The case study proposed by Kolovos et al. describes the transformation of an Ecore model to a GenModel. In order to populate the features of the GenModel elements, Kolovos et al. 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. 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, 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.

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.

---

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.
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 described 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:

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

While the model tree must be traversed programmatically, reverse lookups are not necessary. 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:

```java
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).
2. Solution of the Ecore2GenModel Case

```java
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:

```java
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. 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:

```java
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.
3. Discussion

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](http://dev.eclipse.org/blogs/gef3d/2010/01/20/a-graphical-editor-for-the-gmf-mapping-model/)

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 automatically, such as predefined trace models, smart selection of rules (based on the parameter types and other context information), or other domain specific improvements.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/](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 consists 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).

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

    /* Declaration of metamodels */

    /* 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 multiple 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);
    }
}
```
A. Listing Ecore2GenModel

```java
37 called EPackage2GenPackage(from Ecore.EPackage s, into GenModel.GenModel genModel):
38     (create GenModel.GenPackage t) {
39         genModel.genPackages += t;
40         t.ecorePackage = s;
41         setAnnotation(s, t, "disposableProviderFactory", true);
42         setAnnotation(s, t, "prefix", s.name->firstToUpperCase());
43         copyAnnotations(s, t);
44
45     forEach (var Ecore.EClassifier eClassifier in s.eClassifiers) {
46         EClassifiers2GenClassifiers(eClassifier, t);
47     }
48 }
49
50 /* A virtual rule can be overloaded. Since EClassifier is an abstract class, this rule does
51     not get invoked directly. */
52 called virtual EClassifiers2GenClassifiers(from Ecore.EClassifier s, into GenModel.GenPackage genPackage): (return GenModel.GenClassifier t) {
53     }
54
55 /* This rule overloads the afore defined rule. In Mitra, overloading a rule (or
56     implementing abstract rules) is more flexible as in other OO languages:
57     The rule names (of overloaded/implemented and overloading/implementing rule)
58     may differ, and it is even possible to overload/implement a rule with
59     a rule having a different number of parameters. However, this special feature is
60     not used in the example. */
61 called EClass2GenClass(from Ecore.EClass s, into GenModel.GenPackage genPackage): (create GenModel.GenClass t)
62     overloads EClassifiers2GenClassifiers(Ecore.EClassifier s, GenModel.GenPackage genPackage): (GenModel.GenClassifier t)
63     {
64         genPackage.genClasses += t;
65         t.ecoreClass = s;
66
67         setAnnotation(s, t, "image", ! s.^abstract);
68         copyAnnotations(s, t);
69
70     forEach (var Ecore.EStructuralFeature eStructuralFeature in s.eStructuralFeatures) {
71         EStructuralFeature2GenFeature(eStructuralFeature, t);
72     }
73
74     forEach (var Ecore.EOperation eOperation in s.eOperations) {
75         EOperation2GenOperation(eOperation, t);
76     }
77 }
78
called virtual EStructuralFeature2GenFeature(from Ecore.EStructuralFeature s, into GenModel.GenClass genClass): (create GenModel.GenFeature t) {
79     genClass.genFeatures += t;
80     t.ecoreFeature = s;
81 }
```
A. Listing Ecore2GenModel

/*
  This rules overloads the virtual rule above. The overloaded rule can be called from an
  overloading rule by using the statement 'super'. Just as in Java constructors, the
  overloaded rule must be called at the very beginning of the body of the overloading
  rule. In contrast to constructors in Java, the overloading rule is only called with
  super and not automatically.

  '$' refers to the type in expressions. E.g., '$GenModel.GenPropertyKind' refers to the
  type itself. This way, static fields and methods can be called. It is even possible to
  store the type in a variable, as demonstrated later on.
*/
called EAttribute2GenFeature(from Ecore.EAttribute s, into GenModel.GenClass genClass):
    (create GenModel.GenFeature t)
overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s, 
    GenModel.GenClass genClass): (GenModel.GenFeature t)
{
    super; // calls overloaded rule

    var GenModel.GenPropertyKind defaultProperty;
    if (s.changeable) {
        defaultProperty = $GenModel.GenPropertyKind.Editable; // static access
    } else {
        defaultProperty = $GenModel.GenPropertyKind.Readonly;
    }

    setAnnotation(s, t, "children", false);
    setAnnotation(s, t, "createChild", false);
    setAnnotation(s, t, "notify", true);
    setAnnotation(s, t, "propertySortChoices", false);
    setAnnotation(s, t, "property", defaultProperty);
    copyAnnotations(s, t);
}
called EReference2GenFeature(from Ecore.EReference s, into GenModel.GenClass genClass):
    (create GenModel.GenFeature t)
overloads EStructuralFeature2GenFeature(Ecore.EStructuralFeature s, 
    GenModel.GenClass genClass): (GenModel.GenFeature t)
{
    super;

    var GenModel.GenPropertyKind defaultProperty;
    if (! s.container && ! s.containment) {
        if (s.changeable) {
            defaultProperty = $GenModel.GenPropertyKind.Editable;
        } else {
            defaultProperty = $GenModel.GenPropertyKind.Readonly;
        }
    } else {
        defaultProperty = $GenModel.GenPropertyKind.None;
    }

    setAnnotation(s, t, "children", false);
    setAnnotation(s, t, "createChild", false);
    setAnnotation(s, t, "notify", true);
    setAnnotation(s, t, "propertySortChoices", false);
    setAnnotation(s, t, "property", defaultProperty);
    copyAnnotations(s, t);
A. Listing Ecore2GenModel

```java
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",
        defaultProperty == $GenModel.GenPropertyKind.Editable);
145 setAnnotation(s, t, "property", defaultProperty);
146 copyAnnotations(s, t);
147 }
148
called EOperation2GenOperation(from Ecore.EOperation s, into GenModel.GenClass genClass):
    (create GenModel.GenOperation t) {
        genClass.genOperations += t;
        t.ecoreOperation = s;
        copyAnnotations(s, t);

156 forEach (var Ecore.EParameter eParameter in s.eParameters) {
            EParameter2GenParameter(eParameter, t);
    }
158
called EParameter2GenParameter(from Ecore.EParameter s,
    into GenModel.GenOperation genOperation): (create GenModel.GenParameter t) {
    genOperation.genParameters += t;
    t.ecoreParameter = s;
    copyAnnotations(s, t);

166 }
167
called EEnum2GenEnum(from Ecore.EEnum s, into GenModel.GenPackage genPackage):
    (create GenModel.GenEnum t)

174 overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
    GenModel.GenPackage genPackage): (GenModel.GenClassifier t)

183 }
184
called EEnumLiteral2GenEnumLiteral(from Ecore.EEnumLiteral s,
    into GenModel.GenEnum genEnum): (create GenModel.GenEnumLiteral t) {
    genEnum.genEnumLiterals += t;
    t.ecoreEnumLiteral = s;
    copyAnnotations(s, t);
}
```
A. Listing Ecore2GenModel

called EDataType2GenDataType(from Ecore.EDataType s,
    into GenModel.GenPackage genPackage): (create GenModel.GenDataType t)
overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
    GenModel.GenPackage genPackage): (GenModel.GenClassifier t)
{
    genPackage.genDataTypes += t;
    t.ecoreDataType = s;
    copyAnnotations(s, t);
}

/*
Helper rules, called by the rules above:
*/
called setAnnotation(from Ecore.EModelElement source, into any target, String label,
    any default) {
    var any value = getAnnotation(source, label);
    if (value==null) value = default;
    setFeature(target, label, value);
}

/* This rule uses reflection in order to set a feature of a target.
   '<<..>>' indicates an expression within a feature access statement.
   '->' indicates a native method call, in contrast to operation calls using
   the dot notation. This is similar to OCL
*/
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; }
        }
    }
}

called copyAnnotations(from Ecore.EModelElement source, into any target) {
    forEach (var String feature in target.<>fieldNames()) {
        var any value = getAnnotation(source, feature);
        if (value==null) {
            setFeature(target, feature, value);
        }
    }
}
B. Listing Ecore2GenModelWithMarkers

This second listing shows the transformation discussed in subsection 2.3. It mainly consists 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

```java
/* The select statement used in this rule works quite similar to select in OCL or SQL.
*
called getAnnotation(EModelElement element, String label) : (return any ret) {
  var EAnnotation ann = select first {
    var EAnnotation a in element.eAnnotations where a.source == "emf.gen";
    if (ann!=null) {
      var EStringToStringMapEntry det = select first {
        var EStringToStringMapEntry d in ann.details where d.key == label;
        if (det!=null)
          return det.value;
      }
      return null;
    }
  }
}

B. Listing Ecore2GenModelWithMarkers

module transformations:Ecore2GenModelWithMarkers {
  metamodel ecore:Markers (nsUri="http://feu.de/marker");

  /* This rule simply creates the marker model. Note that in the submission, this rule is called manually and the created marker model is specified as an auto parameter. Since traced rules are cached (that is they are only executed the first time when called multiple times with the same input parameters), it would have been possible to call this rule from the following rules in order to automatically retrieve the marker model.
   */
  manual traced createMarkerModel(use Ecore.EObject element);
    (create Markers.MarkerModel markerModel) {
      markerModel.element = element;
    }
*/
```

Transformation Tool Contest 2010
Ecore to GenModel with markers (cf. 2.3 Markers)
/*
   This virtual rule is overloaded by rules for specific types.
   From the user’s point of view, only manual rules are visible, that is only this
   virtual rule can be selected in the user interface. Also, only this base rule is used
   later on in the trigger.
*/

manual traced virtual attachMarkerGenModel(from Ecore.EObject element,
   into Markers.MarkerModel markerModel): (create Markers.Marker marker)
{
    marker.element = element;
    markerModel.markers += marker;
}

/*
   Overloads afore declared virtual rule, uses ‘super’ to call the overloading rule.
*/

called traced attachMarkerGenModel(use Ecore.EObject, Markers.MarkerModel markerModel);
   (create Markers.Marker marker)
{
   super; // call overloading rule

   // Note: $GenModel.GenPackage returns the type itself
   populateMarker($GenModel.GenPackage, marker); // populate marker properties
   populateMarker($GenModel.GenModel, marker); // also with model features

   // set default values
   setProperty(marker, "complianceLevel", $GenModel.GenJDKLevel.JDK60);
   setProperty(marker, "copyrightFields", false);
   setProperty(marker, "modelPluginID", package.name);
   setProperty(marker, "modelDirectory", "/" + package.name + "/src");
   setProperty(marker, "modelName", package.name->firstToUpperCase());
   setProperty(marker, "importerID", "org.eclipse.emf.importer.ecore");
}

called traced attachMarkerGenModel(use Ecore.EClass s,
   into Markers.MarkerModel markerModel):
   (create Markers.Marker marker)
{
   super;
   populateMarker($GenModel.GenClass, marker);
   setProperty(marker, "image", ! s."abstract");
}
called traced attachMarkerGenModel(use Ecore.EAttribute s, 
    into Markers.MarkerModel markerModel): (create Markers.Marker marker)
overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
    (Markers.Marker)
{
    super;
    populateMarker($GenModel.GenFeature, marker);

    var GenModel.GenPropertyKind defaultProperty;
    if (s.changeable) {
        defaultProperty = $GenModel.GenPropertyKind.Editable;
    } else {
        defaultProperty = $GenModel.GenPropertyKind.Readonly;
    }
    setProperty(marker, "children", false);
    setProperty(marker, "createChild", false);
    setProperty(marker, "notify", true);
    setProperty(marker, "propertySortChoices", false);
    setProperty(marker, "property", defaultProperty);
}
called traced attachMarkerGenModel(use Ecore.EReference s, 
    into Markers.MarkerModel markerModel): (create Markers.Marker marker)
overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel):
    (Markers.Marker)
{
    super;
    populateMarker($GenModel.GenFeature, marker);

    var GenModel.GenPropertyKind defaultProperty;
    if (! s.container || ! s.containment) {
        if (s.changeable) {
            defaultProperty = $GenModel.GenPropertyKind.Editable;
        } else {
            defaultProperty = $GenModel.GenPropertyKind.Readonly;
        }
    } else {
        defaultProperty = $GenModel.GenPropertyKind.None;
    }
    setProperty(marker, "children", s.containment);
    setProperty(marker, "createChild", s.containment && s.changeable);
    setProperty(marker, "notify", s.containment);
    setProperty(marker, "propertySortChoices", 
        defaultProperty == $GenModel.GenPropertyKind.Editable);
    setProperty(marker, "property", defaultProperty);
}
called traced attachMarkerGenModel(use Ecore.EOperation s,  
          into Markers.MarkerModel markerModel): (create Markers.Marker marker)  
          
        overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel): (Markers.Marker)  
        
          {  
            super;  
            populateMarker($GenModel.GenOperation, marker);  
          }  
          
called traced attachMarkerGenModel(use Ecore.EParameter s,  
          into Markers.MarkerModel markerModel): (create Markers.Marker marker)  
          
        overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel): (Markers.Marker)  
        
          {  
            super;  
            populateMarker($GenModel.GenParameter, marker);  
          }  
          
called traced attachMarkerGenModel(use Ecore.EEnum s,  
          into Markers.MarkerModel markerModel): (create Markers.Marker marker)  
          
        overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel): (Markers.Marker)  
        
          {  
            super;  
            populateMarker($GenModel.GenEnum, marker);  
           setProperty(marker, "typeSafeEnumCompatible", false);  
          }  
          
called traced attachMarkerGenModel(use Ecore.EDataType s,  
          into Markers.MarkerModel markerModel): (create Markers.Marker marker)  
          
        overloads attachMarkerGenModel(Ecore.EObject, Markers.MarkerModel): (Markers.Marker)  
        
          {  
            super;  
            populateMarker($GenModel.GenDataType, marker);  
          }  
          
           */  
           
           Helper rules for attaching markers:  
           */  
           
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);  
              }  
            }  
          }  
          
}
Automatically triggered rules, these rules populate the GenModel element's features with the property values of the marker.

This rule is triggered by the rules defined in the trigger section. Since this rule is called automatically, all parameters must be bind by in the trigger. That is, marker is bound using the return value of attachMarkerGenModel(...) and target is bound using the return value of EcoreToGenModel(...)

Note that both trigger rules have a parameter 'element'. That is, the values of this parameter must be equal!

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)
  ) {
    out("copy marker properties into " + target); // only for debugging purposes
    forEach (var String feature in target->fieldNames()) {
      var Markers.StringProperty property = select first {
        var Markers.StringProperty p in marker.properties where p.name==feature);

      out("property for feature is " + feature + " from marker " + property);

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

  /* A second automatically called rule. Note that the second rule in the trigger equals the second rule in the trigger defined in the rule above. A single traced rule can used in several triggers. Also note that the order of the execution of the triggering rules (as the order of the definition in the trigger) does not matter. */

auto copyMarkerPropertiesModel(from Markers.Marker marker, into GenModel.GenModel genModel)

  trigger (  
      (GenModel.GenPackage t),
    attachMarkerGenModel(Ecore.EPackage epackage, Markers.MarkerModel markerModel):
      (Markers.Marker marker)
  ) {
    out("extra Model: copy marker properties into " + genModel);
    forEach (var String feature in genModel->fieldNames()) {
      var Markers.StringProperty property = select first (var Markers.StringProperty p in marker.properties where p.name==
      feature);
out("property\_fon\_feature\_r + feature + `\_isa` + property);

if (property!=null && property.stringValue!=null) {
    out("set\_feature\_r + feature + `\_from\_marken\_r` + 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\rightarrow toString();
}

called setFeature(any target, String feature, any value) {
    if (target.<<feature>\rightarrow isAssignableFrom(value)) {
        target.<<feature>> = value;
    } else {
        if (target.<<feature>\rightarrow isMany()) {
            foreach (var String part in value\rightarrow split(",")) {
                target.<<feature>>+= part;
            }
        } else {
            var any parsed = target.<<feature>\rightarrow type()\rightarrow 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);
/* Just as in the first example the rule to actually create the GenModel */

manual generateGenModel(from Ecore.EPackage s) : (create GenModel.GenModel genModel) {
  var GenModel.GenPackage genPackage = EPackage2GenPackage(s, genModel);
  genModel.genPackages += genPackage;

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

/* Called from afore defined rule. The rules implements the abstract rule in order to
 trigger the auto rules. This pattern is true for all following rules. */
called traced EPackage2GenPackage(from Ecore.EPackage s, into GenModel.GenModel genModel):
  (create GenModel.GenPackage t)
  implements EcoreToGenModel(any, any); (any)
  {
    genModel.genPackages += t;
    t.ecorePackage = s;
    t.disposableProviderFactory = true;
    t.prefix = s.name->firstToUpperCase();
    forEach (var Ecore.EClassifier eClassifier in s.eClassifiers) {
      EClassifiers2GenClassifiers(eClassifier, t);
    }
  }

/* Just as in the previous listing, this virtual rule cannot be called as EClassifier is
abstract. */
called traced virtual EClassifiers2GenClassifiers(from Ecore.EClassifier s, into GenModel.GenPackage genPackage): (return GenModel.GenClassifier t)
  implements EcoreToGenModel(any, any); (any)
  {
    // must not be called
  }

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;
    t.image = ! s.^abstract;
    forEach (var Ecore.EStructuralFeature eStructuralFeature in s.eStructuralFeatures) {
      EStructuralFeature2GenFeature(eStructuralFeature, t);
    }
  }
Listing Ecore2GenModelWithMarkers

```java
B. Listing Ecore2GenModelWithMarkers

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            } else {
373                defaultProperty = $GenModel.GenPropertyKind.Readonly;
374            }
375        }
376
377    }
```
else {
    defaultProperty = $GenModel.GenPropertyKind.None;
}

t.children = s.containment;
t.createChild = t.children && s.changeable;
t.notify = t.children;
t.propertySortChoices = defaultProperty == $GenModel.GenPropertyKind.Editable;
t.property = defaultProperty;

}
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.

```java
/*
Transformation Tool Contest 2010
Ecore2UML (cf. 2.4 3D Benefits)
*/
module transformations:Ecore2UML {

  metamodel uml2:UML (){

    /* The manual rule used in a dropformation. It also triggers the following rule. */
    manual traced EClass2UMLClass(from Ecore.EClass s, into UML.Package p) : (create UML.Class t) {
      t.name = s.name;
      p.packageElement += t;
    }

    called EDataType2GenDataType(from Ecore.EDataType s,
      into GenModel.GenPackage genPackage): (create GenModel.GenDataType t)
    overloads EClassifiers2GenClassifiers(Ecore.EClassifier s,
      GenModel.GenPackage genPackage): (GenModel.GenClassifier t)
    {
      genPackage.genDataTypes += t;
      t.ecoreDataType = s;
    }

    /* Definition of a native (=Java) method. */
called out(String message) native(class="mitra.Log");
}
```

C. Listing Ecore2GenModelWithMarkers
/* Triggered by afore defined rule. The with-block defines the first parameter, the second parameter is bound to the arguments of the trigger rules. */

```
auto traced EReference2UMLAssoc(from Ecore.EReference eRef, into UML.Package p):
    (create UML.Association assoc)
    trigger (
        EClass2UMLClass(Ecore.EClass eClassFrom, UML.Package p) : (UML.Class fromClass),
        EClass2UMLClass(Ecore.EClass eClassTo, UML.Package p) : (UML.Class toClass)
    ) with {
        eRef = select first (var Ecore.EReference e in eClassFrom.eReferences where
            e.eType == eClassTo);
    }

    when {eRef!=null} // explicit when clause
    {
        assoc.name = eRef.name;
        setAssocEnds(assoc, fromClass, toClass);
        p.packagedElement += assoc;
    }

    /* Helper rule */

called setAssocEnds(UML.Association assoc, UML.Class srcClass, UML.Class dstClass) {
    var UML.Property src = new UML.Property();
    src.name = "src";
    src.type = srcClass;
    assoc.ownedEnd += src;

    var UML.Property dst = new UML.Property();
    dst.name = "dst";
    dst.type = dstClass;
    assoc.ownedEnd += dst;
}
```