INF 5120 Oblig 2

Creating a Domain Specific Language (DSL) using the Eclipse Modeling Framework (EMF)
Catalogue

Description of Oblig2 ..................................................................................................3
Task 1 Creating the DSL metamodel with EMF .........................................................4
  1.1 Solution Approach ..........................................................................................4
    1.1.1 Build the ecore model diagram .............................................................4
    1.1.2 Build the gen model ..............................................................................5
    1.1.3 Build and test CJM model .....................................................................6
Task 2 Making a Transformation to extract Statistics ..............................................7
Task 3 Editor, Graphical Export, Model Transformation .......................................8
  3.1 Option 1: Implement a "user-friendly" editor for your DSL .........................8
    3.1.1 Text based editor .................................................................................8
    3.1.2 Form based editor ..............................................................................9
    3.1.3 Graphical editor ..................................................................................11
  3.2 Option 2: Implement a graphical export for the models ...............................13
  3.3 Option 3: Implement support for multiple journeys .......................................15
    3.3.1 Extend DSL metamodel to support flow diagrams .........................15
    3.3.2 Generate individual journey sequences ............................................16
Other solutions for oblig 2 .....................................................................................18
  4.1 Data analysis and query language AML with HTML view .........................18
  4.2 Merging graphical export and the text statistics in HTML view ...................21
Description of Oblig2

The goal of this exercise is to develop a domain specific language and associated tools for capturing, editing and analyzing customer journeys. The DSL will be created as a set of eclipse plugins and using the Eclipse Modeling Framework. As a starting point you should have an installation of the "Eclipse Modeling Tools" bundle which is downloadable at http://www.eclipse.org/downloads/.
Task 1 Creating the DSL metamodel with EMF

The first task of the exercise is to create the metamodel for the DSL using EMF and getting a basic treebased editor for it. The process has been demonstrated during the lecture and is documented in the EMF tutorials (see for example http://www.vogella.de/articles/EclipseEMF/article.html).

It is recommended to work in an iterative way and testing your metamodel as you build it. Start with a small metamodel, check that everything works and then gradually add the concepts you need in order to complete the DSL. Do not attempt to make a complete meta-model on the first iteration!

1.1 Solution Approach

The first step is to build the DSL metamodel. The main point is to capture all the information of the CJM datasets. Because this case is a very simple, so it is better to make your metamodel has good scalability, which means it can be easily extended and used for other case, and what’s also worth mentioned is do not think too much detail at the very beginning. It is recommended to build a basic model first and then improve it gradually.

1.1.1 Build the ecore model diagram

Here is an example for the metamodel, which is extendable and not very complex.

Firstly we set CustomerJourneyModel as the starting point of the whole model. And the model consists of three main elements, which are Journey, Actor and Touchpoint.

CustomerJourneyModel contains Journey and Actor as well as Journey contains Touchpoint, the relationship between them is composition, because they have the same lifecycle, which means each of them cannot exist on its own. For each of them, we define some attributes and sub classes according to the CJM datasets. The whole ecore model diagram is shown in figure 1-1.
After we finish the ecore model diagram, we can check the results in the ecore model, which is shown in figure 1-2.

1.1.2 Build the gen model

The second step is to build the gen model based on the ecore model. And the gen model is used to generate the code of the EMF plugin. The gen model is shown in figure 1-3.
1.1.3 Build and test CJM model

After we build the gen model, we can generate the code, and then launch a new eclipse application in which you can build CJM Model with the treevieww editor. If you can capture all the information of the CJM datasets, your CJM metamodel is fine. The CJM model is shown in figure 1-4.
Task 2 Making a Transformation to extract Statistics

The goal of this part was to extract data from the model we built in part 1 and show statistics.

Solution Approach

To represent the generation of the statistics of the journeys, some methods were added to the .test project which collect data from the journey tree models and print statistics to console. And this statistics may contain basic information of the journey (name, properties), touch-points included in the journey, customer satisfaction for each touch-point and so on.

One example for the statistics is provided in figure 2-1.

![Figure 2-1 an example for statistics extraction](image)

And it is easy to extend this transformation and extraction of statistics like adding ranking system, or comparison of different touch-points in satisfaction or seasonal effects.

Another solution for this part is to generate a HTML report which contains all data in a standalone file for easy sharing, which is provided in section 4.2.
Taks 3 Editor, Graphical Export, Model Transformation

3.1 Option 1: Implement a "user-friendly" editor for your DSL

Provide a usable editor for your DSL. You can decide if you want a text based editor, a form based editor or a fully graphical editor. For text based editor you can use the EMFText tool (see http://www.emftext.org). The web site for the tool includes a lot of example which can help you getting an editor fairly quickly.

3.1.1 Text based editor

- **Solution Approach**
  Firstly download and install the EMFText tool as a Eclipse plugin, since we already have the CustomerJourney.ecore model and the corresponding CustomerJourney.genmodel, right click the CustomerJourney.genmodel,"Generate Syntax"—"HUTX Syntax", then a *.cs file (here is CustomerJourney.cs) will be generated automatically. See Figure 3-1

![Figure 3-1 the CustomerJourney.cs file](image)

The first line “SYNTAXDEF MY” of this file means the generated text file must be followed by a file name extension “MY”, and the second line indicates the URI of the ecore model. ‘TOKENS’ and ‘TOKENSTYLES’ define the pattern matching and
the style of tokens. At the end of the *.cs file, many rules are defined in this file, we can build the text output file by using these rules. Of course we can create or modify rules.

Finally we can generate the text resource as a *.MY file, as demonstrated in Figure 3-2

```
CustomerJourneyModel
{
  contain:
  Actor {
    name:Customer1
    ID:C1
  }

  contain:
  Actor {
    ID:C3
    name:Customer2
  }

  contain:
  Actor {
    name:ServiceCenter
    ID:SC
  }

  contains:
  ExpectedJourney
  {
    journeystart:2012-3-4
    status:completed
    comment:good
    name:EG1
  }
  has:
  ExpectedTP{
    byactor :Customer1
    etime:Touchpoint1
    TID:TP1
    evaluation:good
    date:2012-4-4
    comment:good
  }
  has:
  ExpectedTP{
    byactor :Customer1
    etime:Touchpoint2
    TID:TP2
  }
```

Figure 3-2 The result of the text output of CustomerJourney.ecore model

### 3.1.2 Form based editor

For a form based editor, the easiest method is to extend the default editor generated by EMF. Have a look at "Eclipse Forms" and follow some examples on how to use them in connection with EMF. Here is a link to some more information: http://wiki.eclipse.org/EMF/Recipes (see "Create an Eclipse Forms editor with widgets for your properties").

- **Solution Approach**
  For the form based editor, we can use the EEF Eclipse plugin to do it, see the tutorial:
  
Firstly perform a right click on the .genmodel file generated from the metamodel and select the action EEF > Initialize EEF models, after we have the *.eefgen file, *.components file and the “src-gen” folder, the next step is to open the plugin.xml file and add a dependency to the bundle "org.eclipse.emf.eef.runtime" in the "Dependencies" tab, as shown in Figure 3-3.

![Required Plug-ins](image)

**Figure 3-3 Modification of the Dependencies of plugin.xml file**

Once these steps done, generation can be called with the action "Generate EEF Architecture", just right click on the EEFGen model. The next step is to add a dependency between the plugin.xml and the XXX_properties.plugin.xml as the tutorial told. After we finish these steps, a form will be build, which is shown in the Figure 3-4 and Figure 3-5.

![CustomerJourneyModel](image)

**Figure 3-4 the form of one Customer Journey Model**
3.1.3 Graphical editor

To implement a graphical editor, we use Eugina tool which is one layer above GMF and normally make things a bit easier.
(see http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/).

Also there is another tutorial which is more specific.
(see http://sistemas.uniandes.edu.co/~isis4712/dokuwiki/doku.php?id=tut_eugenia)

- Solution Approach
  1) Firstly we install an eclipse plugin called EMFatic, which is a text editor of ECore Models, and its syntax is similar to Java. We use EMFatic for the automatic generation of the editor. The first step is to import CJM.ecore model and gen model to our project, and then generate EMFatic source code, which is used to define the graphical editor. The source code can be found in our delivery files.

  2) After generate the EMFatic source code, we can use Eugina to generate GMF models. There are four GMF models which are generated automatically. The first one is Graphical Definition model, which is used to define the figures, nodes, links, etc. that you will display on your diagram. The second one is Tooling Definition model,
which is used to specify the palette, creation tools, actions, etc. for your graphical elements. The third one is Mapping Definition model, which is used to bind the models we have so far: the graphical definition, and the tooling definition.

Based on these models we also get the fourth model: gmf gen model, which is used to generate the graphical editor. All the models we build are shown in figure3-6.

Figure3-6 the GMF models

3) Build the CJM model with the graphical editor

We launch a new eclipse application and build the CJM models; you can design the editor according to your own ideas. And the one we build is shown in figure3-7.

Figure3-7 the graphical editor
3.2 Option 2: Implement a graphical export for the models

Customer journey experts typically extract graphical representations from the expected journeys and customer experiences. The goal of this part of the exercise is to implement a transformation which produces a graphical representation from the models.

From a technical point of view you have many options to generate graphical representations. You can use plain JAVA and create graphics with Java2D (you'll find many Java2D tutorial on the internet). You can also generate SVG graphics using one of the SVG libraries for Java (like batik). Another option is to generate towards the Graphviz tool which has a textual input format and options to generate towards different graphical formats (see http://www.graphviz.org/). Or any other technique you like.

Whichever tool you use and representation you make, remember that the goal is to provide some useful insight to the domain experts so make sure you clearly identify what is it that you want to show on the graphical representation and make sure that you have an appropriate and easy to grasp representation.

In this section, we use three different ways to implement a graphical export.

Solution Approach

1) The first solution is using Graphviz tool, which is based on the dot language with simple syntax. Figure 3-8 shows an example generated by Graphviz, and the source code and the guidance of dot language is included in our delivery files.

![Figure 3-8 CJM diagram](image)

2) The second solution is using Eugina, which provide a function that can generate diagram from CJM model automatically. The generated diagram is almost the same as the one we build by using graphical editor, which is shown in figure

3) The third solution is using EMF to Graphviz, which is an eclipse plugin. This tool is more powerful, which can generate graphical representations from the CJM model automatically. The process of generate a picture is shown at Figure 3-9.
Figure 3-9 How to generate a graphical representation from by emf2gv
And the final result is shown in Figure3-10.

Figure 3-10 Emf to GraphViz transformation
3.3 Option3: Implement support for multiple journeys

The goal of task 3.3 is to extend the former DSL metamodel in order to be able to represent such flow diagrams and to write a program which processes these flow diagrams and produce models for all individual journey sequences.

3.3.1 Extend DSL metamodel to support flow diagrams

To extend the DSL metamodel to support flow diagrams, we add Flow as a component in a customer journey model. And build several subclasses as flow elements to organize the diagram. Also, the touchpoint class is divided into two types: JourneyTouchpoint for Journey and FlowTouchPoint for FlowElement. The Flow metamodel segment is shown in Figure 3-11.

![Flow metamodel segment](image)

Figure 3-11 the flow metamodel segment

As we have the metamodel for Flow, we can build a journey model graphical editor as described in section 3.1.3, and then we can draw the flow diagram directly. An example of the flow diagram is shown in figure 3-12.
3.3.2 Generate individual journey sequences

As we have built the flow diagram, we need some methods to extract data from it and to transform it to individual journey sequences.

Each flow diagram would have a corresponding tree-based .journeymodel file, so the idea is to read this file to get the whole flow, and to extract touchpoints from the tree and reorganize them to a generic journey which is also a .journeymodel file and then to generate the graphical view of the journey sequences.

As we can see in figure 3-12, this flow have two dividing points, which divide this flow to 4 different journey sequences, so the result should include 4 journeys, as is shown in figure 3-13, which is generated automatically.
Figure 3-13  four journey sequence
Other solutions for oblig 2

4.1 Data analysis and query language AML with HTML view

This is a good solution for data analysis supported by a query language AML and website view, which is proposed by Henning and Dalimir in Customer Journey System. See Appendix A.

It is difficult to know what kind of statistics that is required at design-time, and therefore found it convenient to design a query language for identifying desirable metrics for processing of the customer journey models. This allows the end users to extend the solution non-intrusively. The alternative would be to update the tooling every time new types of statistics are required - something which is not a good approach.

The query language, named Analysis Modeling Language (AML) comprises concepts for modeling of metrics and criteria; an analysis is built using one or more metrics. Each metric identifies required data or statistics. Each metric can be constrained by an arbitrary number of criteria. Three types of metrics are supported: count, find and calculate. A count metric is used for counting the number of journeys or touchpoints that fulfill one or more requirements. find metrics shows the identified targets that fulfill a set of requirements, while calculate can be used to process the targets in different ways, e.g. calculating percentages. A criterion is modeled using a simple expression language. Three operators are supported: equals, lessThan and greaterThan. Each operator is associated with a value.

Obviously, the language can be extended to support more types of metrics and operators. However, we found this to be out of scope of this exercise. An overview of the AML metamodel is given in Figure 4-1.

How a concrete syntax for the AML language could have looked like in Figure 4-2.
Two kinds of visualizations for the statistics and the modeled journeys: website and graphs.

The Website Generator

The website generator generates a website based on an AML model. That is, the result of performing the AML analysis is shown in the website. An overview of the website is given in Figure 4-3 and Figure 4-4. The second page is the result of performing the analysis.
The Graph Generator

Metamodel of the intermediate graph is in Figure 4-5. Via Parameter metaclass we can specify visual features of the graph. There is a set of pre-defined parameter’s name with given semantic.
For example, parameter with name TEXTCOLOR means color of the text inside the node. In the current system we support two types of graph forms: graphml and dot. The later one is more widely used. Examples of outputs are in Figures 4-6. The red text means that a custom TP is an ad-hoc. Border color means evaluation of the TP where: red is bad, yellow is medium, green is good, and black means without evaluation.

Figure 4-6 an example graph

4.2 Merging graphical export and the text statistics in HTML view

This solution is proposed by Tobias Birmili, Florian Hagenauser and Kacper Surdy in “Creating a Domain Specific Language using the Eclipse Modeling Framework”. A colored graphical export of the journey is implemented, and the comparison between a generic journey and actual customer journey can be shown in the graph. Also, graphical export and text statistics are merged in a HTML view. See Appendix B.

The last part was to merge the graphical export feature and the statistics view. This was implemented into the Analyzer component and generates a new HTML file which can be viewed with a web browser (tested with Chrome 18+, Firefox 12 and Safari). The HTML shows a tabbed view of all the example journeys and it is possible to view every journey on its own. All the statistics from the analyzer can be found here. By hovering over a touchpoint it is highlighted in the graphical export which allows an easy overview even of very complex journeys. The reverse way is also possible. Figure 4-7 shows an example of the merged component. Every page also shows the complete overview and the touchpoints there are also highlighted. To generate the HTML the CLI from the Analyzer component has to be run and a new HTML file named output.html is generated. This file can be viewed with any recent web browser.
Figure 4-7: an example of merged component showing journey statistics