Ecore2GenModel with Mitra and GEF3D

Jens von Pilgrim
Jens.vonPilgrim@FernUni-Hagen.de
FernUniversität in Hagen

The case study proposed by Kolovos et al.[1] describes the transformation of an Ecore model to a GenModel. In order to populate the features of the GenModel elements, Kolovos et all propose to use annotations. The solution described here exactly implements that technique using a transformation language called *Mitra*. Besides, an alternative mechanism is implemented, using generic model markers, which are stored in a separate model.

1. Mitra and GEF3D

The language used in this solution is called Mitra, a short version of micro transformation. Mitra is an imperative language optimized for computer-assisted transformation (CAT), that is a transformation which rules are selectively invoked by the user but executed by a transformation engine.[2] Since the user is highly involved in the transformation process, the user interface is important. Many models used in Model Driven Development (MDD) are visualized with a graphical syntax, e.g., UML or Ecore diagrams. Furthermore, model-to-model (M2M) transformations often operate on multiple models, e.g., a source and a target model. Thus, visualizing a single diagram is not sufficient, instead multiple diagrams are to be displayed. For that reason, Mitra is combined with GEF3D[3], a framework enabling 3D diagram editors. Existing 2D editors can easily be adapted, called 3D-fied, in order to be used in 3D: the 2D output of these editors is projected onto a plane in the 3D scene.[4]

Two different user interface concepts are applied, depending on the parameters of the transformation rule The two concepts are illustrated in Figure 1 and Figure 2.

The first concept is called *dropformation*, a short version of drag-and-drop-and-transform. One rule provided by the solution transforms an Ecore class to an UML:

```
manual traced EClass2UMLClass
    (from Ecore.EClass s, into UML.Package p) : (create UML.Class t)
```

The user interface can exploit the parameter modifiers (**from**, **into**) in order to determine the arguments provided by the user. From the user's point of view, the transformation looks like a simple drag-and-drop operation. The dragged Ecore class (the **from** parameter) is simply dropped onto the UML package (the **into** parameter), and the new UML class is created by the transformation.

---

[1] Dimitrios S. Kolovos, Louis M. Rose, Richard F. Paige, and Juan de Lara. Ecore to genmodel case study. In *Transformation Tool Contest 2010*, 2010

[2] The syntax of Mitra is quite similar to Java, except local variables are to be declared with the keyword **var**. In contrast to Java, multiple return parameters can be declared, however this feature is not used in the example here. Other special features are explained in the following.

[3] `http://www.eclipse.org/gef3d`

[4] In the solution, 3D-fied versions of the Ecore Tools editor and UML2 Tools editors are used. These versions are part of GEF3D's example plugin.

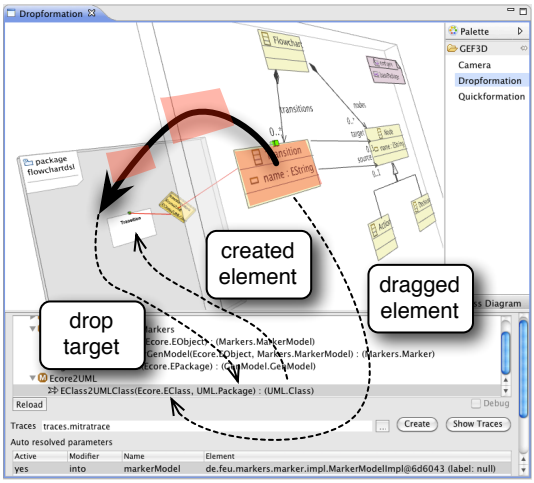

Figure 1: Dropformation

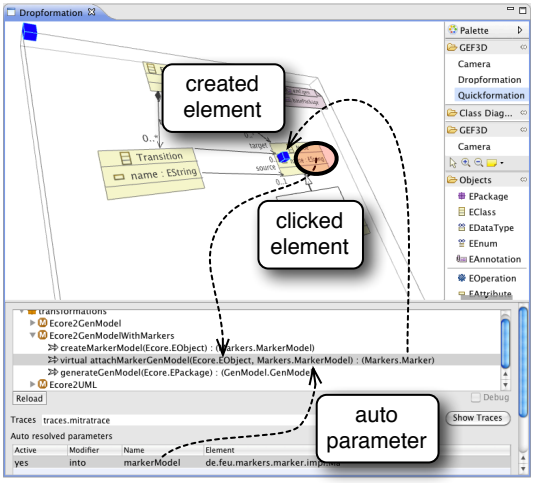

Figure 2: Quickformation

The second concept is called *quickformation*. In this case, no elements are dragged, instead only one argument is provided by the user by simple clicking on the element in the diagram. The following rule creates markers for arbitrary Ecore elements:

```
manual traced virtual attachMarkerGenModel
    (from Ecore.EObject element, into Markers.MarkerModel markerModel):
    (create Markers.Marker marker)
```

Since only one argument (the **from** parameter) is provided, the other argument has to be specified differently. In the example, the second parameter (the **into** parameter) is defined as an *auto (resolved) parameter*. Whenever a quickformation has to be executed, the engine firstly binds selected elements, and then it tries to bind missing parameters using specified auto parameters.[5)]

The diagram presentations of the models is usually very useful for users of the transformation, while transformation developers might prefer working with a representation of the abstract syntax tree, e.g., provided by the sample Ecore Model editor. For that purpose, Mitra provides a so called *Mitra Control* view. Just as in the Dropformation view, rules can be selected and trace models can be loaded and created. Instead of using drop- or quick-formations, rule parameters are presented in a table. The parameters can be set by simply dragging elements from the Ecore Model editor onto the parameter row.

[5)] In the marker example, the marker model could also be resolved automatically using traces and triggers as decribed in subsection 2.3. The auto parameter mechanism is used here mainly for demonstration purposes.

2. Solution of the Ecore2GenModel Case

2.1. Using Annotations

Although Mitra is specialized on semi-automated transformations, it can be used for fully automated transformations as well. The first transformation is a 1:1 copy of the ETL transformation attached to the case study. Just as proposed in the case study, this first transformation copies values defined in annotations to the GenModel elements. Annotations can be set using the properties view of the Ecore Tools editor (embedded into the 3D editor).

The transformation itself looks very much like the Epsilon Transformation Language (ETL) version. Mitra is almost strictly imperative, thus rules must contain code to transform contained elements as the model tree is not traversed automatically. The following loop shows a snippet from the package rule:

```
forEach (var Ecore.EClassifier eClassifier in s.eClassifiers) {
    EClassifiers2GenClassifiers(eClassifier, t);
}
```

While the model tree must be traversed programmatically, reverse lookups are not nessary. Containers are passed as `into`-parameters, which probably makes the code much easier readable for Java experienced programmers. Just as in Java, rule can be overloaded. This feature is used in order to overload the rule transforming classifiers:

```
called virtual EClassifiers2GenClassifiers(from Ecore.EClassifier s,
    into GenModel.GenPackage genPackage):  (return GenModel.GenClassifier t) ...

called EClass2GenClass(from Ecore.EClass s, into GenModel.GenPackage genPackage):
    (create GenModel.GenClass t)
        overloads EClassifiers2GenClassifiers(Ecore.EClassifier s, GenModel.GenPackage
            genPackage): (GenModel.GenClassifier t) ...

called EEnum2GenEnum(from Ecore.EEnum s, into GenModel.GenPackage genPackage):
    (create GenModel.GenEnum t)
        overloads EClassifiers2GenClassifiers(Ecore.EClassifier s, GenModel.GenPackage
            genPackage): (GenModel.GenClassifier t) ...
```

2.2. Reflection

The annotations are copied using the reflection features of Mitra. The following (simplified) snippet shows how features are set using reflection. The conversion is done by calling a native method `parse()` (which can be applied on all types).[6)]

[6)] Mitra distinguished between `target.feature` and `target.<<feature>>`. In the first case, `feature` is the name of a field, while in the second case, `<<feature>>` is an expression. As in Object Constraint Language (OCL), Mitra distinguishes between fields of a model type (using the dot notation) and native methods (using the arrow notation).

```
public called setFeature(any target, String feature, any value) {
    if (target.<<feature>>—>isAssignableFrom(value)) { target.<<feature>> = value; }
    else {
        var any parsed = target.<<feature>>—>type()—>parse(value);
        if (parsed!=null) { target.<<feature>> = parsed; }
    }
}
```

2.3. Markers

Annotations can only be used if the source model supports them. Unfortunately, this is not always the case. Thus we suggest an alternative solution, using markers instead of annotations. In order to use markers, firstly a marker model has to be created using a transformation rule as well. Then, new markers can be added to model elements using quickformations as described above (see Figure 2). In order to provide an information to the user about what features can be set, markers are populated with all features available in the target GenModel elements. This is also done with reflection:

```
called populateMarker(use Type type, into Markers.Marker marker) {
    forEach(var String featureName in type—>fieldNames()) {
        if (type—>fieldChangeable(featureName)) {
            var Type fieldType = type—>fieldType(featureName);
            if (fieldType—>canParse()) { setProperty(marker, featureName, null); }
}   }   }
```

In order to resolve a marker associated to a specific element, Mitra uses *triggers*. Triggers are much like database triggers, however they are not activated by model changes but by certain states of the transformation. Mitra natively supports persistable traces, which are usually used for saving the state of the transformation in order to allow multi-sessions when semi-automatically transforming models.[7] These traces are used to trigger rules automatically. The following code snippet shows a rule which is automatically triggered when an Ecore element is transformed to a GenModel element *and* a marker has been set before:

```
auto copyMarkerProperties(from Markers.Marker marker, into any target)
    trigger (
        EcoreToGenModel(any element, any genmodel_container): (any target),
        attachMarkerGenModel(Ecore.EObject element, Markers.MarkerModel markerModel):
            (Markers.Marker marker)
    ) { /* populate genmodel features with marker properties */     }
```

Triggers can be interpreted as a declarative extension to the otherwise imperative language. The parameters of the rule are to be set by the trigger (or an optional **with**-clause). In the example, the parameters `marker` and `target` are bound to values provided by the rules specified in the trigger. An implicit when-clause clause is defined by similar parameter names, which implies equal values (optionally, an explicit **when**-clause can be defined).

[7] In order to create a trace, a rule must be annotated with the modifier **traced**. Traced rules are also cached, that is when a traced rule is called with the same input parameters multiple times, it is only executed once.

2.4. 3D Benefits

So far, using 3D in the graphical view might be nice but it is not really necessary. Instead of creating 3D cubes, markers could have been visualized with blue rectangles in a 2D view as well. As we noted in section 1, Mitra's main focus is on semi-automated M2M-transformation whereby visualization of multiple diagrams is required. Besides 3D, GEF3D also provides special techniques for combining multiple editors. This is needed for combining the Ecore Tools editor and the marker editor. To give an impression of the benefits of 3D, a small transformation example is included for transforming Ecore classes to UML classes (see Figure 1). This transformation does not only show a typical CAT, but also how traces are visualized with GEF3D (see Figure 3).

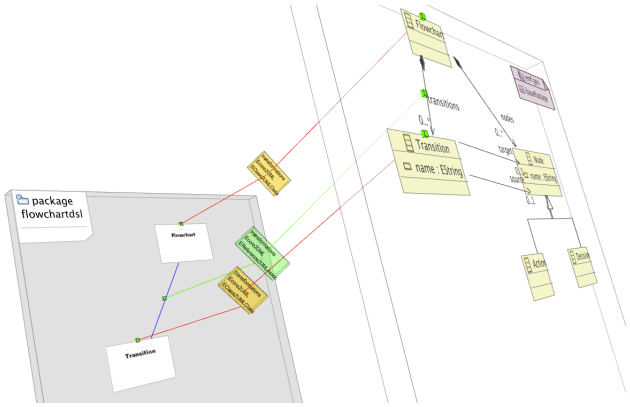

Figure 3: Visualization of traces

3. Discussion

Using the reflection features of Mitra, the case study example could be implemented similarly to the solution proposed by Kolovos et al. Our solution does not use traces for updating the target model automatically, however traces are used by the transformation itself. Although reflection is a powerful mechanism, it disables type checking. E.g., using the non-reflective method of setting a field immediately shows an error if the wrong field name is used. The case study revealed some weaknesses of Metra when handling collections (OCL support is planned but not yet realized) and enumerations.

Providing a good transformation language is only one side of the story. The other side of the story is to provide a good user interface. Since many modeling languages use a graphical notation, transformation tools should (re-) use the concrete notation, i.e. diagrams, as well. GEF3D provides a lot of mechanisms in order to simplify the 3D-fication of existing editors (note that the 2D-diagrams are still fully editable). However, adding transformation support remains a challenge. E.g., the Ecore Tools editor does not automatically reflect model changes while the UML2 Tools editors do. Note that the submitted tool provides only a "raw" drop- and quickformation user interface. For specific applications, many settings could be defined automaticlly, such as predefinded trace models, smart selection of rules (based on the parameter types and other context information), or other domain specific improvments.[8)]

[8)] An example of a domain specific application, a graphical editor for the GMF mapping model using Mitra and GEF3D can be found at `http://dev.eclipse.org/blogs/gef3d/2010/01/20/a-graphical-editor-for-the-gmf-mapping-model/`.

Appendix

A. Listing Ecore2GenModel

This first listing shows the transformation discussed in subsection 2.1. It mainly consits of two parts:

- rules transforming ecore elements to genmodel elements
- helper rules

This is a rather long listing, and usually this would be split up into several modules (Mitra supports include/import of modules).

```
/*
Transformation Tool Contest 2010
Ecore to GenModel (cf 2.1 Using Annotations)
*/
module transformations:Ecore2GenModel {

    /* Declaration of metamodels */
    metamodel ecore:Ecore (nsUri="http://www.eclipse.org/emf/2002/Ecore");
    metamodel ecore:GenModel (nsUri="http://www.eclipse.org/emf/2002/GenModel");

    /* One and only MANUAL rule, which is to be invoked manually using the Quickformation
        mechanism.
        Remarks:
        — the parameter modifier create let Mitra automatically create a new instance; other
            instances are to be created using a Java—like constructor
        — Mitra supports mulitple return parameters, however in this example this is not used
            (but it is helpful if n source elements are transformed to m target elements)
    */
    manual generateGenModel(from Ecore.EPackage s) : (create GenModel.GenModel genModel) {
        var GenModel.GenPackage genPackage = EPackage2GenPackage(s, genModel);
        genModel.genPackages += genPackage;         // add newly created package to container

        // set with defaults
        setAnnotation(s, genModel, "complianceLevel", $GenModel.GenJDKLevel.JDK60);
        setAnnotation(s, genModel, "copyrightFields", false);
        setAnnotation(s, genModel, "modelPluginID", s.name);
        setAnnotation(s, genModel, "modelDirectory", "/" + s.name + "/src");
        setAnnotation(s, genModel, "modelName", s.name—>firstToUpperCase());
        setAnnotation(s, genModel, "importerID", "org.eclipse.emf.importer.ecore");

        // copy others
        copyAnnotations(s, genModel);
    }
```

```
called EPackage2GenPackage(from Ecore.EPackage s, into GenModel.GenModel genModel):
        (create GenModel.GenPackage t) {
    genModel.genPackages += t;
    t.ecorePackage = s;
    setAnnotation(s, t, "disposableProviderFactory", true);
    setAnnotation(s, t, "prefix", s.name->firstToUpperCase());
    copyAnnotations(s, t);

    forEach (var Ecore.EClassifier eClassifier in s.eClassifiers) {
        EClassifiers2GenClassifiers(eClassifier, t);
    }
}

/* A virtual rule can be overloaded. Since EClassifier is an abstract class, this rule does
    not get invoked directly.
*/
called virtual EClassifiers2GenClassifiers(from Ecore.EClassifier s,
        into GenModel.GenPackage genPackage): (return GenModel.GenClassifier t) {
}

/* This rule overloads the afore defined rule. In Mitra, overloading a rule (or
    implementing abstract rules) is more flexibel as in other OO languages:
    The rule names (of overloaded/implemented and overloading/implementing rule)
    may differ, and it is even possible to overload/implement a rule with
    a rule having a different number of parameters. However, this special feature is
    not used in the example.
*/
called EClass2GenClass(from Ecore.EClass s, into GenModel.GenPackage genPackage):
        (create GenModel.GenClass t)
    overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
        GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
{
    genPackage.genClasses += t;
    t.ecoreClass = s;

    setAnnotation(s, t, "image", ! s.^abstract);
    copyAnnotations(s, t);

    forEach (var Ecore.EStructuralFeature eStructuralFeature in s.eStructuralFeatures) {
        EStructuralFeature2GenFeature(eStructuralFeature, t);
    }
    forEach (var Ecore.EOperation eOperation in s.eOperations) {
        EOperation2GenOperation(eOperation, t);
    }
}

called virtual EStructuralFeature2GenFeature(from Ecore.EStructuralFeature s,
        into GenModel.GenClass genClass): (create GenModel.GenFeature t) {
    genClass.genFeatures += t;
    t.ecoreFeature = s;
}
```

```
88
89      /* This rules overloads the virtual rule above. The overloaded rule can be called from an
90          overloading rule by using the statement 'super'. Just as in Java constructors, the
91          overloaded  rule must be called at the very beginning of the body of the overloading
92          rule. In contrast to    constructors in Java, the overloading rule is only called with
93          super and not automatically.
94
95          '$' refers to the type in expressions. E.g., '$GenModel.GenPropertyKind' refers to the
96          type itself. This way, static fields and methods can be called. It is even possible to
97          store the   type in a variable, as demonstrated later on.
98      */
99      called EAttribute2GenFeature(from Ecore.EAttribute s, into GenModel.GenClass genClass):
100             (create GenModel.GenFeature t)
101         overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s,
102             GenModel.GenClass genClass): (GenModel.GenFeature t)
103     {
104         super; // calls overloaded rule
105
106         var GenModel.GenPropertyKind defaultProperty;
107         if (s.changeable) {
108             defaultProperty = $GenModel.GenPropertyKind.Editable; // static access
109         } else {
110             defaultProperty = $GenModel.GenPropertyKind.Readonly;
111         }
112
113         setAnnotation(s, t, "children", false);
114         setAnnotation(s, t, "createChild", false);
115         setAnnotation(s, t, "notify", true);
116         setAnnotation(s, t, "propertySortChoices", false);
117         setAnnotation(s, t, "property", defaultProperty);
118         copyAnnotations(s, t);
119     }
120
121     called EReference2GenFeature(from Ecore.EReference s, into GenModel.GenClass genClass):
122             (create GenModel.GenFeature t)
123         overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s,
124             GenModel.GenClass genClass): (GenModel.GenFeature t)
125     {
126         super;
127
128         var GenModel.GenPropertyKind defaultProperty;
129         if (! s.container &&  ! s.containment) {
130             if (s.changeable) {
131                 defaultProperty = $GenModel.GenPropertyKind.Editable;
132             }
133             else {
134                 defaultProperty = $GenModel.GenPropertyKind.Readonly;
135             }
136         }
137         else {
138             defaultProperty = $GenModel.GenPropertyKind.None;
```

```
139         }
140
141         setAnnotation(s, t, "children", s.containment);
142         setAnnotation(s, t, "createChild", t.children && s.changeable);
143         setAnnotation(s, t, "notify", t.children);
144         setAnnotation(s, t, "propertySortChoices",
145             defaultProperty == $GenModel.GenPropertyKind.Editable);
146         setAnnotation(s, t, "property", defaultProperty);
147         copyAnnotations(s, t);
148     }
149
150     called EOperation2GenOperation(from Ecore.EOperation s, into GenModel.GenClass genClass):
151                 (create GenModel.GenOperation t) {
152             genClass.genOperations += t;
153             t.ecoreOperation = s;
154         copyAnnotations(s, t);
155
156         forEach (var Ecore.EParameter eParamter in s.eParameters) {
157             EParameter2GenParameter(eParamter, t);
158         }
159     }
160
161     called EParameter2GenParameter(from Ecore.EParameter s,
162         into GenModel.GenOperation genOperation):   (create GenModel.GenParameter t) {
163         genOperation.genParameters += t;
164         t.ecoreParameter = s;
165         copyAnnotations(s, t);
166     }
167
168     called EEnum2GenEnum(from Ecore.EEnum s, into GenModel.GenPackage genPackage):
169             (create GenModel.GenEnum t)
170         overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
171             GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
172     {
173         genPackage.genEnums += t;
174         t.ecoreEnum = s;
175
176         setAnnotation(s, t, "typeSafeEnumCompatible", false);
177         copyAnnotations(s, t);
178
179         forEach (var Ecore.EEnumLiteral eLiteral in s.eLiterals) {
180             EEnumLiteral2GenEnumLiteral(eLiteral, t);
181         }
182     }
183
184     called EEnumLiteral2GenEnumLiteral(from Ecore.EEnumLiteral s,
185             into GenModel.GenEnum genEnum): (create GenModel.GenEnumLiteral t)  {
186         genEnum.genEnumLiterals += t;
187         t.ecoreEnumLiteral = s;
188         copyAnnotations(s, t);
189     }
```

```
190
191     called EDataType2GenDataType(from Ecore.EDataType s,
192             into GenModel.GenPackage genPackage): (create GenModel.GenDataType t)
193             overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
194             GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
195     {
196         genPackage.genDataTypes += t;
197         t.ecoreDataType = s;
198         copyAnnotations(s, t);
199     }
200
201     /*-------------------------------------------------------------------------------
202         Helper rules, called by the rules above:
203     */
204
205     called  setAnnotation(from Ecore.EModelElement source, into any target, String label,
206             any default) {
207         var any value = getAnnotation(source, label);
208         if (value==null) value = default;
209         setFeature(target, label, value);
210     }
211
212     /* This rule uses reflection in order to set a feature of a target.
213        — '<<..>>' indicates an expression within a feature access statement.
214        — '—>' indicates a native method call, in contrast to operation calls using
215             the dot notation. This is similar to OCL
216     */
217     public called setFeature(any target, String feature, any value) {
218         if (target.<<feature>>—>isAssignableFrom(value)) {
219             target.<<feature>> = value;
220         } else {
221             if (target.<<feature>>—>isMany()) {
222                 forEach (var String part in value—>split(",")) {
223                     target.<<feature>>+= part;
224                 }
225             } else {
226                 var any parsed = target.<<feature>>—>type()—>parse(value);
227                 if (parsed!=null) { target.<<feature>> = parsed; }
228             }
229         }
230     }
231
232     called copyAnnotations(from Ecore.EModelElement source, into any target) {
233         forEach (var String feature in target—>fieldNames()) {
234             var any value = getAnnotation(source, feature);
235             if (value!=null) {
236                 setFeature(target, feature, value);
237             }
238         }
239     }
240
```

```
241     /* The select statement used in this rule works quite similar to select in
242        OCL or SQL.
243     */
244     called getAnnotation(Ecore.EModelElement element,  String label) : (return any ret) {
245         var Ecore.EAnnotation ann = select first (
246             var Ecore.EAnnotation a in element.eAnnotations where a.source == "emf.gen");
247         if (ann!=null) {
248             var Ecore.EStringToStringMapEntry det = select first (
249                 var Ecore.EStringToStringMapEntry d in ann.details where d.key == label);
250             if (det!=null)
251                 return det.value;
252         }
253         return null;
254     }
255 }
```

B. Listing Ecore2GenModelWithMarkers

This second listing shows the transformation discussed in subsection 2.3. It mainly consits of five parts:

- rules creating the markers
- helper rules for creating markers
- triggered rules to populate features of GenModel element with values of marker properties
- rules transforming ecore elements to genmodel elements
- helper rules for transformation

```
1  /*
2  Transformation Tool Contest 2010
3  Ecore to GenModel with markers (cf. 2.3 Markers)
4  */
5  module transformations:Ecore2GenModelWithMarkers {
6      metamodel ecore:Ecore (nsUri="http://www.eclipse.org/emf/2002/Ecore");
7      metamodel ecore:GenModel (nsUri="http://www.eclipse.org/emf/2002/GenModel");
8      metamodel ecore:Markers (nsUri="http://feu.de/marker");
9
10     /* This rule simply creates the marker model. Note that in the submission, this
11         rule is called manually and the created marker model is specified as an auto
12         parameter. Since traced rules are cached (that is they are only executed the
13         first time when called multiple times with the same input parameters),
14         it would have been possible to call this rule from the following rules in order
15         to automaticlly retrieve the marker model.
16     */
17     manual traced createMarkerModel(use Ecore.EObject element):
18             (create Markers.MarkerModel markerModel) {
19         markerModel.element = element;
20     }
```

```
21      /*
22          This virual rule is overloaded by rules for specific types.
23          From the user's point of view, only manual rules are visible, that is only this
24          virtual rule can be selected in the user interface. Also, only this base rule is used
25          later on in the trigger.
26      */
27      manual traced virtual attachMarkerGenModel(from Ecore.EObject element,
28              into Markers.MarkerModel markerModel): (create Markers.Marker marker)
29      {
30          marker.element = element;
31          markerModel.markers += marker;
32      }
33
34      /*
35          Overloads afore declared virtual rule, uses 'super' to call the overloading rule.
36      */
37      called traced attachMarkerGenModel(use Ecore.EPackage package,
38              into Markers.MarkerModel markerModel): (create Markers.Marker marker)
39          overloads attachMarkerGenModel(Ecore.EObject package,
40              Markers.MarkerModel markerModel): (Markers.Marker marker)
41      {
42          super; // call overloading rule
43
44          // Note: $GenModel.GenPackage returns the type itself
45          populateMarker($GenModel.GenPackage, marker); // populate marker properties
46          populateMarker($GenModel.GenModel, marker); // also with model features
47
48          // set default values
49          setProperty(marker, "complianceLevel", $GenModel.GenJDKLevel.JDK60);
50          setProperty(marker, "copyrightFields", false);
51          setProperty(marker, "modelPluginID", package.name);
52          setProperty(marker, "modelDirectory", "/" + package.name + "/src");
53          setProperty(marker, "modelName", package.name->firstToUpperCase());
54          setProperty(marker, "importerID", "org.eclipse.emf.importer.ecore");
55      }
56
57      called traced attachMarkerGenModel(use Ecore.EClass s,
58              into Markers.MarkerModel markerModel): (create Markers.Marker marker)
59          overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
60              (Markers.Marker)
61      {
62          super;
63          populateMarker($GenModel.GenClass, marker);
64          setProperty(marker, "image", ! s.^abstract);
65      }
66
67
68
69
70
71
```

```
72      called traced attachMarkerGenModel(use Ecore.EAttribute s,
73              into Markers.MarkerModel markerModel): (create Markers.Marker marker)
74          overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
75              (Markers.Marker)
76      {
77          super;
78          populateMarker($GenModel.GenFeature, marker);
79
80          var GenModel.GenPropertyKind defaultProperty;
81          if (s.changeable) {
82              defaultProperty = $GenModel.GenPropertyKind.Editable;
83          } else {
84              defaultProperty = $GenModel.GenPropertyKind.Readonly;
85          }
86          setProperty(marker, "children", false);
87          setProperty(marker, "createChild", false);
88          setProperty(marker, "notify", true);
89          setProperty(marker, "propertySortChoices", false);
90          setProperty(marker, "property", defaultProperty);
91      }
92
93      called traced attachMarkerGenModel(use Ecore.EReference s,
94              into Markers.MarkerModel markerModel): (create Markers.Marker marker)
95          overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
96              (Markers.Marker)
97      {
98          super;
99          populateMarker($GenModel.GenFeature, marker);
100
101         var GenModel.GenPropertyKind defaultProperty;
102         if (! s.container &&  ! s.containment) {
103             if (s.changeable) {
104                 defaultProperty = $GenModel.GenPropertyKind.Editable;
105             }
106             else {
107                 defaultProperty = $GenModel.GenPropertyKind.Readonly;
108             }
109         }
110         else {
111             defaultProperty = $GenModel.GenPropertyKind.None;
112         }
113
114         setProperty(marker, "children", s.containment);
115         setProperty(marker, "createChild", s.containment && s.changeable);
116         setProperty(marker, "notify", s.containment);
117         setProperty(marker, "propertySortChoices",
118             defaultProperty == $GenModel.GenPropertyKind.Editable);
119         setProperty(marker, "property", defaultProperty);
120     }
121
122
```

```
123     called traced attachMarkerGenModel(use Ecore.EOperation s,
124             into Markers.MarkerModel markerModel): (create Markers.Marker marker)
125         overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
126             (Markers.Marker)
127     {
128         super;
129         populateMarker($GenModel.GenOperation, marker);
130     }
131
132     called traced attachMarkerGenModel(use Ecore.EParameter s,
133         into Markers.MarkerModel markerModel): (create Markers.Marker marker)
134         overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel): (Markers.Marker)
135     {
136         super;
137         populateMarker($GenModel.GenParameter, marker);
138     }
139
140     called traced attachMarkerGenModel(use Ecore.EEnum s,
141             into Markers.MarkerModel markerModel): (create Markers.Marker marker)
142         overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
143             (Markers.Marker)
144     {
145         super;
146         populateMarker($GenModel.GenEnum, marker);
147         setProperty(marker, "typeSafeEnumCompatible", false);
148     }
149
150     called traced attachMarkerGenModel(use Ecore.EDataType s,
151             into Markers.MarkerModel markerModel): (create Markers.Marker marker)
152         overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
153             (Markers.Marker)
154     {
155         super;
156         populateMarker($GenModel.GenDataType, marker);
157     }
158
159
160     /*-------------------------------------------------------------------------------
161         Helper rules for attaching markers:
162     */
163
164     called populateMarker(use Type type, into Markers.Marker marker) {
165         forEach(var String featureName in type->fieldNames()) {
166             if (type->fieldChangeable(featureName)) {
167                 var Type fieldType = type->fieldType(featureName);
168                 if (fieldType->canParse()) {
169                     setProperty(marker, featureName, null);
170                 }
171             }
172         }
173     }
```

```
174     /*-------------------------------------------------------------------------------
175         Automatically triggered rules, these rules populate the GenModel element's
176         features with the property values of the marker.
177     */
178
179     /* This rule is triggered by the rules defined in the trigger section. Since this rule is
180         called automatically, all parameters must be bind by in the trigger. That is,
181         - marker is bound using the return value of attachMarkerGenModel(..)
182         - target is bound using the return value of EcoreToGenModel(..)
183         Note that both trigger rules have a parameter 'element'. That is, the values of
184         this parameter must be equal!
185     */
186     auto copyMarkerProperties(from Markers.Marker marker, into any target)
187     trigger (
188         EcoreToGenModel(any element, any genmodel_container): (any target),
189         attachMarkerGenModel(Ecore.EObject element, Markers.MarkerModel markerModel):
190             (Markers.Marker marker)
191     ) {
192         out("copy marker properties into " + target); // only for debugging purposes
193         forEach (var String feature in target->fieldNames()) {
194             var Markers.StringProperty property =   select first (
195                 var Markers.StringProperty p in marker.properties where p.name==feature);
196
197             out("property for feature " + feature + " is " + property);
198
199             if (property!=null && property.stringValue!=null) {
200                 out("set feature " + feature + " from marker " + property.stringValue);
201
202                 setFeature(target, feature, property.stringValue);
203             }
204         }
205     }
206
207     /* A second automatically called rule. Note that the second rule in the trigger equals the
208         second rule in the trigger defined in the rule above. A single traced rule can used in
209         several triggers. Also note that the order of the execution of the triggering rules (as
210         the order of the definition in the trigger) does not matter.
211     */
212     auto copyMarkerPropertiesModel(from Markers.Marker marker, into GenModel.GenModel genModel)
213     trigger (
214         EPackage2GenPackage(Ecore.EPackage epackage, GenModel.GenModel genModel):
215             (GenModel.GenPackage t),
216         attachMarkerGenModel(Ecore.EPackage epackage, Markers.MarkerModel markerModel):
217             (Markers.Marker marker)
218     ) {
219         out("extra: Model: copy marker properties into " + genModel);
220         forEach (var String feature in genModel->fieldNames()) {
221             var Markers.StringProperty property =
222                 select first (var Markers.StringProperty p in marker.properties where p.name==
                        feature);
223
```

```
            out("property for feature " + feature + " is " + property);

            if (property!=null && property.stringValue!=null) {
                out("set feature " + feature + " from marker " + property.stringValue);

                setFeature(genModel, feature, property.stringValue);
            }
        }
    }

    called setProperty(into Markers.Marker marker, String name, any value) {
        var Markers.StringProperty property =   select first (
            var Markers.StringProperty p in marker.properties where p.name==name);
        if (property==null) {
            property = new Markers.StringProperty();
            property.name = name;
            marker.properties += property;
        }
        if (value!=null)
            property.stringValue = value->toString();

    }

    public called setFeature(any target, String feature, any value) {
        if (target.<<feature>>->isAssignableFrom(value)) {
            target.<<feature>> = value;
        } else {
            if (target.<<feature>>->isMany()) {
                forEach (var String part in value->split(",")) {
                    target.<<feature>>+= part;
                }
            } else {
                var any parsed = target.<<feature>>->type()->parse(value);
                if (parsed!=null) {
                    target.<<feature>> = parsed;
                }
            }
        }
    }


    /*-------------------------------------------------------------------------------
        Transformation rules, quite similar to first transformation, except that
        markers are used instead of annotations.
    */

    /* Abstract rule, is implemented by concrete rules later on. This abstract rule is defined
        in order to simplify the definition of the triggers.
    */
    abstract traced EcoreToGenModel(from any ecore_element, into any genmodel_container):
        (return any genmodel_element);
```

```
275     /* Just as in the first example the rule to actually create the GenModel */
276     manual generateGenModel(from Ecore.EPackage s) : (create GenModel.GenModel genModel) {
277         var GenModel.GenPackage genPackage = EPackage2GenPackage(s, genModel);
278         genModel.genPackages += genPackage;
279
280         // set with defaults
281         genModel.complianceLevel = $GenModel.GenJDKLevel.JDK60;
282         genModel.copyrightFields = false;
283         genModel.modelPluginID = s.name;
284         genModel.modelDirectory = "/" + s.name + "/src";
285         genModel.modelName = s.name->firstToUpperCase();
286         genModel.importerID = "org.eclipse.emf.importer.ecore";
287     }
288
289     /* Called from afore defined rule. The rules implements the abstract rule in order to
290         trigger the auto rules. This pattern is true for all following rules.
291     */
292     called traced EPackage2GenPackage(from Ecore.EPackage s, into GenModel.GenModel genModel):
293             (create GenModel.GenPackage t)
294         implements  EcoreToGenModel(any, any ): (any)
295     {
296         genModel.genPackages += t;
297         t.ecorePackage = s;
298         t.disposableProviderFactory = true;
299         t.prefix = s.name->firstToUpperCase();
300         forEach (var Ecore.EClassifier eClassifier in s.eClassifiers) {
301             EClassifiers2GenClassifiers(eClassifier, t);
302         }
303     }
304
305     /* Just as in the previous listing, this virtual rule cannot be called as EClassifier is
306         abstract.
307     */
308     called traced virtual EClassifiers2GenClassifiers(from Ecore.EClassifier s,
309             into GenModel.GenPackage genPackage): (return GenModel.GenClassifier t)
310         implements  EcoreToGenModel(any, any ): (any)
311     {
312         // must not be called
313     }
314
315     called EClass2GenClass(from Ecore.EClass s, into GenModel.GenPackage genPackage):
316             (create GenModel.GenClass t)
317         overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
318             GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
319     {
320         genPackage.genClasses += t;
321         t.ecoreClass = s;
322         t.image = ! s.^abstract;
323
324         forEach (var Ecore.EStructuralFeature eStructuralFeature in s.eStructuralFeatures) {
325             EStructuralFeature2GenFeature(eStructuralFeature, t);
```

```
326         }
327         forEach (var Ecore.EOperation eOperation in s.eOperations) {
328             EOperation2GenOperation(eOperation, t);
329         }
330     }
331 
332     called traced virtual EStructuralFeature2GenFeature(from Ecore.EStructuralFeature s,
333             into GenModel.GenClass genClass): (create GenModel.GenFeature t)
334         implements  EcoreToGenModel(any, any ): (any)
335     {
336         genClass.genFeatures += t;
337         t.ecoreFeature = s;
338     }
339 
340     called EAttribute2GenFeature(from Ecore.EAttribute s, into GenModel.GenClass genClass):
341             (create GenModel.GenFeature t)
342     overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s,
343             GenModel.GenClass genClass): (GenModel.GenFeature t)
344     {
345         super;
346 
347         var GenModel.GenPropertyKind defaultProperty;
348         if (s.changeable) {
349             defaultProperty = $GenModel.GenPropertyKind.Editable;
350         } else {
351             defaultProperty = $GenModel.GenPropertyKind.Readonly;
352         }
353 
354         t.children = false;
355         t.createChild = false;
356         t.notify = true;
357         t.propertySortChoices = false;
358         t.property = defaultProperty;
359     }
360 
361     called EReference2GenFeature(from Ecore.EReference s, into GenModel.GenClass genClass):
362             (create GenModel.GenFeature t)
363         overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s,
364             GenModel.GenClass genClass): (GenModel.GenFeature t)
365     {
366         super;
367 
368         var GenModel.GenPropertyKind defaultProperty;
369         if (! s.container &&  ! s.containment) {
370             if (s.changeable) {
371                 defaultProperty = $GenModel.GenPropertyKind.Editable;
372             }
373             else {
374                 defaultProperty = $GenModel.GenPropertyKind.Readonly;
375             }
376         }
```

```
377         else {
378             defaultProperty = $GenModel.GenPropertyKind.None;
379         }
380 
381         t.children = s.containment;
382         t.createChild = t.children && s.changeable;
383         t.notify = t.children;
384         t.propertySortChoices = defaultProperty == $GenModel.GenPropertyKind.Editable;
385         t.property = defaultProperty;
386 
387     }
388 
389     called traced EOperation2GenOperation(from Ecore.EOperation s,
390             into GenModel.GenClass genClass): (create GenModel.GenOperation t)
391             implements  EcoreToGenModel(any, any ): (any)
392     {
393             genClass.genOperations += t;
394             t.ecoreOperation = s;
395         forEach (var Ecore.EParameter eParamter in s.eParameters) {
396             EParameter2GenParameter(eParamter, t);
397         }
398     }
399 
400     called traced EParameter2GenParameter(from Ecore.EParameter s,
401             into GenModel.GenOperation genOperation):   (create GenModel.GenParameter t)
402             implements  EcoreToGenModel(any, any ): (any)
403     {
404         genOperation.genParameters += t;
405         t.ecoreParameter = s;
406     }
407 
408     called traced EEnum2GenEnum(from Ecore.EEnum s, into GenModel.GenPackage genPackage):
409             (create GenModel.GenEnum t)
410         overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
411                 GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
412     {
413         genPackage.genEnums += t;
414         t.ecoreEnum = s;
415         t.typeSafeEnumCompatible = false;
416         forEach (var Ecore.EEnumLiteral eLiteral in s.eLiterals) {
417             EEnumLiteral2GenEnumLiteral(eLiteral, t);
418         }
419     }
420 
421     called traced EEnumLiteral2GenEnumLiteral(from Ecore.EEnumLiteral s,
422             into GenModel.GenEnum genEnum): (create GenModel.GenEnumLiteral t)
423         implements  EcoreToGenModel(any, any ): (any)
424     {
425         genEnum.genEnumLiterals += t;
426         t.ecoreEnumLiteral = s;
427     }
```

```
428     called EDataType2GenDataType(from Ecore.EDataType s,
429             into GenModel.GenPackage genPackage): (create GenModel.GenDataType t)
430         overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
431             GenModel.GenPackage genPackage):    (GenModel.GenClassifier t)
432     {
433         genPackage.genDataTypes += t;
434         t.ecoreDataType = s;
435     }
436
437     /* Definition of a native (=Java) method.
438     */
439     called out(String message) native(class="mitra.Log");
440 }
```

C. Listing Ecore2GenModelWithMarkers

This second listing shows the transformation discussed in subsection 2.4. This kind of transformation is used for *dropformations*. In this very simple example, only one manually called rule is defined. However, the strength of Mitra is to enable the definition of different manual rules, in order to allow similar elements to be transformed differently without the need for markers or annotations. Using abstract rules, different rules can then be unified in order to trigger the very same auto rules. As in this example, the auto rules can be used to transform relations between node elements automatically.

```
1  /*
2  Transformation Tool Contest 2010
3  Ecore2UML (cf. 2.4 3D Benefits)
4  */
5  module transformations:Ecore2UML {
6
7      metamodel ecore:Ecore (nsUri="http://www.eclipse.org/emf/2002/Ecore");
8      metamodel uml2:UML ();
9
10     /* The manual rule used in a dropformation. It also triggeres the following rule.
11     */
12     manual traced EClass2UMLClass(from Ecore.EClass s, into UML.Package p) : (create UML.Class
           t) {
13         t.name = s.name;
14         p.packagedElement += t;
15     }
16
17
18
19
20
21
22
23
24
```

```
25      /* Triggered by afore defined rule. The with-block defines the first parameter, the second
26          parameter is bound to the arguments of the trigger rules.
27      */
28      auto traced EReference2UMLAssoc(from Ecore.EReference eRef, into UML.Package p):
29              (create UML.Association assoc)
30      trigger (
31          EClass2UMLClass(Ecore.EClass eClassFrom, UML.Package p) : (UML.Class fromClass),
32          EClass2UMLClass(Ecore.EClass eClassTo, UML.Package p) : (UML.Class toClass)
33      ) with {
34          eRef = select first (var Ecore.EReference e in eClassFrom.eReferences where
35              e.eType == eClassTo);
36      }
37      when (eRef!=null ) // explicit when clause
38      {
39          assoc.name = eRef.name;
40          setAssocEnds(assoc, fromClass, toClass);
41          p.packagedElement += assoc;
42      }
43
44      /* Helper rule
45      */
46      called setAssocEnds(UML.Association assoc, UML.Class srcClass, UML.Class dstClass) {
47          var UML.Property src = new UML.Property();
48          src.name = "src";
49          src.type = srcClass;
50          assoc.ownedEnd += src;
51
52          var UML.Property dst = new UML.Property();
53          dst.name = "dst";
54          dst.type = dstClass;
55          assoc.ownedEnd += dst;
56      }
57  }
```