Ecore to Genmodel case study solution using the Viatra2 framework

Ábel Hegedüs, Zoltán Ujhelyi, Gábor Bergmann, and Ákos Horváth

Budapest University of Technology and Economics, Hungary
{hegedusa,ujhelyiz,bergmann,ahorvath}@mit.bme.hu

Abstract. The paper presents a solution of the Ecore to GenModel case study of the Transformation Tool Contest 2010, using the model transformation tool Viatra2.

This work was partially supported by the EC FP6 DIANA (AERO1-030985) and ICT FP7 SecureChange (ICT-FET-231101) European Projects.

1 Introduction

Automated model transformations play an important role in modern model-driven system engineering in order to query, derive and manipulate large, industrial models. Since such transformations are frequently integrated to design environments, they need to provide short reaction time to support software engineers.

The objective of the Viatra2 (VIsual Automated model TRAnsformations [1]) framework is to support the entire life-cycle, i.e. the specification, design, execution, validation and maintenance of model transformations.

Model representation. Viatra2 uses the VPM metamodeling approach [2] for describing modeling languages and models. The main reason for selecting VPM instead of a MOF-based metamodeling approach is that VPM supports arbitrary metalevels in the model space. As a direct consequence, models taken from conceptually different domains (and/or technological spaces) can be easily integrated into the VPM model space. The flexibility of VPM is demonstrated by a large number of already existing model importers accepting the models of different BPM formalisms, UML models of various tools, XSD descriptions, and EMF models.

Graph transformation (GT) [3] based tools have been frequently used for specifying and executing complex model transformations. In GT tools, *graph patterns* capture structural conditions and type constraints in a compact visual way. At execution time, these conditions need to be evaluated by *graph pattern matching*, which aims to retrieve one or all matches of a given pattern to execute a transformation rule.

Transformation description. Specification of model transformations in Viatra2 combines the visual, declarative rule and pattern based paradigm of graph transformation (GT) [3] and the very general, high-level formal paradigm of abstract state machines (ASM) [4] into a single framework for capturing transformations within and between modeling languages.

Transformation Execution. Transformations are executed within the framework by using the Viatra2 interpreter. For pattern matching both (i) *local search based pattern matching* (LS) and (ii) *incremental pattern matching* (INC) are available. This feature provides the transformation designer additional opportunities to fine tune the transformation either for faster execution (INC) or lower memory consumption (LS) [5].

The rest of the paper is structured as follows. Sec. 2 introduces the Case Study problem which is solved in this paper. Sec. 3 gives an architectural overview of the transformation, while Sec. 4 highlights the interesting parts of our implementation and finally Sec. 5 concludes the paper.

2 Case study

In the Eclipse Modeling Framework (EMF) [6] the Ecore metamodeling language is used for defining arbitrary metamodels. Conforming instance models can be handled by reflection or code generation, the latter is performed by a toolkit provided by EMF. The toolkit first transforms the Ecore metamodel into a GenModel model that stores implementation-specific information and also refers back to the original Ecore metamodel. Then the functional Java classes are generated using a JET-based model-to-text transformation that consumes the GenModel.

As a challenge, [7] proposes the reimplementation of the Ecore to GenModel transformation within a model transformation framework. To overcome the reconciliation problem in the existing transformation (i.e. major changes in the Ecore model can lead to the loss of customised attributes in the existing GenModel) the case study specifies GenModel options as annotations in the source Ecore metamodel. These annotations are populated in the generated GenModel by the transformation.

Furthermore, the case study proposes the application of reflection for handling annotations using a generic function instead of explicit one-by-one mapping for each annotation/attribute pair.

The proposed transformation demonstrates two useful features of transformation languages: (1) ability to establish cross-model references and (2) support for reflection.

3 Solution Architecture

We implemented our solution for the case study using the VIATRA2 model transformation framework. Fig. 1 shows the complete architecture with both preexisting and newly created components. The *Transformation Controller* is an extension to the Eclipse framework that provides an easy-to-use graphical interface for executing the underlying transformation (i.e. it appears as a command in the pop-up menu of Ecore EMF files). From the user perspective, the controller is invoked on an *input Ecore* file and the result is an *output GenModel* file.

Note that the transformation is performed on models *inside the VPM modelspace* of VIATRA2 rather than on in-memory EMF models. Although VIATRA2 does not manipulate EMF models directly, it includes a generic support for handling EMF metamodels and instance models.

In order to understand the transformation we briefly outline the metamodeling approach of our solution. The *Ecore metametamodel* is the base of this support, which was defined in accordance with the actual EMF metamodel of Ecore.

Both the *Ecore and GenModel metamodels* are defined as instances of this metametamodel, and are imported into VIATRA2 with the generic Ecore metamodel importer. Then the input file is used to *import the Ecore file into* VIATRA2 and create the *Ecore model* which is the instance of the Ecore metamodel.

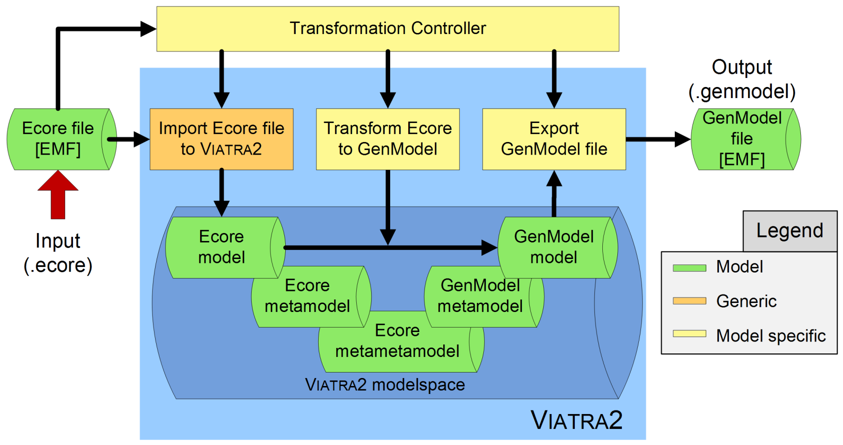


Fig. 1. Solution Architecture

By executing our implemented transformation, we can *transform the Ecore model to a GenModel model* which is an instance of the GenModel metamodel. This *GenModel model* is then *exported* to create the output GenModel file.

Note that currently we have limited generic support for exporting cross-resource references, thus the exporter plugin provided for this case study is GenModel-specific, but we plan to resolve this over time by improving the generic Ecore instance exporter.

4 Transforming Ecore models to GenModels (E2GM)

The *E2GM* transformation generates the GenModel model from the Ecore model in the VIATRA2 framework and is implemented in the VIATRA2 Textual Command Language (VTCL) [8]. E2GM can be separated into two parts, (1) the construction of the GenModel model based on the Ecore model, (2) parsing annotations and creating the corresponding attributes reflectively.

The complete transformation is only 700 lines of VTCL code including whitespaces and comments (see Appendix B). It includes 26 simple type-checking graph patterns (i.e. entity X is type Y, see lines 74-104) and 20 complex patterns (e.g. Ecore annotation that does not have a corresponding GenModel attribute). The type-checking patterns and 10 complex patterns are handled by INC, the other complex patterns by LS. Finally, the actual manipulation is executed by 8 declarative rules (e.g. create GenModel entity for given Ecore entity).

Ecore model traversal is done by navigating through the tree structure of the model and creating GenModel elements with the correct type using a set of mapping key-value pairs (see lines 11-23). The key of the map is the Ecore type (e.g. EClass) and the value is the corresponding GenModel type (GenClass). Although the Ecore metamodel includes a complex type hierarchy (using generalisation), the mapping takes advantage of the fact that for any type in the Ecore model, at most one of its supertypes (or itself) is mapped to a GenModel type (i.e. there is no ambiguity). Therefore, we can define a generic graph pattern that returns the mappable type of any given Ecore element, and this type is used as a key to retrieve the appropriate GenModel type that is instantiated (see lines 174-179).

GenModel and cross-model references. The type of the references between generated GenModel elements are handled similarly to the generic pattern for mapping. Instead of an explicit declaration for every reference type, the Ecore and GenModel metamodels are used (as instances of the Ecore metametamodel) to find the appropriate reference type definition in the GenModel metamodel. Fig. 2 illustrates a graph pattern to find reference type *GenRefType* for containing *GenChild* in *GenParent* (see also lines 210-247).

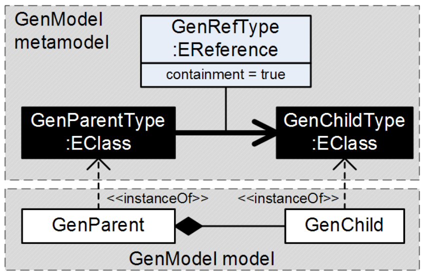


Fig. 2. Pattern for GenModel references

Furthermore, the created GenModel elements have references to the elements in the Ecore model to store the connection between the two models. For example, an *EClass* is transformed into a *GenClass* element in the GenModel with an *ecoreClass* reference between them (see lines 390-401). The transformation creates these references using a generic approach as well (i.e. by retrieving the reference type from the GenModel metamodel).

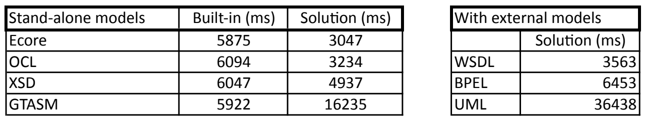
External Ecore models. EMF provides a capability to create complex interconnected models from more than one Ecore model. Ecore models that are referenced from a source Ecore model are called *external* as their definition is not inside the actual model. Such models are transformed to GenModel models in two different ways: (1) if they already have their own GenModels, then these models are references with the *usedGenPackages* from the generated GenModel; (2) otherwise the generated GenModel will include the packages corresponding to the referenced Ecore models as well. In our E2GM transformation we only support the second case, where new *GenPackages* are created for external Ecore models (see lines 204-208).

Parsing annotations. In our solution we used a reflective approach for parsing annotations in the Ecore model, by retrieving the attributes of GenModel entities from the metamodels using the key of Ecore annotations. Apart from implementing a similar technique proposed in [7] we extended that solution using generic graph patterns to decide the appropriate type for the attributes defined by the annotations. These patterns find the attribute type using the GenModel metamodel, and declarative rules create the attributes themselves (see lines 576-652).

We also implemented type checking for the attributes to ensure that boolean and enumeration values are correct to avoid the generation of a syntactically incorrect GenModel (e.g. boolean with a value other than "true" or "false", see lines 698-729). Furthermore, to provide a GenModel that is usable for code generation without further editing, several default attributes are set even if no annotation is defined for them in the Ecore model (e.g see line 132).

Performance. We used several Ecore models with varying size and complexity to test the performance of our implementation. We tested stand-alone metamodels such as Ecore, XSD, OCL and GTASM (the metamodel for the transformation language of VIATRA2), and metamodels with external Ecore models (e.g. BPEL, UML, WSDL).

As a comparison, we measured[1] the performance of the built-in EMF generator on the same metamodels. This generator is a headless Eclipse application which can be invoked from a command prompt. We found that our implementation is faster on smaller models even though we only measured the time the built-in EMF generator required to perform the actual transformation, not the whole time from start up. However, on larger models, our solution was slower by one order of magnitude (see Fig. 3). The main factors for these results are the generic transformation and modeling approach of VIATRA2 and our solution, as opposed to the explicit template-based approach of the built-in generator.



Stand-alone models	Built-in (ms)	Solution (ms)
Ecore	5875	3047
OCL	6094	3234
XSD	6047	4937
GTASM	5922	16235

With external models	
	Solution (ms)
WSDL	3563
BPEL	6453
UML	36438

Fig. 3. Performance results

It is important to note, that this headless generator can not handle external Ecore models, therefore we could only test with stand-alone metamodels. Furthermore, we also measured the performance of the transformation in our solution, without the import-export. We found that the execution time is directly proportional to model size, therefore it scales well.

5 Conclusion

In the current paper we have presented our VIATRA2 based implementation for the Ecore to Genmodel case study [7].

Relying on the high-level metamodeling features of VIATRA2, we have presented relatively simple solutions to all optional parts and more. Our implementation is able to handle cross-model references both between several Ecore models and between the generated GenModel and the original Ecore models. We used reflection when dealing with annotations in Ecore models.

The high points of our transformation are the generic rules for mapping Ecore elements to GenModel, which are easily customisable for changes in GenModel metamodel. We exploited the incremental matching feature of VIATRA2 and employed type checking of boolean and enumeration values in annotations. The implementation is able to handle nested packages and external models as well.

On the other hand, import-export of models is required and referenced GenModels (for external Ecore models) are not handled at the moment.

References

1. VIATRA2 Framework: An Eclipse GMT Subproject: (`http://www.eclipse.org/gmt/`)
2. Varró, D., Pataricza, A.: VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. Journal of Software and Systems Modeling **2**(3) (2003) 187–210

[1] All measurements were carried out on a computer with Intel Centrino Duo 1.66 GHz processor, 3 GB DDR2 memory, Windows XP, Eclipse 3.5.2 and EMF 2.5.

3. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation. Volume 2: Applications, Languages and Tools. World Scientific (1999)
4. Börger, E., Stärk, R.: Abstract State Machines. A method for High-Level System Design and Analysis. Springer-Verlag (2003)
5. Bergmann, G., Horváth, A., Ráth, I., Varró, D.: Experimental assessment of combining pattern matching strategies with VIATRA2. Journal of Software Tools in Technology Transfer (2009) Accepted.
6. The Eclipse Modeling Framework project: (http://www.eclipse.org/emf/)
7. Kolovos, D.S., Rose, L.M., Paige, R.F., de Lara, J.: Ecore to GenModel Case Study for TTC2010 (2010)
8. Balogh, A., Varró, D.: Advanced model transformation language constructs in the VIATRA2 framework. In: ACM Symposium on Applied Computing — Model Transformation Track (SAC 2006), Dijon, France, ACM Press (2006) 1280–1287

A Solution demo and implementation

The deployable implementation and source code is available as an Eclipse online update site (`http://mit.bme.hu/~ujhelyiz/viatra/ttc10-site/`) and as an archive (`http://mit.bme.hu/~ujhelyiz/viatra/ttc10.zip`)

The SHARE image for demonstration purposes is available at `http://is.tm.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=XP-TUe_TTC10_TTC10%3A%3AXP-Ec2Gm_Viatra.vdi`

B Appendix - Ecore to GenModel transformation

```
// metamodel imports
import nemf.packages.ecore;
import nemf.packages.genmodel;
import datatypes;

// Ecore-to-GenModel transformation
@incremental
machine ecore2genmodel {

 // mapping rules for corresponding Ecore and GenModel types
 asmfunction mapping/1 {
  (nemf.packages.ecore.EPackage) = nemf.packages.genmodel.GenPackage;
  (nemf.packages.ecore.ETypeParameter) =
   nemf.packages.genmodel.GenTypeParameter;
  (nemf.packages.ecore.EEnumLiteral) = nemf.packages.genmodel.GenEnumLiteral;
  (nemf.packages.ecore.EDataType) = nemf.packages.genmodel.GenDataType;
  (nemf.packages.ecore.EClass) = nemf.packages.genmodel.GenClass;
  (nemf.packages.ecore.EStructuralFeature) =
   nemf.packages.genmodel.GenFeature;
  (nemf.packages.ecore.EOperation) = nemf.packages.genmodel.GenOperation;
  (nemf.packages.ecore.EParameter) = nemf.packages.genmodel.GenParameter;
  (nemf.packages.ecore.EEnum) = nemf.packages.genmodel.GenEnum;
 }

 // temporal values and entity references
 asmfunction temp/1;
 // reference to ecore input model
 asmfunction ecore/0;
 // reference to output genmodel
 asmfunction genmodel/0;
```

```
// entry point of transformation
// EcoreModel: fully qualified name of the input model
// GenModel: local name of GenModel
// PluginID: plugin identifier of GenModel
rule main(in EcoreModel, in GenModelName, in PluginID) = seq{
 println("[INFO] >>>Ecore2Genmodel Transformation started on " + EcoreModel);
 // find Ecore model in modelspace
 if(ref(EcoreModel) != undef && find EPackage(ref(EcoreModel))) seq{
  update ecore() = ref(EcoreModel);
 } else seq{
  println("[ERROR] EcoreModel not found!");
  fail;
 }
 update temp("pluginID") = PluginID;


 let GenModelRef = "nemf.resources."+GenModelName+"_genmodel" in
 // if GenModel already exists
  if(ref(GenModelRef) != undef &&
   find GenModel(ref(GenModelRef))) seq{
  println("[Warning] Existing GenModel");
  // delete previous GenModel
  delete(ref(GenModelRef));
 }
 let NewGenModel = undef, R = undef in seq{
  // create new model
  new(GenModel(NewGenModel) in nemf.resources);
  rename(NewGenModel, GenModelName+"_genmodel");
  update genmodel() = NewGenModel;
  // initialize GenModel using annotations
  new(relation(R,genmodel(),ecore()));
  call initialiseGenmodel(genmodel(),ecore());
  delete(R);
 }

 let GenType = nemf.packages.genmodel.GenModel in seq{
  // create GenModel equivalent of main EPackage
  call parseEcoreEntity(ecore(), genmodel(), GenType);
 }
 println("[INFO] >>> Ecore2Genmodel Transformation finished.");
}

//-------------------Type checking patterns--------------
pattern EcoreBoolean(EBoolean) = {nemf.ecore.datatypes.EBoolean(EBoolean);}
pattern EcoreString(EString) = {nemf.ecore.datatypes.EString(EString);}
pattern EcoreEnum(EEnum) = {nemf.ecore.datatypes.EEnum(EEnum);}
pattern EcoreEnumLiteral(EEnumLiteral) =
 {nemf.ecore.datatypes.EEnumLiteral(EEnumLiteral);}

pattern EPackage(EPackage) = {EPackage(EPackage);}
pattern ETypeParameter(ETypeParameter) = {ETypeParameter(ETypeParameter);}
pattern EEnumLiteral(EEnumLiteral) = {EEnumLiteral(EEnumLiteral);}
pattern EDataType(EDataType) = {EDataType(EDataType);}
pattern EClass(EClass) = {EClass(EClass);}
pattern EStructuralFeature(EStructuralFeature) =
 {EStructuralFeature(EStructuralFeature);}
pattern EOperation(EOperation) = {EOperation(EOperation);}
pattern EParameter(EParameter) = {EParameter(EParameter);}
pattern EEnum(EEnum) = {EEnum(EEnum);}
pattern EObject(EObject) = {EObject(EObject);}
pattern EReference(EReference) = {EReference(EReference);}

pattern GenPackage(GenPackage) = {GenPackage(GenPackage);}
pattern GenModel(GenModel) = {GenModel(GenModel);}
pattern GenClass(GenClass) = {GenClass(GenClass);}
pattern GenTypeParameter(GenTypeParameter) =
 {GenTypeParameter(GenTypeParameter);}
```

```
pattern GenEnumLiteral(GenEnumLiteral) = {GenEnumLiteral(GenEnumLiteral);}
pattern GenDataType(GenDataType) = {GenDataType(GenDataType);}
pattern GenFeature(GenFeature) = {GenFeature(GenFeature);}
pattern GenOperation(GenOperation) = {GenOperation(GenOperation);}
pattern GenParameter(GenParameter) = {GenParameter(GenParameter);}
pattern GenEnum(GenEnum) = {GenEnum(GenEnum);}


//--------------------------------------------------------

// initialize GenModel
rule initialiseGenmodel(in GenModel, in EcoreModel) = let R = undef in seq{

 // create default attributes required for a proper GenModel
 call parseAnnotation(GenModel,"copyrightFields",EcoreModel,"false","value");
 call parseAnnotation(GenModel,"complianceLevel",EcoreModel,
  nemf.packages.genmodel.GenJDKLevel.JDK60, "entity");
 call parseAnnotation(GenModel,"importerID",EcoreModel,
  "org.eclipse.emf.importer.ecore","value");

 // find modelname
 let Name =  nemf.packages.ecore.ENamedElement.name in
  try choose ModelName with
   find AttributeForType(EcoreModel,Name,ModelName) do
  let NewName = value(ModelName) in seq{
  update NewName = str.toUpperCase(str.substring(NewName,0,1))
   +str.substring(NewName,1);
  call parseAnnotation(GenModel,"modelName",EcoreModel,NewName,"value");
  // set foreign model reference
  let ForeignModel = undef in seq{
   new(nemf.ecore.datatypes.EString(ForeignModel) in GenModel);
   rename(ForeignModel, "foreignModel_"+value(ModelName));
   setValue(ForeignModel,value(ModelName));
   new(GenModel.foreignModel(R,GenModel,ForeignModel));
  }
 }

 call parseAnnotation(GenModel,"modelDirectory",EcoreModel,
  "/"+temp("pluginID")+"/src", "value");
 call parseAnnotation(GenModel,"modelPluginID",EcoreModel,
  temp("pluginID"), "value");
 // parse remaining annotations
 call parseAnnotationList(EcoreModel,GenModel);
}

// pattern for restricting datatypes
@localsearch
pattern EcoreDataType(EDataType) =
 {nemf.ecore.datatypes.EBoolean(EDataType);} or
  {nemf.ecore.datatypes.EString(EDataType);} or
 {nemf.ecore.datatypes.EEnum(EDataType);} or
 {nemf.ecore.datatypes.EEnumLiteral(EDataType);} or
 {nemf.ecore.datatypes.EInt(EDataType);}

// pattern for finding the type of an attribute
@localsearch
pattern AttributeForType(EcoreEntity,FeatureRel,Attribute) = {
 EModelElement(EcoreEntity);
 find MappedEcoreTypedEntity(EcoreEntity);
 nemf.ecore.EClass(EcoreType);
 instanceOf(EcoreEntity,EcoreType);
 nemf.ecore.EDataType(AttributeType);
 nemf.ecore.EClass.EAttribute(FeatureRel,EcoreType,AttributeType);
 entity(Attribute);
 find EcoreDataType(Attribute);
 instanceOf(Attribute,AttributeType);
 relation(Rel,EcoreEntity,Attribute);
 instanceOf(Rel,FeatureRel);
```

```
}

// create Ecore equivalent of EcoreEntity in GenModel
// under GenmodelParent which has type GenParentType
rule parseEcoreEntity(in EcoreEntity, in GenmodelParent, in GenParentType) =
 let GenmodelType = undef, NewGenmodelEntity = undef in seq{

 // find Ecore type from metamodel
 try choose EcoreType below nemf.packages.ecore with
  find EcoreType(EcoreEntity,EcoreType) do seq{
   // find GenModel type using mapping
  update GenmodelType = mapping(EcoreType);
 }
 else seq{
  println("[ERROR] Can't find Genmodel type!");
  fail;
 }

 if(GenmodelType != undef) let GenRefType = undef,
   PreviousRef = undef in seq{
  // create new entity
  new(entity(NewGenmodelEntity) in GenmodelParent);
  rename(NewGenmodelEntity,name(EcoreEntity));
  // set type of entity
  new(instanceOf(NewGenmodelEntity,GenmodelType));
  // create reference between GenModel and Ecore entity
  call createGenmodel2EcoreReference(NewGenmodelEntity,
   EcoreEntity,GenmodelType);
  // initialize Genmodel entity
  call initialiseGenmodelEntity(NewGenmodelEntity,EcoreEntity);

  // create GenModel entities for children
  forall EcoreChildEntity in EcoreEntity with
    find MappedEcoreTypedEntity(EcoreChildEntity) do seq{
   call parseEcoreEntity(EcoreChildEntity, NewGenmodelEntity, GenmodelType);
  }

  // handle external models (parsing is started from root EPackage)
  if(find EReference(EcoreEntity))
   try choose ExternalModel below nemf.resources with
     find ExternalEcoreEntityReference(EcoreEntity,ecore(),ExternalModel) do
   call parseEcoreEntity(ExternalModel, genmodel(), GenmodelType);

  // ordered relations are handled
  // for reference between same types
  if(GenmodelType == GenParentType)
   // find reference type
   try choose GenRefTypeT with
    find GenmodelEntitySelfReferenceType(GenmodelParent,NewGenmodelEntity,
      GenRefTypeT) do seq{
     update GenRefType = GenRefTypeT;
     // find last relation in the ordered list
     try choose PreviousRefT in GenmodelParent with
      find LastOrderedSelfRelationRef(GenParentType, GenmodelParent,
        GenRefType,PreviousRefT) do
       update PreviousRef = PreviousRefT;
   }
  // for reference between different types
  else // find reference type
    try choose GenRefTypeT with
      find GenmodelEntityReferenceType(GenmodelParent,
       NewGenmodelEntity,GenRefTypeT) do seq{
     update GenRefType = GenRefTypeT;
     // find last relation in the ordered list
     try choose PreviousRefT in GenmodelParent with
      find LastOrderedRelationRef(GenParentType,GenmodelType,
        GenmodelParent,GenRefType,PreviousRefT) do
       update PreviousRef = PreviousRefT;
    }
   if (GenRefType != undef) let R = undef in seq{
    // create reference between parent and child GenModel entities
    if (PreviousRef != undef) let RR = undef in seq{
     new(relation(R,GenmodelParent,NewGenmodelEntity));
     new(instanceOf(R,GenRefType));
     // create ordered relation
     new(nemf.ecore.EObject.orderedRelation.next(RR,PreviousRef,R));
    } else seq{
     new(relation(R,GenmodelParent,NewGenmodelEntity));
     new(instanceOf(R,GenRefType));
     }
   }
   // parse annotations for entity
   call parseAnnotationList(EcoreEntity,NewGenmodelEntity);
  }
 }

 // pattern for finding type for entity
 @localsearch
 pattern EcoreType(EcoreEntity, EcoreType) = {
  EModelElement(EcoreEntity);
  nemf.ecore.EClass(EcoreType) below nemf.packages.ecore;
  instanceOf(EcoreEntity,EcoreType);
  check(mapping(EcoreType) != undef);
 }

 // pattern for restricting entities to mapped types
 @localsearch
 pattern MappedEcoreTypedEntity(EcoreEntity) =
  {EPackage(EcoreEntity);} or
  {ETypeParameter(EcoreEntity);} or
  {EEnumLiteral(EcoreEntity);} or
  {EDataType(EcoreEntity);} or
  {EClass(EcoreEntity);} or
  {EStructuralFeature(EcoreEntity);} or
  {EOperation(EcoreEntity);} or
  {EParameter(EcoreEntity);} or
  {EEnum(EcoreEntity);
 }

 //pattern for checking already mapped Ecore entities
 pattern MappedEcoreEntityInGenmodel(EcoreEntity) = {
  EModelElement(EcoreEntity);
  GenBase(GenmodelEntity);
  relation(R,GenmodelEntity,EcoreEntity);
 }

 // pattern for checking external model reference
 @localsearch
 pattern ExternalEcoreEntityReference(EReference, EcoreModel, ExternalModel)={
  EPackage(EcoreModel) below nemf.resources;
  EReference(EReference) below EcoreModel;
  EPackage(ExternalModel) below nemf.resources;
  EClass(ERefType) below ExternalModel;
  EReference.eReferenceType(R,EReference,ERefType);
  // check if the entity is not inside the model
  neg find MappedEcoreEntityInGenmodel(ExternalModel);
 }

 // pattern for finding reference type of GenModel entity and parent
 // with the same entity type
 pattern GenmodelEntitySelfReferenceType(GenParent,GenEntity,GenRefType) = {
  GenBase(GenParent);
  nemf.ecore.EClass(GenParentType);
  instanceOf(GenParent,GenParentType);
  GenBase(GenEntity) in GenParent;
  instanceOf(GenEntity,GenParentType);
 nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenParentType);
 Boolean(True);
 check(True == datatypes.Boolean.true);
 nemf.ecore.EClass.EReference.containment(Containment,GenRefType,True);
}

// pattern for finding reference type of GenModel entity and parent
// with different entity types
pattern GenmodelEntityReferenceType(GenParent,GenEntity,GenRefType) = {
 GenBase(GenParent);
 GenBase(GenEntity) in GenParent;
 nemf.ecore.EClass(GenParentType);
 instanceOf(GenParent,GenParentType);
 nemf.ecore.EClass(GenmodelType);
 instanceOf(GenEntity,GenmodelType);
 nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenmodelType);
 Boolean(True);
 check(True == datatypes.Boolean.true);
 nemf.ecore.EClass.EReference.containment(Containment,GenRefType,True);
}

// pattern for finding last relation among ordered relations of GenRefType
// (between same entity types)
pattern LastOrderedSelfRelationRef(GenParentType, GenParent,
  GenRefType,PreviousRef) = {
 GenBase(GenParent);
 nemf.ecore.EClass(GenParentType);
 nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenParentType);
 instanceOf(GenParent,GenParentType);
 instanceOf(GenEntity,GenParentType);
 relation(PreviousRef,GenParent,GenEntity);
 instanceOf(PreviousRef,GenRefType);
 GenBase(GenEntity);
 find GenmodelEntitySelfReferenceType(GenParent,GenEntity,GenRefType);
 // match only if there is no next relation from the found reference
 neg pattern HasNextRelation(GenParent,GenEntity,PreviousRef,GenRefType) = {
  GenBase(GenParent);
  nemf.ecore.EClass(GenParentType);
  nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenParentType);
  instanceOf(GenParent,GenParentType);
  instanceOf(GenEntity,GenParentType);
  relation(PreviousRef,GenParent,GenEntity);
  instanceOf(PreviousRef,GenRefType);
  GenBase(GenEntity);
  find GenmodelEntitySelfReferenceType(GenParent,GenEntity,GenRefType);
  // --------------------------------------
  GenBase(OtherEntity);
  relation(R2,GenParent,OtherEntity);
  instanceOf(R2,GenRefType);
  nemf.ecore.EObject.orderedRelation.next(Rx,PreviousRef,R2);
 }
}

// pattern for finding last relation among ordered relations of GenRefType
// (between same entity types)
pattern LastOrderedRelationRef(GenParentType,GenmodelType, GenParent,
  GenRefType,PreviousRef) = {
 GenBase(GenParent);
 nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenmodelType);
 instanceOf(GenParent,GenParentType);
 nemf.ecore.EClass(GenmodelType);
 nemf.ecore.EClass(GenParentType);
 instanceOf(GenEntity,GenmodelType);
 relation(PreviousRef,GenParent,GenEntity);
 instanceOf(PreviousRef,GenRefType);
 GenBase(GenEntity);// in GenParent;
 find GenmodelEntityReferenceType(GenParent,GenEntity,GenRefType);
 // match only if there is no next relation from the found reference
```

```
  neg pattern HasNextRelation(GenParent,GenEntity,PreviousRef,GenRefType) = {
   GenBase(GenParent);
   nemf.ecore.EClass.EReference(GenRefType,GenParentType,GenmodelType);
   instanceOf(GenParent,GenParentType);
   nemf.ecore.EClass(GenmodelType);
   nemf.ecore.EClass(GenParentType);
   instanceOf(GenEntity,GenmodelType);
   relation(PreviousRef,GenParent,GenEntity);
   instanceOf(PreviousRef,GenRefType);
   GenBase(GenEntity);
   find GenmodelEntityReferenceType(GenParent,GenEntity,GenRefType);
   // --------------------------------------
   GenBase(OtherEntity);
   relation(R2,GenParent,OtherEntity);
   instanceOf(R2,GenRefType);
   nemf.ecore.EObject.orderedRelation.next(Rx,PreviousRef,R2);
  }
 }

 // create cross-model reference from GenModel to Ecore
 rule createGenmodel2EcoreReference(in GenmodelEntity,
   in EcoreEntity, in GenmodelType) = seq{
  // find reference type
  choose Gen2EcRefType with
   find Genmodel2EcoreReferenceType(GenmodelType,Gen2EcRefType) do
   let R = undef in seq{
    // create reference
    new(relation(R,GenmodelEntity,EcoreEntity));
    new(instanceOf(R,Gen2EcRefType));
   }
 }

 // pattern for finding reference type between GenModel and Ecore entities
 @localsearch
 pattern Genmodel2EcoreReferenceType(GenmodelType,Gen2EcRefType) = {
  nemf.ecore.EClass(GenmodelType) below nemf.packages.genmodel;
  nemf.ecore.EClass(EcoreType) below nemf.packages.ecore;
  nemf.ecore.EClass.EReference(Gen2EcRefType,GenmodelType,EcoreType);
 }

 // initialize GenModel entity (not GenModel typed) using annotations
 rule initialiseGenmodelEntity(in GenmodelEntity, in EcoreEntity) = seq{

  // GenPackage-specific attributes needed for proper GenModel
  if(find GenPackage(GenmodelEntity)) seq{
   call parseAnnotation(GenmodelEntity,"disposableProviderFactory",
    EcoreEntity,"true", "value");
   let Name =  nemf.packages.ecore.ENamedElement.name in
   try choose ModelName with
     find AttributeForType(EcoreEntity,Name,ModelName) do seq{
    call parseAnnotation(GenmodelEntity,"prefix",EcoreEntity,
     value(ModelName), "value");
   }
  }
  // GenClass-specific attributes needed for proper GenModel
  else if(find GenClass(GenmodelEntity)) seq{
   let AbstractRel = nemf.packages.ecore.EClass.abstract in
   try choose Abstract with find AttributeForType(EcoreEntity,
     AbstractRel,Abstract) do seq{
    call parseAnnotation(GenmodelEntity,"image",EcoreEntity,
     toString(!(toBoolean(value(Abstract)))), "value");
   }
  }
  // GenEnum-specific attributes needed for proper GenModel
  else if(find GenEnum(GenmodelEntity)) seq{
   call parseAnnotation(GenmodelEntity,"typeSafeEnumCompatible",EcoreEntity,
    "false", "value");
  }
```

```
// GenFeature-specific attributes needed for proper GenModel
else if(find GenFeature(GenmodelEntity)) let DefaultProperty = undef in seq{
 // EReference-specific attributes
 if(find EReference(EcoreEntity)) seq{
  let ContainerFeature = nemf.packages.ecore.EReference.container,
   ContainmentFeature = nemf.packages.ecore.EReference.containment,
   ChangeableFeature = nemf.packages.ecore.EStructuralFeature.changeable,
   Children = "false" in
  // find container, containment, children attribute types
  try choose Container with
   find AttributeForType(EcoreEntity,ContainerFeature,Container) do
   try choose Containment with
    find AttributeForType(EcoreEntity,ContainmentFeature,Containment) do
    try choose Changeable with
     find AttributeForType(EcoreEntity,ChangeableFeature,Changeable) do seq{
     // default property decided based on attributes
     if(!toBoolean(value(Container)) && !toBoolean(value(Containment))) seq{
      if(toBoolean(value(Changeable))) seq{
       update DefaultProperty =
        nemf.packages.genmodel.GenPropertyKind.Editable;
      }
      else seq{
       update DefaultProperty =
        nemf.packages.genmodel.GenPropertyKind.Readonly;
      }
     } else seq{
      update DefaultProperty = nemf.packages.genmodel.GenPropertyKind.None;
     }
     // children attribute created
     if(toBoolean(value(Containment)))
      call parseAnnotation(GenmodelEntity,"children",EcoreEntity,
       toString(toBoolean(value(Containment))), "value");
     else
      call parseAnnotation(GenmodelEntity,"children",
       EcoreEntity,"","value");
     try choose ChildrenT with
       find ChildrenAttribute(GenmodelEntity,ChildrenT) do seq{
       update Children = value(ChildrenT);
      }
     call parseAnnotation(GenmodelEntity,"createChild",
      EcoreEntity,toString((toBoolean(Children)
       && toBoolean(value(Changeable)))), "value");
     call parseAnnotation(GenmodelEntity,"notify",
      EcoreEntity,toString(toBoolean(Children)), "value");
     }
 }// otherwise (EAttribute)
  else seq{
  let ChangeableRel = nemf.packages.ecore.EStructuralFeature.changeable in
  // default property decided based on attributes
  try choose Changeable with
   find AttributeForType(EcoreEntity,ChangeableRel,Changeable) do seq{
   if(toBoolean(value(Changeable))) seq{
    update DefaultProperty =
     nemf.packages.genmodel.GenPropertyKind.Editable;
   }
   else seq{
    update DefaultProperty =
     nemf.packages.genmodel.GenPropertyKind.Readonly;
   }
   }
  call parseAnnotation(GenmodelEntity,"createChild",
   EcoreEntity,"false","value");
  call parseAnnotation(GenmodelEntity,"notify",EcoreEntity,"true","value");
 }
 if(find EReference(EcoreEntity)
   && DefaultProperty ==
    nemf.packages.genmodel.GenPropertyKind.Editable) seq{
  call parseAnnotation(GenmodelEntity,"propertySortChoices",
    EcoreEntity,"true", "value");
  } else call parseAnnotation(GenmodelEntity,"propertySortChoices",
   EcoreEntity,"false", "value");
  call parseAnnotation(GenmodelEntity,"property",
   EcoreEntity,DefaultProperty, "entity");
 }
}

// pattern to find children attribute for GenFeature entity
pattern ChildrenAttribute(GenEntity, ChildrenAttr) = {
 GenFeature(GenEntity);
 nemf.packages.genmodel.GenFeature.children(ChRel,GenEntity,ChildrenAttr);
 entity(ChildrenAttr);
 find EcoreDataType(ChildrenAttr);
}

// parse annotations for EcoreEntity, create attributes for GenModel entity
rule parseAnnotationList(in EcoreEntity, in GenmodelEntity) = seq{
 // forall annotation
 forall AnnotationKey below EcoreEntity with
   find UnmappedAnnotation(EcoreEntity,GenmodelEntity,AnnotationKey) do seq{
  // copy annotation value to attribute
  call parseAnnotation(GenmodelEntity,
   value(AnnotationKey),EcoreEntity,"","");
 }
}

// pattern to find annotations not yet handled
@localsearch
pattern UnmappedAnnotation(EcoreEntity,GenmodelEntity,AnnotationKey) = {
 EModelElement(EcoreEntity);
 GenBase(GenmodelEntity);
 relation(Re2g,GenmodelEntity,EcoreEntity);
 EModelElement.eAnnotations(Ra,EcoreEntity,EAnnotation);
 EAnnotation(EAnnotation);
 nemf.ecore.datatypes.EString(Source);
 EAnnotation.source(R,EAnnotation,Source);
 check(value(Source) == "emf.gen");
 EStringToStringMapEntry(Details);
 EAnnotation.details(R2,EAnnotation,Details);
 nemf.ecore.datatypes.EString(AnnotationKey);
 EStringToStringMapEntry.key(R3,Details,AnnotationKey);
 neg find ExistingAttributeForName(GenmodelEntity, AnnotationKey);
}

// pattern for checking existing attribute by its name
pattern ExistingAttributeForName(GenmodelEntity, AttributeName) = {
 EModelElement(EcoreEntity);
 relation(Re2g,GenmodelEntity,EcoreEntity);
 EModelElement.eAnnotations(Ra,EcoreEntity,EAnnotation);
 EAnnotation(EAnnotation);
 EStringToStringMapEntry(Details);
 EAnnotation.details(R2,EAnnotation,Details);
 nemf.ecore.datatypes.EString(AttributeName);
 EStringToStringMapEntry.key(R3,Details,AttributeName);
 GenBase(GenmodelEntity);
 nemf.ecore.EClass(GenmodelType);
 instanceOf(GenmodelEntity,GenmodelType);
 nemf.ecore.EDataType(AttributeType);
 nemf.ecore.EClass.EAttribute(AttributeRel,GenmodelType,AttributeType);
 String(Name);
 nemf.ecore.EClass.EStructuralFeature.name(Rn,AttributeRel,Name);
 check(value(Name)==value(AttributeName));
 entity(Attribute);
 instanceOf(Attribute,AttributeType);
 relation(Rel,GenmodelEntity,Attribute);
 instanceOf(Rel,AttributeRel);
}
```

```
// if annotation with given name exist, parse it, otherwise use default
rule parseAnnotation(in GenmodelEntity, in AnnotationKey,
  in EcoreEntity, in Default, in DefaultType) =
 let AnnValue = undef, DefUsed = false in seq{

 // find value from annotation
 try choose AnnValueT with
  find AnnotationValue(EcoreEntity, AnnotationKey, AnnValueT) do seq{
  update AnnValue = value(AnnValueT);
 // use default value otherwise
  } else seq{
  if (Default != "") seq{
   update AnnValue = Default;
   update DefUsed = true;
  }
 }

 if(AnnValue != undef) seq{
  // find attribute type and relation
  try choose AttributeType, AttributeRel with
   find AttributeTypeForName(GenmodelEntity,AnnotationKey,AttributeType,
    AttributeRel) do let Attribute = undef, AttrR = undef in seq{
    // create relation to default entity
    if (DefUsed == true && DefaultType == "entity") seq{
     new(relation(AttrR,GenmodelEntity,Default));
    new(instanceOf(AttrR,AttributeRel));
     }else let Value = str.trim(AnnValue), Rest = str.trim(AnnValue) in
    // handle EReference.many (iterate always finds Many)
    iterate choose Many in datatypes.Boolean with
     find IsAttributeRelMany(AttributeRel,Many) do
     // update params
      let Start = 0, End = str.indexOf(Rest,",") in seq{
     // if many relation and has more values, parse single value
     if(Many == datatypes.Boolean.true && End > Start) seq{
      update Value = str.substring(Rest,Start,End);
     } else seq{
      update Value = Rest;
     }
     // checking value for type safety (Boolean, Enum)
     let Result = undef, Target = undef in seq{
      // handle EBoolean true/false, EEnum values, reference to enums
     call checkAttributeTypeInAnnotationValue(Value,
       AttributeType,Result,Target);
      if(Result == "ok") seq{
       if(Target == undef) seq{
       // create attribute
        new(entity(Attribute) in GenmodelEntity);
       new(instanceOf(Attribute,AttributeType));
       rename(Attribute,AnnotationKey);
       setValue(Attribute,Value);
       // create Attribute relation
       new(relation(AttrR,GenmodelEntity,Attribute));
       new(instanceOf(AttrR,AttributeRel));
       } else if(find EcoreEnumLiteral(Target)) seq{
        // create Attribute relation
       new(relation(AttrR,GenmodelEntity,Target));
       new(instanceOf(AttrR,AttributeRel));
      }
     }// return fault
      else if(str.startsWith(Result,"fault_")) seq{
      println("[ERROR] Annotation Value is wrong: "
       + str.substring(Result,6));
     }
     }
    // update Rest if there is more of it (or comma is the last char)
    if(End > 0 && End < str.length(Rest)-1)
     update Rest = str.substring(Rest,End+1);
     // otherwise exit loop
     else fail;
     }
    }
   else seq{
    println("[Warning] No such attribute (" + AnnotationKey
     + ") in genmodel for (" + GenmodelEntity + ")");
   }
  }
 }

 // pattern for finding annotation value for given key
 @localsearch
 pattern AnnotationValue(EcoreEntity, AnnotationKey, AnnotationValue) = {
  EAnnotation(EAnnotation) in EcoreEntity;
  nemf.ecore.datatypes.EString(Source) in EAnnotation;
  EAnnotation.source(R,EAnnotation,Source);
  check(value(Source) == "emf.gen");
  EStringToStringMapEntry(Details) in EAnnotation;
  EAnnotation.details(R2,EAnnotation,Details);
  nemf.ecore.datatypes.EString(Key) in Details;
  EStringToStringMapEntry.key(R3,Details,Key);
  check(value(Key) == AnnotationKey);
  nemf.ecore.datatypes.EString(AnnotationValue) in Details;
  EStringToStringMapEntry.value(R4,Details,AnnotationValue);
 }

 // pattern for retrieving "many" value of relation
 pattern IsAttributeRelMany(AttributeRel,Many) = {
  nemf.ecore.EClass(EcoreType);
  nemf.ecore.EDataType(AttributeType);
  nemf.ecore.EClass.EAttribute(AttributeRel,EcoreType,AttributeType);
  Boolean(Many);
  nemf.ecore.EClass.EStructuralFeature.many(ManyRel,AttributeRel,Many);
 }

 // pattern for finding attribute type and relation by name
 @localsearch
 pattern AttributeTypeForName(GenmodelEntity, AttributeName,
   AttributeType, AttributeRel) = {
  GenBase(GenmodelEntity);
  nemf.ecore.EClass(GenmodelType);
  instanceOf(GenmodelEntity,GenmodelType);
  nemf.ecore.EDataType(AttributeType);
  nemf.ecore.EClass.EAttribute(AttributeRel,GenmodelType,AttributeType);
  String(Name) in GenmodelType;
  nemf.ecore.EClass.EStructuralFeature.name(R,AttributeRel,Name);
  check(value(Name)==AttributeName);
 }



 // Result is "ok" if type is in order, otherwise another string starting
 // with "fault_" followed by the reason.
 // Target is an EEnumLiteral entity
 // if such an attributetype and value is given
 rule checkAttributeTypeInAnnotationValue(in Value, in AttributeType,
   out Result, out Target) = seq{
  update Result = "ok";
  update Target = undef;
  // handle EBoolean true/false
  if(find EcoreBoolean(AttributeType)) seq{
   if(Value != "true" && Value != "false") seq{
    update Result = "fault_Not Boolean value: "+Value+", expected true/false";
    //update Target = undef;
   }
  }
  // handle EEnum values
  if(find EcoreEnum(AttributeType)) seq{
   // handle reference to enums
   try choose EnumValue in AttributeType with
     find EnumLiteralInEnumType(EnumValue,Value,AttributeType) do seq{
    //update Result = "ok";
    update Target = EnumValue;
   } else seq{
    //update Target = undef;
    update Result = "fault_Value "+Value+" is not "
     + name(AttributeType) + ", expected |";
    forall EnumValue in AttributeType with
      find EnumLiteralsInEnumType(EnumValue,AttributeType) do seq{
     update Result = Result + name(EnumValue) + "|";
    }
    update Result = Result;
   }

  }
 }

 // pattern for finding EnumLiteral type in Enum by its name
 @localsearch
 pattern EnumLiteralInEnumType(EnumValue,Value,AttributeType) = {
  nemf.ecore.datatypes.EEnum(AttributeType);
  nemf.ecore.datatypes.EEnumLiteral(EnumValue);
  instanceOf(EnumValue,AttributeType);
  check(value(EnumValue) == Value);
 }

 // pattern for finding EnumLiterals for an Enum
 pattern EnumLiteralsInEnumType(EnumValue,AttributeType) = {
  nemf.ecore.datatypes.EEnum(AttributeType);
  nemf.ecore.datatypes.EEnumLiteral(EnumValue);
  instanceOf(EnumValue,AttributeType);
 }
}
```

Listing 1.1. Transformation code