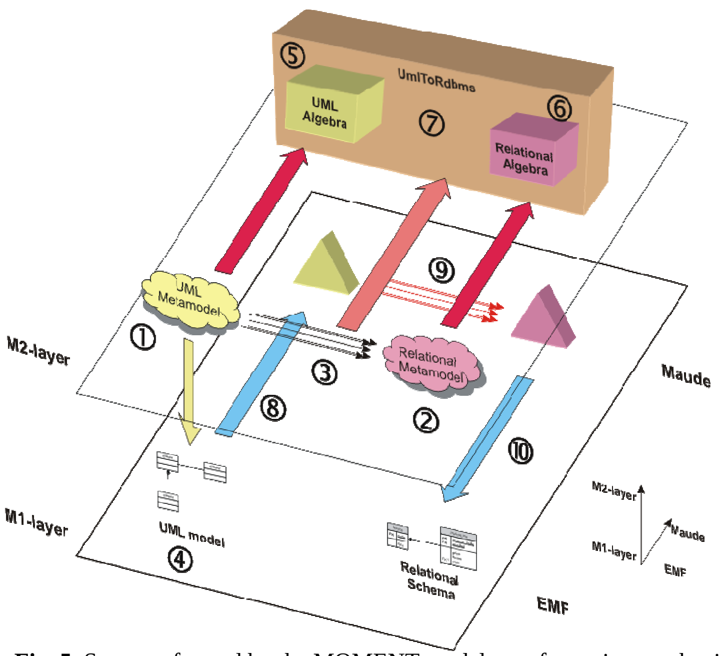


Fig. 5. Steps performed by the MOMENT model transformation mechanism

- *(1) and (2):* We specify both UML and Relational metamodels at the M2-layer by means of the EMF or graphical editors based on this modelling framework. For instance we can also consider XML schemas and Rational Rose models as metamodels.
- *(3)*: The QVT transformation is defined as a model at the M1-layer, but it relates the constructs of the source to the constructs of the target metamodels.
- *(4)*: We define a UML model using a UML graphical editor based on EMF.
- *(5) and (6)*: Both Ecore and UML metamodels, respectively, are projected as algebraic specifications by means of the interoperability bridges that have been implemented in the MOMENT Framework. These algebraic specifications permit each metamodel to be considered as an algebra that provides the constructors to build models as algebraic sets.
- *(7)*: The model that defines the QVT transformation is projected into the Maude TS as the UmlToRdbms algebraic specification, which contains the specification of the *ModelGen* and *ModelGenRule* operators.
- *(8)*: The source UML model, which is defined in step (4) at the M1-layer, is projected into the Maude TS as a term of the UML algebra.
- *(9)*: Maude applies the *ModelGenRule* axioms through its equational deduction mechanism. Thus, Maude constitutes the runtime engine for the MOMENT transformation mechanism.
- *(10)*: This is the last step of the model transformation process. It parses the obtained term in step (9), defining a EMF model in the M1-layer as an instance of the target metamodel defined at the M2-layer.

In the model transformation process, the user only interacts with the MOMENT platform when defining the source and target metamodels (step (1) and (2)), the QVT transformation between both metamodels (step (3)) and the source model (step (4)). The other steps are automatically carried out by the Framework. The output model can also be manually manipulated from a graphical editor.

3.5 Analysis of the Complexity of the *ModelGen* operator

To analyze the complexity of the *ModelGen* operator, we have to take into account the temporal cost of both the *ModelGenRule* operator and the *CompleteReferences* operator.

[13] presents the efficient AC matching algorithm that is used in Maude 2. This algorithm permits an efficient matching of the left-hand side of an equation to a term, when there is a symbol in the term and in the left-hand side of the equation that has been defined as associative and commutative. For instance, the Magma concatenation operation represented by the symbol "," is associative and commutative. The AC matching algorithm permits the substitution of a left-hand side like *Set{self, M}* in a term that represents a model. If we consider that the element *self* must satisfy a guard, the AC matching algorithm determines whether or not an equation can be applied. If we consider *m* as the size of elements of a model, this algorithm can detect the failure of the left-hand side of an equation of this type in *O(log(m))* time. Furthermore, when the equation can be applied, this algorithm can compute a matching substitution in *O(log(m))* time.

Taking into account the complexity of this efficient algorithm, we detect the best and the worst case for the *ModelGenRule* operator. Both cases are constituted by a transformation that contains only one transformation rule. In the best case, the transformation rule fails and the default axiom must be applied, finishing the model transformation in *O(log(m))* time. In the worst case, the transformation rule is applied to all the elements of the model and the computation finishes in *O(m · log(m)) + O(m · R)* time, where *R* is the temporal cost associated to the computation of a transformation rule. OCL queries also use the AC matching algorithm to traverse the models. Therefore, the complexity of a transformation rule *R* is determinant in the analysis of the complexity of the transformation and must be taken into account by the user, since transformation rules are manually coded with the QVT Relations language.

Finally, assuming that the complexity of the *CompleteReferences* is logarithmic, the final temporal cost of a transformation is *O(m · log(m)) + O(m · rule)* in the worst case. To transform the UML model of the example in Section 2, Maude performed 798620 rewritings in 3328 ms.

4 Related works

We provide a brief comparison of the ModelGen operator to other model transformation mechanisms that are the most current in the Model-Driven Engineering field. The goal of the study is to compare their support for traceability and transformation organization.

MTF [14] is the IBM Model Transformation Framework, which implements some of the QVT concepts and is based on the EMF. It provides a simple declarative language for defining mappings between models. An MTF transformation results in a set of *mappings* that relate objects among different models. The direction of the transformation is defined when the transformation is invoked. Bidirectional transformations imply constraining the kind of model transformations that can be solved. For instance, transformations that produce loss of information cannot be taken into account.

ATL [15] is a model transformation language that provides declarative and imperative constructs. A transformation is constituted by several transformation rules but there is no mechanism to organize them by means of modules. Traceability support is not provided implicitly in a transformation; traceability models can be generated by a transformation as in the ModelGen operator. The expressions in a transformation rule are defined in OCL expressions, making the ATL language easy to learn and to use. The ATL model transformer is integrated in the NetBeans MDR repository and in the Eclipse platform, allowing the manipulation of EMF models. Tefkat [16] is a model transformation tool that is quite similar to ATL, which is also built on EMF. It incorporates the concept of tracking classes to define traces between the source model and the target model that is generated by the transformation. However, the traces must be defined explicitly in a transformation rule.

XSLT has become a popular alternative for describing model transformations. Tools, such as MTRANS [17] or UMT [18], follow this approach by serializing metamodels and models into XMI documents and then performing the transformations by means of XSLT specifications. A XSLT specification is also an XML document that describes a transformation using both declarative and imperative constructs. Nevertheless, the verbosity of the XML syntax sometimes leads to specifications that are difficult to read and to maintain [19]. XSLT 2 permits the definition of transformations that generate more than one XML document. Therefore, traceability support can be added explicitly to a XSLT transformation. This approach can also be applied to EMF models, which are persisted in XMI. EMF avoids the serialization of certain data, such as derived attributes or default values. Thus, this functionality must be embedded in the XSLT specification. This task can easily become cumbersome.

All these approaches deal with model transformations in a similar way. MTF is the only one that permits the definition of bidirectional relations to perform transformations and that provides implicit traceability support. In the other approaches, the traceability support must be explicitly codified in the transformation function to generate a traceability model. In MOMENT, the set of axioms that describe the ModelGen is automatically generated from a QVT Relations model. Thus, the traceability support is implicit. On the other hand, none of the above tools provide a solution to reuse transformations. In MOMENT, transformations are defined in algebraic modules that can be imported into others. Furthermore, the ModelGen operator can be easily composed with other operators by using functional composition since we are working in an algebraic context. This feature avoids the development of complex frameworks to deal with transformation composition.

5 Conclusions

In this paper, we have presented the ModelGen operator, which permits the definition of directed declarative transformations that can be applied to several source models, which may or may not conform to the same metamodel. Furthermore, a transformation can be parameterized with additional data that can act as control parameters allowing configuration and tuning. The return value of a transformation is a tuple that is constituted by a model and several traceability models. There is one traceability model for each pair *(source model, target model)*. ModelGen uses the efficient AC matching algorithm that is implemented in Maude 2 as the pattern matching mechanism to apply transformation rules to the source models. The traceability support that is provided in MOMENT [12] permits the definition of an incremental transformation operator in an easy way, similar to the PropagateChanges operator, which is also presented in [12].

Our formal approach for model transformation permits the application of advantageous features to this field, such as transformation composition or modularity. Furthermore, formal features (like confluence and termination) can also be studied. The MOMENT Framework shows that formal methods can be applied to industrial tools not only for proving theoretical aspects, but also for solving actual problems in an efficient manner. Nevertheless, an algebraic setting might not be the most user-friendly environment to work on models. This reason led us to provide support for QVT by using generative programming techniques, as shown in this paper. In this way, Maude remains hidden from the final user although its formal features are used in our framework.

References

1. Frankel, D. S.: Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons OMG Press. January, 2003.
2. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons. 2004.
3. QVT-Merge Group, “Revised Submission for MOF 2.0 Query/View/Transformation RFP(ad/2002-04-10)”, Version 2.0, ad/2005-03-02, March 2005
4. Sendall, S., Kozaczynski, W. Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software. September/October 2003 (Vol. 20, No. 5), pp. 42-45.
5. Bernstein, P.A: Applying Model Management to Classical Meta Data Problems. pp. 209-220, CIDR 2003.
6. The MOMENT web site: http://moment.dsic.upv.es:8080
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science, 285(2):187-243, 2002.
8. Boronat, A., Carsí, J.Á., Ramos, I.: Automatic Reengineering in MDA Using Rewriting Logic as Transformation Engine. IEEE Computer Society Press. 9th European Conference on Software Maintenance and Reengineering. Manchester, UK. 2005.
9. Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: An Initial Appraisal. Int. Federated Conf. (DOA, ODBASE, CoopIS), Industrial track, Irvine, 2002.
10. OMG, “Meta Object Facility 1.4”, http://www.omg.org/technology/documents/formal/mof.htm
11. EMF web site: http://www.eclipse.org/emf/

12. Boronat, A., Carsí, J.Á., Ramos, I.: Automatic Support for Traceability in a Generic Model Management Framework. In Proceedings of European Conference on Model Driven Architecture - Foundations and Applications. Nuremberg (Germany). 2005. (To appear)
13. Eker, S.: Associative-Commutative Rewriting on Large Terms. In Proceedings of the 14th International Conference on Rewriting Techniques and Applications. Valencia, Spain, 2003.
14. The MTF web site: http://www.alphaworks.ibm.com/tech/mtf
15. Bézivin, J., Dupé, G., Jouault, F. , Pitette, G., and Rougui, E.J.: First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In: OOPSLA 2003 Workshop, Anaheim, California.
16. Lawley, M., Steel, J.: Practical declarative model transformation with tefkat. In Model Transformations In Practice Workshop, Montego Bay, Jamaica, October 2005.
17. M. Peltier, Jean Bézevin, and G. Guillaume. MTRANS: A general framework, based on XSLT for model transformations. In WTUML '01, Proceedings of the workshop on Transformations in UML, Genova, Italy, 2001.
18. The UMT web site: umt-qvt.sourceforge.net/
19. A. Gerber, M. Lawley, K. Raymond, J. Steel, A. Wood. Transformation: The Missing Link of MDA, In A. Corradini, H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.): Graph Transformation: First International Conference (ICGT 2002), v1 15 Barcelona, Spain, October 7-12, 2002. Proceedings. LNCS vol. 2505, Springer-Verlag, 2002, pp. 90 - 105

A APPENDIX FOR REVIEWERS

In this section, we provide the complete model transformation in both the QVT Relations language and the corresponding ModelGen operator. The aim of this section is to prove that the MOMENT Framework automates the algebraic support that is provided to EMF. The model transformation chosen is the Uml2Rdbms transformation that is presented in the QVT-Merge proposal [3]. Nevertheless, as we have chosen the Ecore metamodel as implementation of the UML metamodel, we introduce slight differences in the model transformation. In appendix A.1, the complete model transformation that has been used is presented with the QVT Relations language. In appendix A.2, the corresponding algebraic modules that contain the specification of the corresponding ModelGen operator are shown in Maude notation. In appendix A.3, the source model to be transformed is presented in UML notation and as a term of the UML algebra. In appendix A.4, the application of the specified ModelGen operator to the source term in the Maude term rewriting system is shown. Finally, the model, which is generated from parsing the Relational algebra term, is shown in the default EMF graphical modeller.

A.1 The Uml2Rdbms transformation in QVT Relations language

```
transformation UmlToRdbms(uml:Ecore, rdbms:SimpleRDBMS)
{
    key Table (name, schema);
    key Column (name, owner); // owner:Table opposite column:Column
    key Key (name, owner); // key of class ‘Key’;
    // owner:Table opposite key:Key

    top relation PackageToSchema // map each package to a schema
    {
        pn: String;
        checkonly domain uml p:EPackage {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:EClass {
            namespace=p:EPackage {},
            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:EClass {};
        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:EClass
        {
            attribute=a:EAttribute {
                name=an,
                eType=p:EDataType {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 SuperAttributeToColumn
    {
        checkonly domain uml c:EClass {
            general=sc:EClass {}
        };
        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:EReference {
            ePackage=p:EPackage {},
            name=an,
            eContainingClass=sc:EClass {
                name=scn
            },
            eType=dc:EClass {
                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;
    }
}
```

A.2 The algebraic specification of the Uml2Rdbms ModelGen operator

```
fmod UML2RDBMS is
    pr specore .
    pr sprdbms .
    pr spTraceabilityMetamodel .
    pr TUPLE<2> { TRDBMS , TTraceabilityMetamodel } .

    *** **********************************************************
    *** **********************************************************
    ***
    *** VARIABLE DECLARATION SECTION
    ***
    *** **********************************************************
    *** **********************************************************


    vars OID1 OID2 : Qid .
    var PL : ParameterList .
    vars sourceFile targetFile : String .
    var Col : Collection{vM3} .
    var M : Magma{vM3} .
    var N : ecoreNode .
    var TN : rdbmsNode .
    vars Model Model1 Model2 Model11 Model22 Set  : Set{vM3} .
    vars TargetMM TracMM : Set{vM3} .
    var RDBMSCol : Collection{vrdbms} .
    var TR : TransformationRule .
    var primitiveType : String .
    var TableId : Qid .
    var prefix : String .
    var ownerId : Qid .
    vars name1 defaultValue1 instanceClassName1 instanceClass1 : String .
    vars abstract1 interface1 : Bool .
    vars   eAnnotations1   ePackage1   eSuperTypes1   eOperations1   eAllAttributes1   eAllReferences1
eReferences1 eAttributes1 eAllContainments1 eAllOperations1 eAllStructuralFeatures1 eAllSuperTypes1
eIDAttribute1 eStructuralFeatures1 : OrderedSet{QID} .
    var AttId : Qid .
    var MAttId : OrderedMagma{QID} .


    *** **********************************************************
    *** **********************************************************
    ***
    *** HELPER FUNCTIONS SECTION
    ***
    *** **********************************************************
    *** **********************************************************
    op isTable : -> BoolBody{vrdbms} .
    eq TN :: isTable ( PL ; RDBMSCol ) = TN :: rdbms-Table .

    op isKey : -> BoolBody{vrdbms} .
    eq TN :: isKey ( PL ; RDBMSCol ) = TN :: rdbms-Key .

    op PrimitiveTypeToSqlType : String -> String .
    eq PrimitiveTypeToSqlType(primitiveType) =
        if (primitiveType == "http://www.eclipse.org/emf/2002/Ecore#//EInt") then
            "NUMBER"
        else
            if (primitiveType == "http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal") then
                "NUMBER"
            else
                if (primitiveType == "http://www.eclipse.org/emf/2002/Ecore#//EBoolean") then
                    "BOOLEAN"
                else
                    if (primitiveType == "http://www.eclipse.org/emf/2002/Ecore#//EDate") then
```

```
                        "DATE"
                    else
                        "VARCHAR"
                    fi
                fi
            fi
        fi
    .

op IsPrimitiveDatatype : String -> Bool .
eq IsPrimitiveDatatype(primitiveType) =
    if ((find(primitiveType, "http://www.eclipse.org/emf/2002/Ecore#//", 0)) =/= notFound) then
        true
    else
        false
    fi .


*** **************************************************************
*** **************************************************************
***
*** MODELGEN OPERATOR SECTION
***
*** **************************************************************
*** **************************************************************
sort Transformation .

op ModelGen`(_;_`) : Transformation ParameterList -> Tuple{ TRDBMS , TTraceabilityMetamodel } .
op Uml2Rdbms : -> Transformation [ctor] .


eq ModelGen ( Uml2Rdbms ; ? Model ? TargetMM ? TracMM ? sourceFile ? targetFile ) =
    (
        CompleteReferences(
            p1(ModelGenRule (PackageToSchema ; ? Model ? Model ? TargetMM ? TracMM))
            -> including(
                p1(ModelGenRule (ClassToTable ; ? Model ? Model ? TargetMM ? TracMM))
            )
            -> including(
                p1(ModelGenRule (AssocToFKey ; ? Model ? Model ? TargetMM ? TracMM))
            ) -> flatten
            , MM((empty-set).Set{vrdbms})
        )
    ,
        CompleteReferences(
            p2(ModelGenRule (PackageToSchema ; ? Model ? Model ? TargetMM ? TracMM))
            -> including(
                p2(ModelGenRule (ClassToTable ; ? Model ? Model ? TargetMM ? TracMM))
            )
            -> including(
                p2(ModelGenRule (AssocToFKey ; ? Model ? Model ? TargetMM ? TracMM))
            )
            -> including (
                (New("TraceabilityModel", TracMM)).TraceabilityMetamodelNode
                :: OID <-- 'TraceabilityMetamodel-model
                :: operator <-- "ModelGen(Ecore-to-RDBMSQVT)"
                :: domainModel <-- sourceFile
                :: rangeModel <-- targetFile
                :: links <--
```

```
                        ((
                            p2(ModelGenRule (PackageToSchema ; ? Model ? Model ? TargetMM ?
TracMM))
                            -> including(
                                p2(ModelGenRule (ClassToTable ; ? Model ? Model ? TargetMM ?
TracMM))
                            )
                            -> including(
                                p2(ModelGenRule (AssocToFKey ; ? Model ? Model ? TargetMM ?
TracMM))
                            ) -> flatten
                            :: OID) -> asOrderedSet
                        )
                ) -> flatten
                , TracMM
            )
        )
    .

    *** ************************************************************
    *** ************************************************************
    ***
    *** MODELGENRULE OPERATOR SECTION
    ***
    *** ************************************************************
    *** ************************************************************

     sort TransformationRule .
    op ModelGenRule`(_;_`) : TransformationRule ParameterList -> Tuple{ TRDBMS ,
TTraceabilityMetamodel } [memo] .

    *** DEFAULT TRANSFORMATION RULE
    eq ModelGenRule (TR ; PL) =
        (
            (empty-set).Set{vrdbms}
        ,
            (empty-set).Set{vTraceabilityMetamodel}
        )
    [owise] .


    *** TRANSFORMATION RULES
    *** RULE 1: PackageToSchema
  op PackageToSchema : -> TransformationRule [ctor] .
    ceq ModelGenRule (PackageToSchema ; ? Set{ N, M } ? Model ? TargetMM ? TracMM) =
        (
            Set{
                (New("Schema", TargetMM)).rdbmsNode
                :: OID <-- (N :: OID)
                :: name <-- (N :: name)
            }
            -> including (
                p1(ModelGenRule (PackageToSchema ; ? Set{ M } ? Model ? TargetMM ? TracMM))
            ) -> flatten
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{ (N :: OID) }
                :: manipulationRule <-- "EPackage-to-Schema"
            }
            -> including (
                p2(ModelGenRule (PackageToSchema ; ? Set{ M } ? Model ? TargetMM ? TracMM))
            ) -> flatten
        )
    if (N :: ecore-EPackage)
    [metadata "EPackage-to-Schema"] .

    ceq ModelGenRule (PackageToSchema ; ? Set{ N } ? Model ? TargetMM ? TracMM) =
        (
            Set{
                (New("Schema", TargetMM)).rdbmsNode
                :: OID <-- (N :: OID)
                :: name <-- (N :: name)
            }
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{ (N :: OID) }
                :: manipulationRule <-- "EPackage-to-Schema"
            }
        )
    if (N :: ecore-EPackage)
    [metadata "EPackage-to-Schema"] .


    *** RULE 2: ClassToTable
op ClassToTable : -> TransformationRule [ctor] .
    ceq ModelGenRule (ClassToTable ; ? Set{ N, M } ? Model ? TargetMM ? TracMM) =
        (
            Set{
                (
                    ((New("Table", TargetMM)).rdbmsNode
                    :: OID <-- (N :: OID)
                    :: schema <--
                        (p1
                            (ModelGenRule (
                                PackageToSchema ;
                                ? ((N :: ePackage(Model)) -> asSet)
                                ? Model
                                ? MM((empty-set).Set{vrdbms})
                                ? TracMM))
                        )
                    :: name <-- (N :: name)
                    :: key <--
                        Set{
                            (New("Key", TargetMM)).rdbmsNode
                            :: OID <-- qid(string(N :: OID) + "_pk")
                            :: name <-- ((N :: name) + "_pk")
                            :: column <--
                                (Set {
                                    (New("Column", TargetMM)).rdbmsNode
                                    :: OID <-- qid(string(N :: OID) + "_tid")
                                    :: name <-- ((N :: name) + "_tid")
                                    :: type <-- "NUMBER"
```

```
                                        })
                                }
                        )
                        :: column <--
                                (Set {
                                        (New("Column", TargetMM)).rdbmsNode
                                        :: OID <-- qid(string(N :: OID) + "_tid")
                                        :: name <-- ((N :: name) + "_tid")
                                        :: type <-- "NUMBER"
                                })
                ),
                (
                        (New("Column", TargetMM)).rdbmsNode
                        :: OID <-- qid(string(N :: OID) + "_tid")
                        :: name <-- ((N :: name) + "_tid")
                        :: type <-- "NUMBER"
                ),
                (
                        (New("Key", TargetMM)).rdbmsNode
                        :: OID <-- qid(string(N :: OID) + "_pk")
                        :: name <-- ((N :: name) + "_pk")
                        :: column <--
                                Set {
                                        (New("Column", TargetMM)).rdbmsNode
                                        :: OID <-- qid(string(N :: OID) + "_tid")
                                        :: name <-- ((N :: name) + "_tid")
                                        :: type <-- "NUMBER"
                                }
                )
        } -> including (
                p1(ModelGenRule (ClassToTable ; ? Set{ M } ? Model ? TargetMM ? TracMM))    ***
where
        )
        -> including (
                p1(ModelGenRule (AttributeToColumn ; ? Set{ N } ? Model ? 'none ? (N :: name) ?
TargetMM ? TracMM))   *** where
        ) -> flatten
    ,
        Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{ (N :: OID) :: qid(string(N :: OID) + "_tid") :: qid(string(N ::
OID) + "_pk") }
                :: manipulationRule <-- "EClass-to-Table"
        } -> including (
                p2(ModelGenRule (ClassToTable ; ? Set{ M } ? Model ? TargetMM ? TracMM))    ***
where
        )
        -> including (
                p2(ModelGenRule (PrimitiveAttributeToColumn ; ? Set{ N } ? Model ? 'none ? (N ::
name) ? TargetMM ? TracMM))   *** where
        ) -> flatten
    )
  if (N :: ecore-EClass)
  [metadata "EClass-to-Table"] .

  ceq ModelGenRule (ClassToTable ; ? Set{ N } ? Model ? TargetMM ? TracMM) =
    (
    Set{
        (
            ((New("Table", TargetMM)).rdbmsNode
            :: OID <-- (N :: OID)
            :: schema <--                                          *** when
                (p1
                    (ModelGenRule (
                        PackageToSchema ;
                        ? ((N :: ePackage(Model)) -> asSet)
                        ? Model
                        ? MM((empty-set).Set{vrdbms})
                        ? TracMM))
                )
            :: name <-- (N :: name)
            :: key <--
                Set{
                    (New("Key", TargetMM)).rdbmsNode
                    :: OID <-- qid(string(N :: OID) + "_pk")
                    :: name <-- ((N :: name) + "_pk")
                    :: column <--
                        (Set {
                            (New("Column", TargetMM)).rdbmsNode
                            :: OID <-- qid(string(N :: OID) + "_tid")
                            :: name <-- ((N :: name) + "_tid")
                            :: type <-- "NUMBER"
                        })
                }
            )
            :: column <--
                (Set {
                    (New("Column", TargetMM)).rdbmsNode
                    :: OID <-- qid(string(N :: OID) + "_tid")
                    :: name <-- ((N :: name) + "_tid")
                    :: type <-- "NUMBER"
                })
        ),
        (
            (New("Column", TargetMM)).rdbmsNode
            :: OID <-- qid(string(N :: OID) + "_tid")
            :: name <-- ((N :: name) + "_tid")
            :: type <-- "NUMBER"
        ),
        (
            (New("Key", TargetMM)).rdbmsNode
            :: OID <-- qid(string(N :: OID) + "_pk")
            :: name <-- ((N :: name) + "_pk")
            :: column <--
                Set {
                    (New("Column", TargetMM)).rdbmsNode
                    :: OID <-- qid(string(N :: OID) + "_tid")
                    :: name <-- ((N :: name) + "_tid")
                    :: type <-- "NUMBER"
                }
        )
    }
    -> including (
        p1(ModelGenRule(AttributeToColumn ; ? Set{ N } ? Model ? 'none ? (N :: name) ?
TargetMM ? TracMM))   *** where
    ) -> flatten
```

```
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{ (N :: OID) :: qid(string(N :: OID) + "_tid") :: qid(string(N ::
OID) + "_pk") }
                :: manipulationRule <-- "EClass-to-Table"
            }
            -> including (
                p2(ModelGenRule (AttributeToColumn ; ? Set{ N } ? Model ? 'none ? (N :: name) ?
TargetMM ? TracMM))   *** where
            )  -> flatten
        )
    if (N :: ecore-EClass)
    [metadata "EClass-to-Table"] .


    *** RULE 3 : AttributeToColumn
  op AttributeToColumn : -> TransformationRule [ctor] .
    ceq ModelGenRule (AttributeToColumn ; ? Set{ N, M } ? Model ? TableId ? prefix ? TargetMM ?
TracMM) =
        (
            p1(ModelGenRule (PrimitiveAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            -> including(
                p1(ModelGenRule (SuperAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            )
            -> including(
                p1(ModelGenRule (AttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            ) -> flatten
        ,
            p2(ModelGenRule (PrimitiveAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            -> including(
                p2(ModelGenRule (SuperAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            )
            -> including(
                p2(ModelGenRule (AttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            ) -> flatten
        )
    if (N :: ecore-EClass)
    [metadata "EAttribute-to-Column"] .

    ceq ModelGenRule (AttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ? TargetMM ?
TracMM) =
        (
            p1(ModelGenRule (PrimitiveAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            -> including(
                p1(ModelGenRule (SuperAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            ) -> flatten
        ,
            p2(ModelGenRule (PrimitiveAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            -> including(
                p2(ModelGenRule (SuperAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ?
TargetMM ? TracMM ))
            ) -> flatten
        )
    if (N :: ecore-EClass)
    [metadata "EAttribute-to-Column"] .


    *** RULE 4: PrimitiveAttributeToColumn

  op PrimitiveAttributeToColumn : -> TransformationRule [ctor] .
    *** Set{(EClass .. (AttId :: MAttId) ), M}
    eq ModelGenRule (
        PrimitiveAttributeToColumn ;
        ? Set{
        (ecore-EClass OID1  name1 instanceClassName1 instanceClass1 defaultValue1 abstract1
interface1 eAnnotations1
        ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1 eReferences1
        OrderedSet{ AttId :: MAttId } eAllContainments1 eAllOperations1 eAllStructuralFeatures1
eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1),
        M
        }
        ? Model ? ownerId ? prefix ? TargetMM ? TracMM) =
        if (IsPrimitiveDatatype(string(((Model -> any (hasId ; ? Set{ AttId } ; Model)) :: eType) -> first)))
then
        (
            Set{
                (New("Column", TargetMM)).rdbmsNode
                :: OID <-- ((Model -> any (hasId ; ? Set{ AttId } ; Model)) :: OID)
                :: name <--
                    (if (prefix == "") then
                        ((Model -> any (hasId ; ? Set{ AttId } ; Model)) :: name)
                    else
                        (prefix + "_" + ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name))
                    fi)
                :: type <-- (PrimitiveTypeToSqlType(string(((Model -> any (hasId ; ? Set{AttId} ;
Model)) :: eType) -> first)))
                :: owner <--
                    (if (ownerId == 'none) then
                        OrderedSet{ OID1 }
                    else
                        OrderedSet{ ownerId }
                    fi)
            }
            -> including (
                p1(ModelGenRule (
                    PrimitiveAttributeToColumn ;
                    ? Set{ (ecore-EClass OID1  name1 instanceClassName1 instanceClass1
defaultValue1 abstract1 interface1 eAnnotations1
                        ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1
eReferences1
                        OrderedSet{ MAttId } eAllContainments1 eAllOperations1
eAllStructuralFeatures1
                        eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1), M }
                    ? Model ? ownerId ? prefix ? TargetMM ? TracMM))
            ) -> flatten
```

```
          ,
              Set{
                  (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                  :: OID <-- qid(string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) + "-to-" +
string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID))
                  :: domain <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                  :: range <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                  :: manipulationRule <-- "EAttribute-to-Column"
              }
              -> including (
                  p2(ModelGenRule (
                      PrimitiveAttributeToColumn ;
                      ? Set{ (ecore-EClass OID1 name1 instanceClassName1 instanceClass1
defaultValue1 abstract1 interface1 eAnnotations1
                          ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1
eReferences1
                          OrderedSet{ MAttId } eAllContainments1 eAllOperations1
eAllStructuralFeatures1
                          eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1), M }
                      ? Model ? ownerId ? prefix ? TargetMM ? TracMM))
              ) -> flatten
          )
          else
              (ModelGenRule (
                  PrimitiveAttributeToColumn ;
                  ? Set{
                      (ecore-EClass OID1 name1 instanceClassName1 instanceClass1 defaultValue1
abstract1 interface1 eAnnotations1 ePackage1 eSuperTypes1
                      eOperations1 eAllAttributes1 eAllReferences1 eReferences1 OrderedSet{ AttId }
eAllContainments1 eAllOperations1 eAllStructuralFeatures1
                      eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1), M
                  }
                  ? Model ? ownerId ? prefix ? TargetMM ? TracMM
              ))
          fi
      [metadata "EAttribute-to-Column"] .

      *** Set{(EClass .. (AttId :: MAttId) ) }
      eq ModelGenRule (
          PrimitiveAttributeToColumn ;
          ? Set{
              (ecore-EClass OID1 name1 instanceClassName1 instanceClass1 defaultValue1 abstract1
interface1 eAnnotations1
              ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1 eReferences1
              OrderedSet{ AttId :: MAttId } eAllContainments1 eAllOperations1 eAllStructuralFeatures1
eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1)
          }
          ? Model ? ownerId ? prefix ? TargetMM ? TracMM) =
          if (IsPrimitiveDatatype(string(((Model -> any (hasId ; ? Set{AttId} ; Model)) :: eType) -> first)))
then
          (
              Set{
                  (New("Column", TargetMM)).rdbmsNode
                  :: OID <-- ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID)
                  :: name <--
                      (if (prefix == "") then
                          ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name)
                      else
                          (prefix + "_" + ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name))
```

```
                    fi)
                :: type <-- (PrimitiveTypeToSqlType(string(((Model -> any (hasId ; ? Set{AttId} ;
Model)) :: eType) -> first)))
                :: owner <--
                    (if (ownerId == 'none) then
                        OrderedSet{ OID1 }
                    else
                        OrderedSet{ ownerId }
                    fi)
            }
            -> including (
                p1(ModelGenRule (
                    PrimitiveAttributeToColumn ;
                    ? Set{ (ecore-EClass OID1  name1 instanceClassName1 instanceClass1
defaultValue1 abstract1 interface1  eAnnotations1
                        ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1
eReferences1
                        OrderedSet{ MAttId } eAllContainments1 eAllOperations1
eAllStructuralFeatures1
                        eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1) }
                    ? Model ? ownerId ? prefix ? TargetMM ? TracMM))
            )  -> flatten
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) + "-to-" +
string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID))
                :: domain <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: range <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: manipulationRule <-- "EAttribute-to-Column"
            }
            -> including (
                p2(ModelGenRule (
                    PrimitiveAttributeToColumn ;
                    ? Set{ (ecore-EClass OID1  name1 instanceClassName1 instanceClass1
defaultValue1 abstract1 interface1  eAnnotations1
                        ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1
eReferences1
                        OrderedSet{ MAttId } eAllContainments1 eAllOperations1
eAllStructuralFeatures1
                        eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1) }
                    ? Model ? ownerId ? prefix ? TargetMM ? TracMM))
            ) -> flatten
        )
        else
        ModelGenRule (
        PrimitiveAttributeToColumn ;
        ? Set{
        (ecore-EClass OID1  name1 instanceClassName1 instanceClass1 defaultValue1
        abstract1 interface1  eAnnotations1 ePackage1 eSuperTypes1
        eOperations1 eAllAttributes1 eAllReferences1 eReferences1
        OrderedSet{ AttId } eAllContainments1 eAllOperations1 eAllStructuralFeatures1
        eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1)
        }
        ? Model ? ownerId ? prefix ? TargetMM ? TracMM)
        fi
    [metadata "EAttribute-to-Column"] .

    *** Set{(EClass .. (AttId) ), M}
    eq ModelGenRule (
        PrimitiveAttributeToColumn ;
        ? Set{
        (ecore-EClass  OID1   name1  instanceClassName1  instanceClass1  defaultValue1  abstract1
interface1
        eAnnotations1  ePackage1  eSuperTypes1  eOperations1  eAllAttributes1  eAllReferences1
eReferences1
        OrderedSet{   AttId   }   eAllContainments1   eAllOperations1   eAllStructuralFeatures1
eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1),
        M
        }
        ? Model ? ownerId ? prefix ? TargetMM ? TracMM) =
    if (IsPrimitiveDatatype(string(((Model -> any (hasId ; ? Set{AttId} ; Model)) :: eType) -> first))) then
        (
            Set{
                (New("Column", TargetMM)).rdbmsNode
                :: OID <-- ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID)
                :: name <--
                    (if (prefix == "") then
                        ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name)
                    else
                        (prefix + "_" + ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name))
                    fi)
                ::  type  <--  (PrimitiveTypeToSqlType(string(((Model  ->  any  (hasId  ;  ?  Set{AttId}  ;
Model)) :: eType) -> first)))
                :: owner <--
                    (if (ownerId == 'none) then
                        OrderedSet{ OID1 }
                    else
                        OrderedSet{ ownerId }
                    fi)
            }
            -> including (
                p1(ModelGenRule (PrimitiveAttributeToColumn ;   ? Set{ M }   ?  Model  ?  ownerId  ?
prefix ? TargetMM ? TracMM))
            ) -> flatten
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) + "-to-" +
string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID))
                :: domain <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: range <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: manipulationRule <-- "EAttribute-to-Column"
            }
            -> including (
                p2(ModelGenRule (PrimitiveAttributeToColumn ;   ? Set{ M }   ?  Model  ?  ownerId  ?
prefix ? TargetMM ? TracMM))
            ) -> flatten
        )
        else
        (
            (empty-set).Set{vrdbms}
        ,
            (empty-set).Set{vTraceabilityMetamodel}
        )
        fi
    [metadata "EAttribute-to-Column"] .
    *** Set{(EClass .. (AttId) )}
    eq ModelGenRule (
        PrimitiveAttributeToColumn ;
        ? Set{
        (ecore-EClass OID1 name1 instanceClassName1 instanceClass1 defaultValue1 abstract1
interface1
        eAnnotations1 ePackage1 eSuperTypes1 eOperations1 eAllAttributes1 eAllReferences1
eReferences1
        OrderedSet{ AttId } eAllContainments1 eAllOperations1 eAllStructuralFeatures1
eAllSuperTypes1 eIDAttribute1 eStructuralFeatures1)
        }
        ? Model ? ownerId ? prefix ? TargetMM ? TracMM) =
    if (IsPrimitiveDatatype(string(((Model -> any (hasId ; ? Set{AttId} ; Model)) :: eType) -> first))) then
        (
            Set{
                (New("Column", TargetMM)).rdbmsNode
                :: OID <-- ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID)
                :: name <--
                    (if (prefix == "") then
                        ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name)
                    else
                        (prefix + "_" + ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: name))
                    fi)
                :: type <-- (PrimitiveTypeToSqlType(string(((Model -> any (hasId ; ? Set{AttId} ;
Model)) :: eType) -> first)))
                :: owner <--
                    (if (ownerId == 'none) then
                        OrderedSet{ OID1 }
                    else
                        OrderedSet{ ownerId }
                    fi)
            }
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) + "-to-" +
string((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID))
                :: domain <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: range <-- OrderedSet{ ((Model -> any (hasId ; ? Set{AttId} ; Model)) :: OID) }
                :: manipulationRule <-- "EAttribute-to-Column"
            }
        )
        else
        (
            (empty-set).Set{vrdbms}
        ,
            (empty-set).Set{vTraceabilityMetamodel}
        )
        fi
    [metadata "EAttribute-to-Column"] .


    *** RULE 4: SuperAttributeToColumn
  op SuperAttributeToColumn : -> TransformationRule [ctor] .

    ceq ModelGenRule (SuperAttributeToColumn ; ? Set{ N, M } ? Model ? TableId ? prefix ? TargetMM
? TracMM) =
        if ( TableId =/= 'none ) then
            (
                p1(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? TableId ? prefix ? TargetMM ? TracMM ))
                -> including(
                    p1(ModelGenRule (SuperAttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix
? TargetMM ? TracMM ))
                ) -> flatten
            ,
                p2(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? TableId ? prefix ? TargetMM ? TracMM ))
                -> including(
                    p2(ModelGenRule (SuperAttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix
? TargetMM ? TracMM ))
                ) -> flatten
            )
        else
            (
                p1(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? (N :: OID) ? prefix ? TargetMM ? TracMM ))
                -> including(
                    p1(ModelGenRule (SuperAttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix
? TargetMM ? TracMM ))
                ) -> flatten
            ,
                p2(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? (N :: OID) ? prefix ? TargetMM ? TracMM ))
                -> including(
                    p2(ModelGenRule (SuperAttributeToColumn ; ? Set{ M } ? Model ? TableId ? prefix
? TargetMM ? TracMM ))
                ) -> flatten
            )
        fi
    if (N :: ecore-EClass) and (N :: eSuperTypes -> notEmpty)
    [metadata "EAttribute-to-Column"] .

    ceq ModelGenRule (SuperAttributeToColumn ; ? Set{ N } ? Model ? TableId ? prefix ? TargetMM ?
TracMM) =
        if ( TableId =/= 'none ) then
            (
                p1(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? TableId ? prefix ? TargetMM ? TracMM ))
            ,
                p2(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? TableId ? prefix ? TargetMM ? TracMM ))
            )
        else
            (
                p1(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? (N :: OID) ? prefix ? TargetMM ? TracMM ))
            ,
                p2(ModelGenRule (AttributeToColumn ; ? ((N :: eSuperTypes(Model)) -> asSet) ? Model
? (N :: OID) ? prefix ? TargetMM ? TracMM ))
            )
        fi
    if (N :: ecore-EClass) and (N :: eSuperTypes -> notEmpty)
    [metadata "EAttribute-to-Column"] .


    *** RULE 5: AssocToFKey
  op AssocToFKey : -> TransformationRule [ctor] .
```

```
ceq ModelGenRule (AssocToFKey ; ? Set{ N, M } ? Model ? TargetMM ? TracMM) =
    (
        Set{
            (
                (New("ForeignKey", TargetMM)).rdbmsNode
                :: OID <--
                    qid(
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                    )
                :: name <--
                    (
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                    )
                :: owner <--
                    ((p1(ModelGenRule (ClassToTable ; ? ((N :: eContainingClass(Model)) ->
asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                    -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )
                :: refersTo <--
                    ((p1(ModelGenRule (ClassToTable ; ? ((N :: eType(Model) -> asSet)) ? Model ?
MM((empty-set).Set{vrdbms}) ? TracMM)) )
                    -> select(isKey ; empty-params ; (empty-set).Set{vrdbms}) )
                :: column <--
                    Set{
                        (
                            (New("Column", TargetMM)).rdbmsNode
                            :: OID <--
                                qid(
                                    (((N :: eContainingClass(Model)) -> first) :: name)
                                    + "_" +
                                    (N :: name)
                                    + "_" +
                                    (((N :: eType(Model)) -> first) :: name)
                                    + "_tid"
                                )
                            :: name <--
                                (
                                    (((N :: eContainingClass(Model)) -> first) :: name)
                                    + "_" +
                                    (N :: name)
                                    + "_" +
                                    (((N :: eType(Model)) -> first) :: name)
                                    + "_tid"
                                )
                            :: type <-- "NUMBER"
                            :: owner <--
                                ((p1(ModelGenRule (ClassToTable ; ? ((N ::
eContainingClass(Model)) -> asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                                -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )

                        )
                    }
                ),
                (
                (New("Column", TargetMM)).rdbmsNode
                :: OID <--
                    qid(
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                        + "_tid"
                    )
                :: name <--
                    (
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                        + "_tid"
                    )
                :: type <-- "NUMBER"
                :: owner <--
                    ((p1(ModelGenRule  (ClassToTable  ;  ?  ((N  ::  eContainingClass(Model))  ->
asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                    -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )
                )
            }
            -> including (
                p1(ModelGenRule (AssocToFKey ; ? Set{ M } ? Model ? TargetMM ? TracMM))
            )  -> flatten
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{
                    qid(
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                    )
                    ::
                    qid(
                        (((N :: eContainingClass(Model)) -> first) :: name)
                        + "_" +
                        (N :: name)
                        + "_" +
                        (((N :: eType(Model)) -> first) :: name)
                        + "_tid"
                    )
             }
                :: manipulationRule <-- "EReference-to-ForeignKey"
            }
            -> including (
                p2(ModelGenRule (AssocToFKey ; ? Set{ M } ? Model ? TargetMM ? TracMM))
            ) -> flatten
        )
    if (N :: ecore-EReference) and (N :: eOpposite == empty-orderedset)
    [metadata "EReference-to-ForeignKey"] .

    ceq ModelGenRule (AssocToFKey ; ? Set{ N } ? Model ? TargetMM ? TracMM) =
        (
            Set{
                (
                    (New("ForeignKey", TargetMM)).rdbmsNode
                    :: OID <--
                        qid(
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                        )
                    :: name <--
                        (
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                        )
                    :: owner <--
                        ((p1(ModelGenRule  (ClassToTable  ;  ?  ((N  ::  eContainingClass(Model))  ->
asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                        -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )
                    :: refersTo <--
                        ((p1(ModelGenRule (ClassToTable ; ? ((N :: eType(Model) -> asSet)) ? Model ?
MM((empty-set).Set{vrdbms}) ? TracMM)) )
                        -> select(isKey ; empty-params ; (empty-set).Set{vrdbms}) )
                    :: column <--
                        Set{
                            (
                                (New("Column", TargetMM)).rdbmsNode
                                :: OID <--
                                    qid(
                                        (((N :: eContainingClass(Model)) -> first) :: name)
                                        + "_" +
                                        (N :: name)
                                        + "_" +
                                        (((N :: eType(Model)) -> first) :: name)
                                        + "_tid"
                                    )
                                :: name <--
                                    (
                                        (((N :: eContainingClass(Model)) -> first) :: name)
                                        + "_" +
                                        (N :: name)
                                        + "_" +
                                        (((N :: eType(Model)) -> first) :: name)
                                        + "_tid"
                                    )
                                :: type <-- "NUMBER"
                                :: owner <--
```

```
                                    ((p1(ModelGenRule (ClassToTable ; ? ((N ::
eContainingClass(Model)) -> asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                                    -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )

                            )

                        }


                ),
                (
                    (New("Column", TargetMM)).rdbmsNode
                    :: OID <--
                        qid(
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                            + "_tid"
                        )
                    :: name <--
                        (
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                            + "_tid"
                        )
                    :: type <-- "NUMBER"
                    :: owner <--
                        ((p1(ModelGenRule (ClassToTable ; ? ((N :: eContainingClass(Model)) ->
asSet) ? Model ? MM((empty-set).Set{vrdbms}) ? TracMM)) )
                        -> select(isTable ; empty-params ; (empty-set).Set{vrdbms}) )
                )
            }
        ,
            Set{
                (New("TraceabilityLink", TracMM)).TraceabilityMetamodelNode
                :: OID <-- qid(string(N :: OID) + "-to-" + string(N :: OID))
                :: domain <-- OrderedSet{ (N :: OID) }
                :: range <-- OrderedSet{
                        qid(
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                        )
                        ::
                        qid(
                            (((N :: eContainingClass(Model)) -> first) :: name)
                            + "_" +
                            (N :: name)
                            + "_" +
                            (((N :: eType(Model)) -> first) :: name)
                            + "_tid"
                        )
                    }
                    :: manipulationRule <-- "EReference-to-ForeignKey"
               }
          )
     if (N :: ecore-EReference) and (N :: eOpposite == empty-orderedset)
     [metadata "EReference-to-ForeignKey"] .

endfm
```

A.3 Algebraic representation of the source UML model

Fig 6. shows the source UML Model in UML notation and the corresponding Ecore model using the EMF default modeller.

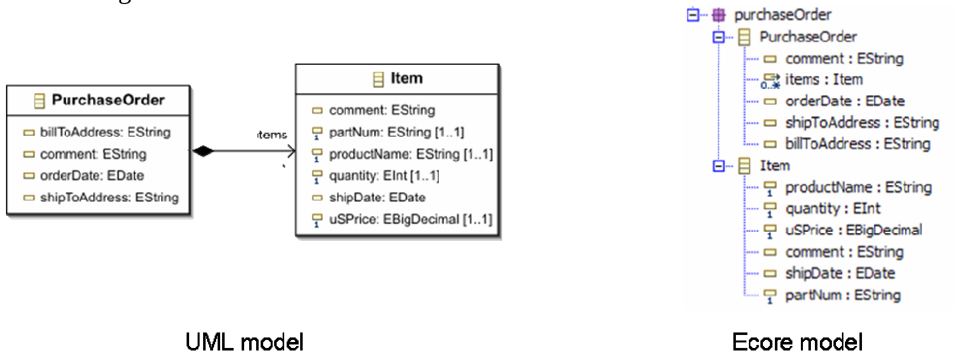


Fig. 6. The source UML model used in the example, which is shown by means of a UML editor and the default EMF editor.

The algebraic term that represents the above ecore model is defined in the following algebraic specification. This model has been automatically obtained by means of the MOMENT bridges that have been developed between the EMF and the Maude Technical Spaces.

```
fmod PROVA-MODELS is
pr UML2RDBMS .

op uml-model : -> Set{vM3} .
eq uml-model =
Set {
(ecore-EPackage 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#/ "purchaseOrder" "http://es.upv.dsic.issi/moment/purchaseOrder.ecore"
"purchaseOrder" empty-orderedset OrderedSet {'#// } OrderedSet { 'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item :: 'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder } empty-orderedset empty-
orderedset ),

(ecore-EClass 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//Item "Item" "" "0" "0" false false empty-orderedset OrderedSet
{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#/ } empty-
orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-
source-models/po_old.ecore#//Item/productName :: 'platform:/resource/MOMENT-
```

QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum } empty-orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum } empty-orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum } empty-orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName "productName" true false 1 1 false true true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity "quantity" true false 1 1 false true true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EInt } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EInt }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice "uSPrice" true false 1 1 false true true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment "comment" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate "shipDate" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EDate } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EDate }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum "partNum" true false 1 1 false true true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString }),

(ecore-EClass 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder "PurchaseOrder" "" "0" "0" false false empty-orderedset OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#/ } empty-orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress } OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items } OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items } OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress } OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items } empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress } empty-orderedset empty-orderedset OrderedSet { 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment "comment" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString }),

(ecore-EReference 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/items "items" true true 0 -1 true false true false false "" "0" false false true false false empty-orderedset OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-

models/po_old.ecore#//PurchaseOrder } empty-orderedset OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate "orderDate" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EDate } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EDate }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress "shipToAddress" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString }),

(ecore-EAttribute 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress "billToAddress" true false 0 1 false false true false false "" "0" false false false empty-orderedset OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString } OrderedSet {'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder } OrderedSet {'http://www.eclipse.org/emf/2002/Ecore#//EString })
} .
endfm

A.4 Model transformation execution in Maude

In this section, we show the application of the specified Uml2Rdbms ModelGen operator to the above mentioned UML term by means of the Maude term rewriting system.

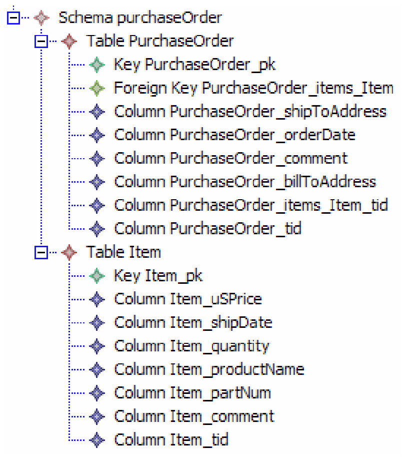
reduce in PROVA-MODELS : p1 ModelGen(Uml2Rdbms ; ? sourceM ? RdbmsMM ? "po.ecore" ? "po.rdbms") .
rewrites: 798620 in -290922340999ms cpu (3328ms real) (~ rewrites/second)

result Set{vrdbms}:
Set{(rdbms-Schema 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#/ "purchaseOrder" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Key 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_pk "Item_pk" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_tid}),(rdbms-Key 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_pk "PurchaseOrder_pk" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_tid}),(rdbms-ForeignKey 'PurchaseOrder_items_Item "PurchaseOrder_items_Item" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_pk} OrderedSet{'PurchaseOrder_items_Item_tid} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder}),(

```
rdbms-Table 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item "Item" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_pk} empty-orderedset
OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_tid} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#/}),(
rdbms-Table 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder "PurchaseOrder" "0" OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_pk} OrderedSet{'PurchaseOrder_items_Item} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress
:: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/billToAddress :: 'PurchaseOrder_items_Item_tid :: 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_tid} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#/}),(rdbms-Column 'PurchaseOrder_items_Item_tid "PurchaseOrder_items_Item_tid" "0"
"NUMBER" OrderedSet{'PurchaseOrder_items_Item} empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/comment "Item_comment" "0" "VARCHAR"
empty-orderedset empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/partNum "Item_partNum" "0" "VARCHAR" empty-orderedset empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/productName "Item_productName" "0" "VARCHAR"
empty-orderedset empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/quantity "Item_quantity" "0" "NUMBER" empty-orderedset empty-orderedset
OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/shipDate "Item_shipDate" "0" "DATE"
empty-orderedset empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item/uSPrice "Item_uSPrice" "0" "NUMBER" empty-orderedset empty-orderedset OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column 'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_tid "Item_tid" "0" "NUMBER" empty-orderedset
OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item_pk} OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//Item}),(rdbms-Column
```

```
'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder/billToAddress  "PurchaseOrder_billToAddress"  "0"  "VARCHAR"
empty-orderedset
        empty-orderedset  OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder}),(rdbms-Column                         'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/comment
"PurchaseOrder_comment" "0" "VARCHAR" empty-orderedset empty-orderedset
        OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder}),(rdbms-Column                         'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/orderDate
        "PurchaseOrder_orderDate" "0" "DATE" empty-orderedset empty-orderedset
        OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder}),(rdbms-Column                         'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder/shipToAddress
        "PurchaseOrder_shipToAddress" "0" "VARCHAR" empty-orderedset
        empty-orderedset  OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder}),(rdbms-Column                         'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_tid       "PurchaseOrder_tid"
"0"             "NUMBER"             empty-orderedset             OrderedSet{'platform:/resource/MOMENT-
QVT/UML2RDBMS/simplified-source-models/po_old.ecore#//PurchaseOrder_pk}
OrderedSet{'platform:/resource/MOMENT-QVT/UML2RDBMS/simplified-source-
models/po_old.ecore#//PurchaseOrder})}
```

The resulting term is parsed by MOMENT and the corresponding Relational model is generated. The generated model using the default EMF graphical modeller is shown in Fig. 7.



```
Schema purchaseOrder
  Table PurchaseOrder
    Key PurchaseOrder_pk
    Foreign Key PurchaseOrder_items_Item
    Column PurchaseOrder_shipToAddress
    Column PurchaseOrder_orderDate
    Column PurchaseOrder_comment
    Column PurchaseOrder_billToAddress
    Column PurchaseOrder_items_Item_tid
    Column PurchaseOrder_tid
  Table Item
    Key Item_pk
    Column Item_uSPrice
    Column Item_shipDate
    Column Item_quantity
    Column Item_productName
    Column Item_partNum
    Column Item_comment
    Column Item_tid
```

Fig. 7. The generated relational model.