Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Model Driven Software Development course

Introduction to the Eclipse Modeling Framework

Oszkár Semeráth Gábor Szárnyas

August 11, 2014

Contents

Chapter 1

Introduction to the Eclipse Modeling Framework

Author: Oszkár Semeráth

1.1 About the EMF

From Wikipedia (http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework) „Eclipse Modeling Framework (EMF) is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model." EMF's data model is lightweight as it only defines a few but well-defined modeling elements. However, it has an extensive tooling support and community. For example, you can define the textual or graphical syntax of a language and generate the appropriate editors.

EMF can generate Java code from the model with only a click of a button. The generated code is capable of serialisation to XMI and deserialisation from XMI files.

EMF home page: http://www.eclipse.org/modeling/emf/

1.2 Description of the task

The goal of this exercise is to create the metamodel of customized Entity-Relationship Diagrams (ERD). Those diagrams can aid the development of software components that working with complex data structures. A later exercise will show you how complete database schemes, full classes and the automated mapping between those can be derived from those documents.

The following image presents an example of Entity Relation diagram.

1.3 Prerequisites

The Eclipse Modeling Tools edition contains every required plug-in.

1.4 Ecore model: step-by-step

1. Create a new **Empty EMF Project** by **File | New | Other... | Eclipse Modeling Framework | Empty EMF Project**. Name it to `hu.bme.mit.mdsd.erdiagram`.

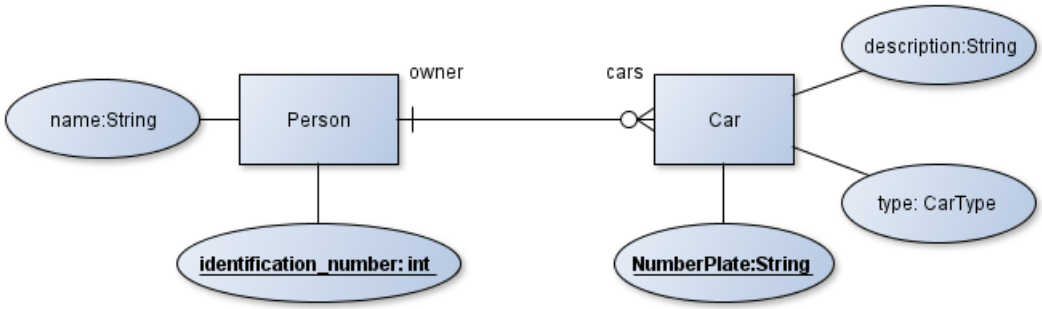


Figure 1.1: Example of an Entity-Relationship diagram

2. There is a folder in the project named **model**. Create a new **ECore Model** in it by right click to the folder | New | Other... | ECore Model. Name it to `ERDiagram.ecore`.

3. A new editor opens that shows that the model resource has a yet unnamed empty package. Fill the missing properties in the property view:

   - Name: `ERDiagram`
   - Ns Prefix: `hu.bme.mit.mdsd.erdiagram`
   - Ns URI: `hu.bme.mit.mdsd.erdiagram`

   To show unavailable view go to **Window | Show View | Other... | General**.

4. It is possible to create the model in this tree editor but there is a more convenient editor for this purpose. Right click to the ecore file and choose the **Initialize ECore diagram File...** option. Name it to **ERDiagram.ecorediag**.

5. Let's make the following part by dropping metamodel elements from the palette to the diagram:

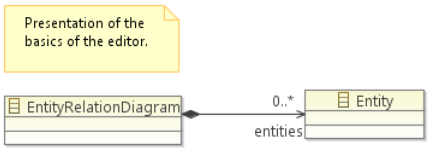


Figure 1.2: A very simple Ecore model with two `EClasses` and an `EReference` between them

6. If you click on a model element you can edit its properties in the **Property view**.

   - Specify the names of the `EClasses` and the `EReference`.
   - The `EClasses` can be set to **Abstract** or **Interface** int this view.
   - The multiplicity of the relation is set to `0..*`
   - The `EOpposite` feature should be presented.
   - The objects of the instance models of the metamodel have to be in a tree hieracy with respect of the containment references. Set the `entities` relation to **Is Containment**.
   - **Appearance:** you can edit the view of the diagram.
   - **Advanced Options:** Direct editing for the properties of the elements of the model. The features should be presented:

7. The effect of the diagram editing on the `.ecore` file can be observed in the **Outline** view. The next figure shows the actual state of the metamodel in this view.

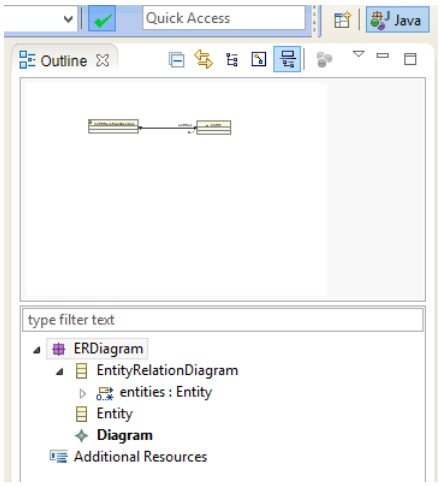


Figure 1.3: Outline view and the diagram validation button

8. The editor can validate the model with the check symbol visible in the upper part of the following figure
9. Note that deleting from model and the diagram are different things.
10. Create the metamodel of the Entity Relation Diagram on your own like it was a class diagram. A possible result is visible on Figure 1.4.

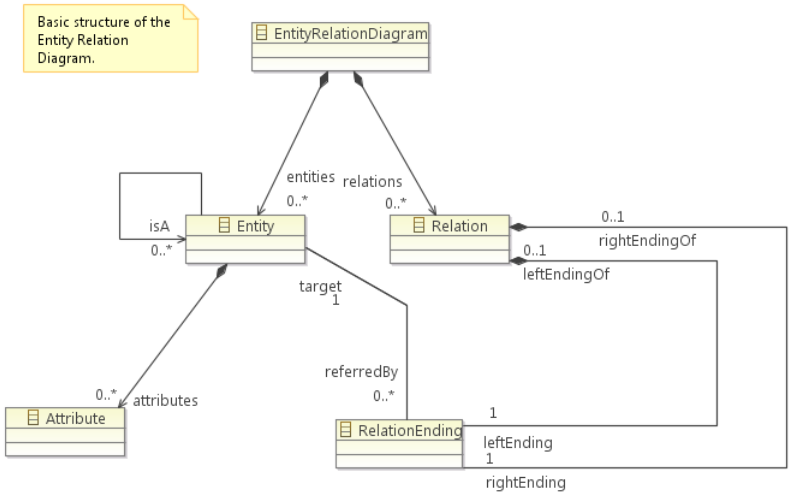
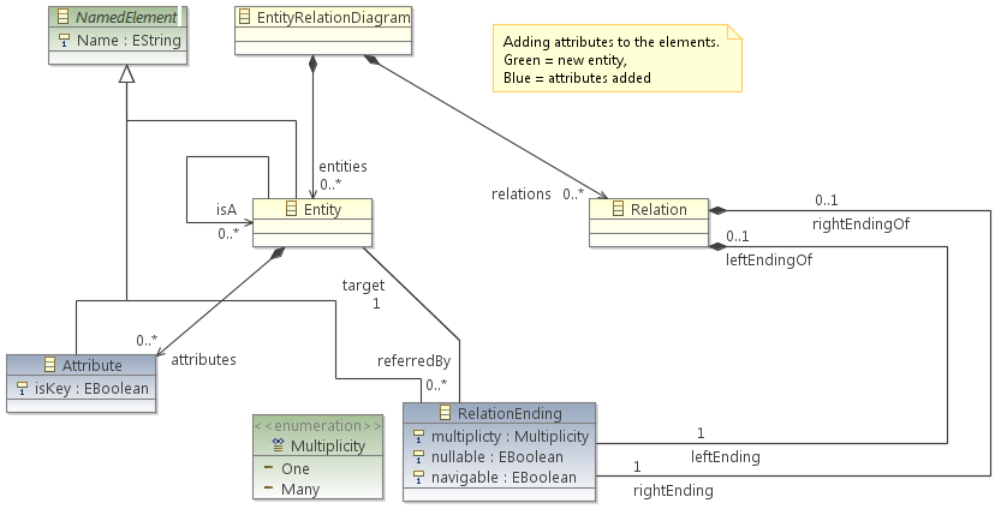


Figure 1.4: The metamodel of the ER diagram

11. Add the `EEnum` named `Multiplicity` to the metamodel, and add two literals to it: `One` and `Many`.
12. Add an abstract `NamedElement` class to the metamodel and add inheritance relations to the Fill the The difference between the `EAttribute` and `EReference` is that the EAttribute is referring to an `EDataTypes` opposed to `EReferences` that endings to `EClasses`. At this phase we have all visible details of the Entity Relation



Diagrams.

13. The metamodel lacks of `EOperations`, because it is basically a data model.

14. Adding namespace to the diagram, this could be the name of the diagram. The types of the attributes should be defined outside of the model, and referred by the diagram.

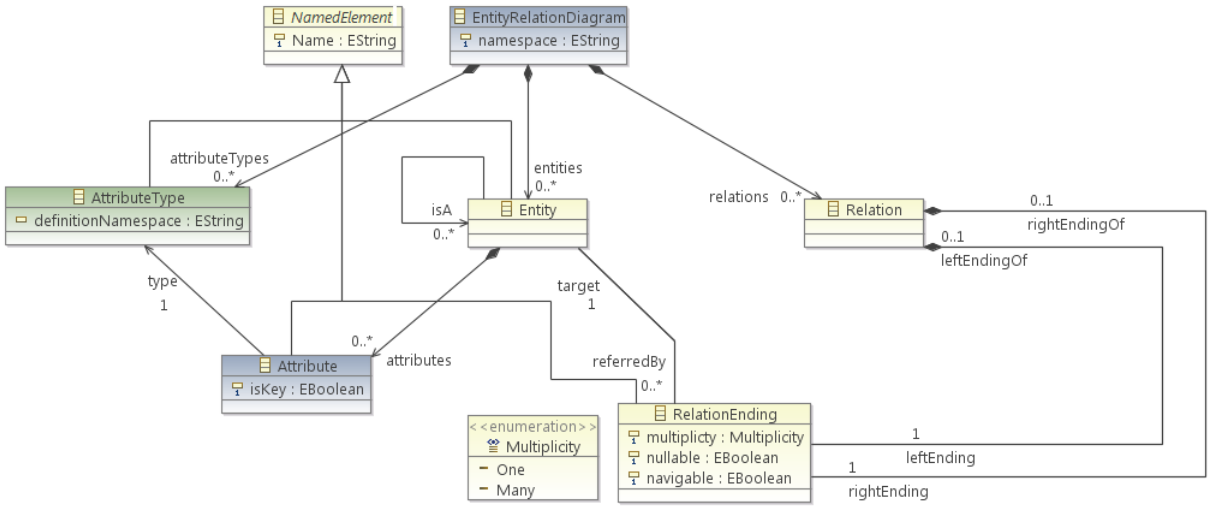


Figure 1.5: The model with the `AttributeType` class and the `namespace` attribute

15. The diagram may refer to an existing database with existing tables. The referred table and column names might be described in the diagram (for example the `User` entity is stored in the `USER_TABLE` because its name is reserved). Note the multiple inheritance at the `Attribute` entity.

1.5 Editor: step-by-step

This example shows how to generate classes and an editor from Ecore models.

1. The ecore files are the blueprints of the domain specific languages. To use the tooling support available in Eclipse some kind of Java class representation of those “boxes” are needed. Fortunately those classes can be automatically generated.

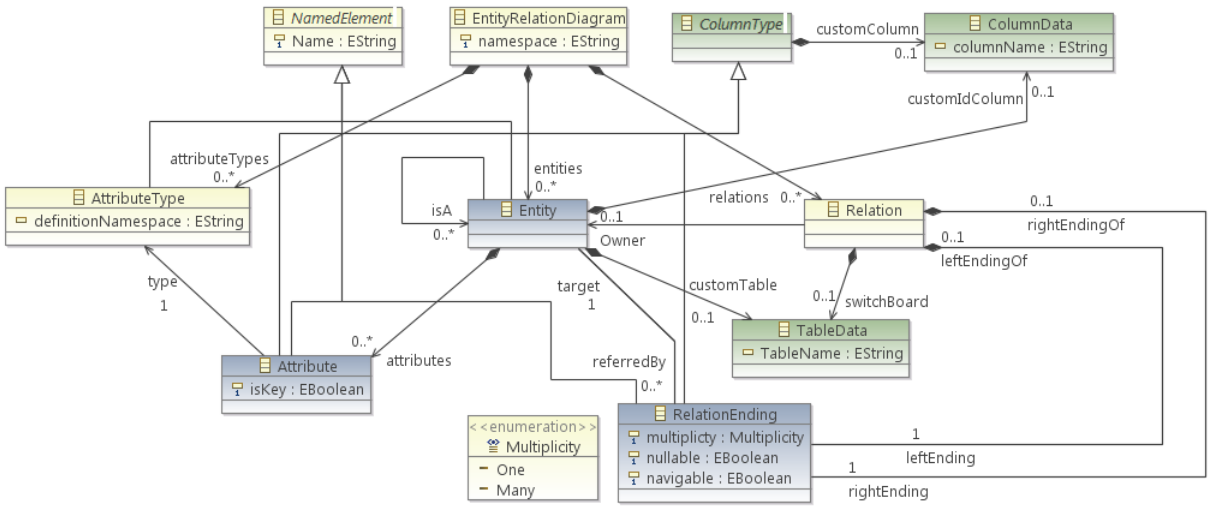


Figure 1.6: The model now supports the connection to databases

Right click the ecore file and **New | Other | Eclipse Modeling Framework | EMF Generator Model**. The default `ERDiagram.genmodel` is fine. At the next step choose that the generator generate from an Ecore model. In the third step the URI of the Ecore model have to be added. Click on load and next. Choose the only avaliable package to generate and hit finish.

2. Another tree editor opens similar to the ecore editor. Browse some of the setting in the property editor. Right click to the root, and choose the **Generate Model** command. Three package has been generated in the source folder. Browse for example the `hu.bme.mit.mdsd.erdiagram/src/ERDiagram/EntityRelationDiagram.java` file, and you can see that nothing strange has been generated. The implementation class has some unusual field, but the implementations of the functions of the interface are quite simple.

3. Generate an **editor**. Right click to the root of the genmodel file, and generate edit and editor in this order.

4. Right click to the project, and choose **Run as | Eclipse** application.

5. Create an empty project by **File | New | Other... | General | Project** and name it to **Diagrams**.

6. Create a new Entity Relation Diagram into the new project by right clicking on it and picking **New | Other | Example EMF Model Creation Wizard | ERDiagram Model**. The name can be the default `My.erdiagram`, and the model object (what we want to edit) should be **Entity Relation Diagram**.

7. Create the instance model. The editor is quite self-explanatory to use.

1.6 Model manipulation: step-by-step

The following example shows how to edit the model from code.

1. Create a new **Plug-in Project** by right click | New | Plug-in Project. Name it to `hu.bme.mit.mdsd.erdiagrammanipulator`.

2. Add the following dependencies:

`hu.bme.mit.mdsd.erdiagram`	The edited domain.
`org.eclipse.emf.ecore.xmi`	The instance model is serialised as an XMI document.

3. Create a class to the source folder:

package	`hu.bme.mit.jpadatamodelmanipulator`
name	`ModelEditor`

4. Create an initialisation method for model loading.

```
public void init() {
   // For the initialisation of the model.
   // Without this the following error happens:
   //  "Package with uri 'hu.bme.mit.mdsd.erdiagram' not found."
   ERDiagramPackage.eINSTANCE.eClass();

   // Defining that the files with the .erdiagram extension should be parsed as an xmi.
   Resource.Factory.Registry reg = Resource.Factory.Registry.INSTANCE;
   reg.getExtensionToFactoryMap().put("erdiagram", new XMIResourceFactoryImpl());
}
```

5. The model is in an xmi file that can be generally handled as a resource. A resource can be referenced by an URI. Write a method that loads a resource:

```
public Resource getResourceFromURI(URI uri) {
   ResourceSet resSet = new ResourceSetImpl();
   Resource resource = resSet.getResource(uri, true);
   return resource;
}
```

6. The resource simply can be saved:

```
public void saveResource(Resource resource) {
   try {
     resource.save(Collections.EMPTY_MAP);
   } catch (IOException e) {
      System.out.println("The following error occured during saving the resource: "
        + e.getMessage());
   }
}
```

7. The content of the resource should be the ED diagram object.

```
public EntityRelationDiagram getModelFromResource(Resource resource) {
   // check the content!
   EntityRelationDiagram root = (EntityRelationDiagram) resource.getContents().get(0);
   return root;
}
```

8. The ER diagram object should be edited through the interface and instantinated by the generated factory methods. This method creates a custom table data object for every entity that doesn't already have one:

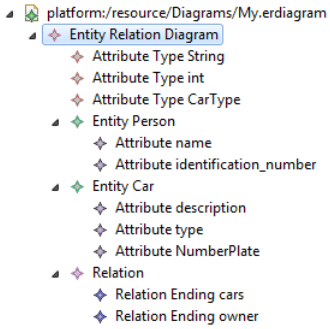


Figure 1.7: The tree editor for the Ecore model

```
public void editDiagram(EntityRelationDiagram diagram) {
   for(Entity entity : diagram.getEntities()) {
      if(entity.getCustomTable() == null) {
         TableData tableData = ERDiagramFactory.eINSTANCE.createTableData();
         tableData.setTableName(entity.getName().toUpperCase()+"_TABLE");
         entity.setCustomTable(tableData);
      }
   }
}
```

The result can be printed to the output by this method:

```
public void printTables(EntityRelationDiagram diagram) {
   for(Entity entity : diagram.getEntities()) {
      System.out.println(entity.getCustomTable().getTableName());
   }
}
```

9. You can get the URI by right click | **Properties** and copy the file to a string. For example my URI is:

   ```
   URI uri = URI.createFileURI("C:/workspace/Diagrams/My.erdiagram");
   ```

   The main method looks like:

   ```
   public static void main(String[] args) {
      ModelEditor editor = new ModelEditor();
      editor.init();
      URI uri = URI.createFileURI("C:/workspace/Diagrams/My.erdiagram");
      Resource resource = editor.getResourceFromURI(uri);
      EntityRelationDiagram diagram = editor.getModelFromResource(resource);
      editor.editDiagram(diagram);
      editor.printTables(diagram);
      editor.saveResource(resource);
   }
   ```

   Right click to the class and choose **Run as | Java Application**. This will run our code as a simple Java application that loads modifes and saves a model.

1.7 Summary

At the end the following steps have been made:

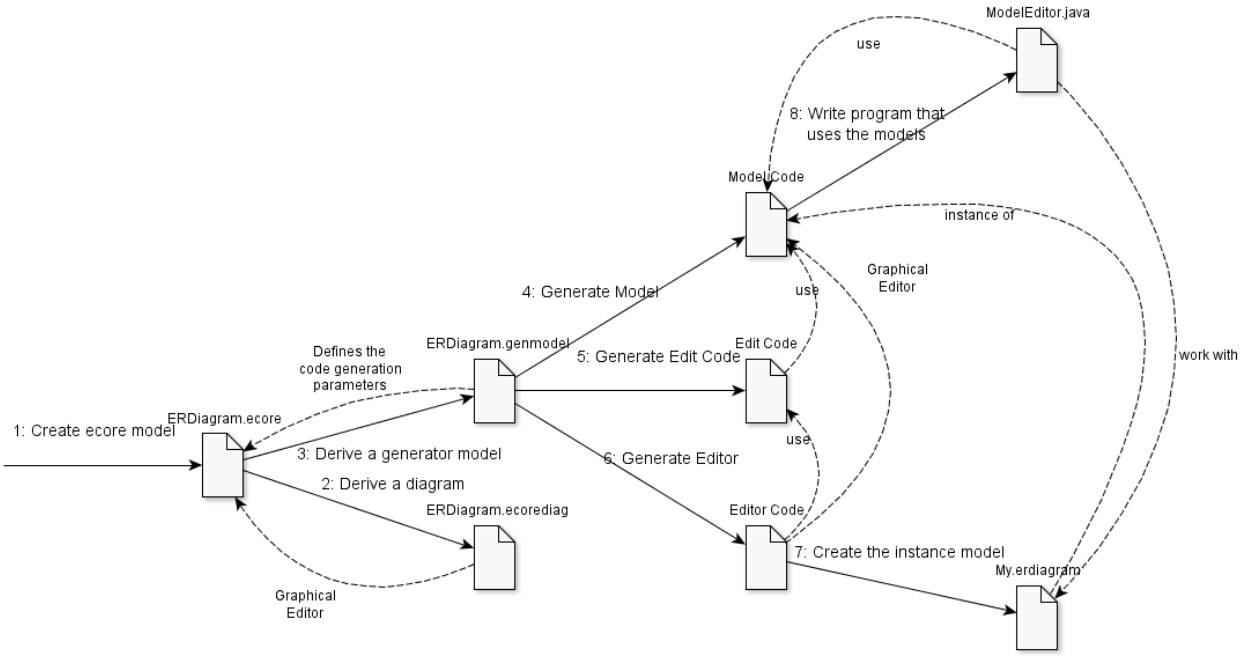


Figure 1.8: The workflow of the laboratory and the dependencies between the artifacts (marked with dashed lines)

The final metamodel is:

1.8 General tips

```
* If anything goes wrong with the regeneration and there is problem with your code you have two options
* If the document was not edited by hand or it isn't valuable delete it. Generate the code again, and it
* In other case don't be afraid of rewriting. For example if you delete an item from the metamodel the X
```

1.9 References

- Tutorial: http://eclipsesource.com/blogs/tutorials/emf-tutorial/

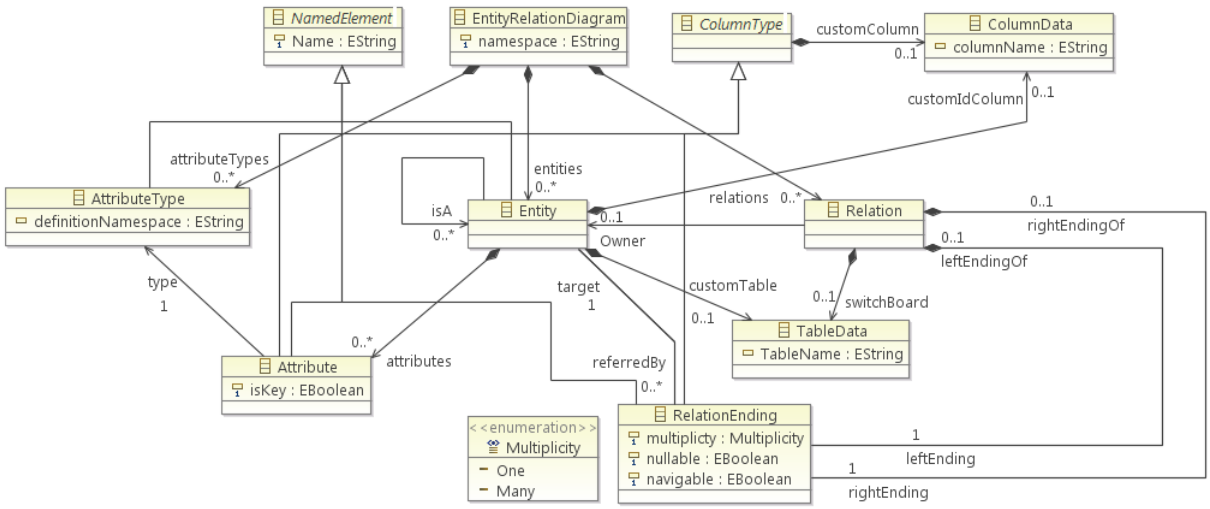


Figure 1.9: The final Ecore model