Metamodeling with EMF: Gener reusable Java snippets

Ken McNeill

Published on April 08, 2008 / *Updated: September 21, 2010*

EMF is an integral part of the Eclipse platform, as we cornerstone of related Eclipse Visual Editor, SDO, XSD, and UML — many of re integrated into IBM Developer and WebSphere® Business Modeler. Toda has grown to encompa enumerated types, annotations, and generics. If you N to EMF, see Related started.

In most documents and tutorials, EMF is used to model *data* and *interfaces* (e.g. *L* documentation), and not *behavior*. Of course, there are some default method imp but these concern relationships between model elements. Moreover, there are ve being used as a "meta-metamodel" — with the exception of the Eclipse Foundatio with EMF" (see Related topics) — but not a single example showing how to extend

Finally, the process of using and extending the EMF JET templates is not well doc

Contents

Why extend the Ecore metamodel?

The Ecore metamodel is a powerful tool for designing Model-Driven Architecture (which can be used as a starting point for software development. Typically, we wou the objects (of type `EClass`) in our domain of application, their attributes, and thei relationships. We would also define the specific operations that belong to those o using the `EOperation` model element. By default, EMF will generate *skeletons,* or r signatures for those operations, but we have to go back and implement those ope often recoding similar logic time and again.

But what if we want to specify some sort of arbitrary implementation behavior rig model? One approach is to add text-based annotations (of type `EAnnotation`) to m objects and to interpret those annotations in templates during code generation. F example of this, see the Eclipse Foundation article "Implementing Model Integrity with MDT OCL" (see Related topics). However, our object is not to validate model e as described in the article but rather to model implementation itself, in order to re metamodel elements with any concrete model. To do this, we need to extend the metamodel.

The extended metamode

Accompanying this article is a highly simplified pr ıatic model that extends metamodel or framework; it is strictly a prototypical set of elements for illustratin implementation with EMF. Figure 1 shows a snapshot of our sample extended me description of each element.

Figure 1. EcoreX model

Contents

EcoreX elements

`EPackageX` extends `EPackage`

This is a simple "marker" extension of Ecore element `EPackage` with no additional
because by default, the EMF Editor Plug-in for element `EPackage` will not allow a s
element (see `EClassX`, below). By providing a model element that extends `EPacka`
allow adding an `EClassX` child element to an `EPackageX`.

`EClassX` extends `EClass`

Again, this is a simple marker extension of Ecore e `EClass` with no addition
element is necessary because by default, the Edit -in for `EClass` will not allo
which is our objective in this article.

`EOperationImpl` extends `EOperation`

This is the principal entity and entry point for adding concrete metafunctionality t
conferred with attributes that do not exist in the base `EOperation` element of Ecor
belong to `EOperationImpl` and are used to compose programmatic implementatio
variables and statements, and can return a reference or value.

`LocalVariable` extends `ETypedElement`

`LocalVariable` is a locally scoped variable. A variable has a name and a Java type
these attributes exist already in its super-superclass `EParameter`, `LocalVariable` c

`Statement` is an abstract super-class.

`LiteralAssignment` extends `Statement`

A `LiteralAssignment` refers to a variable and has a `String` attribute allowing the u
assigned to a variable (e.g., "hello," "4.5" could be assigned to a `String` or `float`, r

`Access` extends `Statement`

An `Access` represents the act of referencing a Java field or operation.

`FieldReferenceAssignment` extends `Access`

Accessing a field in order to assign a value (e.g., `var1 = var2.name`).

`Invoke` extends `Access`

Invokes an operation (Java method). The result of an `Invoke` can be assigned to a

Figure 2 offers a more UML-like representation of the EcoreX metamodel.

Figure 2. Ecorex model diagram

Contents

Getting started

There are six high-level steps to this article:

1. Extend the Ecore metamodel, adding new sem
2. Create a `genmodel` for the extended metamodel.
3. Generate an EMF Editor for the metamodel, and install its plug-in.
4. Using the new editor, build a concrete model describing programmatic behavi
5. Create and configure a `genmodel` for the concrete model.
6. Generate concrete Java code, based on the concrete model.

You can create or import the metamodel described above. In both cases, you sho

and skip to Build and launch the Editor Metamodel plug-in or follow the steps belo

Extending the Ecore metamodel — Starting from scratch

Right-click the project and from the context menu, select **New > Other > Example Model**. (On Eclipse V3.5+ [Galileo, Helios], the selection is **New > Other > Eclipse**

Choose the `model` folder and the name `EcoreX.ecore`.

By default, we call the model package `ecorex`. Right-click in the model window an **Registered Packages**. Choose the Ecore Model with namespace `http://www.ecli`

Once you have imported the Ecore metamodel, you are ready to extend it. To recr right-clicking over the package element `ecorex` and select **New Child EClass**. Call element descriptions, above). You then need to add the base element `EPackage` as

Use the same procedure for creating new element `EClassX` by designating `EClass` other `EClasses` in the EcoreX model by subclassing Ecore objects when necessary know which attributes to create for which `EClass`.

Build and launch the Editor Metam plug-in

In the build step, we will create the metamodel ge and build the model and model and select **New > Other > Eclipse Modelin ework > EMF Model**. (For is **New > Other > Eclipse Modeling Framework > EMF Generator Model**.) You ma name `EcoreX.genmodel`. The EcoreX model should be pre-selected as a base mod the `EcoreX.ecore` metamodel.

Figure 3. New EMF model

Contents

When asked to specify which packages to generate and which to reference from o
EcoreX package under Root packages and Ecore u eferenced generator moc

At this point, the wizard will create a `genmodel` for tamodel. You can now au
selecting **Generate All** from the context menu afte ighting the top-level elen
four Eclipse projects, according to the behavior co d in the `genmodel`. This a
project, so you may wish not to generate that plug

Now we move on to the launch step. In most Eclipse tutorials, you are asked to la
separate Eclipse process. In this section, we will take a different approach: We wi
Eclipse and workspace. This makes it easier to integrate the pre-built metamodel
the next section. To do this:

Double-click the EMFX plugin.xml to open the plug-in configuration editor.

Click the **Export Wizard** under the **Exporting** tab.

Select the principal modeling plug-in and the two editor plug-ins.

Figure 4. Export

Contents

Image shows Export method

When you click **Finish**, the generated plug-in JARs ilt and copied to the plug
you need to restart Eclipse to activate the new plu Now we are ready to laun
project to hold our concrete model (ours will be na est2).

Within this new project, navigate to **New > Other Example EMF Model Creation W**
model name. *Note: In recent versions of EMF (V2.5+), the file extension of the conc*
.ecorex; otherwise, the concrete `genmodel` *cannot be created in a later step.*. Select
empty concrete model. The following sections explain how to build the programm
My.ecore can be found in Related topics.

Model the concrete test model

timestamp — useful for debugging messages, for example. The following is a repr

Listing 1. printTimestampMessage

```
1  void printTimestampMessage(String message) {
2
3       System.out.print(message);
4       System.out.print("; Timestamp= ");
5       System.out.println(System.currentTimeMillis());
6  }
```

The second example takes three date-based parameters and returns a numerical upon which that date falls.

Listing 2. getDayOfWeek

```
1  int getDayOfWeek(int year, int month, int date) {
2
3       int result;
4       Calendar calendar = Calendar.getInstance();
5       calendar.set(year, month, date);
6       result = calendar.get(Calendar.DAY_OF_WEEK);
7       return result;
8  }
```

The first step is to fill in the three required attribut er the new EPackageX ele last section. If you don't see the **Properties** tab be e modeling window, you the context menu. In this example, our package is mypackage.

Figure 5. EPackageX properties

Contents

Next, add a new `EClassX` to mypackage. You can d ısing the context menu w
name attribute to give the class a name (e.g., MyC dd two `EOperationImpl` e
the method names `printTimeStampMessage` and `ge :Week`, above. Then, for ea

Figure 6. `EOperationImpl getDayOfWeek()`

Contents

Figure 7. getDayOfWeek() properties

Image shows getDayOfWeek() properties

Above, the operation `printTimestampMessage()` takes one parameter of type `EStr` parameters of type `EInt`. In addition, operation `getDayOfWeek` returns an `EInt`, cor (see Figure 7).

Anatomy of an `EOperationImpl`

Until this point, we have only dealt with inherited Ecore elements and attributes. N

LocalVariable

Looking at Figure 8, the `printTimestampMessage()` will require two `LocalVariable` of type `ELong`.

Figure 8. `printTimestampMessage()`

Image shows printTimestampMessage()

Figure 9. `LiteralAssignment`

Image shows LiteralAssignment

In Figure 9, the string for attribute `Value` is in-lined into the `LiteralAssignment`. Y which literal values (constants) are modeled as separate elements.

Next, we insert an element of type `LiteralAssignment`, which allows us to choose

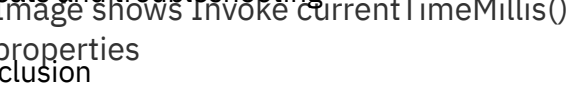
DataType

Again looking at the figure above, notice that there is an Ecore `DataType` called `Sys` java.lang.System. This must be added to our mypackage package because it will b follow.

Statement

The first `Statement` added to this operation is an `Invoke` of static method `currentT` above.

Figure 10. Invoke `currentTimeMillis()` properties



According to our metamodel (once we provide the emplates in the next sec the Java statement: `timestamp = java.lang.System.currentTimeMillis();`.

The next `Invoke` is subtly different from the previous. First of all, there is no `Assig` `message` parameter as an argument for the property called `Args`.

Figure 11. Invoke `out.print` properties

Contents

Also notice that the value of the `Access Name` property is `out.print`— we are chea
referencing field `out` from Java `System`, then invoking method `print`. As it stands,
`FieldReferenceAssignment` together with a `LocalVariable` of type `PrintStream`.

The third and final `Invoke` in the operation is a `println()` using the `LocalVariable`
completes the model of concrete operation `printTimestampMessage()`.

Let's look at the complete model of the second `EOperationImplgetDayOfWeek()`.

Figure 12. `getDayOfWeek()`

Image shows getDayOfWeek()

operation.

`LocalVariables`

Among the three `LocalVariables` in the operation model, we are particularly conc
will hold the value to be returned after the last statement of the operation. Among
called `ReturnParameter`, and in our implementation, we use the pull-down menu to sele

`Statements`

In Figure 12, following the three `LocalVariables` are three `Statements`. The first is
element `CalendarType`, which assigns a value to variable `calendar`, analogous to v

Next, there is an `Invoke` of method `set()` on variable `calendar`, this time passing t
`EOperationImpl` parameters: `year`, `month`, and `date`.

Figure 13. `set()` with parameters

Image shows set() with parameters

Figure 14. `FieldReferenceAssignment`

Contents

According to our metamodel, this element will yield Java code similar to `DAY = Ca`

In Figure 15, the `DAY` variable is used in the last `Invoke` of this `EOperationImpl`: a g variable `result` (the `Return Ref` of our implementation).

Figure 15. `Return Ref`

Image shows Return Ref

Implementing dynamic templates

We have now designed an extended metamodel and used it to describe a concret

By default, EMF does not use dynamic templates when generating code for model started customizing JET templates, we need to copy some files from a plug-in JAR org.eclipse.emf.codegen.ecore_2.3.0.XYZ.jar, where *XYZ* is the time stamp of you This article uses org.eclipse.emf.codegen.ecore_2.3.0.v200706262000.jar. To co utility and:

Extract the templates directory from this JAR into the Java project for your co

Create a directory called **Class** in templates/model.

Create new empty file in the **Class** folder called implementedGenOperation.TC

As the name suggests, the new file in step three is the JET template where we wil objects `EOperationImpl`. By default, this file does not exist because EMF simply pr each `EOperation`. Once we activate the dynamic templates facility, our new file wi the Java method as defined by an `EOperationImpl`.

The following is the complete code of `implementedGenOperation.TODO.override.`

Listing 3. `implementedGenOperation`

```
1  // created by implementedGenOperation.      )verride.javajetinc
2  <%if ( ! (genOperation.getEcoreOperati      .nstanceof EOperationIn
3      // TODO: implement this method
4      // Ensure that you remove @generat      mark it @generated NO
5          thrownew UnsupportedOperationE      .on();
6  <% } else { %>
7      // ** EOperationX implementation *...
8  <% EOperationImpl opx = (EOperationImpl)genOperation.getEcoreOper
9  Statement stm = null;
10 Iterator iterator = null;
11
12 EList<LocalVariable> pList = opx.getLocalVariables();
13 LocalVariable lvar = null;
14
15 String iname = null;
16 StringBuffer paramsString = null;
17 StringBuffer varString = null;
18
19 for (int i = 0;i < pList.size(); i++) {
20     lvar = pList.get(i);
```

```
24  iterator = opx.getStatements().iterator();
25
26  while (iterator.hasNext()) {
27
28      paramsString = new StringBuffer();
29      varString = new StringBuffer();
30      iname = null;
31
32      stm = (Statement)iterator.next();
33      if (stm instanceof LiteralAssignment) {%><%= stm.getAssignmen
34  ((LiteralAssignment)stm).getValue()%>;
35      <%} else
36      //
37      if (stm instanceof FieldReferenceAssignment) {
38          Access ax = (Access)stm;
39          if (stm.getAssignment() != null) {
40          varString.append(stm.getAssignment().getName());
41          varString.append(" = ");
42          }
43              if ( ax.getStaticType() != null) {
44              // STATIC
45              iname = ax.getStaticType().getInstanceClassName();
46              } else {
47              // NON STATIC
48              iname = ax.getTarget().getName();
49              } %><%=varString.toString()%><%=iname%>.<%=ax.getAcce
50          <% } elseif (stm instanceof Invoke) {
51          // INVOKE
52          Invoke iv = ((Invoke)stm);
53
54          if (stm.getAssignment() != nul
55              varString.append(stm.getAs      nt().getName());
56              varString.append(" = ");
57          }
58
59          for (int p = 0; p < iv.getArgs      e(); p++) {\
60  paramsString.append(iv.getArgs().get(p      lame());
61              if ( p + 1 < iv.getArgs().        ) {
62              paramsString.append(" , ");
63              }
64          }
65                  if (iv.getStaticType() != null) {
66                  // STATIC
67                  iname = iv.getStaticType().getInstanceClassName()
68                  } else {
69              // NON STATIC
70                  iname = iv.getTarget().getName();
71                  } %><%=varString.toString()%><%=iname%>.<%=iv.get
72  %>(<%=paramsString.toString()%>);
73      <% }
74
75  } // STATEMENTS
```

```
79 | } // EOPERATIONIMPL %>
```

Contents

The specifics of JET are beyond the scope of this article. However, since the JET t
we will review the content of the template in terms of pseudo-code, keeping in mi
pre-initialized by Ecore/JET before the template is processed.

Listing 4. genOperation is pre-initialized by Ecore/JET

```
1  Is this genOperation is an EOperationImpl?If false, emit default
2       STOP;
3
4  Else, cast it to EOperationImpl;
5       continue;
6
7  Find and declare all elements of type LocalVariable, initializing
8
9  Iterate through all Statements;
10      Emit java code according to the subtype;
11
12 Does the implementation return something?
13      If yes, emit the return statement;
```

There are a few actions to perform before building ncrete model. First, at th
we must add to the list of imports (the first two m n bold):

```
1 | <%@jetpackage="org.eclipse.emf.codegen.      templates.model"
2 | imports="ecorex.*  org.eclipse.emf.comm      l.* \
3 | java.util.* org.eclipse.emf.codegen.ecc      model.*"…
```

The package EcoreX is, of course, the extended metamodel. Next, we need to crea
our concrete model (`My.ecore`, of type '.ecorex'). To do this, right-click on the mod
Modeling Framework > EMF Model. (With EMF V2.5+ [Galileo, Helios], the selecti
Framework > EMF Generator Model.) Once created, three properties need to be
Templates & Merge, and the fourth under **Model**.

Figure 16. `GenModel`— Templates & Merge

Contents

Set **Dynamic Templates** to true.

Specify the **Template Directory**.

Add EMFX (extended metamodel plug-in ID) tc **late Plug-in Variables**.

Recent versions: under the Model group prope **Suppress Interfaces** to

At this point, you're ready to build by right-clicking on the `GenModel` and choosing the source folder (src) of your concrete Test project (ours is called Test2), you sho packages and classes, including one called mypackage.impl.MyClassImpl.java. O generated methods.

Listing 5. MyClassImpl.java

```
1 publicvoid printTimestampMessage(String message) {
2     // created by implementedGenOperation.TODO.override.javajetin
```

```
6       timestampStr = "; Timestamp = ";
7
8       timestamp = java.lang.System.currentTimeMillis();
9
10      java.lang.System.out.print(message);
11
12      java.lang.System.out.print(timestampStr);
13
14      java.lang.System.out.println(timestamp);
15
16      }
17    publicint getDayOfWeek(int year, int month, int date) {
18      // created by implementedGenOperation.TODO.override.javajetin
19      // ** EOperationX implementation **
20      int result;
21      int DAY;
22      java.util.Calendar calendar = null;
23      calendar= java.util.Calendar.getInstance();
24
25      calendar.set(year , month , date);
26
27      DAY = java.util.Calendar.DAY_OF_WEEK;
28
29      result = calendar.get(DAY);
30
31      return result;
32
33      }
```

You can test this class by adding a main method.

Caveats and troubleshoo ;

Ecore file naming (EMF V2.5+)

Prior to EMF V2.5, as shown in several screenshots above, a concrete model prod
keep the file extension '.ecorex' (as proposed by the wizard at creation time). This
from a 'first level' Ecore model. However, in recent versions of EMF, the `genmodel`
will not accept file extensions other than .ecore.

To obtain color-coded syntax highlighting of JET templates, you need to have the Plugin has recently moved from EMF to M2T).

However, at the time of this writing, the most recent version of the JET Editor doe or on-the-fly compilation for nested JET includes, such as .javajetinc files. Additio parent file (e.g., Class.javajet, above) and not in the included file in order for the b

You can, in fact, turn an EMF dynamic templates project (Test2 in our example) in configuration (i.e., using the context menu for the project). In practice, the above- lack of integration between EMF and M2T/JET, makes this impractical today.

As a result, it can be difficult to catch and correct errors in included template files into an intermediate Java file (located by default in a hidden Java project called JI you can see these compilation errors by removing the filter from the Package Expl formatting error in the template file, you will get an Eclipse pop-up window during more evolved functionality in a future release of JET.

No model validation

The examples in this article do not use the EMF Va n Framework or the OCL model will cause build failures. For example, an `EC` `onImpl` may declare a ce property may refer to a different type or be null. Th rors would not be caught code would not compile. A more evolved metamo ild enforce integrity and

Conclusion

We have seen how to extend the Ecore metamodel to conceptualize simple progra method. We extended several Ecore model elements — most notably `EOperation`- our metamodel, and then used the editor to design a concrete test model, includi an `EOperationImpl`. We configured and built JET templates for generating code fo

PDF of this content

Contents

Sample templates (os-eclipse-emfmetamodel.zip | 3KB)

The Eclipse Foundation article "Using EMF" introduces the Eclipse Modelling F

"Modeling Rule-Based Systems with EMF" defines a metamodel in ECore for n

The Eclipse Foundation article "Implementing Model Integrity in EMF with MD generated from an Ecore specification without requiring any post-generation c

The Eclipse Modeling Framework (EMF) is home to all EMF-related documenta

The Eclipse Foundation Model To Text (M2T) project focuses on the generatior including JET.

The Eclipse Modeling Framework Technology (EMFT) Project is a Eclipse proje technologies.

Read the developerWorks article "Build metamodels with dynamic EMF" to lea models on demand without generating Java in ntation classes.

Download Model to Text (M2T), an Eclipse Fou n project that focuses on models.

Check out the latest Eclipse technology downl : IBM alphaWorks.

Check out the "Recommended Eclipse reading

Download Eclipse Platform and other projects from the Eclipse Foundation.

Browse all the Eclipse content on developerWorks.

Follow developerWorks on Twitter.

New to Eclipse? Read the developerWorks article "Get started with the Eclipse architecture, and how to extend Eclipse with plug-ins.

Visit developerWorks Open for extensive how-to information, tools, and projec source technologies and use them with IBM's products.

Contents

Comments

Code Patterns

Articles

Tutorials

Recipes

Open Source Projects

Videos

Newsletters

Events

Cities

Developer Answers