EMF/GEF/GMF

Andrey Pines
University of Paderborn
andreyp@mail.upb.de

December, 23 2013

Abstract

The Graphical Modeling Framework or GMF is a framework that allows easy and fast development of graphical editors in Eclipse. Since we decided to use GMF in the Soundgates project, I describe in this document the concepts and the frameworks GMF is based on (Eclipse Modeling Framework and Graphical Editing Framework), its benefits and disadvantages.

1 Introduction

One task of our project group is to develop a graphical design environment similar to Cycling 74 MAX or Pure Data where the user can build up complex sound generator networks consisting of basic sound components to emulate analog and digital synthesizers. We decided to use the Graphical Modeling Framework (GMF)[1] to implement this task. GMF is a framework for developing graphical editors in Eclipse[2]. We made this decision based on the following facts:

1. The creation of a component-based editor with GMF is easy and can be done rapidly
2. The models that can be built with such an editor can be used to generate every kind of code or text

In this paper I explain the concepts and the tools GMF is based on.
Section 2 describes the concept of software modeling. In section 3 I introduce the Eclipse modeling Framework (EMF) that is used to build models. Section 4 describes Graphical Editing Framework (GEF) which is used to implement a graphical editor. In section 5 I explain how GMF is used to generate a GEF editor. Finally, section 6 describes how we use the mentioned frameworks in the Soundgates project.

[1] http://www.eclipse.org/modeling/gmp/
[2] http://www.eclipse.org/

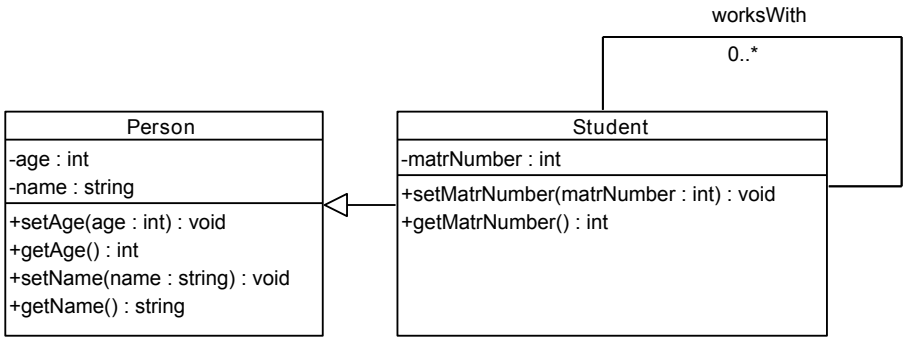

Figure 1: An UML class diagram

2 Modeling

With the growth of computer science the software got larger and more difficult to understand and to maintain. For solving the complexity problem the computer scientists increased the abstraction in the development process: instead of assembler compiled high level languages were used, frameworks and libraries were developed etc.. Modeling is one of the concepts to reduce complexity by abstraction [4].
A model is an abstraction of the modeled entity with a given abstraction aim. Details that are not relevant to the abstraction are left out such that analysis and comprehensibility are simplified. A typical example for models used in computer science are UML diagrams. An UML class represents a Java class but the actual implementation of the methods is left out. The aim of visualizing a Java program as an UML diagram is to get a better overview over the program's structure and relations of its parts.
For each model a meta-model must exist. "A metamodel is a precise definition of the constructs and rules needed for creating semantic models"[3]. A model is an instance of the meta-model, it means that each model element is an instance of the corresponding meta-model element.
Figure 1 shows a simple UML class diagram, which represents a meta-model. The class "Person" has attributes "age" and "name". The class "Student" inherits the class "Person" and has an additional attribute "matrNumber". The class "Student" has an association "worksWith" to itself. Both classes have getters and setters for their attributes. Figure 2 shows an instance of this meta-model. It contains two instances of the class "Student", where all attributes have concrete values. The two instances are connected with each other through the association "worksWith".

[3] http://infogrid.org/trac/wiki/Reference/WhatIsMetaModeling

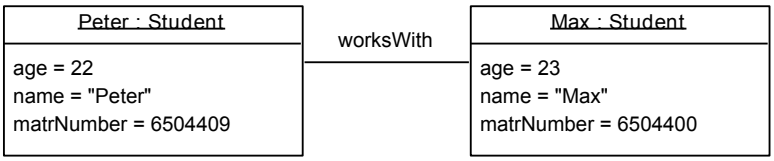

Figure 2: An UML object diagram

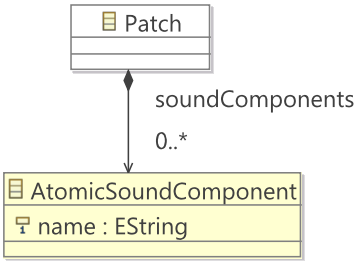

Figure 3: An EMF meta-model example

3 EMF

The Eclipse Modeling Framework (EMF)[4] is a modeling framework that is used to describe and build structured models [3]. This open-source project is developed by the Eclipse Foundation.

3.1 EMF meta-models and models

EMF allows definitions of a meta-models. The developer can define named classes, their attributes, operations and references to other classes. Figure 3 shows an EMF meta-model example. The class "Patch" has a list of instances of the class "AtomicSoundComponent". The class "AtomicSoundComponent" has an attribute "name" of type EString. From this meta-model Java code can be generated. Among others the developer can generate model code and editor code. The model code contains the interfaces of the classes and the implementations of the interfaces. The editor code implements a tree editor where the user can create instances of the meta-model, set values of their attributes and define relations between them. Figure 4 shows a tree editor for the meta-model described above. The model shown contains a "Patch" instance which has two "AtomicSoundComponent" instances: the first one has the name "SineGenerator", the second one has the name "Mixer". For serialization of an EMF model the XML Metadata Interchange (XMI) format is used [3]. This format is standardized by the Object Management Group (OMG)[5]. The text version of the model file has the following content:

[4]http://www.eclipse.org/modeling/emf/
[5]http://www.omg.org/

Figure 4: Tree editor

```
<?xml version="1.0" encoding="UTF-8"?>

<soundgates:Patch xmi:version="2.0"
 xmlns:xmi="http://www.omg.org/XMI"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:soundgates="soundgates">

  <soundComponents xsi:type="soundgates:AtomicSoundComponent"
    name="SineGenerator"/>
  <soundComponents xsi:type="soundgates:AtomicSoundComponent"
    name="Mixer"/>
</soundgates:Patch>
```

As one can see, this file has a XML structure. The first line is the header. The following tag represents the root element "Patch". It has some meta informations like the XMI version etc.. The other tags are enclosed by the "Patch" tag. There are two "soundComponents" tags. Both tags have a reference to the meta-model class they represent, in this case it is "AtomicSoundComponent", as well as the value of the attribute "name".

3.2 Code generation

Once you have a model or an instance of your meta-model, you want to generate code or other text out of it. It is possible to program a code generator "by hand", for example in Java. But there are also special approaches to do this. The most widely-used ones are template engines. In such a template static text is mixed with variable parts which are filled according to a model instance. The popular engines are "openArchitectureWare"[6], "JET"[7], "AndroMDA"[8] etc.. As an example I show how the template engine of "openArchitectureWare" works [2].
Consider the following template for the meta-model from Figure 3 which we apply to the model from Figure 4.

[6]http://www.eclipse.org/workinggroups/oaw/
[7]http://www.eclipse.org/modeling/m2t/?project=jet#jet
[8]http://www.andromda.org/index.html

```
<<DEFINE Root FOR Patch>>
  <<FILE "Patch.txt">>
    # exported Soundgates patch
    <<FOREACH this.soundComponents AS sC ITERATOR it>>
      <<it.counter0>> = <<sC.name>>
    <<ENDFOREACH>>
  <<ENDFILE>>
<<ENDDEFINE>>
```

With DEFINE we start a new template; the root element is Patch. With FILE we define the new file where the output will be written. The line "# exported Soundgates patch" is written to the file. We iterate through the sound components list of the Patch with FOREACH and use an iterator. For each sound component its number and its type are written to the file. The output of the code generator would be:

```
# exported Soundgates patch
0 SineGenerator
1 Mixer
```

4 GEF

The Graphical Editing Framework (GEF)[9] provides basic classes for the implementation of a graphical editor in Eclipse [3]. Such an editor represents an EMF model graphically and allows to modify it. The developer must implement the editor functions like drag&drop, undo/redo etc. within the framework. Figure 5 shows a screenshot of an examplary GEF editor for the model seen in Figure 4. In the diagram there are two rectangles which represent the two "AtomicSoundComponent" instances "SineGenerator" and "Mixer". On the right there is a tool palette where the user can choose the option to drag a new "AtomicSoundComponent" into the diagram. Furthermore the user can modify the property "name" of an "AtomicSoundComponent" instance.
GEF is based on the Model-View-Controller (MVC) pattern [3]. Figure 6 shows the MVC pattern. Model stands for the data, in the context of GEF it is an EMF model. View is a graphical representation of the data; there can be several views for the same data. Controller manages and synchronizes Model and View. In GEF the Controller is implemented through so called EditParts: EditParts are connections between Model and View. An EditPart has a reference to one data element and to one view element, for example in the running example there would be a class named "AtomicSoundComponentEditPart" that has a reference to an "AtomicSoundComponent" and a reference to the corresponding rectangle. The GEF developer must implement all the EditPart classes to implement the Controller part of the MVC pattern.

[9] http://www.eclipse.org/gef/

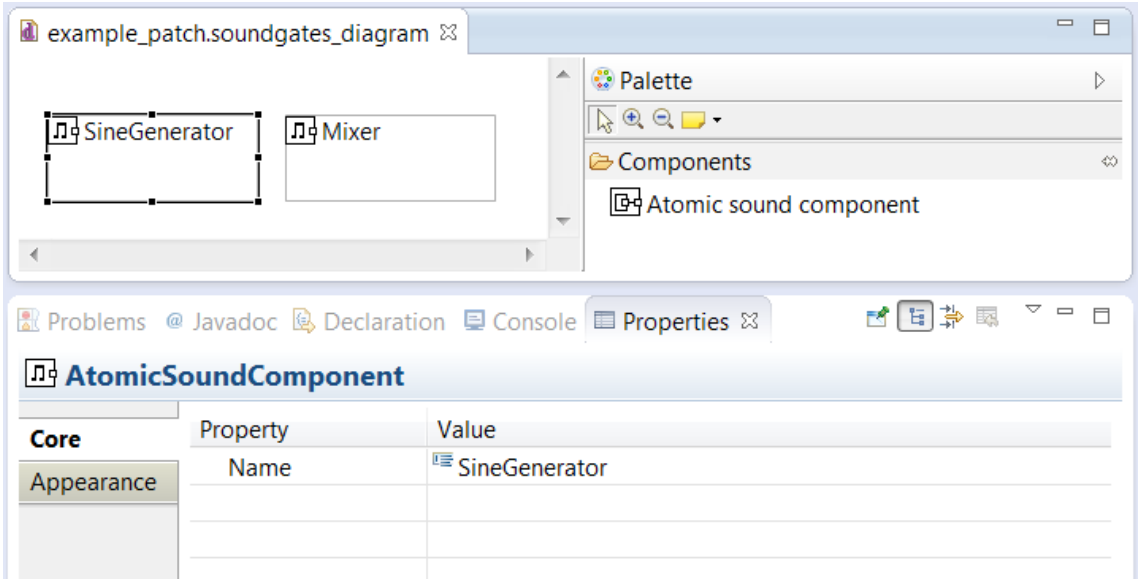

Figure 5: GEF editor

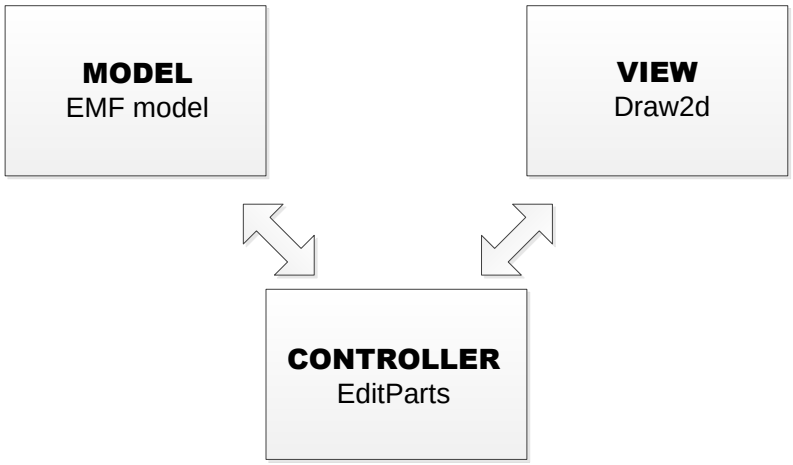

Figure 6: Model-View-Controller

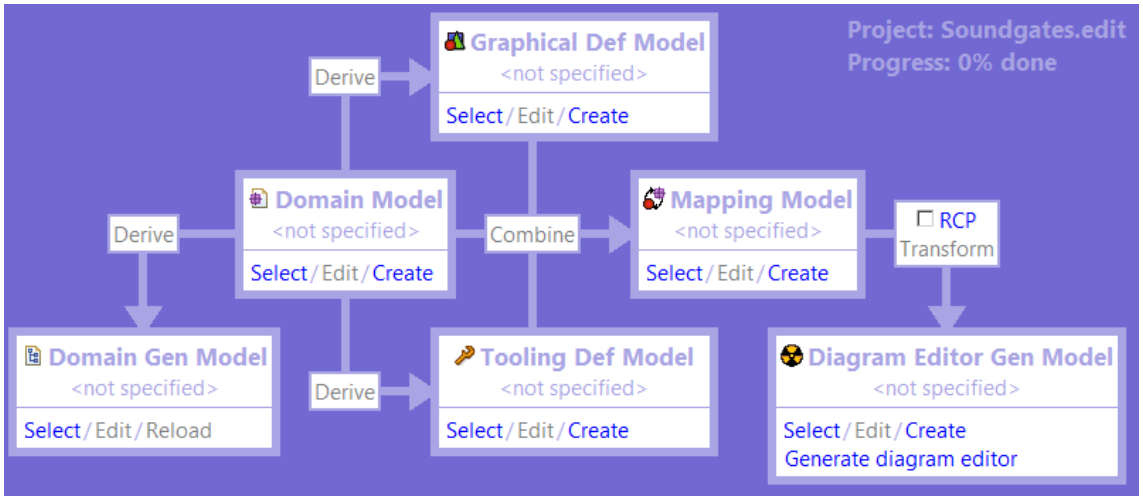

Figure 7: GMF dashboard

All editor actions like creating new components, changing their properties etc. are associated with GEF commands. If the user performs an action, a command is fired which modifies the model. The EditPart that is associated with the modified model element gets a notification. According to the information in the notification about the model change the EditPart updates the view. The GEF developer must implement all GEF commands which are associated with user actions.
As one can see, it is a difficult task to implement a GEF editor by hand. Developers have realized that there are many recurring patterns in such an editor code. That is why an automated approach for editor implementation was developed, which I explain in the next section.

5 GMF

The Graphical Modeling Framework (GMF)[10] is a "meta"-framework for an automatical generation of a GEF editor. Figure 7 shows the GMF dashboard which is a starting point for such a development.

The "Domain Model" is the meta-model on which the editor will be based. The "Domain Gen Model" is derived from the meta-model automatically. The first tasks of the developer are to define the "Graphical Def Model", the "Tooling Def Model" and the "Mapping Model" [1].

- "Graphical Def Model" defines which graphical elements will appear in the diagram and their properties.
- "Tooling Def Model" defines which tools will be available to the user in the tool palette.

[10]http://www.eclipse.org/modeling/gmp/

- "Mapping Model" defines the relationships between the model elements, the graphical elements and the tools.

From these models the "Diagram Editor Gen Model" can be derived. In the "Diagram Editor Gen Model" the developer can set further advanced properties of the editor. From this model the ready-to-use implementation of a GEF editor can be generated.
A disadvantage of GMF is that the generated code is hard to understand and hard to handle if you want to implement your own specific editor features. The reason is that there is no complete documentation about the generated code and the internal processes.

6 Use in the Soundgates project

We used GMF to generate the base of our component-based editor. Later we changed the code manually to get the functionalities which were not generated, e.g. the library of predefined atomic sound components.

6.1 Meta-model

Figure 8 shows the most part of our current implemented meta-model. "Patch" is the root class which contains a list of "Elements". An "Element" can be a "SoundComponent" or a "Link". Each "SoundComponent" can have a number of "Ports". A "Port" has a direction and a data type as attributes. Both classes "SoundComponent" and "Port" inherit the abstract class "NamedElement", which has an attribute "name". That means that all instances of the classes "SoundComponent" and "Port" have the attribute "name". A "SoundComponent" is either an "AtomicSoundComponent" with a concrete type or a "CompositeSoundComponent". A "CompositeSoundComponent" contains embedded "SoundComponents", "Links" and "Delegations". "Links" and "Delegations" are "Connections" which are used to connect "Ports" with each other. A "Link" connects two "Ports" of "SoundComponents" which are on the same level and a "Delegation" connects a "Port" of a "CompositeSoundComponent" with a "Port" of one of its embedded "SoundComponents".

6.2 Example model

Figure 9 shows an examplary model of a Soundgates Patch. "freq_input" is an I/O atomic sound component, which receives user input, for example sensor data. Its port "Value" sends the received value to the port "Frequency" of the component "SineGenerator". "SineGenerator" generates a sine wave depending on the frequency and sends it through the port "Output" to the port "Sound" of the component "SoundOutput", which plays the incoming sound.

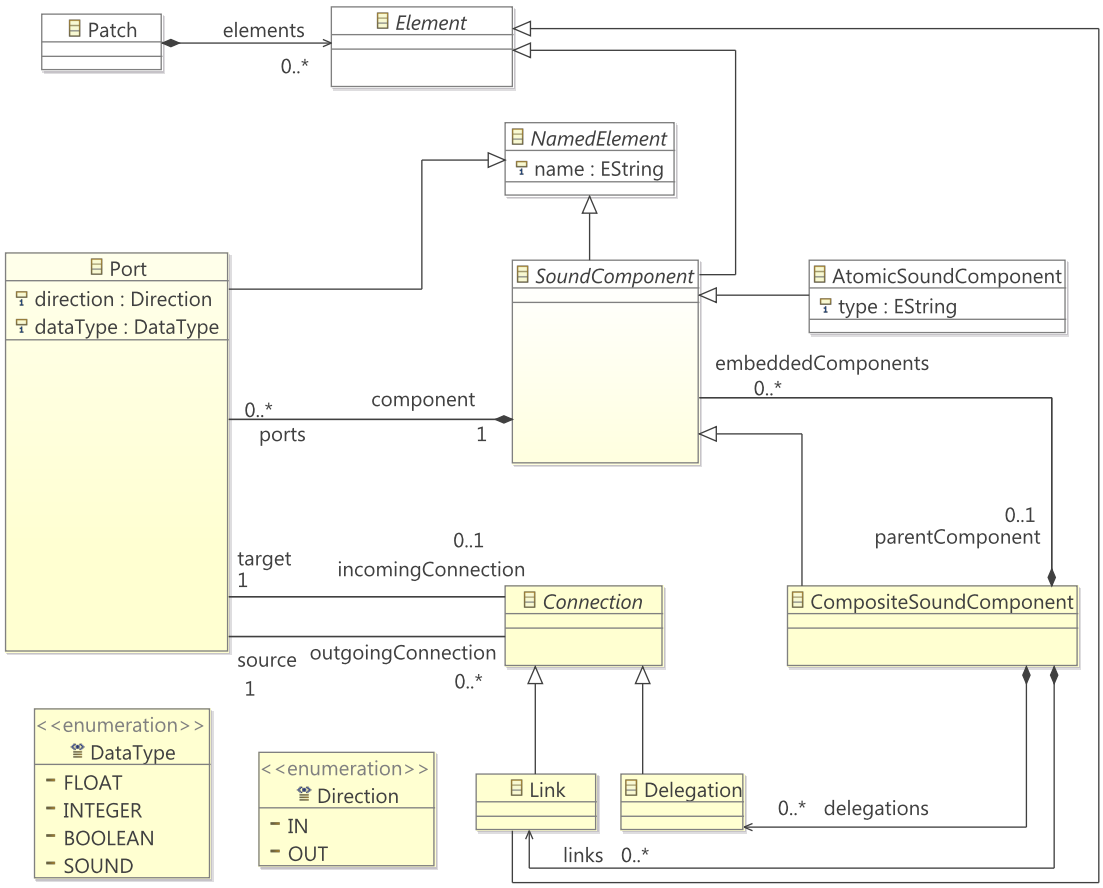

Figure 8: Soundgates meta-model

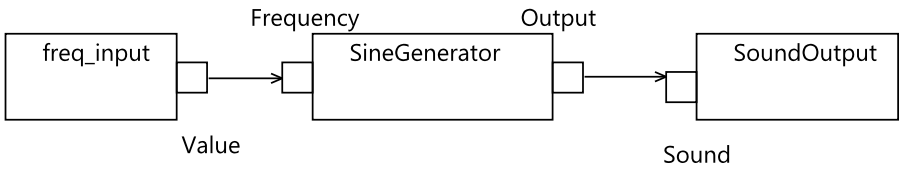

Figure 9: Example Soundgates model

6.3 Exports and code generation

A patch model as seen in Fig. 9 can be exported to a file in our self-defined format such that a synthesizer can be built dynamically based on this patch. A user has the possibility to generate Pure Data[11] code out of the patch model to simulate the patch in Pure Data before running it on a FPGA. Furthermore it is possible to export a patch model as a simple XML file, such that users can easily exchange their patches.

References

[1] Marko Boger and Jan Köhnlein. *Modell-Editoren für Eclipse entwickeln mit GMF*, 2007.

[2] Sven Efftinge and Clemens Kadura. *OpenArchitectureWare 4.1 Xpand Language Reference*, 2006.

[3] Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, and Philippe Vanderheyden. Eclipse development using the graphical editing framework and the eclipse modeling framework. Technical report, IBM, January 2004.

[4] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[11] www.puredata.info