Development of a 2D Meta Model Browser for the MTV framework

Bachelor report presented to
CUI
Geneva University

by

Xavier Eduardo Monnet

Supervisors:
Prof. Dr. Didier Buchs
Assistant Luis Pedro

Geneva, October 2006

Acknowledgements

This project takes me a long time to realize. It started on February 2006 and ended on October 2006. During this time, some people helped me. I want here remind them.

First of all, I want to thank Prof. Dr. Didier Buchs for putting to my disposition a computer with witch I realized all this project and the presentations.

Thanks to my supervisor Luis Pedro too, who have been much more than a supervisor. I will say that he has been a bit like a coach, motivating me during this long work. I appreciate his disponibility. He also helped me with this report correcting my english. Indeed, this was the first time I wrote a report in english. I found this has been an interesting challenge for me.

I want to thank Ed Merk, project lead of the Eclipse Modeling Framework project, for his helpful answers to my questions in the Eclipse Tools EMF newsgroup. Thanks to him I understood better the different types of references in EMF.

Thanks also to a Batik SVG contributor called Andres Toussaint for his "Batik Drag Tutorial: How to move shapes in the Canvas" whitch has been very useful for implement the drag and drop feature of the developed 2D meta model browser.

I don't forget the people who answered to my questions about SVG in the Yahoo and Google SVG groups.

And finally thanks to Sergio Coelho for his encouragements and help during this project and also more generally during my university studies.

Abstract

The aim of this bachelor's project is to develop a generic graphical browser for meta-models. The 2D graphical representation of the meta-models will be supporting in its first version meta-models expressed in Ecore formalism but should be possible to extend it in order to support others (e.g. MOF meta-models).

This work will be accomplished by using Scalable Vector Graphics (SVG) technology which is a Scalable Vector Graphics. SVG is a language for describing two-dimensional graphics in XML and an open standard created by the World Wide Web Consortium, which is also responsible for standards like HTML and XHTML

The graphical representation of the meta-models will have a UML-like layout using a subset of the Unified Modeling Language (UML) Class diagrams.

This 2D Meta-Model Browser will be integrated to the Model Model Transformation for Verification (MTV) project which is being developed by the Software Modeling and Verification (SMV) group at the University of Geneva.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main objective of this Bachelor Project is to develop a 2D Meta Model Browser. The general idea is to take the description of a meta model stored in a XML Metadata Interchange (XMI) [1] format produced by any Eclipse Modeling Framework (EMF) Ecore [2] or Meta Object Facilities (MOF) [3] base tool. This work will be integrated in the MTV [4] framework being developed by the Software Modeling and Verification (SMV) [5] group of the University of Geneva and to generate a graphical representation inspired by the Unified Modeling Language (UML) [6] diagram class style.

In this chapter some concepts and keywords that are useful to understand the work done during this Bachelor's Project are introduced.

1.1 Model-Driven Architecture

The Model-Driven Architecture (MDA) [7] is a software design approach, proposed and sponsored by the Object Management Group (OMG) [8]. A lot of different software design techniques have been proposed since the first steps of the software engineering. So, the question we can ask ourselves is: why is the OMG proposing MDA ?

The computer technologies progress and evolve in a very fast way. The work of the engineers often consist in adapting projects developed months ago with technologies that are no more actual. To solve this problem and to try to develop projects without being dependent on the technologies evolutions, MDA has been proposed. In fact MDA allows to separate design from architecture and realization technologies, facilitating that design and architecture can evolve independently.

The general idea of MDA methodology is to, after having specified the clients requirements in a Computation Independent Model (CIM), model at a high level of abstraction the logical and behavioral functionalities of a system. This high level of abstraction is called Platform Independent Model (PIM) and its main objective is to allow the specified functional requirements to survive to the changes done by the realization technologies and architectures. Once this done, we can transform the PIM to a Platform Specific Model (PSM). The huge interest of this technique is that this transformation can be performed automatically.

Some transformation softwares with the purpose of supporting MDA philosophy are being developed - the main goal is to allow that the ideas of specifying at a high-level of abstraction and then automatically generate different realizations of them can be a reality. Some of the most known softwares that support the MDA approach are, among others, MetaEdit+ [9], Generic Modeling Environment [10], ParadigmPlus [11] and DOmain Modeling Environment [12]. The MTV is also one of the framework that supports MDA approach although with the main goal of using the specification at a high-level of abstraction in order to be able to perform transformations specifically for software verification.

We can point that MDA can be seen as a good solution because when evolution of technologies occurs (and usually the occur very often), we should just need to regenerate the new PSM that respects the new technology. In principle, this makes a great economy of time.

Another thing I can add about MDA after interesting myself on reading what was written by specialized magazines, is that the MDA is seen as a great solution for the future of the software engineering. But some people seems to be not filled with enthusiasm. Some of their arguments are that MDA requires to much qualifications and knowledges. It is a fact that the development teams that will adopt MDA methodology will have to control concepts and languages as UML, Object Constraint Language (OCL), CIM, PIM, etc... But is this really a problem ? I don't think so. In the informatics world, one of the principal rules is that people are always having to learn new concepts, new technologies... that is nothing new. Moreover I will say that this is the reason why informatics science makes such extraordinary progresses in a such fast way.

1.2 Of models and meta models

This section is not the continuation of the John Steinbeck's book "Of mice and men", it is rather to write about some fundamental concepts that must be known. Before we write about what is a meta model, it is important to remind what a model is.

A model is an abstraction of phenomena in the real world. In other words, it is a theoretical construct that represents some processes or behaviors of a given system, with a set of variables and a set of logical and quantitative relationships between them. There are a lot of modeling languages. Each one have his own advantages and defects. Some are formal but harder to use, other easier to understand but having ambiguity. For examples, Petri Nets, Z or OCL are modeling languages. Actually, the more used and accepted in the software development world is the Unified Modeling Language (UML) of the OMG. Modeling a system is the next logical step after producing the requirements documents. Knowing that approximately sixty per cent of the part of a software development budget is alloted to software's maintainability (corrective, adaptive and perfective) [13] we can easily understand why good modeling techniques and the Model-Driven Architecture (MDA) approach are fundamental.

A meta model is yet another abstraction, highlighting properties of the model itself. A model is said to conform to its metamodel like a program conforms to the grammar of the programming language in which it is written. In other words, a meta model is a collection of "concepts" (things, terms, etc.) within a certain domain. We will see further in this project report how we have defined a meta model that describes all the graphical elements useful for creating the graphical representation of a meta model. The OMG usually uses following modeling architecture to present and explain the meta models the (also shown in Fig. 1.1) :

M0 level: Data layer contains usually source code derived from the *M1* level fro different program languages;

M1 level: Model layer, describes the information in *M0* layer. In other words, contains the models from which the source code in *M0* layer is generated. The models available in this layer can be transformed in several different source code languages, depending only on the fact if the languages support the concepts presented in the model;

M2 level: Meta model layer that is a description of the models in layer *M1*. It is this meta model that defines the abstract syntax of a language and details what artifacts can be present in the model;

M3 level: Meta meta-model layer are MOF Ecore themselves. In this approach the assumption is that MOF is self-described. This means that we don't need need another level for MOF description. In summary, the *M3* layer is the description of the Meta model, i.e. the *M2* layer.

The meta meta models are one more step of abstraction. The meta meta model describe s the meta model in the same way that the meta model describes a model. The particularity consists on the fact that if we try to make another step in the abstraction, we will find the same meta meta model. This is why the meta meta models are said to be an auto-descriptive layer: the meta meta model is self described.

There are different approaches and implementations for using meta modeling techniques. One of them is the Meta Object Facilities (MOF) [3] supported by the OMG. Another one is the Eclipse Modeling Framework (EMF) Ecore [2] that is proposed by Eclipse project. We will introduce Ecore that is the meta model used by EMF in the next section.

As an illustration of this methodology take table 1.1

For who might be interested on going deeper and reading more about meta models and MOF, a good work has been develeped by Stephane Heck in his Bachelor's report [4]. He writes about the MOF architecture and gives a very interesting example in witch he starts with a model and climb trough the levels of the modeling architecture showed above.

1.3 Eclipse Modeling Framework and Ecore

The EMF [2] [14] is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

We have adopted EMF Ecore to define a meta model that will be later used to transform the meta model stored using XMI to an SVG [15] description.

Table 1.1. Meta Modeling Architecture Layers with some examples

Meta-Level	**Description**	**Examples**
M0	Data / instances	Records in a DB table, instances of Java classes (abc -instance of java.lang.String)
M1	Metadata / models	Tables, columns in a database (records in a system catalog), concrete classes, methods, fields in Java program (java.lang.String instance of Java Class language construct) classes (abc -instance of java.lang.String)
M2	Meta-Metadata/ metamodels/ languages	Description of database (definition of things like Table, Schema, Column, etc.), description of language constructs (definition of Class, its attributes, contained elements methods, fields, etc.)
M3	Meta-metamodel	MOF or Ecore

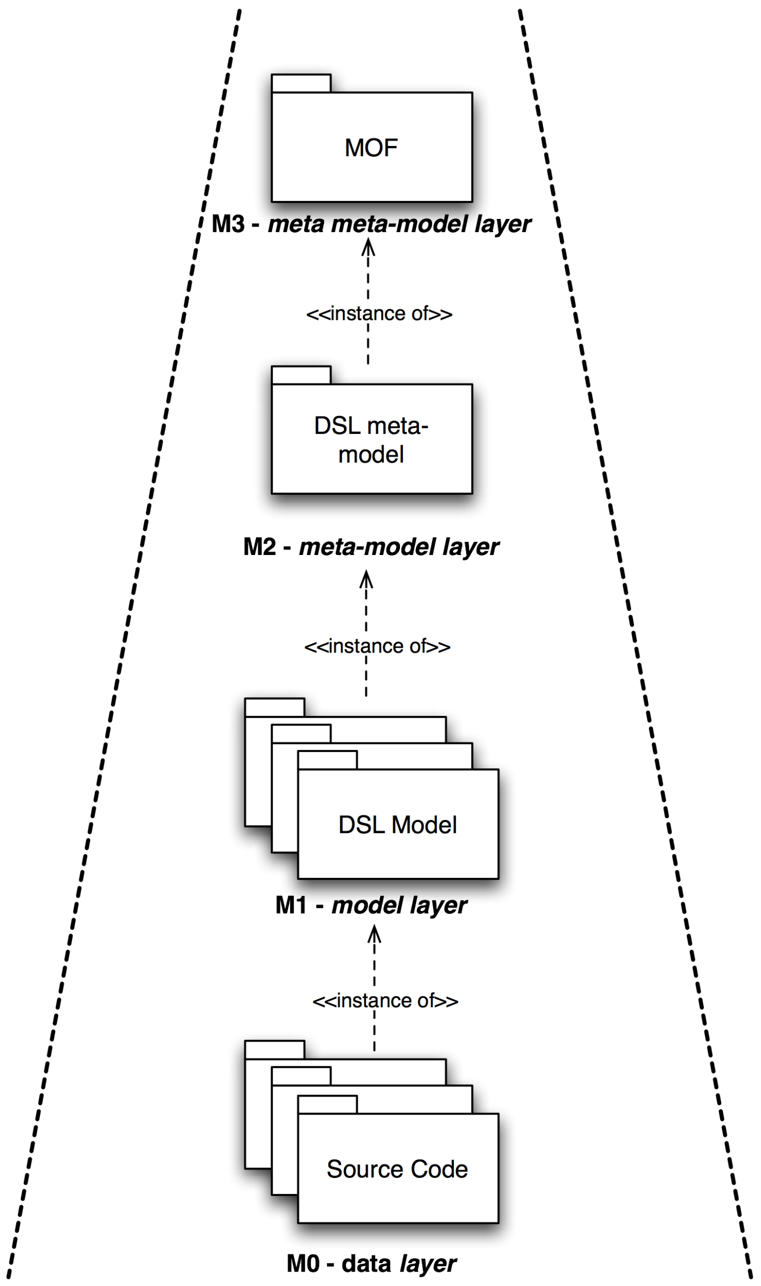


Figure 1.1. Architecture - Meta Object Facilities

This is what will afterwards allow us to provide its graphical representation. The details will be presented in chapter 3 (Problem approach). EMF has been useful also for the automatic generation of the Java classes and interfaces that allow browsing and creation of instances of this meta model.

The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective Application Programming Interface (API) for manipulating EMF objects generically.

Ecore is also a meta meta model because it is its own meta model, in other words it describes itself. In Fig. 1.2 it is possible to see a simplified representation of the Ecore meta model.

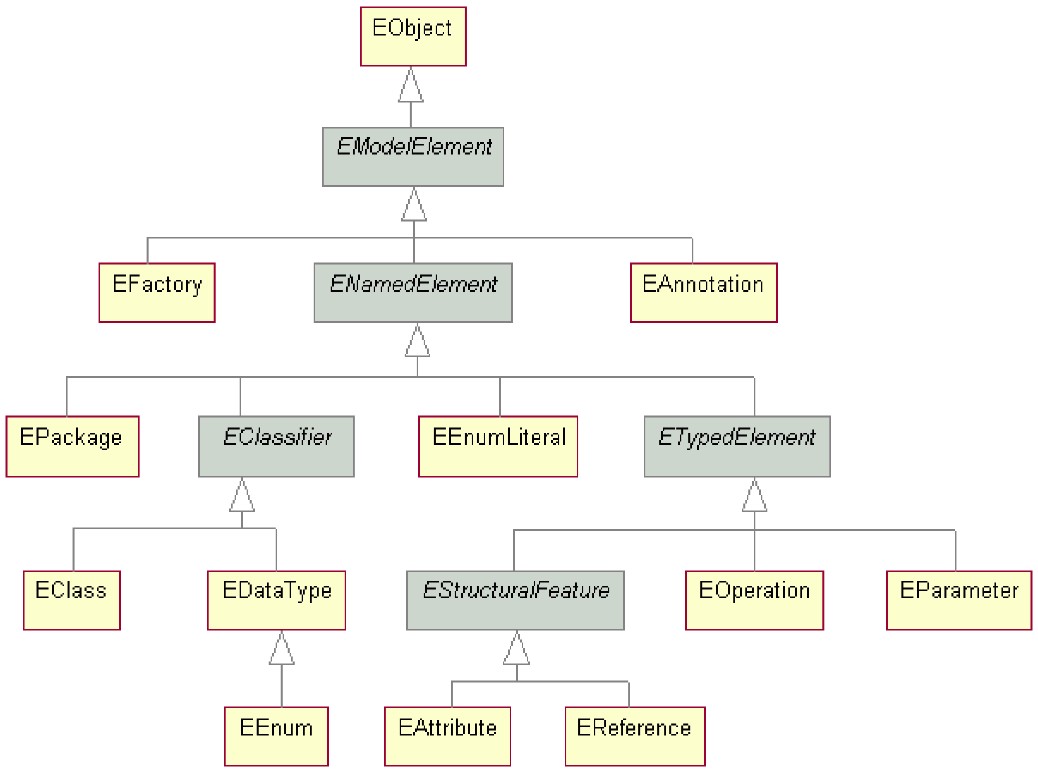


Figure 1.2. Simplified Ecore meta model

An instance of an Ecore metamodel will be a model that is composed by ENamedElement objects. Some important elements are the EPackage, Eclassifier and the ETyped Elements. We see that all the concepts needed for modeling are described in Ecore.

1.4 Scalable Vector Graphics

The Scalable Vector Graphics (SVG) [15,16] is an Extensible Markup Language (XML) [17] markup language for describing two-dimensional vector graphics, both static and animated, and either declarative or scripted. It is an open standard created by the World Wide Web Consortium (W3C) [18] in 1998. An example of a graphic produce with SVG can be seen on Fig. 1.3.

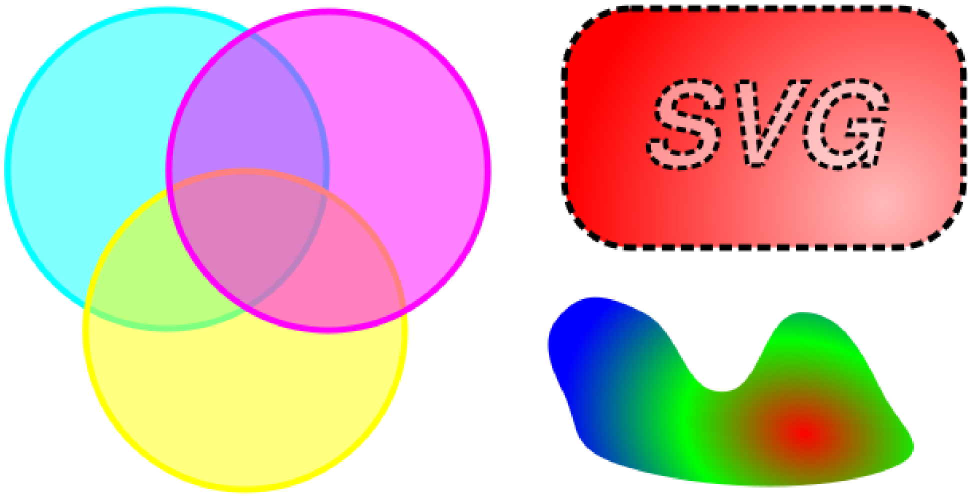


Figure 1.3. Static image generated from an SVG example

SVG allows three types of graphic objects:

- Vector graphic shapes (e.g. paths consisting of straight lines and curves, and areas bounded by them);
- Raster graphics images / digital images;

- Text.

Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility.

The Document Object Model (DOM) [19] for SVG, which includes the full XML DOM, allows straightforward and efficient vector graphics animation. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object.

As an example, we can point out that SVG is used to do the icons of Linux GNOME and KDE. The fact of making the icons being vectorial graphics allows to have resized icons having always the optimal quality. For the same reason, SVG is used to develop applications for mobile phones and Personal Digital Assistants (PDAs). Some people thinks that SVG is called in a not so far future to replace Flash [20] in the web development industry.

To finish this small introduction to SVG, we present an example on Listing 1.4 of an SVG specification file. It is the SVG file used to produce the picture shown on Fig. 1.3. Let's speak a bit of the structure of this SVG example. The structure of an SVG document will seem familiar to anyone who has looked at an HTML file before. The structure of SVG code is a series of elements, some of which enclose other data. In every SVG file, the content is enclosed with the SVG element's tag : <svg> . . . </svg>. That is the root element. All the SVG content must appear between these two tags. All the other tags can possess some attributes with their values and are of the following form :

- <tag> ... </tag> or <tag />.

In the example in Listing 1.4 appears the XML data defining the document type and nature, which are for this example the XML version 1.0 and using the encoding UTF-8. It is the only information allowed to be outside of the two SVG root tags. After, we have the SVG root tag and we can take a look to some of the elements contained in this example. We have some definition between the defs tag. As it is possible to see some linear gradients are defined used for rendering elements. Let's describe now some of the used SVG tags in this example.

The 'g' element : is a container element for grouping together related graphics elements; a group of elements, as well as individual objects, can be given a name using the id attribute; named groups are needed for several purposes such as animation and re-usable objects; a 'g' element can contain other 'g' elements nested within it, to an arbitrary depth; any element that is not contained within a 'g' is treated (at least conceptually) as if it were in its own group;

The 'path' element : represent the outline of a shape which can be filled, stroked, used as a clipping path, or any combination of the three; a path is described using the concept of a current point; in an analogy with drawing on paper, the current point can be thought of as the location of the pen; the position of the pen can be changed, and the outline of a shape (open or closed) can be traced by dragging the pen in either straight lines or curves; paths represent the geometry of the outline of an object, defined in terms of moveto (set a new current point), lineto (draw a straight line), curveto (draw a curve using a cubic Bzier), arc (elliptical or circular arc) and closepath (close the current shape by drawing a line to the last moveto) elements; compound paths (i.e., a path with multiple subpaths) are possible to allow effects such as "donut holes" in objects;

The 'rect' element : defines a rectangle which is axis-aligned with the current user coordinate system; rounded rectangles can be achieved; SVG contains a set of basic shape elements as rectangles, circles, ellipses, lines, polylines or polygons; mathematically, these shape elements are equivalent to a 'path' element that would construct the same shape; the basic shapes may be stroked, filled and used as clip paths; all of the properties available for 'path' elements also apply to the basic shapes;

The 'text' element : defines a graphics element consisting of text; the attributes and properties on the 'text' element indicate such things as the writing direction, font specification and painting attributes which describe how exactly to render the characters; since 'text' elements are rendered using the same rendering methods as other graphics elements, all of the same coordinate system transformations, painting, clipping and masking features that apply to shapes such as paths and rectangles also apply to 'text' elements.

More information about SVG and its elements can be found in the W3C website about SVG [15].

Listing 1.1. Extract of an SVG example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->
<svg
    ...
  <defs
      id="defs4">
    <linearGradient
        id="linearGradient2790">
      <stop
          style="stop-color:#ff0000;stop-opacity:1.0000000"
          offset="0.00000000"
          id="stop2792" />
      <stop
          style="stop-color:#00ff00;stop-opacity:1.0000000"
          offset="0.50000000"
          id="stop2802" />
      <stop
          style="stop-color:#0000ff;stop-opacity:1.0000000"
          offset="1.0000000"
          id="stop2794" />
    </linearGradient>
    <linearGradient
        id="linearGradient2775">
   ...
  </defs>
  <g
      id="layer1"
      transform="translate(6.201104,5.167586)">
    <path
        d="M 300.00000,252.36218 C 300.00000,307.59065
            255.22847,352.36218 200.00000,352.36218 C
            144.77153,352.36218 100.00000,307.59065
            100.00000,252.36218 C 100.00000,197.13371
            144.77153,152.36218 200.00000,152.36218 C
            255.22847,152.36218 300.00000,197.13371
            300.00000,252.36218 L 300.00000,252.36218 z "
        transform="translate(-98.00000,-149.0000)"
        style="fill:#00ffff;fill-opacity:0.49999997;fill-rule:
            evenodd;stroke:#00ffff;stroke-width:4.0000000;stroke-
            linecap:butt;stroke-miterlimit:4.0000000;stroke-
            dasharray:none;stroke-dashoffset:0.00000000;stroke-
            opacity:1.0000000"
         id="path1306" />
      ...
      <rect
         width="250.00000"
         height="150.00000"
         ry="41.428570"
         x="351.00000"
         y="2.3621826"
         style="fill:url(#radialGradient2785);fill-opacity
            :1.0000000;fill-rule:evenodd;stroke:#000000;stroke-
            width:4.0000000;stroke-linecap:butt;stroke-miterlimit
            :4.0000000;stroke-dasharray:8.0000000, 4.0000000 ;
            stroke-dashoffset:0.00000000;stroke-opacity:1.0000000"
         id="rect2041"
         rx="41.428570" />
      <text
         x="398.91016"
         y="101.34265"
         style="font-size:72.000000px;font-style:oblique;font-
            variant:normal;font-weight:bold;font-stretch:normal;
            text-align:start;line-height:125.00000%;writing-mode:
            lr-tb;text-anchor:start;fill:#ffffff;fill-opacity
            :0.49999997;stroke:#000000;stroke-width:3.0000000;
            stroke-linecap:butt;stroke-linejoin:miter;stroke-
            miterlimit:4.0000000;stroke-dasharray:6.0000000,
            3.0000000 ;stroke-dashoffset:0.00000000;stroke-opacity
            :1.0000000;font-family:Bitstream Vera Sans"
         id="text2043"
         xml:space="preserve"
         sodipodi:linespacing="125.00000%"><tspan
           x="398.91016"
           y="101.34265"
           id="tspan2045">SVG</tspan></text>
      ...
   </g>
</svg>
```

1.5 Batik SVG Toolkit

The Batik toolkit [21] is a Java technology based toolkit for applications or applets that want to use images in the SVG format for various purposes, such as viewing, generation or manipulation. It gives to developers a set of core modules which can be used together or individually to support specific SVG solutions.

Let's describe the Batik architecture (Fig. 1.4) and the role of each of its modules. The Batik modules are of one of three types :

Application Modules : illustrate how to use the Core Modules and let users evaluate the Batik software by experimenting with its features; for example, the SVG Browser is built using several Batik Core Modules (such as the JSVGCanvas GUI component and the ImageTranscoders) and illustrates how Batik lets you not only view, zoom, pan and rotate SVG documents, but also search them and convert them to other formats (such as JPEG, Tiff or PNG); the SVG Pretty Printer is another example that shows how Batik lets you manipulate and transform SVG content, here for the purposed of tidying up potentially disorganized SVG files; the SVG Font Converter illustrates how Batik can help you embed SVG Font definitions in an SVG file by providing an application that converts ranges of characters from a True Type Font format to the SVG Font format; finally, the SVG Rasterizer shows how to leverage the Transcoder API to convert to and from SVG content; note that even though the Application Modules are meant to be usefull and fun to use, they are not the primary deliverables of the Batik project; instead, they are illustrations of how the Batik core modules might be used and combined;

Core Modules : are the heart of Batik and the primary deliverables for the projects. These are the modules developers use to manipulate, generate, create, convert, render and view SVG content; they can be used individually or in combinations for various purposes, and the Application Modules offer some usage examples : SVG Generator is a modules which contains SVGCanvas2D that lets all Java technology applications or applets easily convert their graphics to the SVG format, as easily as they draw to a screen or a printer, by leveraging the Java 2D API's extensible design; SVG DOM an implementation of the SVG DOM API defined in the SVG recommendation; it lets the programmer manipulate SVG documents in a Java program; JSVGCanvas is a UI component that can display SVG content and let the user interact with that content (zoom, pan, rotate, text selection, etc...); Bridge module is rarely used directly; it is responsible for creating and maintaining an appropriate object corresponding to an Element; the Bridge converts an SVG document into the internal representation Batik uses for graphics (GVT, the Graphic Vector Toolkit); Transcoder is a module that provide a generic API for transcoding an input to an output; this module transcodes an input stream or a document into a particular ouput format;

Low Level Modules : are used internally by the Core Modules to accomplish their work; these modules are not typically used by developers directly; the Low Level Modules include: the Graphic Vector Toolkit (GVT) module, which represents a view of the DOM tree that is more suitable for rendering and event handling purposes; this module describes DOM tree in terms of a tree of Java objects; the Renderer module is responsible for rendering a GVT tree and any related task; for example, a raster based Renderer may perform some caching (the default Renderer in Batik does that); however, a Renderer could perform any task it deems necessary and does not have to be raster based; the SVG Parser module contains 'Micro Parsers'; these are parsers for complex SVG attributes such as transform or color attributes. Higher level modules rely on the SVG Parser module.

Now that we have introduced Batik, let us talk about why we are using Batik in this Bachelor's project. We are going to use the Core Modules in order to develop the 2D Meta Model Browser. Lets have closer look to the modules used for the project:

- The SVG Generator is used to produce the SVG graphic file. After creating an instance of the SVG Generator, we can draw lines, polygons, shapes or text and produce the corresponding SVG file. We use this in the project for generating the SVG document that describe the 2D representation of the meta model we are browsing.
- The User Interface Component is used for displaying a graphic described in a SVG document. To be more precise, an instance of JSVG-Canvas allows us to display the graphic. It let the user interact with that content too (zoom, pan, rotate, text selection, etc...). We use this in the project to display the created SVG description of the meta model

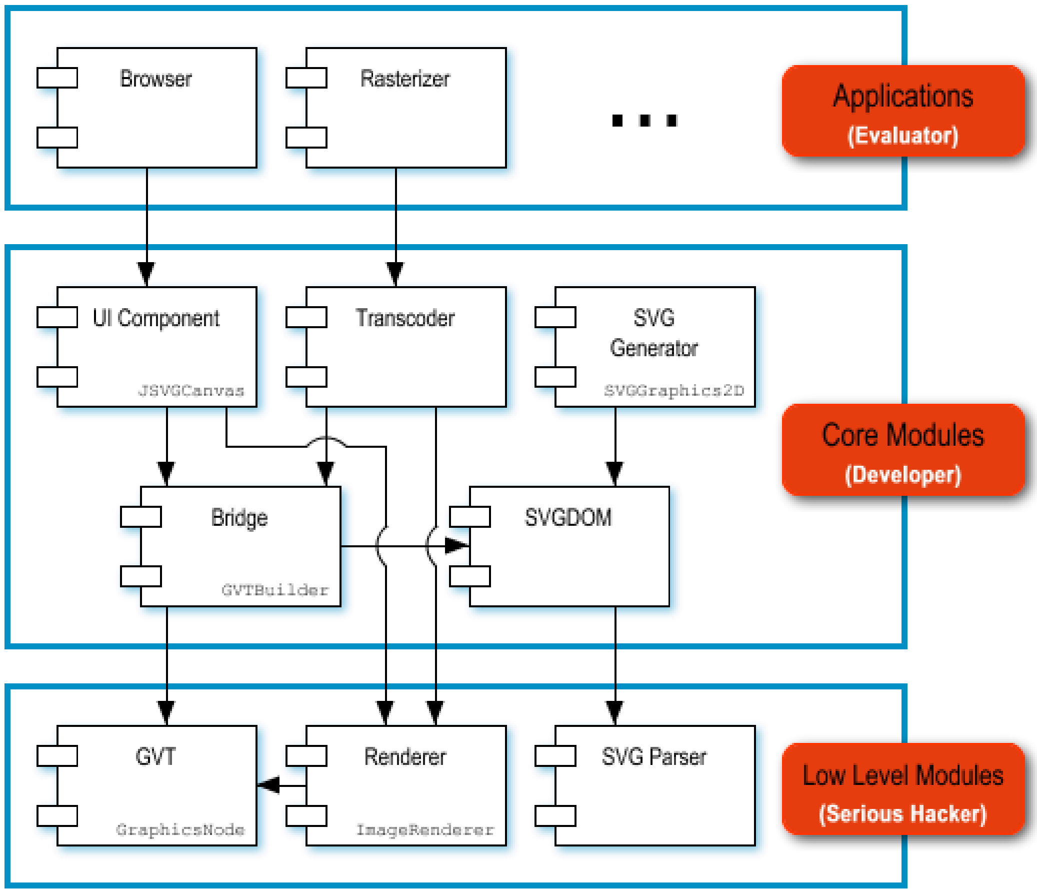


Figure 1.4. Architecture of Batik

we are browsing and to allow manipulations and interactions with the elements of the 2D representation.

- The next logical module we need to use, is the SVGDOM. It allows us to manipulate the DOM of the SVG displayed. This is used in the project to access and modify the DOM document of our 2D graphic representation of the browsed meta model.

- And finally, the Transcoder API is used for rasterizing a SVG graphic to an image file (png, jpg...). In the project, the transcoder is used to allow the Export feature of the browser. This allow to export any 2D graphic representation of our meta model in SVG to an image format like png or jpeg.

To conclude this chapter, I will say that I think the main keys to open this project's box have been given now. Let your curiosity be stronger than you and open the box of this project reading the next chapters. Do not be afraid, I can promise you that it will not be like the Pandora box.

Chapter 2

Description of the project

The main goal of this project is to develop a generic two-dimensional (2D) Java based browser for meta-modeling purposes. It aims to deploy a generic meta-model browser for the Meta Object Facilities (MOF) specifications as well as for its equivalent in the Eclipse Modeling Framework (EMF): the Ecore specifications.

The Meta-Models are provided in XMI (or XML) data files and are interpreted using:

For MOF models: the Java Meta Data Interfaces (JMI);

For Ecore models: the similar Java Interfaces generated by EMF.

Both of these Java interfaces are platform-independent infrastructure to manage the creation, storage, access, discovery, and exchange of metadata;

This project will be integrated in the Model Transformation for Verification (MTV) [4] being developed by the SMV group. The MTV project already incorporates generic model and meta-model browsing (tree base fashion) functionalities using default meta-data management operations. Developing a 2D meta-model browser is the logic next step in order to enhance the meta-modeling power of the MTV project.

The 2D browser was proposed to be developed using Scalable Vector Graphics (SVG) - a language for describing two-dimensional graphics and graphical applications in XML.

2.1 Project Phases

Here are the different phases that were identified before the start of the project :

- State of the art analysis regarding SVG and Java integration;
- Analysis of the problem and research in the areas of MOF, EMF, Java 2D and SVG technologies;
- Definition of a meta-model for mapping MOF and Ecore models into SVG (named **SVG4MTV** meta-model);
- Problem analysis including definition of *Use Cases* with all their description and justification;
- Definition of an API for meta-model graphical 2D browser generation;
- Implementation and integration with MTV using an UML-style meta-model representation with the possibility for:
  - Configuration of the elements to present;
  - Configuration of the font size and type for the elements;
  - Zoom in and Zoom out functionalities;
  - Export to pdf and png file types;
- Implementation of proper algorithm for layout and placement of simplified UML class diagrams
- Writing of the Bachelor's Project report;

2.2 Expected Results

For this project, I was expected to deliver an API that renders 2D meta-models in a UML like visual representation. In the aim of doing this, here are the concrete expected results:

- A meta-model that allows the mapping between MOF and Ecore models into SVG. This meta-model could be either defined with MOF or with ECore. It should also be used as an example/case study of the applicability of the project;

- An API that allows SVG files generation from MOF and Ecore models;
- Integration with MTV project;
- The possibility for modifying the positioning of the meta-model components and being able to save the state;

In the next chapters we will explain how these functionalities were accomplished. We will present in detail the different project phases from analysis to implementation, always trying to justify our approaches and choices.

Chapter 3

Problem approach

The 2D Meta-Model Browser for MTV must be able browse meta-models with an UML Class diagram-like graphical syntax. We will see in this chapter the theoretical approaches chosen for the different problems that appeared during the analysis phase (that will be presented on next chapter) and during the realization of the project.

3.1 The transformations

”Nothing is lost, nothing is created, all is transformed.” said the famous scientist Antoine-Laurent de Lavoisier in the XVIII century. He was right saying that all is transformed. In our case, we have as input of our browser an XMI file containing the description of a meta model and we want to get as an output a graphical representation of it in a SVG file. What we now need to know is how the browser produces the SVG file containing this 2D representation. As MTV stores meta-models in MOF or Ecore files, we can state that there must be a transformation of the MOF or Ecore XMI files into SVG ones. The Fig. 3.1 presents a graphical schema on the simplified version of this transformation is performed.

Although this approach seams to be the preferred one since is the simpler of all, it presents some problems that lead us to change our minds and to choose a different one in order to accomplish our goal. Some of the problems are:

- The transformation algorithm would be dependent on the meta-modeling language chosen;

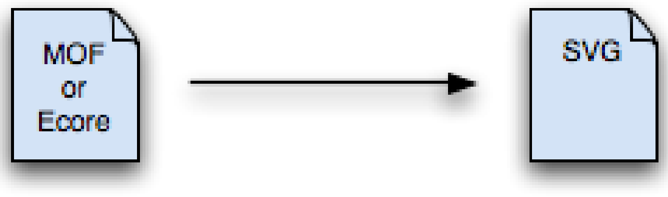


Figure 3.1. Direct transformation

- if in future there is new versions of the meta-modeling language, all the system would need to be re-written;
- The same applies for SVG;
- It is not easy to cope with the idiosyncrasies of both Ecore and MOF.

The solution we propose for this project is to introduce an intermediate level. This level will be defined by a meta-model that describes the SVG's concepts that we need to represent a 2D UML like class diagram of our meta-model. During this report we will be calling this meta-model SVG4MTV. It is not meant to be the complete meta-model of SVG but rather to represent the SVG concepts that we need to concretize this work - SVG offers a lot of functionalities that we don't need and that we will not use in our project.

The SVG4MTV meta model is presented with more details in next section. As it was already mentioned, we will tackle the proble by decomposing the transformation from MOF/Ecore into SVG. This decomposition will use SVG4MTV meta-model and its scheema is presented in 3.2.

In this second approach, the one we have chosen to to implement our 2D meta-model browser, the first phase is to perform the transformation between the meta-modeling language (MOF, Ecore, ...) and the SVG4MTV. This first transformation allows to get all the graphical information needed in a format independent both from the meta-modeling language in witch the meta-model is described and from the SVG specification. Another advantage of using this intermediate format is that, in future, we can imagine to make the system supports different other meta-modeling languages by just writing the transformation from the new meta-modeling language(s) to SVG4MTV, i.e. adding another arrow in top part of the schema represented by Fig. 3.2

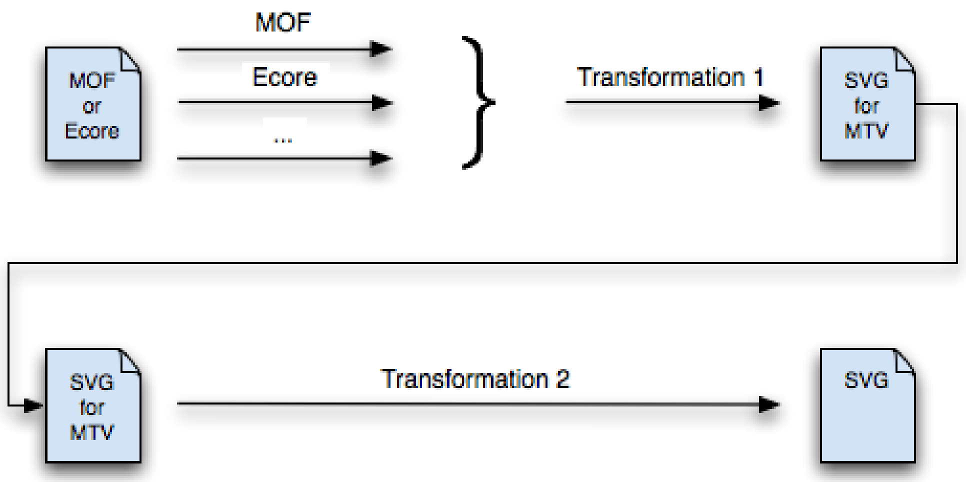


Figure 3.2. Two steps MOF/Ecore to SVG transformation

The second phase (represented as the arrow *2* in Fig. 3.2) consists in performing the transformation an instance of SVG4MTV to SVG itself. This makes more easy the possible modifications of the SVG specification in future new versions.

In summary, this solution brings more modularity and genericity to our system. But we could have a problem if we want to change the SVG4MTV metamodel. But this decision depends on us, so we are not dependent on other people.

3.2 The SVG4MTV meta model

As we saw in the previous section, we are going to need to define a meta model for storing the graphical information of our meta models. The so called SVG4MTV has this name because it is a meta model that will allow to describe meta model in a SVG file and because it is for MTV. It is very important to remind the fact that this is not a meta model for SVG. It is a meta model that will allow us to map the elements present in any meta model with the SVG elements and concepts we need for its graphical representation a two dimensions canvas. Lets take a simple example. In Fig 3.3 it is presented a little example of what we can expect as result in our 2D meta model browser.

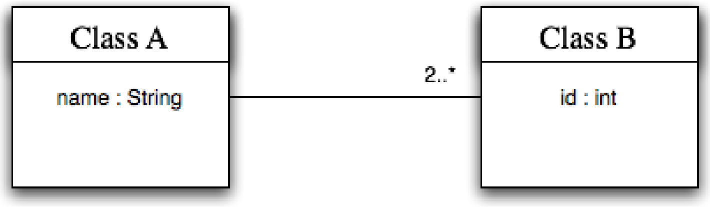


Figure 3.3. Example of expected result of the browser

We can see from this little example that a meta model is represented by rectangles, lines and text. These objects are graphics element existing in SVG. So we can describe this little example by saying that there are two instances of big rectangles, two instances of small rectangles, one instance of a polyline (a polyline is the name given in SVG to a set of points forming a line) and various instances of texts elements. For each element, we can add a set of attributes as (x,y) positions, color, etc.. This description we have done is not more that a textual and non formal description of what corresponds to an SVG4MTV meta model.

The Fig 3.3 corresponds to the XMI (in Ecore formalism) presented in Listing 3.2. That's a good example of what our browser can take as input. You can see in the listing about the Ecore formalism of the example that we have an EPackage that contains two EClassifiers corresponding to the two classes (ClassA and ClassB). Each EClassifier contains two eStructuralFeatures : one describing its attribute, and the other for describing its association end and the EReference for the association between the two classes.

Listing 3.1. Example of XMI file : Ecore of the example on Fig 3.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.
        org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="
        reportExample1"
    nsURI="http:///reportExample1.ecore" nsPrefix="
        reportExample1">
  <eClassifiers xsi:type="ecore:EClass" name="ClassA">
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="2"
        upperBound="-1" eType="#//ClassB" eOpposite="#//ClassB/_
            "/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
         ordered="false" unique="false"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse
            .org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ClassB">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="id"
        ordered="false" unique="false"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse
            .org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="1"
        eType="#//ClassA" eOpposite="#//ClassA/_"/>
  </eClassifiers>
</ecore:EPackage>
```

Now we can see the SVG4MTV file corresponding to the Fig 3.3. The SVG4MTV is presented in Listing 3.2. Let's see if we still find the corresponding elements of the simple example shown in Fig 3.3. You can see in the listing that we have some SVG4MTV elements (I will present them in a detailed way further) as ClassElement and ClassRectangle for describing the two classes (ClassA and ClassB) and elements as Association, AssociationEnds and AssociationPolyLine for describing the association. You also probably noticed that now we have some spacial and two dimensional coordinates informations. They will be used for the placing of the SVG graphic elements and the way we calculate these positions and rectangle's sizes is presented in further sections of this report.

Listing 3.2. Example of XMI file : SVG4MTV of the example on Fig 3.3.

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:SVG4MTV="http:///SVG4MTV.ecore">
  <SVG4MTV:ClassElement name="/7" elementId="1"
      class_big_rectangle="/1" classes_connections="/17"/>
  <SVG4MTV:ClassRectangle x="9" y="10" width="165" height="30"
      class_element="/0">
    <little_rectangle x="9" y="10" width="165" height="15">
      <name_of_class name="/2" type="/3" x="24" y="25"/>
    </little_rectangle>
    <little_rectangle x="9" y="25" width="165" height="15">
      <attributes_of_class name="/4" type="/5" visibility="/6" x
```

```
          =”24” y=”40”/>
    </little_rectangle>
  </SVG4MTV:ClassRectangle>
  <SVG4MTV:SVGClassName text=”ClassA” fontType=”Arial” fontSize=
      ”15” fontColor=”black”/>
  <SVG4MTV:SVGClassType text=”” fontType=”Arial” fontSize=”15”
      fontColor=”black”/>
  <SVG4MTV:SVGClassAttribute text=”name” fontName=”attribute”
      fontType=”Arial” fontSize=”15” fontColor=”black”/>
  <SVG4MTV:SVGClassAttributeType text=”EString” fontName=”
      attributetype” fontType=”Arial” fontSize=”15” fontColor=”
      black”/>
  <SVG4MTV:SVGTextVisibility/>
  <SVG4MTV:SVGText text=”ClassA”/>
  <SVG4MTV:ClassElement name=”/15” elementId=”2”
      class_big_rectangle=”/9” classes_connections=”/17”/>
  <SVG4MTV:ClassRectangle x=”9” y=”120” width=”90” height=”30”
      class_element=”/8”>
    <little_rectangle x=”9” y=”120” width=”90” height=”15”>
      <name_of_class name=”/10” type=”/11” x=”24” y=”135”/>
    </little_rectangle>
    <little_rectangle x=”9” y=”135” width=”90” height=”15”>
      <attributes_of_class name=”/12” type=”/13” visibility=”/14
          ” x=”24” y=”150”/>
    </little_rectangle>
  </SVG4MTV:ClassRectangle>
  <SVG4MTV:SVGClassName text=”ClassB” fontType=”Arial” fontSize=
      ”15” fontColor=”black”/>
  <SVG4MTV:SVGClassType text=”” fontType=”Arial” fontSize=”15”
      fontColor=”black”/>
  <SVG4MTV:SVGClassAttribute text=”id” fontName=”attribute”
      fontType=”Arial” fontSize=”15” fontColor=”black”/>
  <SVG4MTV:SVGClassAttributeType text=”EInt” fontName=”
      attributetype” fontType=”Arial” fontSize=”15” fontColor=”
      black”/>
  <SVG4MTV:SVGTextVisibility/>
  <SVG4MTV:SVGText text=”ClassB”/>
  <SVG4MTV:SVGText text=”_”/>
  <SVG4MTV:Association name=”/16” elementId=”3”
      connected_classes_elements=”/0 /8” is_target=”/19”
      is_source=”/18” source=”1” target=”2”
      polyline_of_association=”/24”/>
  <SVG4MTV:AssociationEnds x=”62” y=”102” name=”/20”
      multiplicity=”/22” sourceId=”1” targetId=”2”
      is_association_src=”/17”/>
  <SVG4MTV:AssociationEnds x="101" y="55" name="/21"
      multiplicity="/23" sourceId="2" targetId="1"
      is_association_tar="/17"/>
  <SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
      "8" fontColor="black"/>
  <SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
      "8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="2..*" fontType="
      Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
      fontSize="8" fontColor="black"/>
  <SVG4MTV:AssociationPolyline association="/17">
    <points_of_association_polyline xValue="86" yValue="39"/>
    <points_of_association_polyline xValue="59" yValue="121"/>
  </SVG4MTV:AssociationPolyline>
</xmi:XMI>
```

Finaly we can see the SVG file corresponding to the Fig 3.3. The SVG code is presented in Listing 3.2. You can see in the listing that we have all the SVG elements for describing the example of Fig 3.3. There is a group for the with the id equal to class1 that corresponds to ClassA, another group of SVG elements with id equal to class2 corresponding to ClassB and finally the last group with id connection3 witch describes the association and the association ends.

Listing 3.3. SVG file for the example on Fig 3.3.

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

<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 1.0//EN' 'http://www.w3.
    org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd'>
<svg xmlns:xlink="http://www.w3.org/1999/xlink" style="fill-
    opacity:1; color-rendering:auto; color-interpolation:auto;
    text-rendering:auto; stroke:black; stroke-linecap:square;
    stroke-miterlimit:10; shape-rendering:auto; stroke-opacity:1;
     fill:black; stroke-dasharray:none; font-weight:normal;
    stroke-width:1; font-family:&apos;Dialog&apos;; font-
    style:normal; stroke-linejoin:miter; font-size:12; stroke-
    dashoffset:0; image-rendering:auto;" xmlns="http://www.w3.org
    /2000/svg">
  <!--Generated by MTV with Batik SVG Generator-->
  <defs id="genericDefs" />
  <g>
    <g style="fill:white; stroke:white;">
      <rect x="0" y="0" width="2110" style="stroke:none;" height
```

```
        ="1320" />
    </g>
    <g id="class1" style="font-size:14; filter:url(/Users/monnet
        /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
        MyDropShadow);">
      <g style="fill:papayawhip; stroke:black;">
        <g>
          <rect x="9" y="10" width="165" style="stroke:none;"
              height="30" />
        </g>
      </g>
      <g>
        <rect x="9" y="10" width="165" style="fill:none;" height
            ="15" />
        <text xml:space="preserve" x="57" y="23" style="
            stroke:none;">ClassA</text>
        <rect x="9" y="25" width="165" style="fill:none;" height
            ="15" />
        <text xml:space="preserve" x="24" y="38" style="
            stroke:none;">name: EString</text>
      </g>
    </g>
    <g id="class2" style="font-size:14; filter:url(/Users/monnet
        /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
        MyDropShadow);">
      <g style="fill:papayawhip; stroke:black;">
        <g>
          <rect x="9" y="120" width="90" style="stroke:none;"
              height="30" />
        </g>
      </g>
      <g>
        <rect x="9" y="120" width="90" style="fill:none;" height
            ="15" />
        <text xml:space="preserve" x="19" y="133" style="
            stroke:none;">ClassB</text>
        <rect x="9" y="135" width="90" style="fill:none;" height
            ="15" />
        <text xml:space="preserve" x="24" y="148" style="
            stroke:none;">id: EInt</text>
      </g>
    </g>
    <g id="connection3" style="font-size:8;">
      <g>
        <path d="M86 39 L59 121" style="fill:none;" />
        <text xml:space="preserve" x="62" y="102" style="
            stroke:none;">2..*</text>
        <text xml:space="preserve" x="62" y="110" style="
            stroke:none;">_</text>
        <text xml:space="preserve" x="101" y="55" style="
            stroke:none;">1</text>
        <text xml:space="preserve" x="101" y="63" style="
            stroke:none;">_</text>
      </g>
    </g>
  </g>
</svg>
```

As we have seen with the last example, we have to make a list of all we are going to need to make a 2D graphic representation for the meta models we want to browse. This list has been done and we are going to see now the SVG4MTV meta model looks like in detail. The Fig 3.4 gives an overview of the SVG4MTV meta model.

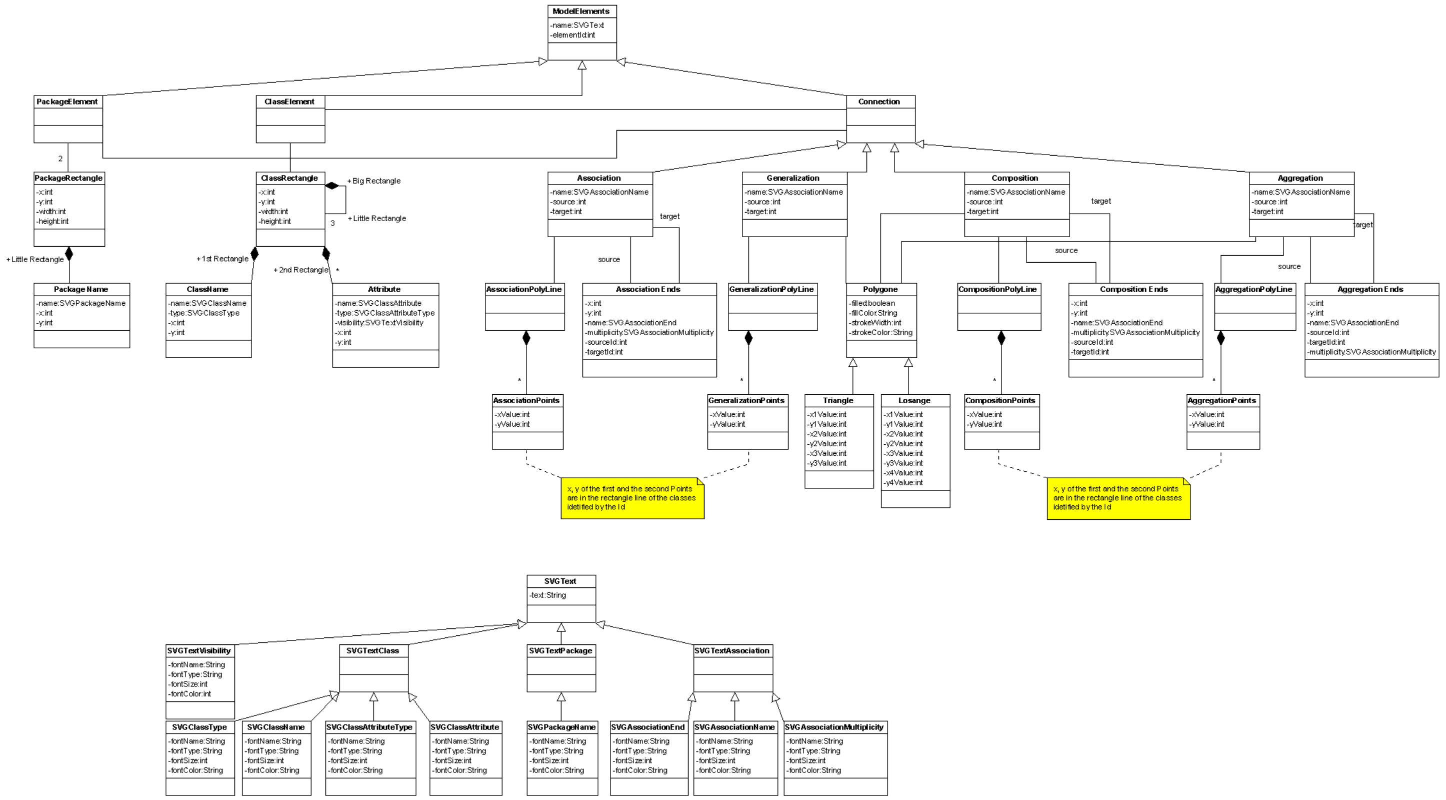


Figure 3.4. Global view of the SVG4MTV meta model

We are now going to see and explain this meta model in parts. We will see in the first part how we have described the packages and the classes of the meta models we want to present in the browser. Then, we will talk about all the stuff concerning the connections between the classes. And, finally, the presentation of the description on how we will deal with the text elements. The Fig 3.5 presents the first part.

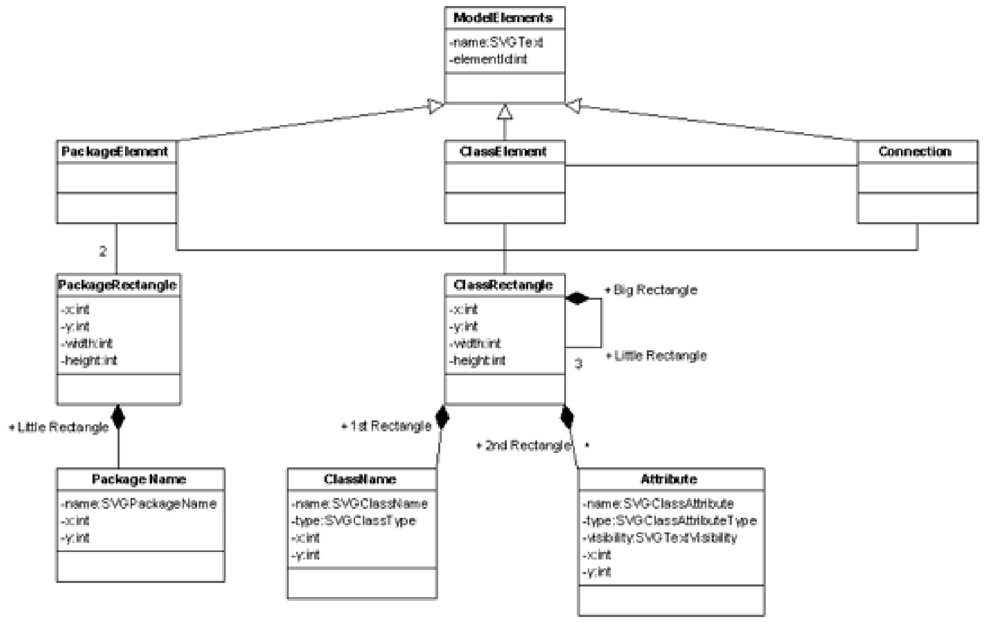


Figure 3.5. PackageElement and Class Element in the SVG4MTV meta model

In Fig 3.5 we can see some model elements we are going to need to make SVG representations of meta models. We have the PackageElement, the Class Element and the Connections. They are all subclasses of ModelElements that possesses an elementId. Two elements cannot have the same id. The elementId allow to identify each Model Element in a unique and easy way. It is specially useful for the Connection to identify the source and the target classes with their ids as we will see further. The PackageElement is associated by two rectangles and the little one will contain the Package name. The ClassElement is associated to a rectangle that will be the bigger of the class. It is the rectangle that marks the bounds of the class element. This big rectangle is composed by two other small rectangles: the first rectangle is the one that will contain the class name and the second one is going to contain all the attributes of the class element. This can be seen in Fig. 3.6. The big rectangle is represented in the figure with red stroke. The first rectangle for the name of the class is represented with a light yellow color and the second rectangle for the attributes of the class is represented with a light green color.

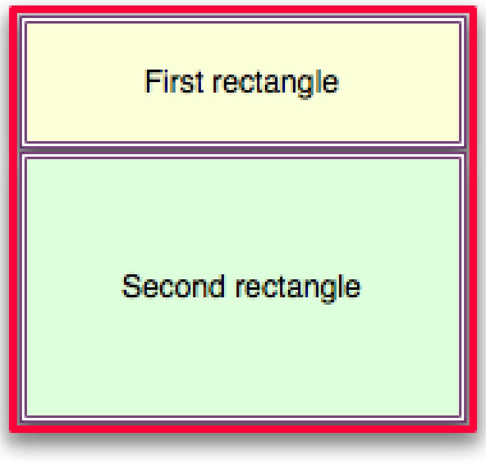


Figure 3.6. Composition of Class Elements in terms of rectangles

We can now have a look to the Connection element. The Fig. 3.7 . There are four types of relations between classes in UML class diagrams:

- Associations;
- Aggregations;
- Compositions;
- Generalizations.

We can now have a closer look to how we represent each one of the relations in terms of SVG4MTV elements. It is important to see that every connection has the id of the source model element (that could be a class element or a package) and the id of the target model element.

Associations : each association is represented by an AssociationPolyLine which represents the line of the association, and two AssociationEnds, one for the source and one for the target;

Aggregations : to represent an aggregation, we need a AggregationPolyline and two AggregationEnds as for the the associations, but we need more : we have an AggregationPolygone that is used to represent the not filled rhombus that is the UML aggregation symbol;

Compositions : the compositions are very similar to the aggregations in terms of SVG4MTV model elements. The only difference is that the symbol for composition is a filled rhombus.

Generalizations : for generalizations, we don't need associations ends, and the symbol for this relation is a not filled triangle.

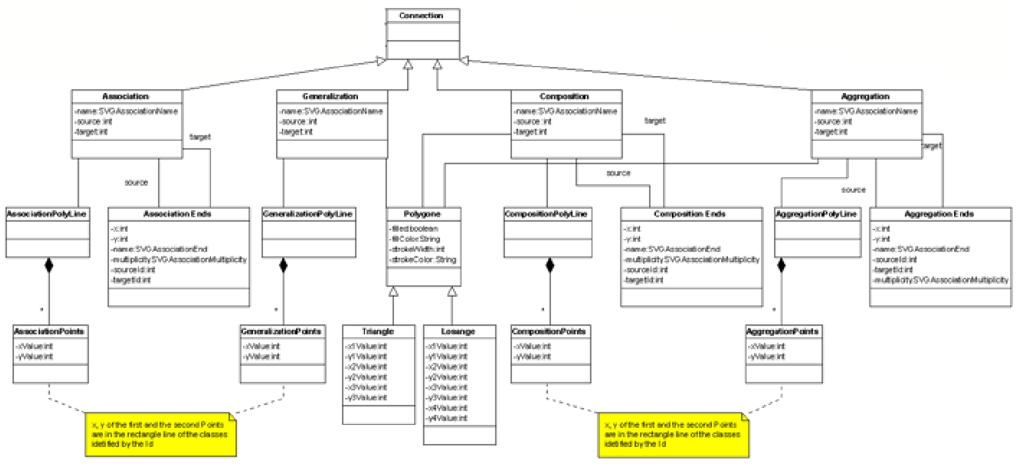


Figure 3.7. Connections in the SVG4MTV meta model

Finally, in Fig. 3.8 we can see all the text elements needed to represent all information that will be placed on the two-dimensional graphic representation. We have defined different types of SVGText elements as the SVG-ClassName for the text of the name of the classes with the aim of giving the possibility of personalizing each text element. The SVG font type, size and color can be different for each text. Indeed, for example, the text for the associations ends is usually smaller than the one for the class names of for the attributes of a class.

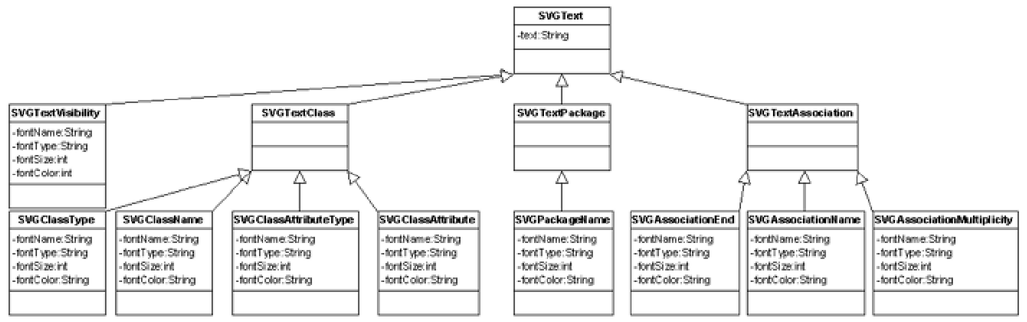


Figure 3.8. Text elements in the SVG4MTV meta model

3.3 Spacial distribution of the meta model elements

Now that we defined the SVG4MTV meta model, we have all the tools that will allow us to perform the two transformations described at the beginning of this chapter. Indeed, we just have to parse the XMI description of the meta model we want to browse and to create an instance of the SVG4MTV meta model containing all the information of the XMI file. But we are now confronted to a complex problem. In the input file, we have information about the meta model itself as the existing classes with their names and attributes or their generalization or associations relations. But, we don't have any information about spacial placement of the elements of our meta model representation !

We saw in the previous section that all the SVG4MTV model elements must have x and y positions. We will have to calculate all the positions of the elements of our 2D graphic representation of the browsed meta model! And the problem of setting the positions of the classes, of the connections, of the associations ends and of all the needed elements is that we need to have them in a way that will not produce superposition of elements. Basically we need to have a suitable algorithm to place UML classes in a 2D canvas. One possible solution is to place all the elements in a random way on a grid and to make the connections. But this is not sufficient and very uninteresting in terms of UML class diagrams. We also need to minimize the crossing edges of the graph too. And for showing an UML style diagram, we need to try to have some sense as having the more abstracted classes on the top.

All this problematic has been solved using the Sugiyama algortihm [22]. The Sugiyama algorithm allow to calculate a placement of a hierarchical graph in order to reduce the edge crossing. It helps to produce readable diagrams. Let's talk a bit about the Sugiyama algorithm. Here are the main goals of the algorithm :

- to have a hierarchical layout of vertices;
- to minimize the crossing edges;
- to make the connected vertices to be closer together and optimize the load balancing.

Here you can see the general steps of the algorithm :

Step 1 : Form a proper hierarchy by injecting nodes; this means that if we have the situation presented in Fig. 3.9, we will inject nodes in order to consider always only two levels of the graph as we will see in next step;

Step 2 : Permute vertices on each level to minimize the crossing edges; this step is explained further;

Step 3 : Assign horizontal positions in order to have closer connected vertices and maximal edge balancing;

Step 4 : Remove dummy nodes.

Let's see how we permute the vertices on each level in order to reduce the crossing edges. We have to introduce the following terminology :

Up-barycenter : is the average of the parent positions;

Down-barycenter : is the average of the child positions.

The position can either be index of the x value of the node position. You can take a look to the Fig. 3.10 to see how to calculate the barycenter's values. The algorithm will in a first phase, sort level per level with up and down barycenter the nodes in order to reduce the crossing edges. If we arrive in a situation where there is no more crossings, the algorithm stops. If not, we iterate the sort till a predetermined number of iterations. The second phase consists in swapping the nodes witch have the same barycenter value, level per level. If the situation is better after the swap, we keep the swap and continue. We do this till we have no more crossing or till a fixed number of iterations. Then, if needed, we have to reorder horizontally the nodes in order to optimize the load balancing but without creating crossing. And after removing the eventually existing dummy nodes, the algorithm takes end.

Following the good advice of not reinventing the wheel, we don't have implemented this algorithm because we found a Java library very useful for our work. This API allows us to generate a JGraph [23] of our meta model. JGraph API use an implementation of the Sugiyama algorithm. Using it we can get the positions of the elements we are going to use in our SVG representation. Calculating the width and height of our classes and having their positions is all what we need to produce our SVG file.

Then the second step that consists on making the transformation of the SVG4MTV file to the SVG file can be done easily, because it is basically a simple one-to-one transformation.

3.4 The browser options

Having the SVG representation of our meta model, there is still to think about the options our 2D Meta Model Browser offers. The functionalities of zooming in and out and exporting to image file format options. This can be done using Batik as we saw previously.

Another very important thing that rests to be described is the option of selecting a class of the diagram and to be able to move it to a new position. For this we need to know using the SVG4MTV instance witch are the connections linked to the class we are moving and to know witch is the situation of the classes and of the connection concerned by the movement. All this problematic and the algorithm used to do this will be presented on chapter 5 witch is about Implementation.

All the needed concepts are now introduced and we have all in our possession to develop our browser. Lets see now the Analysis and Design document the will allow to better understand what exactly we need to implement.

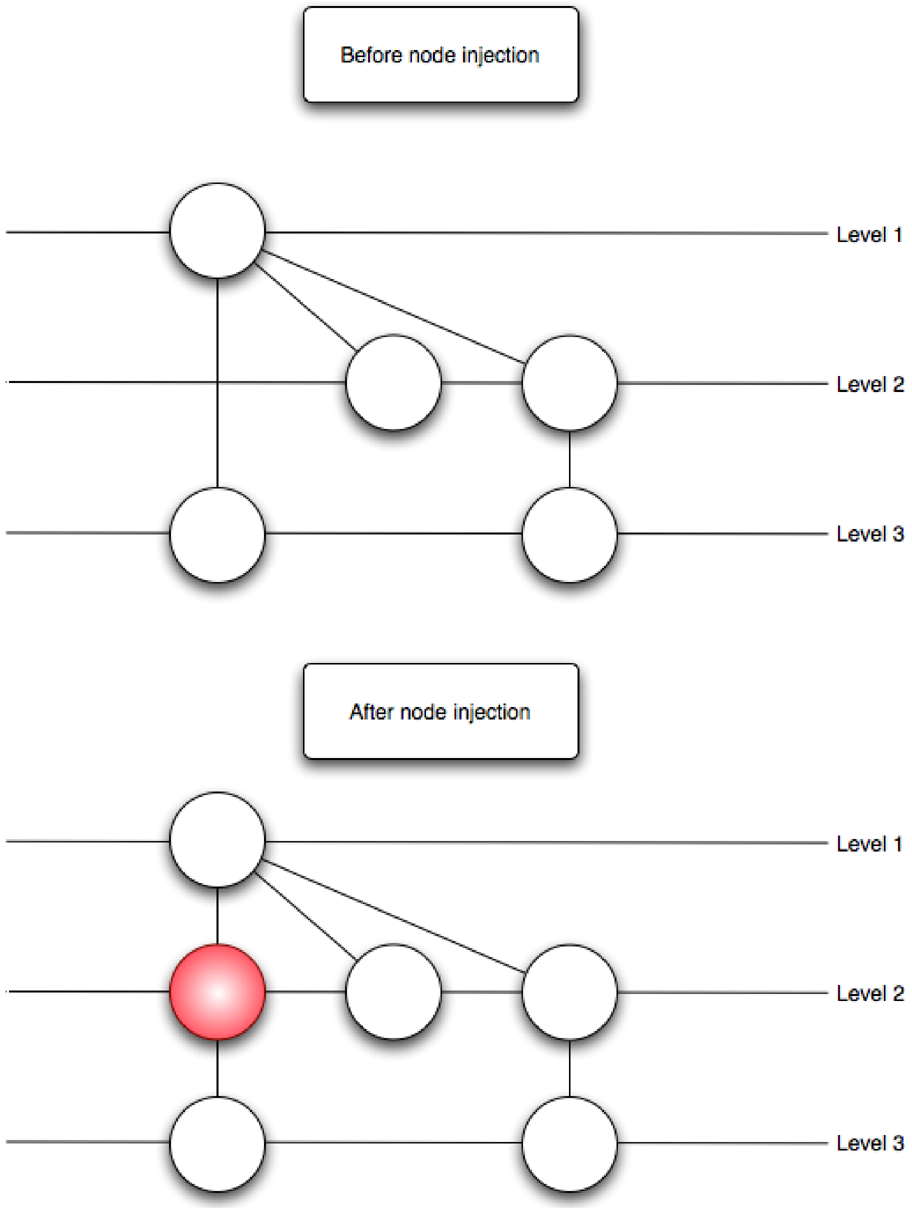


Figure 3.9. Injection of nodes - Sugiyama algorithm step 1

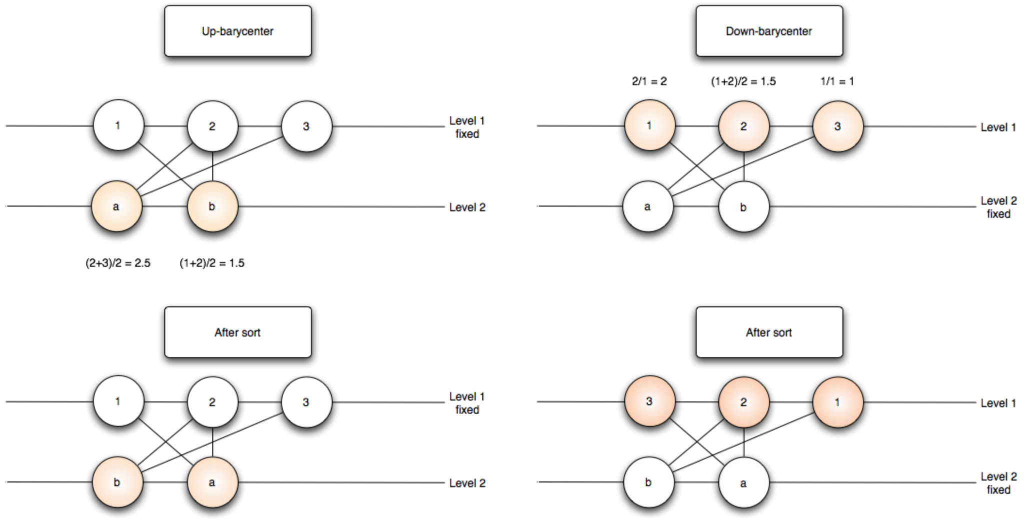


Figure 3.10. Permute vertices to minimize crossing edges - Sugiyama algorithm step 2

Chapter 4

Analysis

This chapter presents all the work done for the Analysis phase of the project. In the first section I am presenting the Use Cases and their actors. Then are showed and explained the following UML diagrams : the Use Cases Diagram, the Activity Diagram and the State Diagram. This chapter presents trough the analysis documents what is expected of our browser and how he will do the various tasks.

4.1 The actors of the use cases

In this section are presented all the actors of the use cases for the 2D Meta Model Browser system.

4.1.1 Primary Actors

The User : use the MTV to browse metamodels.

4.1.2 Secondary Actors

JMI / EMF API : allows to access and manipulate the content of the Meta Data Repository;

MTVSVG Manager API : allows the MTV application to browse metamodels in an UML-Style 2D graphics way;

Batik SVG Generator : allows applications to generate and display SVG graphics;

Batik SVG DOM API : allows to define the logical structure of documents and the way a document is accessed and manipulated. It allows creating SVG documents;

Batik Transcoder API : allows to rasterize an SVG document fragment to a raster image such as JPEG, PNG or Tiff.

4.2 Use cases

In this section we present all the use cases for the 2D Meta Model Browser system.

4.2.1 Transform MOF / Ecore to SVG4MTV

Description :
This Use Case describes how MTV can transform the MOF or Ecore description of the metamodel to an SVG4MTV description of the metamodel.

Definition :
- Name : Transform MOF/Ecore to SVG4MTV.
- Trigger : MTVSVG Manager needs to do this transformation to browse a metamodel.
- Actors : MTVSVG Manager (primary)

Assumptions :
1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[MOF] and XMI[Ecore] files)

Event flow :
1. The Use Case starts when the user selects the option to graphically browse a meta model.
2. The System explores the MOF or Ecore meta model.

3. The System performs the transformation and produces an SVG4MTV file using the SVG4MTV Metamodel and the MOF or Ecore file and the Use Case ends.

4.2.2 Transform SVG4MTV to SVG

Description :

This Use Case describes how MTV can transform the SVG4MTV description of the metamodel to an SVG file.

Definition :

- Name : Transform SVG4MTV to SVG.
- Trigger : MTVSVG Manager finishes the Ecore / MOF to SVG4MTV transformation.
- Actors : MTVSVG Manager (primary)

Assumptions :

1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)
3. An SVG4MTV file has been produced.

Event flow :

1. The Use Case starts when the Ecore / MOF transformation has been finished.
2. The System explores the SVG4MTV file.
3. The System performs the transformation and produces an SVG file using the MTVSVG Manager and the Use Case ends.

4.2.3 Load/Reload the 2D representation of the metamodel

Description :

This Use Case describes how MTV can load/reload the 2D representation of the metamodel. This situation occurs when the user wants to browse a meta model and selects the corresponding option or wants to reload the graphical representation of the meta model selecting the button for reloading.

Definition :

- Name : Load/Reload the 2D representation of the meta model
- Trigger : MTVSVG Manager needs to load/reload the meta model when the user select the corresponding option.
- Actors : MTVSVG Manager (primary), Batik SVG Generator (secondary), Batik SVG DOM (secondary)

Assumptions :

1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)
3. The SVG file for the metamodel is available.

Event flow :

1. The Use Case starts when MTV needs to load/reload the metamodel representation.
2. The System explores the SVG file and parses it.
3. The System produces the DOM document of the SVG file.
4. The System displays the metamodel in the browser panel using Batik SVG Generator and the Use Case ends.

4.2.4 Browse metamodel in MTV with the 2D Meta Model Browser

Description :

This Use Case describes how a user can browse a metamodel with the 2D Meta Model Browser of the MTV application. This situation occurs when the user selects the option to browse the metamodel of the current project with the 2D Meta Model Browser.

Definition :

- Name : Use the 2D Meta Model Browser to display a metamodel in MTV in a 2D UML like graphic representation
- Trigger : The User selects the option to use the 2D Meta Model Browser of MTV

- Actors : User (primary), JMI / EMF (secondary), MTVSVG Manager (secondary), Batik SVG Generator (secondary), Batik SVG DOM (secondary)

Assumptions :

1. A project in the MTV is already open

2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)

Event flow :

1. The Use Case starts when the User selects the option to use the 2D Meta Model Browser of MTV

2. The System retrieves the metamodel from the MTV repository using JMI or EMF.

3. The System performs the transformation of the metamodel described in the XMI[MOF] file to SVG4MTV and save the produced file using the MTVSVG Manager.

4. The System performs the transformation of the SVG4MTV file to SVG and produces an SVG file using the MTVSVG Manager.

5. The System creates a DOM document using the Batik SVG DOM and loads the 2D graphics representation in the 2D Meta Model Browser panel using Batik SVG Generator.

6. When the 2D UML like representation of the metamodel is displayed, the Use Case ends.

Extensions :

3.a The System is unable to perform the tranformation.
- 1. The System writes an error message in the log window.
- 2. back in 1.

4.a The System is unable to produce the SVG file.
- 1. The System writes an error message in the log window.
- 2. back in 1.

4.2.5 Moving elements in the 2D graphical representation of the meta model

Description :

This Use Case describes how the user can change the position of the classes and associations of the 2D representation of the meta model. The goal is to obtain a better quality in the visibility of the graphical representation in the case of complex metamodels. This situation occurs when the user is in the process of browsing a metamodel with the 2D Meta Model Browser.

Definition :

- Name : Moving elements in the 2D graphical representation of the meta model
- Trigger : The User selects a graphical object of the 2D representation of the meta model
- Actors : User (primary), Batik SVG DOM (secondary), Batik SVG Generator (secondary)

Assumptions :

1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)
3. The option of graphically browsing the meta model was selected.
4. Use cases 4.2.3 and 4.2.4 were executed.

Event flow :

1. The Use Case starts when the User selects a graphical object of the 2D representation of the meta model by clicking on the elements that can be moved with the left mouse button and maintaining it clicked.
2. The System displays the selected object with a transparent style.
3. The User moves the cursor of the mouse to the desired position and releases the lefts mouse button.
4. The System displays again the moved objetc in the normal style at his new position and the Use Case ends.

4.2.6 Modify the 2D representation of the meta model: changing the level of details

Description :

This Use Case describes how a user can display more or less details in the metamodel representation. This situation occurs when the user presses the details configuration button.

Definition :

- Name : Modify the level of details of the 2D representation of the metamodel in the 2D Meta Model Browser
- Trigger : The User selects the button of the details option.
- Actors : User (primary), MTVSVG Manager (secondary), Batik SVG Generator (secondary), Batik SVG DOM (secondary)

Assumptions :

1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)
3. The metamodel of the opened project was browsed with the 2D Meta Model Browser and the 2D representation is displayed in the MTVSVG pannel

Event flow :

1. The Use Case starts when the User selects the button of the details level option.
2. The System displays a menu with the different levels of detail that are possible for the metamodel.
3. The User selects the level of details he wants.
4. The System modifies the DOM document using the Batik SVG DOM.
5. The System reloads the metamodel and displays the elements of the metamodel with the selected details level using the Batik SVG Generator and the Use Case ends.

4.2.7 Zoom IN and OUT the meta model representation

Description :

This Use Case describes how a user can change the Zoom level to obtain a suitable size of the graphical representation of the metamodel depending on the size of the metamodel. This situation occurs when the user uses the zoom IN or OUT options.

Definition :

- Name : Zoom IN and OUT the meta model representation
- Trigger : The User uses the more or less Zoom option of the 2D Meta Model Browser.
- Actors : User (primary), Batik SVG Generator (secondary)

Assumptions :

1. A project in the MTV is already open
2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)
3. The metamodel of the opened project was browsed with the 2D Meta Model Browser and the 2D representation is displayed in the browser panel

Event flow :

1. The Use Case starts when the User selects the more or less Zoom option.
2. The System reloads the metamodel and displays it adapting the size of the elements to the desired zoom degree and the Use Case ends.

4.2.8 Save the modified 2D representation of the metamodel

Description :

This Use Case describes how a user can save the modifications done on the graphical representation of the metamodel. This situation occurs when the user selects the save metamodel representation button. Usually this is needed after moving elements of the meta model (Use case 4.2.5).

Definition :

- Name : Save the 2D representation of the metamodel in a SVG Document.
- Trigger : The User selects the save metamodel representation button.
- Actors : User (primary), MTVSVG Manager (secondary), Batik SVG DOM (secondary), Batik SVG Generator (secondary)

Assumptions :

1. A project in the MTV is already open

2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)

3. The metamodel of the opened project was browsed with the 2D Meta Model Browser and the 2D representation is displayed in the browser panel

Event flow :

1. The Use Case starts when the User selects the save model representation button.

2. The System displays an alert box and informs the User that this operation will replace the previous representation.

3. The System asks the User to confirm the operation of saving the new representation

4. The System copies the SVG4MTV in a backup file.

5. The System updates the SVG4MTV file with the new positions of the objects using the MTVSVG Manager.

6. The System displays the message SVG representation saved in the log panel.

7. The System performs the transformation to SVG and produces the SVG file.

8. The System reloads the metamodel and the Use Case ends.

Extensions :

3.a The User click the cancel button.

- 1. exit.

5.a The System is unable to produce the SVG file.

- 1. The System writes an error message in the log window.

- 2. back in 1.

4.2.9 Export the 2D representation of the metamodel to an image format

Description :

This Use Case describes how a user can export the metamodel representation to an image. The user can choose an image format as png or jpeg. This situation occurs when the user selects the export model representation button.

Definition :

- Name : Export the 2D representation of the metamodel to an image format.

- Trigger : The User selects the export button.

- Actors : User (primary), MTVSVG Manager (secondary), Batik Transcoder (secondary)

Assumptions :

1. A project in the MTV is already open

2. To the project a metamodel is available (XMI[Ecore] and XMI[MOF] files)

3. The metamodel of the opened project was browsed with the 2D Meta Model Browser and the 2D representation is displayed in the browser panel

Event flow :

1. The Use Case starts when the User selects the export model representation button.

2. The System verifies that the metamodel representation displayed is saved in the SVG file.

3. The System asks the User the path where he wants to get the exported file.

4. The User selects the desired format of the exported file selecting it in a list.

5. The User selects the confirm export button.

6. The System transcodes the SVG file to the desired data format and creates the file using the Batik Transcoder.

7. The System informs the User that the export is success and that the file has been created and the Use Case ends.

Extensions :

2.a The last modifications of the metamodel representation have not been saved.

- 1. The System asks the User to save the metamodel representation.

- 2. go to 3.

3.a The User gives a bad path.

- 1. The System informs the User that the path is not correct displaying a message in the log panel.

- 2. back in 3.
6.a The System is unable to export the metamodel representation.
- 1. The System displays an error message in the log panel.
- 2. exit.

4.3 The Use Cases Diagram

The use cases diagram can be seen in Fig. 4.1. It's important to remember that this system is part of the MTV system. The MTVSVG Manager is the actor that makes the link between the MTV system and the 2D Meta Model Browser System.

4.4 The Activity Diagram

An Activity Diagram is a type of diagram in the UML. Activity diagrams represent the business and operational workflows of a system. An Activity Diagram shows the overall flow of control. In UML 1.x, an Activity diagram is a variation of the UML State diagram where the "states" represent operations, and the transitions represent the activities that happen when the operation is complete. The UML 2.0 Activity diagram, while similar looking to the UML 1.x Activity diagram, now has semantics based on Petri nets.

As it can be seen in the activity diagram from Fig. 4.2, the activities are divided in six columns that represents who is responsible of the concerned activities. The following list, presents a description of each one of the columns.

Column 1 - The User : The user starts the activity diagram when he clicks on the button for 2D meta model browser to open the graphic representation of the meta model he is working on; This button is part of the options available in the MTV framework; Then he is also responsible of the choice of witch button to click to activate a browser option as moving elements or saving the SVG representation;

Column 2 - The MTVSVG Manager : The MTVSVG Manager is responsible of the transformations; It has to deal with the backup files from different versions of the graphical representation;

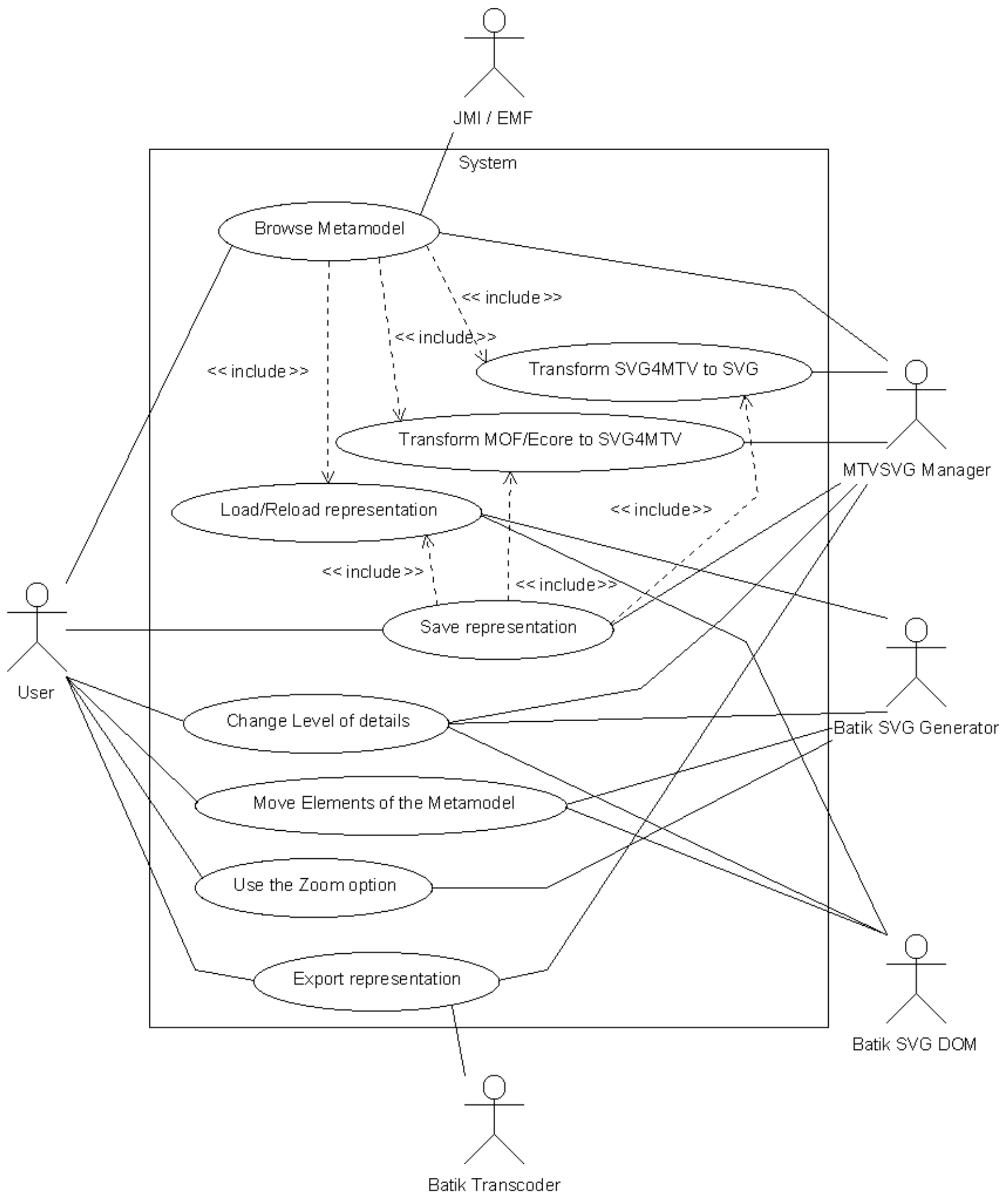


Figure 4.1. Use Case Diagram

Column 3 - The JMI /EMF : The JMI for MOF or EMF for Ecore is responsible of retrieving the XMI meta model's description file to our browser; When this is done, the MTVSVG Manager can perform the transformations;

Column 4 - The SVG DOM : The SVG DOM produces the DOM document for the SVG file and allows the dynamic move of the elements of the graphic representation of the meta model;

Column 5 - The SVG Generator : The SVG Generator allows to load the SVG DOM document and to update the SVG file;

Column 6 - The Batik Transcoder : The Transcoder is responsible for the transcoding of our SVG files to image file formats as png, jpg, etc..;

For example, here is a example of scenario of what can be a use of the browser in terms of activity. The user click to open the 2D Meta Model Browser. Then the JMI / EMF retrieve the meta model from its repository. At this point, the MTVSVG Manager will perform the two transformations to produce the SVG graphical description of the meta model representation. In order to display it, the SVG DOM must produce the SVG DOM document in memory. The SVG Generator can now load the SVG document and display it in a panel. At this step, the system needs a decision from the user. If he is happy with the graphical representation of the meta model, he can for example click on the export button and select the image format he wants. Then, the MTVSVG Manager sends the export command to the Batik Transcoder. The Batik Transcoder transcodes the SVG to the image format. The image is produced and we come back to the user decision's point. He can use all the browser option he wants, but now he has his image of the graphical representation of the meta model and wants to add it to his report, so he presses the Quit button. The activity's oriented scenario takes end now.

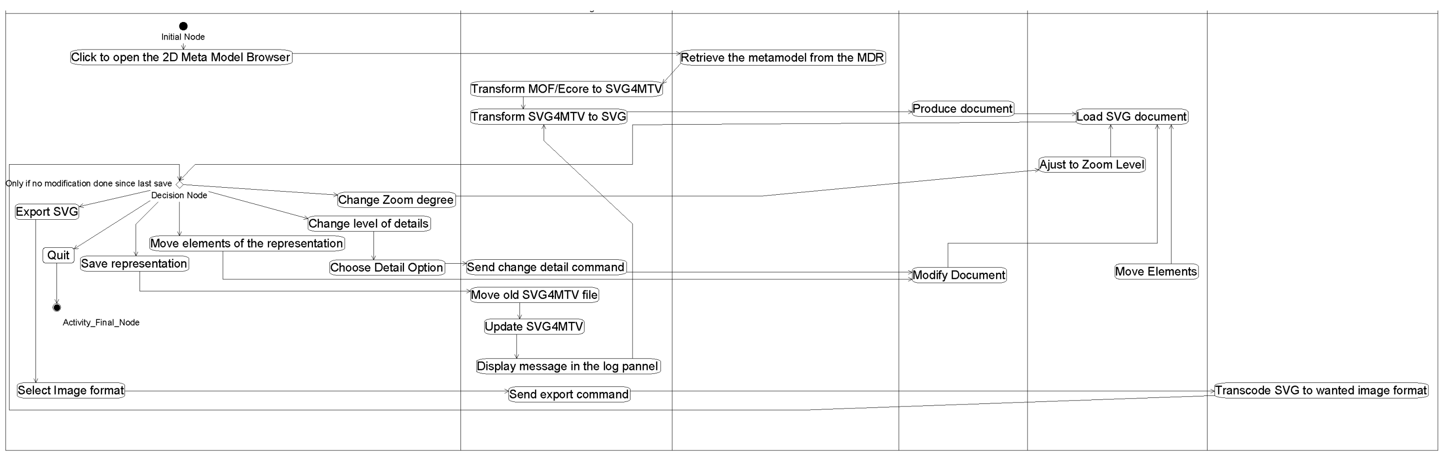


Figure 4.2. Activity Diagram

4.5 The State Chart Diagram

The UML state diagram is essentially a state diagram with standardised notation that can describe a lot of things, from computer programs to business processes. In this case it describes the different states of our browser. You can find the State Chart Diagram in Fig. 4.3.

We see that these are the important states our browser will have when browsing a meta model :

- MOF or Ecore file loaded;
- SVG4MTV file created that corresponds to the result of the first transformation;
- SVG created that corresponds to the result of the second transformation;
- SVG parsed;
- SVG DOM Document created;
- Meta model representation displayed.

Then after this, depending on the choice of the user, the browser can be in the following states :

- Exporting SVG to an image file format;
- Modifying the meta model graphic representation;
- Saving the meta model representation.

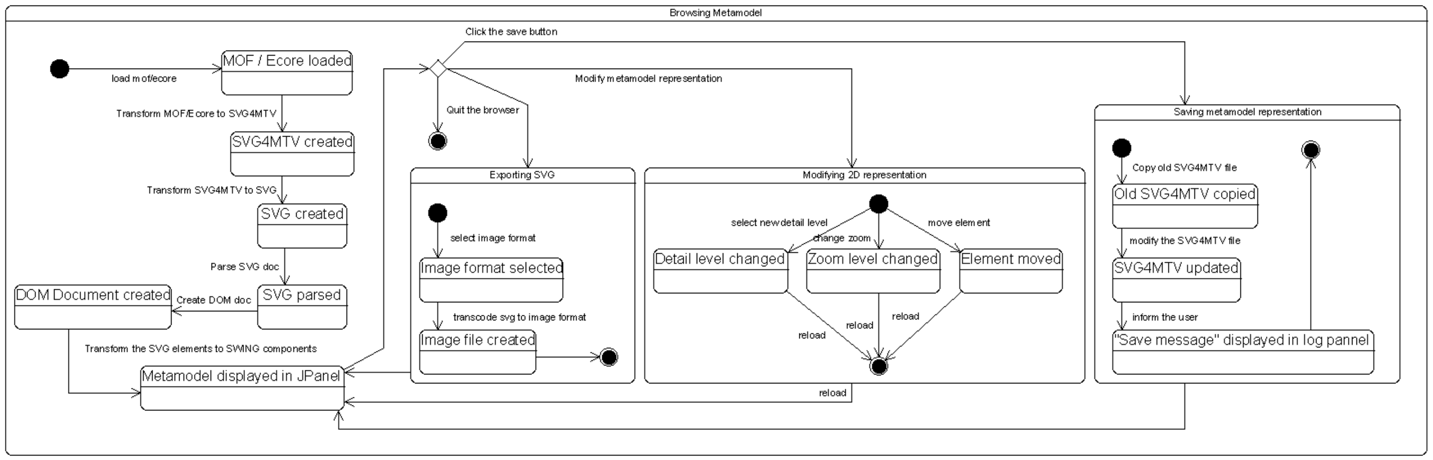


Figure 4.3. State Diagram

Chapter 5

Implementation

This chapter makes a presentation of the main implementation aspects as well as the architecture's choices. It also explains the algorithms used in the browser for placing the elements.

5.1 The transformation manager

As seen in the previous chapter, a very important part of the project is the implementation of the two transformations that allows to get an SVG description giving a MOF / Ecore meta model representation.

As it can be seen in Fig. 5.1, we have a class named TransformationManager. The TransformationManager is used to create instances of Transformations. We used a Singleton Patern to be sure to have only one instance of the TransformationManager which has a transformationList. This list is composed by all the instances of Transformation we need.

Let us show in more detail now the abstract class Transformation. The performTransformation method is abstract and is implemented in a subclass. This allows to have as many subclasses as we need different types of transformations. In our case, we just need two subclasses[1], one for each type of transformation :

- transformation from Ecore description of the meta model to SVG4MTV file;

[1]althought MOF was one of the goals of this project, it was not possible to realize it in the time slot that the project was for.

- transformation from SVG4MTV to SVG.

In terms of classes, we have one subclass called ECOREtoSVG4MTV and another one called SVG4MTVtoSVG. Both having as attributes an input and an output file name, witch are the names of the correspondent Ecore, SVG4MTV and SVG files read or produced when we used the 2D meta model browser in MTV. The performTransformation method is implemented at this level.

Each transformation will use a set of objects with different utilities to help them to perform their transformation. There is the XMILoader that allows to load an XMI based file and to get its resource. There is also the EcoreMetaModelBrowser and the SVG4MTVBrowser that allows to get the elements contained in the correspondent XML based files. We have also implemented two other classes called SVG4MTVProducer and SVGProducer that are used to create the corresponding transformed elements. Finally there is the PositionCalculator that uses the Sugiyama algorithm returns the 2D positions of the classes. More details regarding the implementation details will be provided in the following sub-sections.

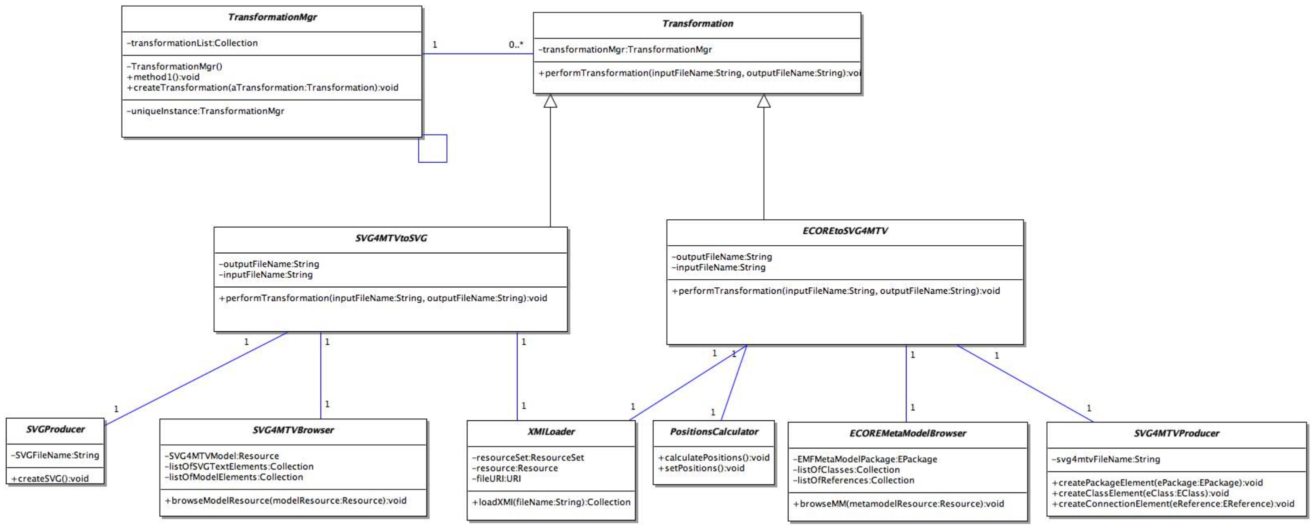

TransformationMgr
-transformationList:Collection
-TransformationMgr()
+method1():void
+createTransformation(aTransformation:Transformation):void
-uniqueInstance:TransformationMgr
1
0..*
Transformation
-transformationMgr:TransformationMgr
+performTransformation(inputFileName:String, outputFileName:String):voi
SVG4MTVtoSVG
-outputFileName:String
-inputFileName:String
+performTransformation(inputFileName:String, outputFileName:String):void
ECOREtoSVG4MTV
-outputFileName:String
-inputFileName:String
+performTransformation(inputFileName:String, outputFileName:String):void
SVGProducer
-SVGFileName:String
+createSVG():void
SVG4MTVBrowser
-SVG4MTVModel:Resource
-listOfSVGTextElements:Collection
-listOfModelElements:Collection
+browseModelResource(modelResource:Resource):void
XMILoader
-resourceSet:ResourceSet
-resource:Resource
-fileURI:URI
+loadXMI(fileName:String):Collection
PositionsCalculator
+calculatePositions():void
+setPositions():void
ECOREMetaModelBrowser
-EMFMetaModelPackage:EPackage
-listOfClasses:Collection
-listOfReferences:Collection
+browseMM(metamodelResource:Resource):void
SVG4MTVProducer
-svg4mtvFileName:String
+createPackageElement(ePackage:EPackage):void
+createClassElement(eClass:EClass):void
+createConnectionElement(eReference:EReference):void

Figure 5.1. Transformation Manager Class Diagram

5.2 Transformation sequence

In this section, it is presented the global algorithm used to perform the transformations.

Below, it is the step-by-step algorithm for the abstract performTransformation method :

Step 1 - Loading resource : we need to load the corresponding resource for the input file;

Step 2 - Browsing the resource : we browse the resource to get the elements contained in it; we store the elements in lists;

Step 3 - Executing the transformation : this step has little differences for the two transformations:

- for the first one, we need to calculate the positions of the elements and to create the corresponding SVG4MTV elements;
- for the second one, it's easier because we just need to make a one-to-one transformation between SVG4MTV and SVG.

Step 4 - Creating the output file : finally we create the output file to end the transformation; at this step, we will have an SVG4MTV file or a SVG file.

5.2.1 From Ecore to SVG4MTV

In this subsection we present the algorithm used to perform the transformation witch takes as input an Ecore file and produces the corresponding description of the meta model we want to browse. This description is in fact an instance of the SVG4MTV meta model. Let us show the commented code for the performTransformation method.

We see in this listing that the performTransformation method takes as input the name of the input file we want to transform (in this case the Ecore file) and the name of the output we will have to produce (the SVG4MTV file). Let's see how this method behaves.

From the input file we first get its resource. To do this we use the loadXMI method of the XMILoader object. This method returns the resource of the Ecore file. This resource allows the ECOREMetaModelBrowser to browse the meta model (with the browseMM method). This operation will fill the attribute of the ECOREMetaModelBrowser object used to store all the elements in a Vector list.

The SVG producer is then initialized with the init method. That creates the SVG4MTV resource.

After the initialization process, we need to calculate the positions of the elements. For this we use a JGraph and the Sugiyama algorithm. The PositionCalculator class is used to accomplish this. We set the font size of the text elements from a configuration file. We then create a JGraph from the list of elements we parsed before and we stored in the ECOREMetaModelBrowser object. We then use the Sugiyama layout for this JGraph. When this is done, we use the setCalculator method of the SVG4MTVProducer to set the sizes and positions of the elements. The positions of the elements are getted by using the generated JGraph. Knowing for each node of the JGraph its centered position and the number of attributes, we can calculate the position of every element of the classes of the browsed meta model. For example, for a class element, knowing its center position in the JGraph, we can calculate:

- the width of the class element by multiplying the number of chars of the longest string's attribute line by the default font size;
- the height of the class element by multiplying the number of attribute's line plus one (for the class name) by the default font size.

Having the center, the width and the height, we have all we need for calculating the position of the class name, of the attributes and of the rectangles forming the class element.

At this step, we can start to create the elements of our SVG4MTV file. Using an iterator trough the Vector list of EClasses of the ECOREMetaModelBrowser object, we create for each EClass the corresponding SVG4MTV class element. For this we use the createClassElement of our SVG4MTVProducer.

At this point, we still do not have created the associations between the classes. We use an iterator on the list of EReferences an use the createConnectionElement method to create the corresponding SVG4MTV connection element. We have to be careful not to insert the same connection twice, since in the Ecore file the same reference is referred twice if we look for the reference browsing the ECore elements. For example, an EReference between the A and B classes will appear twice looking for the connections of the A class and then looking for the connections of the B class. Indeed, the same

EReference is referred in a EClass and in its EOpposite. For the SVG draw of the connection, we do not need to draw two connections for the same association, even if it's bidirectional association.

The generalizations are not expressed in terms of EReferences in Ecore files. They are expressed as ESuperTypes for the classes. That's why we treat them separately calling the method createConnectionElementGeneralization of the SVG4MTVProducer.

Finally, when all the elements are transformed to SVG4MTV elements, we call the createSVG4MTVFile method to produce the SVG4MTV file witch has as name the outputFileName parameter of the performTransformation method.

Listing 5.1. Code for the performTransformation method for the ecore to SVG4MTV

```
    public void performTransformation(String inputFileName,
        String outputFileName) {
          ...
          // Loading the resource
          Resource metamodelResource = lnkXMILoader.loadXMI(
              inputFileName);
          // Browsing it
          lnkECOREMetaModelBrowser.browseMM(metamodelResource);
          lnkSVG4MTVProducer.init(outputFileName);
          // Creating the Sugiyama 2D graphic representation and
              getting the positions
          lnkPositionsCalculator.setDefaultFontSize(
              lnkSVG4MTVProducer.getDefaultFontSize());
          lnkPositionsCalculator.doSugiyamaAlgorithm(
              lnkPositionsCalculator.createJGraph(
              lnkECOREMetaModelBrowser.getListOfEClass()));
          lnkSVG4MTVProducer.setCalculator(lnkPositionsCalculator)
              ;
          // Creating a class element in SVG4MTV for every EClass
              found in the Ecore file by the browser
          for(Iterator itc = lnkECOREMetaModelBrowser.
              getListOfEClass().iterator();itc.hasNext();){
              lnkSVG4MTVProducer.createClassElement((EClass) itc.
                  next());
          }
          Vector refVisited = new Vector();
          // Creating the connections elements in SVG4MTV for
              every EReference found in the Ecore file
          for(Iterator itc = lnkECOREMetaModelBrowser.
            getListOfEReference().iterator();itc.hasNext();){
             EReference ref = (EReference) itc.next();
             if(!(refVisited.contains(ref) || refVisited.contains
                 (ref.getEOpposite()))) {
                 lnkSVG4MTVProducer.createConnectionElement(ref);
             }
             refVisited.add(ref);
        }
        // This is for the generalizations because in Ecore
            there is no EReference for generalizations
        for(Iterator itc = lnkECOREMetaModelBrowser.
            getListOfEClass().iterator();itc.hasNext();){
             lnkSVG4MTVProducer.
                 createConnectionElementGeneralization((EClass)
                 itc.next());
        }
        // Creating the SVG4MTV file
        lnkSVG4MTVProducer.createSVG4MTVFile();
    }
```

5.2.2 From SVG4MTV to SVG

The algorithm used to perform the transformation which takes as input an SVG4MTV file and produces the corresponding SVG file is very similar to the one explained in the previous subsection: nevertheless, this transformation is much more easier. We don't need to calculate any positions because we are just taking the elements of the SVG4MTV instance and producing the corresponding SVG elements. The SVG4MTV elements have their position set during the first transformation and we just take them and express it in SVG syntax.

An example of the different type of file describing the meta models can be seen in Chapter 3 (Problem approach) on Section 3.2 (The SVG4MTV meta model). You can see for the same little example of meta model its representation in the Ecore XMI formalism, the result of the first transformation to SVG4MTV file, and the result in SVG description. The Chapter 6 (Case study and story boards) presents another example with a Petri Nets meta model.

5.3 Browsing the resources

We saw in the previous section that we need to browse the Ecore meta model. For the second transformation, we also need to browse the SVG4MTV instance. In this section both algorithms are presented.

5.3.1 The Ecore meta model

As we saw it in the section about the Ecore meta model, an ecore file contains the description of a meta model. This description is done in an EPackage that contains all EClasses and EReferences for a given meta model. In Listing 5.3.1 we can see the algorithm for browsing an ecore file.

The browsePackage method works as follows. Its parameter is the package which is of type EPackage. The goal of this method is to browse the package of the Ecore file and to fill some Vector list with the Ecore elements found.

- We first start with an iterator on the list of EClassifiers of the package. We add each EClassifier to the list containing all the elements of the package (called listOfModelElements). Then, we are going to fill some more specialized list as the list of EClasses or the list of EReferences;
- If the EClassifier is an instance of EClass, then we can cast it to EClass and add it to the list of EClasses called listOfEClass. Then, knowing that an EClass can posses some EAttributes, we search for them using another iterator. That's the way we fill the listOfEAttribute. An EClass can have also some EReferences, so we want to add them to the listOfEReference. We just make some test to be sure not to take the same Ereference twice;
- If the EClassifier is not an instance of EClass, we also check if it is instance of EEnum or of EDataType. If so, we add these elements to the corresponding list. At this stage of the project, we never use them, but for creating more advanced features for the browser it will be useful to have this information stored.

Listing 5.2. Code for browsing the meta model described in an ecore file.

```
    private void browsePackage(EPackage aPackage) {
        // now we have the package loaded, so we want to browse
            it
        // we get all the EClassifiers contained in the package
        for(Iterator iter = aPackage.getEClassifiers().iterator
            (); iter.hasNext();){
            EClassifier classifier = (EClassifier) iter.next();
            // adding the classifier to the list
            listOfModelElements.add(classifier);
            // if it is a class, we add it to the list of
                classes
            if(classifier instanceof EClass) {
                EClass eClass = (EClass) classifier;
                listOfEClass.add(eClass);
                // we search for its attributes and we add them
                    to the corresponding list of elements
                for(Iterator aIter = eClass.getEAttributes().
                    iterator(); aIter.hasNext();){
                    EAttribute attribute = (EAttribute) aIter.
                        next();
                    listOfEAttribute.add(attribute);
                }
                // we search for its references (associations)
                    and we add them to the corresponding list of
                    elements
                for(Iterator rIter = eClass.getEReferences().
                    iterator(); rIter.hasNext();){
                    EReference reference = (EReference) rIter.
                        next();
                    // we just are careful because we doesn't
                        want to take the same reference twice
                    if(!listOfEReference.contains(reference.
                        getEOpposite())) {
                        if(reference.getEOpposite().
                            isContainment() || reference.
                            getEOpposite().getEContainingClass().
                            getESuperTypes().size()!=0) {
                             listOfEReference.add(reference.
                                 getEOpposite());
                        }
                        else {
                             listOfEReference.add(reference);
                        }
                    }
                }
            }
            // we search for enumerations and we add them to the
                corresponding list of elements
            else if(classifier instanceof EEnum) {
                EEnum eEnum = (EEnum) classifier;
                listOfEEnum.add(eEnum);
                for(Iterator eIter = eEnum.getELiterals().
                    iterator(); eIter.hasNext();){
                    EEnumLiteral literal = (EEnumLiteral) eIter.
                        next();
                    listOfEEnumLiteral.add(literal);
                }
            }
            // we search for data types and we add them to the
                corresponding list of elements
            else if(classifier instanceof EDataType){
                EDataType eDataType = (EDataType) classifier;
                listOfEDataType.add(eDataType);
            }
        }
    }
```

5.3.2 The SVG4MTV instance

Browsing the SVG4MTV instance is an easy task. When we have the resource, we just need to get all the elements it contain. Afterwork, we need to look for each element if it is an instance of a ClassElement or if it is an instance of a Connection. Depending on what type of instance we have, we can fill the corresponding list of SVG4MTV elements (list of ClassElement or list of ConnectionElement).

5.4 Executing the transformation

This section is divided in two subsections. The first one refers to the first transformation, the second one for the transformation from SVG4MTV to SVG.

5.4.1 From ecore elements to SVG4MTV elements

Although this transformation is not very hard, there is a lot of small things to implement, and it becomes complex very fast. To execute this transformation, we use the SVG4MTVProducer class. Here are the main methods of this class :

- init(String svg4mtvFileName);
- createClassElement(EClass eClass);
- createConnectionElement(EReference eReference);
- createConnectionElementGeneralization(EClass eClass).

The init(String svg4mtvFileName) method : is used to open the file where we are going to place the instance of the SVG4MTV meta model. But it is also used to get all the default values for things such as font type, font size, etc.. All this values are read in a text configuration file witch is called "configMTVBrowser.txt". These parameters are stored in this external file. This allows the user to set his own preferences. Indeed there is a functionality in the browser that allow the user to see the configuration parameters and to change them. Please refer to the Chapter 6 (Case study and story boards);

The createClassElement(EClass eClass) method : as input we have a EClass comming from the ecore file. We are going to express this EClass in terms of SVG4MTV. It is a ClassElement and we will have to set and define all the needed information. Here is the algorithm used:

Step 1 - Getting the SVG4MTV factory : as explained previously, we defined a meta model called SVG4MTV and all set of Java interfaces have been generated using the Together Architect for Eclipse; now, in the aim of creating an instance of the SVG4MTV meta model, we need to get the SVG4MTV factory as shown in the Listing 5.4.1;

Step 2 - Creating the ClassElement : the SVG4MTV ClassElement is created and we set the id for it; the id allows us to easily identify and get the class elements when we will work on the moving of classes of the meta model;

Step 3 - Creating the ClassRectangles : we create big Rectangle first and then two smaller ones; for this we need to get their position calculated by the PositionsCalculator and we have still to calculate their width and height; the first small rectangle is for the name of the class and the other small rectangle is for the attributes;

Step 4 - Creating the SVGClassName : this is for setting the text of the class name, its position, and its default font type, size and color.

Step 5 - Creating the SVGClassAttribute : this is for setting the text of all the attributes of the class, their position, and their default font type, size and color as for the class name.

Listing 5.3. How to get the SVG4MTV factory and to use it.

```
        // Getting the factory
        SVG4MTVFactory factorySVG4MTV = SVG4MTVPackage.eINSTANCE
            .getSVG4MTVFactory();
        // Create a new SVG4MTV class element
        ClassElement aClassElement = factorySVG4MTV.
            createClassElement();
```

We must now come back to the Step 3 for creating the rectangles. In order to better precise it :

- We have the position of the big rectangle calculated by the Sugiyama algorithm, but we doesn't know the width and the height of it. The width is calculated taking in account the length of the bigger text line of the class (witch can be the class name or one of the attributes). This biggest length is multiplied by the default font size and there we have the width;
- For the height, we just count the number of lines we need (the class name plus all the attributes of the class) and we multiply it by the default font size;
- All the positioning of the texts in the big rectangle and of the two little rectangle in the big one are also calculated here.

The createConnectionElement methods : we just saw how to create the class elements, but still one important thing to do: to create the connection elements. As already explained, the Ecore EReferences are not representing the generalization connections. The generalization are represented using the Super Types on the EClasses. So let us see know the createConnectionElement(EReference eReference) method that is used for creating the simple associations and the compositions. The first thing to do is to make the difference between an EReference representing an association and another representing a composition. You can see in the Listing 5.4.1 how this is done.

Listing 5.4. How to make the difference between an EReference representing an association and another representing a composition.

```
    public void createConnectionElement(EReference eReference) {
        // this method allows to create a normal association or
            a composition
        // the generalization is treated in an another method
            because there is no eReference for it
        if(eReference.isContainment()) {
            createComposition(eReference);
        }
        else {
            createAssociation(eReference);
        }
    }
```

The two methods for creating an association and a composition are very similar to one for creating the class elements. We create the corresponding connection elements, and all we need for it (association ends, poly line, etc..). The only difference is the connection termination.

Finally the method for creating the generalizations createConnectionElementGeneralization(EClass eClass) is similar too. The only difference is that we have as input an EClass and we get its Super Types. Knowing the two classes, it is clear how to create the poly line and where to position the triangle of the generalization relation.

5.4.2 From SVG4MTV to SVG elements

This transformation is easier than the previous one. It is a simple one-to-one transformation. The only tricky thing is that we had to make SVG groups with the aim of facilitating the moving of some elements in the 2D SVG representation of the meta model browsed. To perform this transformation, we use the SVGProducer class. Here are the main methods of this class :

- init();
- createClass(ClassElement daClass);

- createConnection(Connection daConnection).

The init() method creates the SVG DOM document and place as background of the image a big white rectangle.

The createClass(ClassElement daClass) read all the elements of the ClassElement given as input and create using the Batik SVG Generator the correspondent SVG elements in the document. It is the same for the createConnection(Connection daConnection) method witch first identify the type of the connection (association, composition, generalization) and as for the classes create with the SVG Generator the SVG elements. As mentioned before, we just have to create the groups of SVG elements for make easier the moving functionality. Each group has an id witch is of one of those two formats :

- "class"+"id";
- "connection"+"id";

5.5 The User Interface of the 2D Meta Model Browser

It is important to remind that the 2D meta model browser will be integrated to the MTV framework. So when we are speaking of the user interface of our browser, we have to study how is the user interface of the existing MTV software. You can find in Stephane Heck Bachelor's report [4] some screen shots and explanations about the user interface of MTV.

The 2D meta model browser will take place in a panel and will have some buttons for actions as "Loading the meta model", "Using the zoom option" or "Exporting the diagram to an image format". In the main panel the SVG diagram will be displayed.

For the moment, the 2D meta model browser is done with a JFrame witch contains a space for the buttons on top, the SVG representation of the meta model in the main center panel, and a status information bar at the bottom of the JFrame. The SWING API and Batik are used for displaying the SVG. This is done in the SVGBrowse class.

Indeed we used the Batik JSVGCanvas for displaying the SVG document for the corresponding file. We also use all the habitual listener's "arsenal" for knowing witch button is used by the user and to fire the corresponding event.

5.6 Moving elements

Here we are going to provide a description of what we have realized in the context of moving the meta model elements in the SVG canvas. Fig 5.2 is an illustration of what is expected to be done. In this little example we have a set of two classes being associated and the expected result after moving them. From this example we can see that the action of moving a class element can be realized in a two step algorithm :

Step 1 : Drag and drop the class element to his new position;

Step 2 : Update the positions of the connections affected by the move and of the association ends.

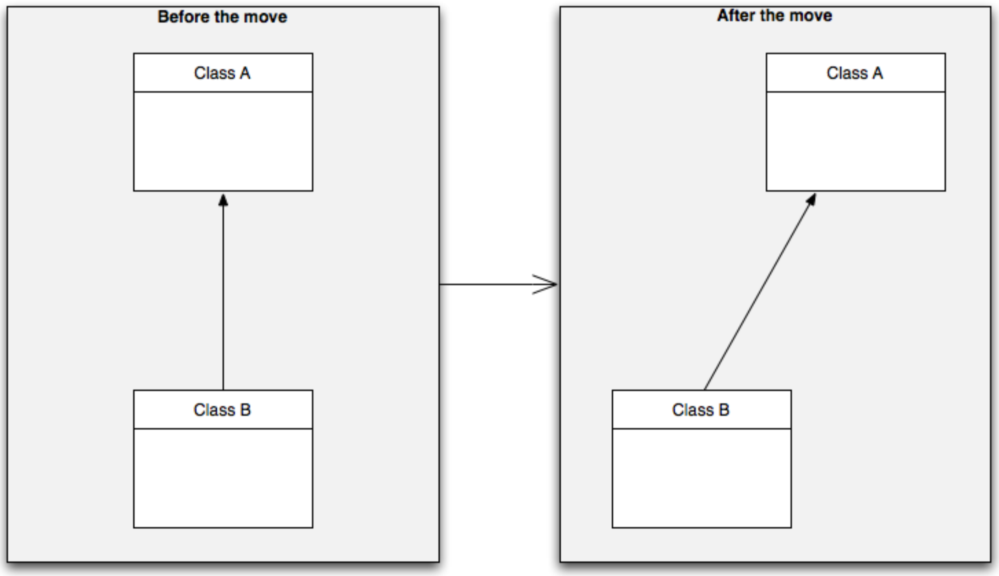


Figure 5.2. Example of the moving of a class element

These two main steps will be explained in more details in the next two subsections.

5.6.1 Drag and drop the class element

The important question here is: how can we move a class element from a point to another. Here are the logical steps we have to execute :

Identify if the user clicked on a class element : this means to have a listener for the click event and to check if we are clicking on a valid element;

Know which class element has been clicked : this is done by checking the position where the user clicked;

Look where the user wants to put the class element : by the time user releases the mouse button the position is marked;

Perform the corespondent actions to set the new positions : this is done with the perform action methods of the listeners.

In the implementation, we had then three important event listeners. Here are the three corresponding classes :

- OnDownAction;
- OnMoveAction;
- OnUpAction;

Let's see now each one of these classes with some more details. Here is the summary of what is done in the OnDownAction class for moving a class element. This is done when the user clicks on a class element.

Step 1 - Getting the id of the selected group : first we get the element clicked using the event target feature; then, using a method called get-GroupID(Element theElement) we climb back in the tree for finding to witch group the clicked element belongs to; this allows to know the id of the group that we are going to move;

Step 2 - Storing the elements of the selected group : in this step we store in a Java Vector called selectedItems all the elements (rectangles and texts) of the class group; this will be useful for the operations that will be done during the move and after the release of the mouse button;

Step 3 - Modify style of the selected element : this is for styling with a different color or style the selected element;

Step 4 - Turning the drag flag to true : useful for the OnMoveAction to know if the user clicked;

Step 5 - Getting the position of the clicked point : we get the coordinates of the point where the user clicked and store it in a initialDragPoint attribute.

In this class we have also things done for the connection positions updating, but we will see it in the next subsection. Let's now focus on the OnMoveAction class. After the user clicked a class element and the previous actions were performed, we can see what's done when the user move the cursor without releasing the mouse button.

Step 1 - Verifying if there is an element to drag : this is done with a simple if condition looking for the drag boolean flag to be true; if drag is false, we do nothing;

Step 2 - Getting the actual position : we get the coordinates of the new position of the cursor and store them in a variable;

Step 3 - Modifying the positions of the dragged elements : we update the positions of all the elements stored in the selectedItems Vector filled in the OnDownAction.

As for the OnDownAction, we have in this class some actions done for the connection updating that we are also going to discuss in the next subsection. We can now finally see what is done in the OnUpAction.

Step 1 - Setting back the original style for the dragged element : we set back his original color or style to the element we have moved;

Step 2 - Turning the drag flag to false : this has to be done to say that the dragging action is finished and put all in conditions for a new drag and drop operation.

All this part of the drag and drop implementation has been realized using some documentation in the Batik website and with a contributor that I want to thank here:

Andres Toussaint for his "Batik Drag Tutorial: How to move shapes in the Canvas" whitch has been very useful for implement the drag and drop feature of the developed 2D meta model browser.

5.6.2 Update the positions of the connections

At this point of the report, we saw how to drag and drop a group of elements of the SVG 2D meta model graphic representation. But we didn't focused yet in : updating the positions of the connections and of the associations ends of the moved class. This was another important piece of the interesting puzzle of this Bachelor's project. We needed an algorithm that allows the user to move the class elements if the automatic placement produced by the Sugiyama algorithm is not the appreciate by the user. This can occurs for some complex meta models or with some graphs.

We had to find a solution that allows the user to place the classes but keeping some rules applied by the Sugiyama algorithm as trying to reduce the number of crossing for the connections.

First attempt : solution rejected

First, a solution witch was not bad, but not good enough was implemented. As can be seen in Fig 5.3, we can take into consideration four situations. a) The one having the moved class drop at the top of the connected and not moving class. b) Another on the left, c) the third on the right d) and finally one on the bottom. Knowing in which situation we are for the connection, allow us to know where the point of the path of the connection we are going to move must be placed on the perimeter of the dragged class. The first problem of this solution is that, knowing in witch face to place the point of the connection, we still don't know exactly how to place it. And more problematic is the fact that, if there is more than one connection to update, we will have the problem of placing the various points on the same face. The solution for this was to calculate the position on the face for the point with equation in Fig 5.4.

The big problem we had with this, is that we respect no more the rule of reducing the crossing of the connection. This was really problematic and motivate me to find another solution.

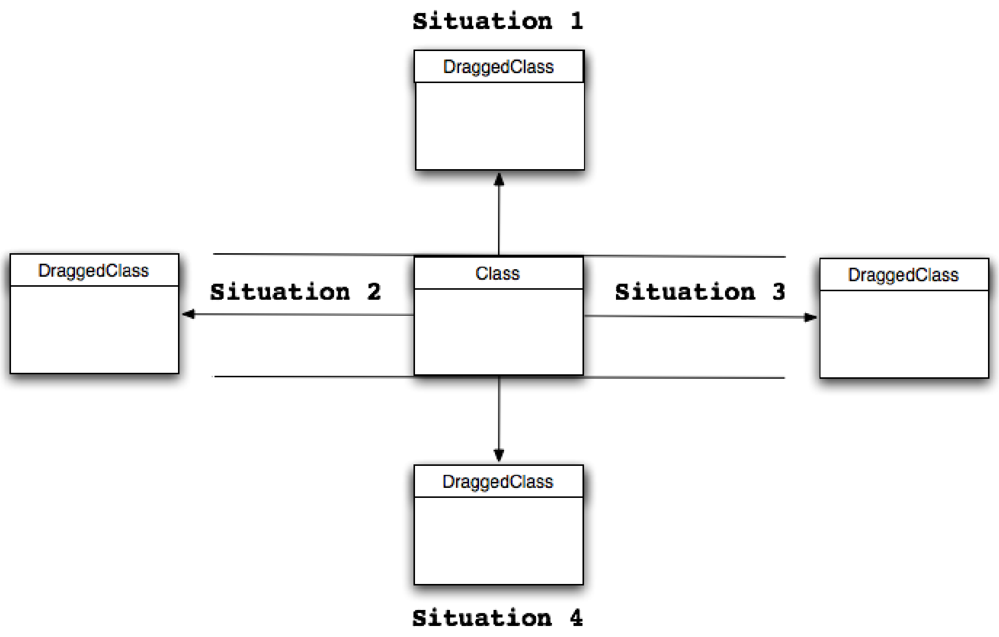
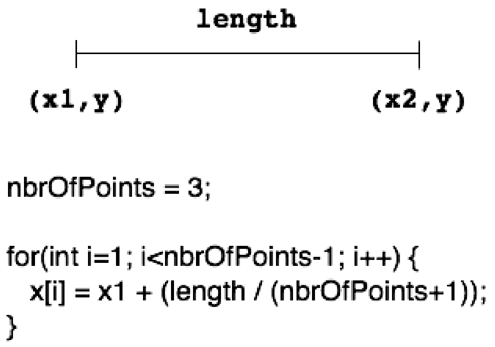


Figure 5.3. Rejected solution

Second attempt : a better solution

The previous option was not good enough and problematic. Another better solution was found that allows to preserve the rule of minimizing the crossing witch was the more important feature to keep. The idea is not really complex, even if it's not easy to explain in a text. So I will do my best to explain you this algorithm. Let's go !

The very first thing we have to do is to get all the ids of the connections that are involved in the operation of moving a class element. Once this is done, we will have to know for each connection which one of the two extremities of the association will be updated. Remind that each connection group contains a path (formed by at least two points), associations ends, and in case of working with a composition or a generalization a polygon. How can we know which one of the extremities of a path will be updated ? The answer is logical : we will have to update the point that is the nearest of the initial drag point, witch is positioned in the biggest rectangle of the class element we want to move. You can look to the Listing 5.6.2 to see how we calculate which point we will have to move. This is all done in the OnDownAction



```
nbrOfPoints = 3;

for(int i=1; i<nbrOfPoints-1; i++) {
  x[i] = x1 + (length / (nbrOfPoints+1));
}
```

Figure 5.4. Numeric formula used in rejected solution. Example for placing 3 points of connection on the face of a class element

class.

Listing 5.5. Code for knowing witch point of a connection we will have to update during a class move.

```
...
// Getting the ids of the connections of the dragged class
Vector theConnectionsIDs = getTheConnections(selectedId);
int j = -1;
// Getting the information to know witch extremity of the
    connection we have to update for each connection concerned by
     the move of the class
for(Iterator itCon = theConnectionsIDs.iterator(); itCon.hasNext
    ();) {
    j++;
    String cur = (String) itCon.next();
    Element movingConnection = document.getElementById("
        connection"+cur);
    if(movingConnection!=null) {
        connectPoint = new boolean[movingConnection.
            getElementsByTagName("path").getLength()];
        // Normally, a path has only two points, but we could
            imagine to have a path with more points in further
            developments of the browser
        for (int i1=0; i1<movingConnection.getElementsByTagName("
            path").getLength();i1++) {
            SVGOMPathElement path =  (SVGOMPathElement)
                movingConnection.getElementsByTagName("path").item(
                i1);
            // Getting the 2 extreme points of the path of the
            polyline
        SVGPoint point1 = path.getPointAtLength(0);
        SVGPoint point2 = path.getPointAtLength(path.
            getTotalLength());
        // We have to know wich one we will change
        // For this, we are going to check wich of the 2 points
             is the nearest point of the moving class
        Point pointClass = new Point();
        pointClass.setLocation(initialDragPoint.getX(),
            initialDragPoint.getY());
        double distance1 = calculDistance(point1,pointClass);
        double distance2 = calculDistance(point2,pointClass);
        if(distance1>=distance2) {
            connectPoint[i1] = true; // p1 doesn't move
        }
        else {
            connectPoint[i1] = false; // p2 doesn't move
        }
        // adding the information for this connection in the
            list witch contains the information for all the
            connections
        connectPointVector.add(j,connectPoint);
      }
    }
}
...
// Method for calculating the distance between two points using
    the Pythagorian rules
private double calculDistance(SVGPoint point, Point pointClass)
    {
    if(((point.getX()-pointClass.getX()))*((point.getX()-
        pointClass.getX())) + ((point.getY()-pointClass.getY()))
        *((point.getY()-pointClass.getY()))!=0){
        return Math.sqrt( ((point.getX()-pointClass.getX()))*((
            point.getX()-pointClass.getX())) + ((point.getY()-
            pointClass.getY()))*((point.getY()-pointClass.getY()))
            );
    }
    else {
        return 0;
    }
}
```

Now that we know which point of each association we will have to move, in other words witch extremity of the association is affected by the move of the class element, we have to find a way to calculate on which side of the perimeter of the class we are going to place the new points of the connections. All this part is implemented in the OnMoveAction class.

With the aim of being as clear as possible, the algorithm is divided in five main steps. All these steps have to be done for each connection concerned by the move.

Step 1 - Identifying the quadrant : we know the position of the top-left corner point of the dragged class, its width and its height; so we can calculate the center point of the class; afterthis we want to know in which quadrant is situated the extremity of the connection that will not be touched, in other words the point that is the more far from the dragged point; this can be done easily with some if conditions; the Fig 5.5 illustrate this step;

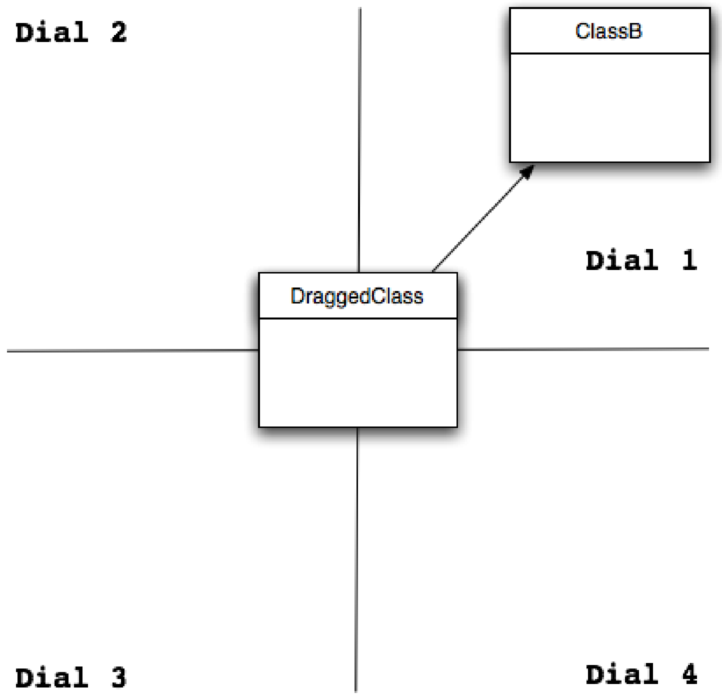


Figure 5.5. Position of the dragged class compared to the quadrants

Step 2 - Calculating the diagonal line : now that we know in which quadrant we are, we are going to calculate the coefficients of the diagonal line; we can calculate them because we know that the equation for this type of line is y = ax + b and because we have for each quadrant two points of the line - for each quadrant we can take the center point plus one of the four corner point of the class depending on the quadrant; this calculation is shown in the Listing 5.6.2;

Listing 5.6. Code for calculating the coefficients of diagonal line.

```java
private double[] calculPente(int cadran, SVGOMPoint
    dragpt, int width, int height) {
     double[] result = new double[2];
     Point pBorder = new Point();
     // We have the same diagonal for each couple of
         dials so we can take the same points
     if(cadran==1 || cadran==3) {
          float x = (float) (dragpt.getX() + width);
          float y = (float) (dragpt.getY());
          pBorder.setLocation(x,y);
     }
     if(cadran==2 || cadran==4) {
          float x = (float) (dragpt.getX());
          float y = (float) (dragpt.getY());
          pBorder.setLocation(x,y);
     }
     Point pCenter = calculCenterOfClass(dragpt,width,
         height);
     double pente;
     // Now we have the two points of the line so we can
         calculate the a of the y = ax + b with this
         equation for P1(x1,y1) and P2(x2,y2) the two
         points
     // a = (y1-y2) / (x1-x2)
     if(pBorder.getX() - pCenter.getX()!=0) {
          pente = (pBorder.getY() - pCenter.getY())/(
              pBorder.getX() - pCenter.getX());
     }
     else {
          pente=1;
     }
     // Calculate the b element of : y = ax + b replacing
          the a calculate previously in the equation with
         one of the two points
     double b = pCenter.getY() - pente*pCenter.getX();
     result[0]=pente;
     result[1]=b;
```

```
            return result;
        }
```

Step 3 - Testing position compared to the diagonal : at this point, we can check if the point corresponding to the other extremity of the connection is situated above, below or on the diagonal; to know this we just calculate the point on the diagonal with the same x coordinate; we have then the y of the point and the y coordinate of the point on the diagonal which is equal to ax+b with the x of the point of the connection; after knowing this, we can know in which portion of the perimeter of the dragged class will be the nearest extremity of the connection as can be seen in Fig 5.6;

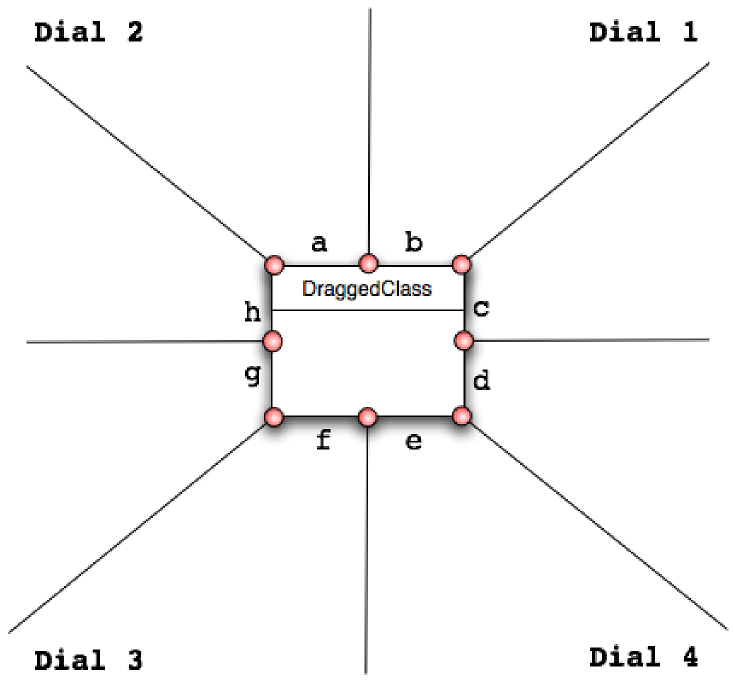


Figure 5.6. All the possible positions

Step 4 - Calculating the ratio : if the other extremity is on one of the quadrants limit or on one of the diagonal, the new position of the dragged extremity of the connection will go on the corresponding red point shown in the Fig 5.6; if not, we are going to place in the corresponding perimeter's part noted with letters in the Fig 5.6; but we still doesn't know exactly where; so we are going to calculate the ratio at the level of the other point and put the point on perimeter with the same ratio; a graphic will explain it better than thousands of words, so just take a look to the Fig 5.7;

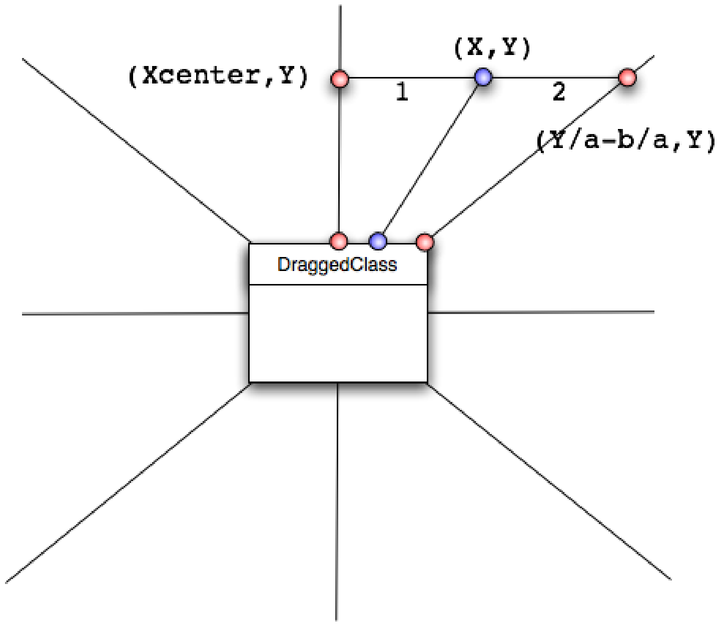


Figure 5.7. Calculating the ratio we know where to place the point on the perimeter of the class

Step 5 - Setting the new position : finally, having the ratio, we can set the new position of the nearest extremity of the connection compared to the dragged class..

We still have to place the dragged association end to its new position, that can be easily done now that we know where to place it. This solution allows us to move the class elements as the user wants, and to still have a placement of "good" quality. What should also be mentioned is that we choose to change only the dragged part of the connection witch can create some situation in witch the connection cross on its class. But this is not a problem because we just need to move a bit the other class to move the other part of the connection. A possible optimization is to do this automatically moving the two extremities of the connection if needed.

5.7 Exporting the SVG representation

The exporting SVG to an image file format is implemented in the SVGBrowse class. There is a button and has an action listener that uses the Batik Transcoder to perform the export. The produced file is called ”out” and has the corresponding file extension.

5.8 Changing the configuration file

As for the exporting option, we have a button that allows the user to access to the configuration manager. It is a JFrame showing the content of the configuration file with information as default font type, size, etc.. We can change these values. This is also implemented using SWING API and listeners.

Chapter 6

Case study and story boards

In this chapter, we are going trough a case study to show how to use in a concrete case the 2D Meta Model Browser. The case chosen is about a meta model for the Petri Nets. One of the objectives of this chapter is to show how to use the browser developed during this Bachelor work.

6.1 Petri Nets

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical representations of discrete distributed systems. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. As such, a Petri net has place nodes, transition nodes, and directed arcs connecting places with transitions. Petri nets were invented in 1962 by Carl Adam Petri in his Ph.D thesis. An example of a simple Petri Net can be seen in Fig. 6.1.

A Petri net consists of places, transitions and directed arcs. Arcs run between places and transitions - not between places and places or transitions and transitions. The places from which an arc run to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition.

Places may contain any number of tokens. A distribution of tokens over the places of a net is called a marking. Transitions act on input tokens by a process known as firing. A transition is enabled if it can fire, i.e., there are tokens in every input place. When a transition fires, it consumes the tokens from its input places, performs some processing task, and places a specified number of tokens into each of its output places. It does this atomically, i.e. in one single non-preemptible step.

Execution of Petri nets is nondeterministic. This means two things:

1. multiple transitions can be enabled at the same time, any one of which can fire;

2. none are required to firethey fire at will, between time 0 and infinity, or not at all(!) i.e. it is totally possible that nothing fires at all.

Since firing is nondeterministic, Petri nets are well suited for modeling the concurrent behavior of distributed systems.

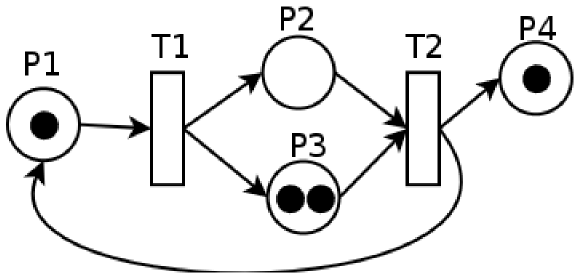


Figure 6.1. A simple Petri Net

6.2 Together Architect for Eclipse

Together is a product line from Borland that integrates a Java IDE, which originally had its roots in JBuilder with a UML modelling tool.

The product line comes in various levels of functionality, called Together Developer, Together Designer, and Together Architect. Earlier versions of the Together products were completely proprietary self-contained applications, whereas from the 2006 version onwards they are based on Eclipse. The installation even allows to install Together using an existing Eclipse installation.

Technically, Together is a set of Eclipse plugins. Together Developer provides UML 1.4 modelling, multilanguage support, physical data modelling, design patterns, source code design pattern recognition, code template design and reuse, documentation generation, and code audits and metrics. Together Designer adds language-neutral UML 2.0 diagramming, business process modelling, and logical data modelling, but has no physical data modelling. Borland Architect, the top product of the line, includes all features plus logical to physical data model transformation and custom pattern support.

In Fig. 6.2, you can see a meta model for the Petri Nets realized with Together Architect.

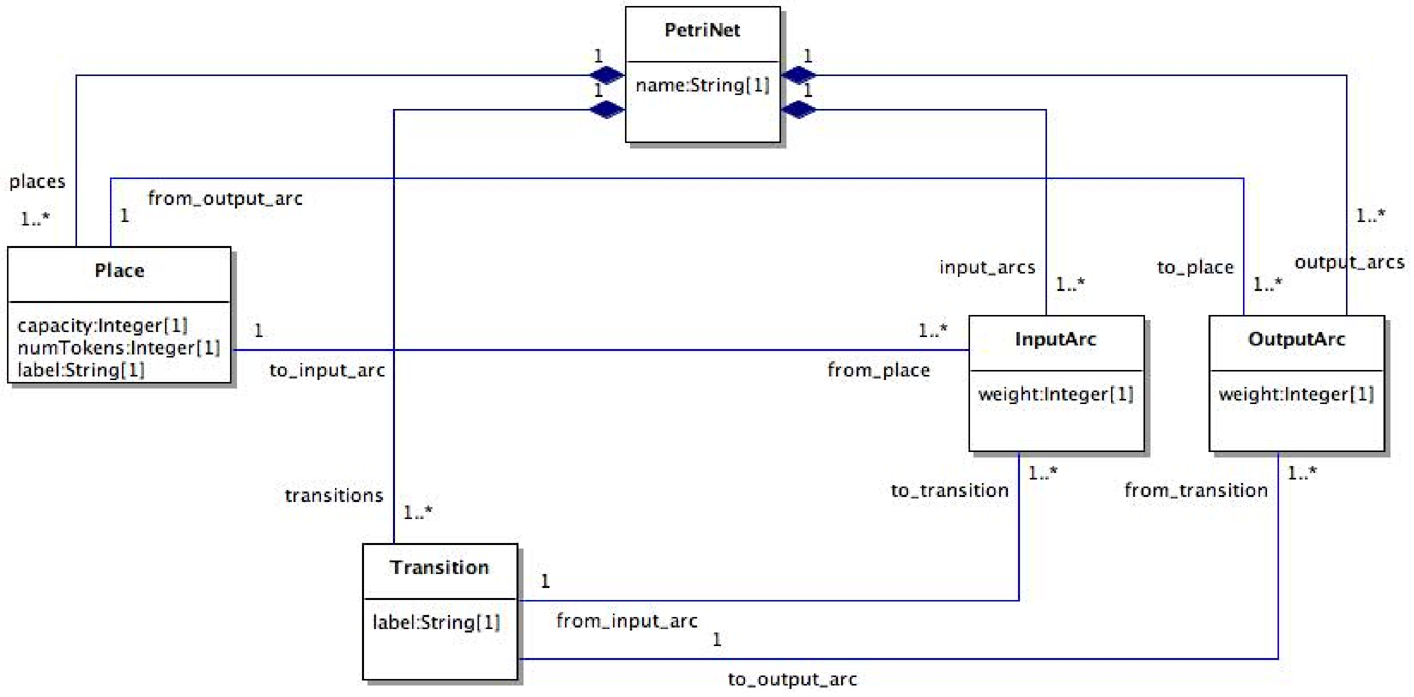


Figure 6.2. Meta model for Petri Nets done in Together Architect

6.3 UML2 XMI

Here is the UML2 file produced by Together Architect for the meta model of the Petri Nets.

Listing 6.1. UML2 XMI file for the Petri Nets

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML" xmi:id="
    _ELAOUFyJEduFw85wVo7zbQ" name="PetriNets" appliedProfile="
    _ELA1YFyJEduFw85wVo7zbQ">
  <packageImport xmi:type="uml:ProfileApplication" xmi:id="
      _ELA1YFyJEduFw85wVo7zbQ">
    <eAnnotations xmi:id="_ELA1YVyJEduFw85wVo7zbQ" source="
        attributes">
      <details xmi:id="_ELA1YlyJEduFw85wVo7zbQ" key="version"
          value="0"/>
    </eAnnotations>
    <importedPackage xmi:type="uml:Profile" href="PetriNets.
        profile.uml2#_EKpB8FyJEduFw85wVo7zbQ"/>
    <importedProfile href="PetriNets.profile.uml2#
        _EKpB8FyJEduFw85wVo7zbQ"/>
  </packageImport>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1Y1yJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELBci1yJEduFw85wVo7zbQ _ELCDnFyJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1ZFyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELA1bFyJEduFw85wVo7zbQ _ELCDg1yJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1ZVyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELBcdVyJEduFw85wVo7zbQ _ELCDhlyJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1ZlyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELCDilyJEduFw85wVo7zbQ _ELCDn1yJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1Z1yJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELBceFyJEduFw85wVo7zbQ _ELCDkVyJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1aFyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELA1b1yJEduFw85wVo7zbQ _ELBch1yJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
      _ELA1aVyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
      _ELA1clyJEduFw85wVo7zbQ _ELCDlFyJEduFw85wVo7zbQ"/>
  <ownedMember xmi:type="uml:Association" xmi:id="
```

```
    _ELA1alyJEduFw85wVo7zbQ" name="Unknown Name" memberEnd="
    _ELA1dVyJEduFw85wVo7zbQ _ELCqkVyJEduFw85wVo7zbQ"/>
<ownedMember xmi:type="uml:Class" xmi:id="
    _ELA1a1yJEduFw85wVo7zbQ" name="PetriNet">
  <ownedAttribute xmi:id="_ELA1bFyJEduFw85wVo7zbQ" name="
      input_arcs" type="_ELBcilyJEduFw85wVo7zbQ" association="
      _ELA1ZFyJEduFw85wVo7zbQ" aggregation="composite">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
        "_ELA1bVyJEduFw85wVo7zbQ" value="-1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
        _ELA1blyJEduFw85wVo7zbQ" value="1"/>
  </ownedAttribute>
  <ownedAttribute xmi:id="_ELA1b1yJEduFw85wVo7zbQ" name="
      places" type="_ELBcdFyJEduFw85wVo7zbQ" association="
      _ELA1aFyJEduFw85wVo7zbQ" aggregation="composite">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
        "_ELA1cFyJEduFw85wVo7zbQ" value="-1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
        _ELA1cVyJEduFw85wVo7zbQ" value="1"/>
  </ownedAttribute>
  <ownedAttribute xmi:id="_ELA1clyJEduFw85wVo7zbQ" name="
      output_arcs" type="_ELCDiVyJEduFw85wVo7zbQ" association="
      _ELA1aVyJEduFw85wVo7zbQ" aggregation="composite">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
        "_ELA1c1yJEduFw85wVo7zbQ" value="-1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
        _ELA1dFyJEduFw85wVo7zbQ" value="1"/>
  </ownedAttribute>
  <ownedAttribute xmi:id="_ELA1dVyJEduFw85wVo7zbQ" name="
      transitions" type="_ELCDl1yJEduFw85wVo7zbQ" association="
      _ELA1alyJEduFw85wVo7zbQ" aggregation="composite">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
        "_ELA1dlyJEduFw85wVo7zbQ" value="-1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
        _ELA1d1yJEduFw85wVo7zbQ" value="1"/>
  </ownedAttribute>
  <ownedAttribute xmi:id="_ELBccFyJEduFw85wVo7zbQ" name="name"
       visibility="package" isUnique="false">
    <eAnnotations xmi:id="_ELBccVyJEduFw85wVo7zbQ">
      <details xmi:id="_ELBcclyJEduFw85wVo7zbQ" key="
          isAttribute" value="true"/>
    </eAnnotations>
    <type xmi:type="uml:PrimitiveType" href="PetriNets.profile
        .uml2#_EKppCFyJEduFw85wVo7zbQ"/>
    <lowerValue xmi:type="uml:LiteralString" xmi:id="
```

```
          _ELBcc1yJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
  </ownedMember>
  <ownedMember xmi:type="uml:Class" xmi:id="
      _ELBcdFyJEduFw85wVo7zbQ" name="Place">
    <ownedAttribute xmi:id="_ELBcdVyJEduFw85wVo7zbQ" name="
        from_place" type="_ELBcilyJEduFw85wVo7zbQ" association="
        _ELA1ZVyJEduFw85wVo7zbQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
          "_ELBcdlyJEduFw85wVo7zbQ" value="-1"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
          _ELBcd1yJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELBceFyJEduFw85wVo7zbQ" name="
        to_place" type="_ELCDiVyJEduFw85wVo7zbQ" association="
        _ELA1Z1yJEduFw85wVo7zbQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
          "_ELBceVyJEduFw85wVo7zbQ" value="-1"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
          _ELBcelyJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELBce1yJEduFw85wVo7zbQ" name="
        capacity" visibility="package" isUnique="false">
      <eAnnotations xmi:id="_ELBcfFyJEduFw85wVo7zbQ">
        <details xmi:id="_ELBcfVyJEduFw85wVo7zbQ" key="
            isAttribute" value="true"/>
      </eAnnotations>
      <type xmi:type="uml:PrimitiveType" href="PetriNets.profile
          .uml2#_EKppCVyJEduFw85wVo7zbQ"/>
      <lowerValue xmi:type="uml:LiteralString" xmi:id="
          _ELBcflyJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELBcf1yJEduFw85wVo7zbQ" name="
        numTokens" visibility="package" isUnique="false">
      <eAnnotations xmi:id="_ELBcgFyJEduFw85wVo7zbQ">
        <details xmi:id="_ELBcgVyJEduFw85wVo7zbQ" key="
            isAttribute" value="true"/>
      </eAnnotations>
      <type xmi:type="uml:PrimitiveType" href="PetriNets.profile
          .uml2#_EKppCVyJEduFw85wVo7zbQ"/>
      <lowerValue xmi:type="uml:LiteralString" xmi:id="
          _ELBcglyJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELBcg1yJEduFw85wVo7zbQ" name="label
        " visibility="package" isUnique="false">
        <eAnnotations xmi:id=”_ELBchFyJEduFw85wVo7zbQ”>
          <details xmi:id=”_ELBchVyJEduFw85wVo7zbQ” key=”
              isAttribute” value=”true”/>
        </eAnnotations>
        <type xmi:type=”uml:PrimitiveType” href=”PetriNets.profile
            .uml2#_EKppCFyJEduFw85wVo7zbQ”/>
        <lowerValue xmi:type=”uml:LiteralString” xmi:id=”
            _ELBchlyJEduFw85wVo7zbQ” value=”1”/>
      </ownedAttribute>
      <ownedAttribute xmi:id=”_ELBch1yJEduFw85wVo7zbQ” type=”
          _ELA1a1yJEduFw85wVo7zbQ” association=”
          _ELA1aFyJEduFw85wVo7zbQ”>
        <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
            ”_ELBciFyJEduFw85wVo7zbQ” value=”1”/>
        <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
            _ELBciVyJEduFw85wVo7zbQ” value=”1”/>
      </ownedAttribute>
    </ownedMember>
    <ownedMember xmi:type=”uml:Class” xmi:id=”
        _ELBcilyJEduFw85wVo7zbQ” name=”InputArc”>
      <ownedAttribute xmi:id=”_ELBci1yJEduFw85wVo7zbQ” name=”
          from_input_arc” type=”_ELCDl1yJEduFw85wVo7zbQ”
          association=”_ELA1Y1yJEduFw85wVo7zbQ”>
        <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
            ”_ELBcjFyJEduFw85wVo7zbQ” value=”1”/>
        <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
            _ELBcjVyJEduFw85wVo7zbQ” value=”1”/>
      </ownedAttribute>
      <ownedAttribute xmi:id=”_ELBcjlyJEduFw85wVo7zbQ” name=”
          weight” visibility=”package” isUnique=”false”>
        <eAnnotations xmi:id=”_ELCDgFyJEduFw85wVo7zbQ”>
          <details xmi:id=”_ELCDgVyJEduFw85wVo7zbQ” key=”
              isAttribute” value=”true”/>
        </eAnnotations>
        <type xmi:type=”uml:PrimitiveType” href=”PetriNets.profile
            .uml2#_EKppCVyJEduFw85wVo7zbQ”/>
        <lowerValue xmi:type=”uml:LiteralString” xmi:id=”
            _ELCDglyJEduFw85wVo7zbQ” value=”1”/>
      </ownedAttribute>
      <ownedAttribute xmi:id=”_ELCDg1yJEduFw85wVo7zbQ” type=”
          _ELA1a1yJEduFw85wVo7zbQ” association=”
          _ELA1ZFyJEduFw85wVo7zbQ”>
        <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
            ”_ELCDhFyJEduFw85wVo7zbQ” value=”1”/>
        <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
```

```
          _ELCDhVyJEduFw85wVo7zbQ” value=”1”/>
    </ownedAttribute>
    <ownedAttribute xmi:id=”_ELCDhlyJEduFw85wVo7zbQ” name=”
        to_input_arc” type=”_ELBcdFyJEduFw85wVo7zbQ” association=
        ”_ELA1ZVyJEduFw85wVo7zbQ”>
      <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
          ”_ELCDh1yJEduFw85wVo7zbQ” value=”1”/>
      <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
          _ELCDiFyJEduFw85wVo7zbQ” value=”1”/>
    </ownedAttribute>
  </ownedMember>
  <ownedMember xmi:type=”uml:Class” xmi:id=”
      _ELCDiVyJEduFw85wVo7zbQ” name=”OutputArc”>
    <ownedAttribute xmi:id=”_ELCDilyJEduFw85wVo7zbQ” name=”
        to_output_arc” type=”_ELCDl1yJEduFw85wVo7zbQ” association
        =”_ELA1ZlyJEduFw85wVo7zbQ”>
      <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
          ”_ELCDi1yJEduFw85wVo7zbQ” value=”1”/>
      <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
          _ELCDjFyJEduFw85wVo7zbQ” value=”1”/>
    </ownedAttribute>
    <ownedAttribute xmi:id=”_ELCDjVyJEduFw85wVo7zbQ” name=”
        weight” visibility=”package” isUnique=”false”>
      <eAnnotations xmi:id=”_ELCDjlyJEduFw85wVo7zbQ”>
        <details xmi:id=”_ELCDj1yJEduFw85wVo7zbQ” key=”
            isAttribute” value=”true”/>
      </eAnnotations>
      <type xmi:type=”uml:PrimitiveType” href=”PetriNets.profile
          .uml2#_EKppCVyJEduFw85wVo7zbQ”/>
      <lowerValue xmi:type=”uml:LiteralString” xmi:id=”
          _ELCDkFyJEduFw85wVo7zbQ” value=”1”/>
    </ownedAttribute>
    <ownedAttribute xmi:id=”_ELCDkVyJEduFw85wVo7zbQ” name=”
        from_output_arc” type=”_ELBcdFyJEduFw85wVo7zbQ”
        association=”_ELA1Z1yJEduFw85wVo7zbQ”>
      <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
          ”_ELCDklyJEduFw85wVo7zbQ” value=”1”/>
      <lowerValue xmi:type=”uml:LiteralInteger” xmi:id=”
          _ELCDk1yJEduFw85wVo7zbQ” value=”1”/>
    </ownedAttribute>
    <ownedAttribute xmi:id=”_ELCDlFyJEduFw85wVo7zbQ” type=”
        _ELA1a1yJEduFw85wVo7zbQ” association=”
        _ELA1aVyJEduFw85wVo7zbQ”>
      <upperValue xmi:type=”uml:LiteralUnlimitedNatural” xmi:id=
          ”_ELCDlVyJEduFw85wVo7zbQ” value=”1”/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
            _ELCDllyJEduFw85wVo7zbQ" value="1"/>
      </ownedAttribute>
  </ownedMember>
  <ownedMember xmi:type="uml:Class" xmi:id="
      _ELCDl1yJEduFw85wVo7zbQ" name="Transition">
    <ownedAttribute xmi:id="_ELCDmFyJEduFw85wVo7zbQ" name="label
        " visibility="package" isUnique="false">
      <eAnnotations xmi:id="_ELCDmVyJEduFw85wVo7zbQ">
        <details xmi:id="_ELCDmlyJEduFw85wVo7zbQ" key="
            isAttribute" value="true"/>
      </eAnnotations>
      <type xmi:type="uml:PrimitiveType" href="PetriNets.profile
          .uml2#_EKppCFyJEduFw85wVo7zbQ"/>
      <lowerValue xmi:type="uml:LiteralString" xmi:id="
          _ELCDm1yJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELCDnFyJEduFw85wVo7zbQ" name="
        to_transition" type="_ELBcilyJEduFw85wVo7zbQ" association
        ="_ELA1Y1yJEduFw85wVo7zbQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
          "_ELCDnVyJEduFw85wVo7zbQ" value="-1"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
          _ELCDnlyJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELCDn1yJEduFw85wVo7zbQ" name="
        from_transition" type="_ELCDiVyJEduFw85wVo7zbQ"
        association="_ELA1ZlyJEduFw85wVo7zbQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
          "_ELCDoFyJEduFw85wVo7zbQ" value="-1"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
          _ELCqkFyJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="_ELCqkVyJEduFw85wVo7zbQ" type="
        _ELA1a1yJEduFw85wVo7zbQ" association="
        _ELA1alyJEduFw85wVo7zbQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id=
          "_ELCqklyJEduFw85wVo7zbQ" value="1"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="
          _ELCqk1yJEduFw85wVo7zbQ" value="1"/>
    </ownedAttribute>
  </ownedMember>
</uml:Model>
```

6.4 ECore XMI

You can see in Listing 6.4 the Ecore file generated by Together. This file is the one we are going to load in our 2D Meta Model Browser. For this, we just click on the ”Load” button of the browser. You can see in Fig. 6.3 a screen shot of the browser.

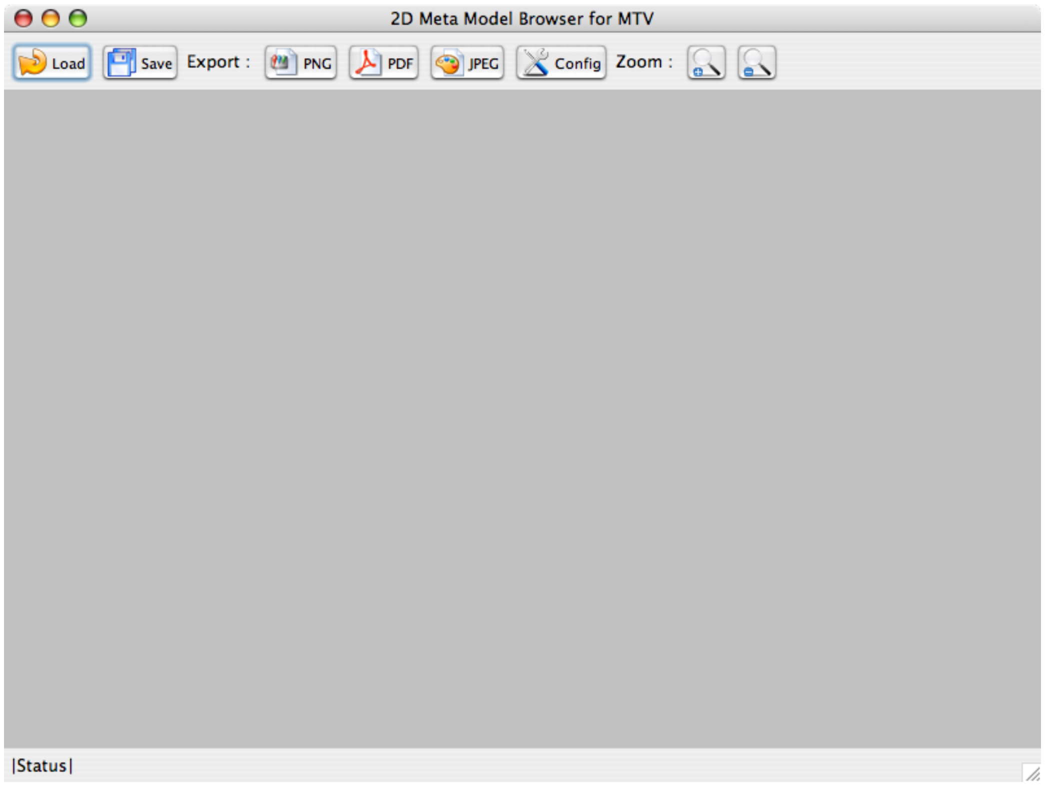


Figure 6.3. Screen shot of the 2D Meta Model Browser.

Listing 6.2. ECore XMI file for the Petri Nets

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.
        org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="
        PetriNets"
    nsURI="http:///PetriNets.ecore" nsPrefix="PetriNets">
  <eClassifiers xsi:type="ecore:EClass" name="PetriNet">
```

```
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        input_arcs" ordered="false"
         lowerBound="1" upperBound="-1" eType="#//InputArc"
             containment="true" eOpposite="#//InputArc/_"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        places" ordered="false"
         lowerBound="1" upperBound="-1" eType="#//Place"
             containment="true" eOpposite="#//Place/_"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        output_arcs" ordered="false"
         lowerBound="1" upperBound="-1" eType="#//OutputArc"
             containment="true" eOpposite="#//OutputArc/_"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        transitions" ordered="false"
         lowerBound="1" upperBound="-1" eType="#//Transition"
             containment="true" eOpposite="#//Transition/_"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
         ordered="false" unique="false"
         lowerBound="1" eType="ecore:EDataType http://www.eclipse
             .org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="InputArc">
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        from_input_arc" ordered="false"
         lowerBound="1" eType="#//Transition" eOpposite="#//
             Transition/to_transition"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        to_input_arc" ordered="false"
         lowerBound="1" eType="#//Place" eOpposite="#//Place/
             from_place"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
        weight" ordered="false"
         unique="false" lowerBound="1" eType="ecore:EDataType
             http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="1"
         eType="#//PetriNet" transient="true" eOpposite="#//
             PetriNet/input_arcs"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Transition">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="label
        " ordered="false"
         unique="false" lowerBound="1" eType="ecore:EDataType
             http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
```

```
        from_transition" ordered="false"
        lowerBound="1" upperBound="-1" eType="#//OutputArc"
            eOpposite="#//OutputArc/to_output_arc"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="1"
        eType="#//PetriNet" transient="true" eOpposite="#//
            PetriNet/transitions"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        to_transition" ordered="false"
        lowerBound="1" upperBound="-1" eType="#//InputArc"
            eOpposite="#//InputArc/from_input_arc"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="OutputArc">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
        weight" ordered="false"
        unique="false" lowerBound="1" eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        from_output_arc" ordered="false"
        lowerBound="1" eType="#//Place" eOpposite="#//Place/
            to_place"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="1"
        eType="#//PetriNet" transient="true" eOpposite="#//
            PetriNet/output_arcs"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        to_output_arc" ordered="false"
        lowerBound="1" eType="#//Transition" eOpposite="#//
            Transition/from_transition"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Place">
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        from_place" ordered="false"
        lowerBound="1" upperBound="-1" eType="#//InputArc"
            eOpposite="#//InputArc/to_input_arc"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
        capacity" ordered="false"
        unique="false" lowerBound="1" eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
        numTokens" ordered="false"
        unique="false" lowerBound="1" eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="label
        " ordered="false"
        unique="false" lowerBound="1" eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="_"
        ordered="false" lowerBound="1"
        eType="#//PetriNet" transient="true" eOpposite="#//
            PetriNet/places"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        to_place" ordered="false"
        lowerBound="1" upperBound="-1" eType="#//OutputArc"
            eOpposite="#//OutputArc/from_output_arc"/>
  </eClassifiers>
</ecore:EPackage>
```

6.5 SVG4MTV XMI

You can see the Listing 6.5 of the SVG4MTV file for the Petri Nets meta model. This file is the result of the first transformation of our browser.

Listing 6.3. SVG4MTV XMI file for the Petri Nets

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:SVG4MTV="http:///SVG4MTV.ecore">
  <SVG4MTV:ClassElement name="/7" elementId="1"
      class_big_rectangle="/1" classes_connections="/47 /57 /67
      /77"/>
  <SVG4MTV:ClassRectangle x="175" y="119" width="132" height="24
      " class_element="/0">
    <little_rectangle x="175" y="119" width="132" height="12">
      <name_of_class name="/2" type="/3" x="187" y="131"/>
    </little_rectangle>
    <little_rectangle x="175" y="131" width="132" height="12">
      <attributes_of_class name="/4" type="/5" visibility="/6" x
          ="187" y="143"/>
    </little_rectangle>
  </SVG4MTV:ClassRectangle>
  <SVG4MTV:SVGClassName text="PetriNet" fontType="Arial"
      fontSize="12" fontColor="black"/>
  <SVG4MTV:SVGClassType text="" fontType="Arial" fontSize="12"
      fontColor="black"/>
  <SVG4MTV:SVGClassAttribute text="name" fontName="attribute"
      fontType="Arial" fontSize="10" fontColor="black"/>
  <SVG4MTV:SVGClassAttributeType text="EString" fontName="
      attributetype" fontType="Arial" fontSize="10" fontColor="
```

```
        black"/>
    <SVG4MTV:SVGTextVisibility/>
    <SVG4MTV:SVGText text="PetriNet"/>
    <SVG4MTV:ClassElement name="/15" elementId="2"
        class_big_rectangle="/9" classes_connections="/47 /87 /96"/
        >
    <SVG4MTV:ClassRectangle x="9" y="9" width="120" height="24"
        class_element="/8">
      <little_rectangle x="9" y="9" width="120" height="12">
        <name_of_class name="/10" type="/11" x="21" y="21"/>
      </little_rectangle>
      <little_rectangle x="9" y="21" width="120" height="12">
        <attributes_of_class name="/12" type="/13" visibility="/14
            " x="21" y="33"/>
      </little_rectangle>
    </SVG4MTV:ClassRectangle>
    <SVG4MTV:SVGClassName text="InputArc" fontType="Arial"
        fontSize="12" fontColor="black"/>
    <SVG4MTV:SVGClassType text="" fontType="Arial" fontSize="12"
        fontColor="black"/>
    <SVG4MTV:SVGClassAttribute text="weight" fontName="attribute"
        fontType="Arial" fontSize="10" fontColor="black"/>
    <SVG4MTV:SVGClassAttributeType text="EInt" fontName="
        attributetype" fontType="Arial" fontSize="10" fontColor="
        black"/>
    <SVG4MTV:SVGTextVisibility/>
    <SVG4MTV:SVGText text="InputArc"/>
    <SVG4MTV:ClassElement name="/23" elementId="3"
        class_big_rectangle="/17" classes_connections="/77 /87 /105
        "/>
    <SVG4MTV:ClassRectangle x="9" y="119" width="144" height="24"
        class_element="/16">
      <little_rectangle x="9" y="119" width="144" height="12">
        <name_of_class name="/18" type="/19" x="21" y="131"/>
      </little_rectangle>
      <little_rectangle x="9" y="131" width="144" height="12">
        <attributes_of_class name="/20" type="/21" visibility="/22
            " x="21" y="143"/>
      </little_rectangle>
    </SVG4MTV:ClassRectangle>
    <SVG4MTV:SVGClassName text="Transition" fontType="Arial"
        fontSize="12" fontColor="black"/>
    <SVG4MTV:SVGClassType text="" fontType="Arial" fontSize="12"
        fontColor="black"/>
    <SVG4MTV:SVGClassAttribute text="label" fontName="attribute"
    fontType="Arial" fontSize="10" fontColor="black"/>
<SVG4MTV:SVGClassAttributeType text="EString" fontName="
    attributetype" fontType="Arial" fontSize="10" fontColor="
    black"/>
<SVG4MTV:SVGTextVisibility/>
<SVG4MTV:SVGText text="Transition"/>
<SVG4MTV:ClassElement name="/31" elementId="4"
    class_big_rectangle="/25" classes_connections="/67 /105
    /114"/>
<SVG4MTV:ClassRectangle x="9" y="229" width="120" height="24"
    class_element="/24">
  <little_rectangle x="9" y="229" width="120" height="12">
    <name_of_class name="/26" type="/27" x="21" y="241"/>
  </little_rectangle>
  <little_rectangle x="9" y="241" width="120" height="12">
    <attributes_of_class name="/28" type="/29" visibility="/30
        " x="21" y="253"/>
  </little_rectangle>
</SVG4MTV:ClassRectangle>
<SVG4MTV:SVGClassName text="OutputArc" fontType="Arial"
    fontSize="12" fontColor="black"/>
<SVG4MTV:SVGClassType text="" fontType="Arial" fontSize="12"
    fontColor="black"/>
<SVG4MTV:SVGClassAttribute text="weight" fontName="attribute"
    fontType="Arial" fontSize="10" fontColor="black"/>
<SVG4MTV:SVGClassAttributeType text="EInt" fontName="
    attributetype" fontType="Arial" fontSize="10" fontColor="
    black"/>
<SVG4MTV:SVGTextVisibility/>
<SVG4MTV:SVGText text="OutputArc"/>
<SVG4MTV:ClassElement name="/39" elementId="5"
    class_big_rectangle="/33" classes_connections="/57 /96 /114
    "/>
<SVG4MTV:ClassRectangle x="9" y="339" width="156" height="48"
    class_element="/32">
  <little_rectangle x="9" y="339" width="156" height="12">
    <name_of_class name="/34" type="/35" x="21" y="351"/>
  </little_rectangle>
  <little_rectangle x="9" y="351" width="156" height="36">
    <attributes_of_class name="/36" type="/37" visibility="/38
        " x="21" y="363"/>
    <attributes_of_class name="/40" type="/41" visibility="/42
        " x="21" y="375"/>
    <attributes_of_class name="/43" type="/44" visibility="/45
        " x="21" y="387"/>
    </little_rectangle>
  </SVG4MTV:ClassRectangle>
  <SVG4MTV:SVGClassName text="Place" fontType="Arial" fontSize="
      12" fontColor="black"/>
  <SVG4MTV:SVGClassType text="" fontType="Arial" fontSize="12"
      fontColor="black"/>
  <SVG4MTV:SVGClassAttribute text="capacity" fontName="attribute
      " fontType="Arial" fontSize="10" fontColor="black"/>
  <SVG4MTV:SVGClassAttributeType text="EInt" fontName="
      attributetype" fontType="Arial" fontSize="10" fontColor="
      black"/>
  <SVG4MTV:SVGTextVisibility/>
  <SVG4MTV:SVGText text="Place"/>
  <SVG4MTV:SVGClassAttribute text="numTokens" fontName="
      attribute" fontType="Arial" fontSize="10" fontColor="black"
      />
  <SVG4MTV:SVGClassAttributeType text="EInt" fontName="
      attributetype" fontType="Arial" fontSize="10" fontColor="
      black"/>
  <SVG4MTV:SVGTextVisibility/>
  <SVG4MTV:SVGClassAttribute text="label" fontName="attribute"
      fontType="Arial" fontSize="10" fontColor="black"/>
  <SVG4MTV:SVGClassAttributeType text="EString" fontName="
      attributetype" fontType="Arial" fontSize="10" fontColor="
      black"/>
  <SVG4MTV:SVGTextVisibility/>
  <SVG4MTV:SVGText text="input_arcs"/>
  <SVG4MTV:Composition name="/46" elementId="6"
      connected_classes_elements="/0 /8" polyline_of_composition=
      "/54" is_source="/48" is_target="/49" source="1" target="2"
       com_losange="/55"/>
  <SVG4MTV:CompositionEnds x="225" y="104" name="/50"
      multiplicity="/52" sourceId="1" targetId="2"
      is_composition_tar="/47"/>
  <SVG4MTV:CompositionEnds x="99" y="45" name="/51" multiplicity
      ="/53" sourceId="2" targetId="1" is_composition_src="/47"/>
  <SVG4MTV:SVGAssociationEnd text="input_arcs" fontType="Arial"
      fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
      "8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
      Arial" fontSize="12" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
      fontSize="12" fontColor="black"/>
  <SVG4MTV:CompositionPolyLine composition="/47">
```

```
    <composition_points xValue="222" yValue="114"/>
    <composition_points xValue="87" yValue="33"/>
  </SVG4MTV:CompositionPolyLine>
  <SVG4MTV:Losange filled="true" fillColor="black" strokeWidth="
      3" strokeColor="black" composition="/47" x1Value="222"
      y1Value="119" x2Value="227" y2Value="114" x3Value="222"
      y3Value="109" x4Value="217" y4Value="114"/>
  <SVG4MTV:SVGText text="places"/>
  <SVG4MTV:Composition name="/56" elementId="7"
      connected_classes_elements="/0 /32" polyline_of_composition
      ="/64" is_source="/58" is_target="/59" source="1" target="5
      " com_losange="/65"/>
  <SVG4MTV:CompositionEnds x="236" y="128" name="/60"
      multiplicity="/62" sourceId="1" targetId="5"
      is_composition_tar="/57"/>
  <SVG4MTV:CompositionEnds x="114" y="351" name="/61"
      multiplicity="/63" sourceId="5" targetId="1"
      is_composition_src="/57"/>
  <SVG4MTV:SVGAssociationEnd text="places" fontType="Arial"
      fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
      "8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
      Arial" fontSize="12" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
      fontSize="12" fontColor="black"/>
  <SVG4MTV:CompositionPolyLine composition="/57">
    <composition_points xValue="233" yValue="138"/>
    <composition_points xValue="102" yValue="339"/>
  </SVG4MTV:CompositionPolyLine>
  <SVG4MTV:Losange filled="true" fillColor="black" strokeWidth="
      3" strokeColor="black" composition="/57" x1Value="233"
      y1Value="143" x2Value="238" y2Value="138" x3Value="233"
      y3Value="133" x4Value="228" y4Value="138"/>
  <SVG4MTV:SVGText text="output_arcs"/>
  <SVG4MTV:Composition name="/66" elementId="8"
      connected_classes_elements="/0 /24" polyline_of_composition
      ="/74" is_source="/68" is_target="/69" source="1" target="4
      " com_losange="/75"/>
  <SVG4MTV:CompositionEnds x="225" y="128" name="/70"
      multiplicity="/72" sourceId="1" targetId="4"
      is_composition_tar="/67"/>
  <SVG4MTV:CompositionEnds x="99" y="241" name="/71"
      multiplicity="/73" sourceId="4" targetId="1"
      is_composition_src="/67"/>
```

```
<SVG4MTV:SVGAssociationEnd text="output_arcs" fontType="Arial"
     fontSize="8" fontColor="black"/>
<SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
    "8" fontColor="black"/>
<SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
    Arial" fontSize="12" fontColor="black"/>
<SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
    fontSize="12" fontColor="black"/>
<SVG4MTV:CompositionPolyLine composition="/67">
  <composition_points xValue="222" yValue="138"/>
  <composition_points xValue="87" yValue="229"/>
</SVG4MTV:CompositionPolyLine>
<SVG4MTV:Losange filled="true" fillColor="black" strokeWidth="
    3" strokeColor="black" composition="/67" x1Value="222"
    y1Value="143" x2Value="227" y2Value="138" x3Value="222"
    y3Value="133" x4Value="217" y4Value="138"/>
<SVG4MTV:SVGText text="transitions"/>
<SVG4MTV:Composition name="/76" elementId="9"
    connected_classes_elements="/0 /16" polyline_of_composition
    ="/84" is_source="/78" is_target="/79" source="1" target="3
    " com_losange="/85"/>
<SVG4MTV:CompositionEnds x="178" y="116" name="/80"
    multiplicity="/82" sourceId="1" targetId="3"
    is_composition_tar="/77"/>
<SVG4MTV:CompositionEnds x="165" y="143" name="/81"
    multiplicity="/83" sourceId="3" targetId="1"
    is_composition_src="/77"/>
<SVG4MTV:SVGAssociationEnd text="transitions" fontType="Arial"
     fontSize="8" fontColor="black"/>
<SVG4MTV:SVGAssociationEnd text="_" fontType="Arial" fontSize=
    "8" fontColor="black"/>
<SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
    Arial" fontSize="12" fontColor="black"/>
<SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
    fontSize="12" fontColor="black"/>
<SVG4MTV:CompositionPolyLine composition="/77">
  <composition_points xValue="175" yValue="126"/>
  <composition_points xValue="153" yValue="131"/>
</SVG4MTV:CompositionPolyLine>
<SVG4MTV:Losange filled="true" fillColor="black" strokeWidth="
    3" strokeColor="black" composition="/77" x1Value="175"
    y1Value="131" x2Value="180" y2Value="126" x3Value="175"
    y3Value="121" x4Value="170" y4Value="126"/>
<SVG4MTV:SVGText text="from_input_arc"/>
<SVG4MTV:Association name="/86" elementId="10"
```

```
        connected_classes_elements="/8 /16" is_target="/89"
        is_source="/88" source="2" target="3"
        polyline_of_association="/94"/>
    <SVG4MTV:AssociationEnds x="82" y="104" name="/90"
        multiplicity="/92" sourceId="2" targetId="3"
        is_association_src="/87"/>
    <SVG4MTV:AssociationEnds x="82" y="45" name="/91" multiplicity
       ="/93" sourceId="3" targetId="2" is_association_tar="/87"/>
    <SVG4MTV:SVGAssociationEnd text="from_input_arc" fontType="
        Arial" fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationEnd text="to_transition" fontType="
        Arial" fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
        fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
        Arial" fontSize="8" fontColor="black"/>
    <SVG4MTV:AssociationPolyline association="/87">
      <points_of_association_polyline xValue="70" yValue="32"/>
      <points_of_association_polyline xValue="79" yValue="120"/>
    </SVG4MTV:AssociationPolyline>
    <SVG4MTV:SVGText text="to_input_arc"/>
    <SVG4MTV:Association name="/95" elementId="11"
        connected_classes_elements="/8 /32" is_target="/98"
        is_source="/97" source="2" target="5"
        polyline_of_association="/103"/>
    <SVG4MTV:AssociationEnds x="88" y="324" name="/99"
        multiplicity="/101" sourceId="2" targetId="5"
        is_association_src="/96"/>
    <SVG4MTV:AssociationEnds x="81" y="45" name="/100"
        multiplicity="/102" sourceId="5" targetId="2"
        is_association_tar="/96"/>
    <SVG4MTV:SVGAssociationEnd text="to_input_arc" fontType="Arial
       " fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationEnd text="from_place" fontType="Arial"
        fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
        fontSize="8" fontColor="black"/>
    <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
        Arial" fontSize="8" fontColor="black"/>
    <SVG4MTV:AssociationPolyline association="/96">
      <points_of_association_polyline xValue="69" yValue="32"/>
      <points_of_association_polyline xValue="85" yValue="340"/>
    </SVG4MTV:AssociationPolyline>
    <SVG4MTV:SVGText text="from_transition"/>
    <SVG4MTV:Association name="/104" elementId="12"
```

```
        connected_classes_elements="/16 /24" is_target="/107"
        is_source="/106" source="3" target="4"
        polyline_of_association="/112"/>
  <SVG4MTV:AssociationEnds x="73" y="214" name="/108"
        multiplicity="/110" sourceId="3" targetId="4"
        is_association_src="/105"/>
  <SVG4MTV:AssociationEnds x="91" y="155" name="/109"
        multiplicity="/111" sourceId="4" targetId="3"
        is_association_tar="/105"/>
  <SVG4MTV:SVGAssociationEnd text="from_transition" fontType="
        Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationEnd text="to_output_arc" fontType="
        Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
        Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
        fontSize="8" fontColor="black"/>
  <SVG4MTV:AssociationPolyline association="/105">
    <points_of_association_polyline xValue="79" yValue="142"/>
    <points_of_association_polyline xValue="70" yValue="230"/>
  </SVG4MTV:AssociationPolyline>
  <SVG4MTV:SVGText text="from_output_arc"/>
  <SVG4MTV:Association name="/113" elementId="13"
        connected_classes_elements="/24 /32" is_target="/116"
        is_source="/115" source="4" target="5"
        polyline_of_association="/121"/>
  <SVG4MTV:AssociationEnds x="86" y="324" name="/117"
        multiplicity="/119" sourceId="4" targetId="5"
        is_association_src="/114"/>
  <SVG4MTV:AssociationEnds x="82" y="265" name="/118"
        multiplicity="/120" sourceId="5" targetId="4"
        is_association_tar="/114"/>
  <SVG4MTV:SVGAssociationEnd text="from_output_arc" fontType="
        Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationEnd text="to_place" fontType="Arial"
        fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1" fontType="Arial"
        fontSize="8" fontColor="black"/>
  <SVG4MTV:SVGAssociationMultiplicity text="1..*" fontType="
        Arial" fontSize="8" fontColor="black"/>
  <SVG4MTV:AssociationPolyline association="/114">
    <points_of_association_polyline xValue="70" yValue="252"/>
    <points_of_association_polyline xValue="83" yValue="340"/>
  </SVG4MTV:AssociationPolyline>
</xmi:XMI>
```

6.6 SVG XML

Here is the SVG of the generated graphical 2D representation of the meta model. You can see the Listing 6.6.

Listing 6.4. SVG XML file for the Petri Nets

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

<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 1.0//EN' 'http://www.w3.
    org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd'>
<svg style="fill-opacity:1; color-rendering:auto; color-
    interpolation:auto; text-rendering:auto; stroke:black; stroke
    -linecap:square; stroke-miterlimit:10; shape-rendering:auto;
    stroke-opacity:1; fill:black; stroke-dasharray:none; font-
    weight:normal; stroke-width:1; font-family:&apos;Dialog&apos
    ;; font-style:normal; stroke-linejoin:miter; font-size:12;
    stroke-dashoffset:0; image-rendering:auto;" xmlns="http://www
    .w3.org/2000/svg" width="772" xmlns:xlink="http://www.w3.org
    /1999/xlink" height="488">
  <!--Generated by MTV with Batik SVG Generator-->
  <defs id="genericDefs" />
  <g>
    <g style="fill:url(/Users/monnet/IdeaProjects/
        SVG4MTV_EMF_GLOBAL/shadowFilter.svg#grid);">
      <g>
        <rect x="0" y="0" width="772" style="stroke:none;"
            height="488" />
      </g>
    </g>
    <g id="class1" style="font-size:10; filter:url(/Users/monnet
        /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
        MyDropShadow);">
      <g style="fill:papayawhip; stroke:black;">
        <g>
          <rect x="175" y="119" width="132" style="stroke:none;"
              height="24" />
        </g>
      </g>
      <g>
        <rect x="175" y="119" width="132" style="fill:none;"
            height="12" />
        <text xml:space="preserve" x="203" y="129" style="
            stroke:none;">PetriNet</text>
        <rect x="175" y="131" width="132" style="fill:none;"
            height="12" />
        <text xml:space="preserve" x="187" y="140" style="
            stroke:none;">name: EString</text>
      </g>
    </g>
    <g id="class2" style="font-size:10; filter:url(/Users/monnet
        /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
        MyDropShadow);">
      <g style="fill:papayawhip; stroke:black;">
        <g>
          <rect x="9" y="9" width="120" style="stroke:none;"
              height="24" />
        </g>
      </g>
      <g>
        <rect x="9" y="9" width="120" style="fill:none;" height=
            "12" />
        <text xml:space="preserve" x="31" y="19" style="
            stroke:none;">InputArc</text>
        <rect x="9" y="21" width="120" style="fill:none;" height
            ="12" />
        <text xml:space="preserve" x="21" y="30" style="
            stroke:none;">weight: EInt</text>
      </g>
    </g>
    <g id="class3" style="font-size:10; filter:url(/Users/monnet
        /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
        MyDropShadow);">
      <g style="fill:papayawhip; stroke:black;">
        <g>
          <rect x="9" y="119" width="144" style="stroke:none;"
              height="24" />
        </g>
      </g>
      <g>
        <rect x="9" y="119" width="144" style="fill:none;"
            height="12" />
        <text xml:space="preserve" x="32" y="129" style="
            stroke:none;">Transition</text>
        <rect x="9" y="131" width="144" style="fill:none;"
            height="12" />
        <text xml:space="preserve" x="21" y="140" style="
            stroke:none;">label: EString</text>
      </g>
    </g>
    <g id="class4" style="font-size:10; filter:url(/Users/monnet
          /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
          MyDropShadow);">
        <g style="fill:papayawhip; stroke:black;">
          <g>
            <rect x="9" y="229" width="120" style="stroke:none;"
                height="24" />
          </g>
        </g>
        <g>
          <rect x="9" y="229" width="120" style="fill:none;"
              height="12" />
          <text xml:space="preserve" x="25" y="239" style="
              stroke:none;">OutputArc</text>
          <rect x="9" y="241" width="120" style="fill:none;"
              height="12" />
          <text xml:space="preserve" x="21" y="250" style="
              stroke:none;">weight: EInt</text>
        </g>
      </g>
      <g id="class5" style="font-size:10; filter:url(/Users/monnet
          /IdeaProjects/SVG4MTV_EMF_GLOBAL/shadowFilter.svg#
          MyDropShadow);">
        <g style="fill:papayawhip; stroke:black;">
          <g>
            <rect x="9" y="339" width="156" style="stroke:none;"
                height="48" />
          </g>
        </g>
        <g>
          <rect x="9" y="339" width="156" style="fill:none;"
              height="12" />
          <text xml:space="preserve" x="65" y="349" style="
              stroke:none;">Place</text>
          <rect x="9" y="351" width="156" style="fill:none;"
              height="36" />
          <text xml:space="preserve" x="21" y="360" style="
              stroke:none;">capacity: EInt</text>
          <text xml:space="preserve" x="21" y="372" style="
              stroke:none;">numTokens: EInt</text>
          <text xml:space="preserve" x="21" y="384" style="
              stroke:none;">label: EString</text>
        </g>
      </g>
      <g id="connection6" style="font-size:8;">
        <g>
```

```
          <path d="M222 114 L87 33" style="fill:none;" />
        </g>
        <g id="1losange">
          <g>
            <polygon style="stroke:none;" points=" 222 119 227 114
                222 109 217 114" />
          </g>
        </g>
        <text x="99" id="1" y="57" style="stroke:none;" xml:space=
           "preserve">_</text>
        <text x="99" id="1" y="45" style="stroke:none;" xml:space=
           "preserve">1</text>
        <text x="225" id="2" y="116" style="stroke:none;"
           xml:space="preserve">input_arcs</text>
        <text x="225" id="2" y="104" style="stroke:none;"
           xml:space="preserve">1..*</text>
      </g>
      <g id="connection7" style="font-size:8;">
        <g>
          <path d="M233 138 L102 339" style="fill:none;" />
        </g>
        <g id="1losange">
          <g>
            <polygon style="stroke:none;" points=" 233 143 238 138
                233 133 228 138" />
          </g>
        </g>
        <text x="114" id="1" y="363" style="stroke:none;"
           xml:space="preserve">_</text>
        <text x="114" id="1" y="351" style="stroke:none;"
           xml:space="preserve">1</text>
        <text x="236" id="5" y="140" style="stroke:none;"
           xml:space="preserve">places</text>
        <text x="236" id="5" y="128" style="stroke:none;"
           xml:space="preserve">1..*</text>
      </g>
      <g id="connection8" style="font-size:8;">
        <g>
          <path d="M222 138 L87 229" style="fill:none;" />
        </g>
        <g id="1losange">
          <g>
            <polygon style="stroke:none;" points=" 222 143 227 138
                222 133 217 138" />
          </g>
```

```
    </g>
    <text x="99" id="1" y="253" style="stroke:none;" xml:space="preserve">_</text>
    <text x="99" id="1" y="241" style="stroke:none;" xml:space="preserve">1</text>
    <text x="225" id="4" y="140" style="stroke:none;" xml:space="preserve">output_arcs</text>
    <text x="225" id="4" y="128" style="stroke:none;" xml:space="preserve">1..*</text>
  </g>
  <g id="connection9" style="font-size:8;">
    <g>
      <path d="M175 126 L153 131" style="fill:none;" />
    </g>
    <g id="1losange">
      <g>
        <polygon style="stroke:none;" points=" 175 131 180 126 175 121 170 126" />
      </g>
    </g>
    <text x="165" id="1" y="155" style="stroke:none;" xml:space="preserve">_</text>
    <text x="165" id="1" y="143" style="stroke:none;" xml:space="preserve">1</text>
    <text x="178" id="3" y="128" style="stroke:none;" xml:space="preserve">transitions</text>
    <text x="178" id="3" y="116" style="stroke:none;" xml:space="preserve">1..*</text>
  </g>
  <g id="connection10" style="font-size:8;">
    <g>
      <path d="M70 32 L79 120" style="fill:none;" />
    </g>
    <text x="82" id="2" y="53" style="stroke:none;" xml:space="preserve">to_transition</text>
    <text x="82" id="2" y="45" style="stroke:none;" xml:space="preserve">1..*</text>
    <text x="82" id="3" y="112" style="stroke:none;" xml:space="preserve">from_input_arc</text>
    <text x="82" id="3" y="104" style="stroke:none;" xml:space="preserve">1</text>
  </g>
  <g id="connection11" style="font-size:8;">
    <g>
      <path d="M69 32 L85 340" style="fill:none;" />
      </g>
      <text x="81" id="2" y="53" style="stroke:none;" xml:space="preserve">from_place</text>
      <text x="81" id="2" y="45" style="stroke:none;" xml:space="preserve">1..*</text>
      <text x="88" id="5" y="332" style="stroke:none;" xml:space="preserve">to_input_arc</text>
      <text x="88" id="5" y="324" style="stroke:none;" xml:space="preserve">1</text>
    </g>
    <g id="connection12" style="font-size:8;">
      <g>
        <path d="M79 142 L70 230" style="fill:none;" />
      </g>
      <text x="91" id="3" y="163" style="stroke:none;" xml:space="preserve">to_output_arc</text>
      <text x="91" id="3" y="155" style="stroke:none;" xml:space="preserve">1</text>
      <text x="73" id="4" y="222" style="stroke:none;" xml:space="preserve">from_transition</text>
      <text x="73" id="4" y="214" style="stroke:none;" xml:space="preserve">1..*</text>
    </g>
    <g id="connection13" style="font-size:8;">
      <g>
        <path d="M70 252 L83 340" style="fill:none;" />
      </g>
      <text x="82" id="4" y="273" style="stroke:none;" xml:space="preserve">to_place</text>
      <text x="82" id="4" y="265" style="stroke:none;" xml:space="preserve">1..*</text>
      <text x="86" id="5" y="332" style="stroke:none;" xml:space="preserve">from_output_arc</text>
      <text x="86" id="5" y="324" style="stroke:none;" xml:space="preserve">1</text>
    </g>
  </g>
</svg>
```

6.7 SVG PNG of the automatically generated placement

Here is the automatically generated placement for the Petri Nets meta model given to the browser in Fig. 6.4.

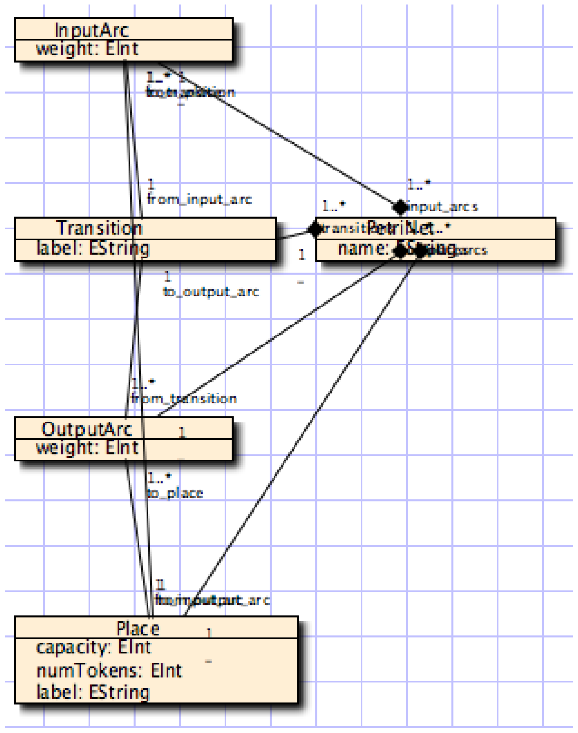


Figure 6.4. *Result of the export to PNG option for the automatically generated placement.*

6.8 SVG PNG after moving elements

After moving the elements in order to make a more readable diagram for the meta model loaded, you can see the result in Fig. 6.5.

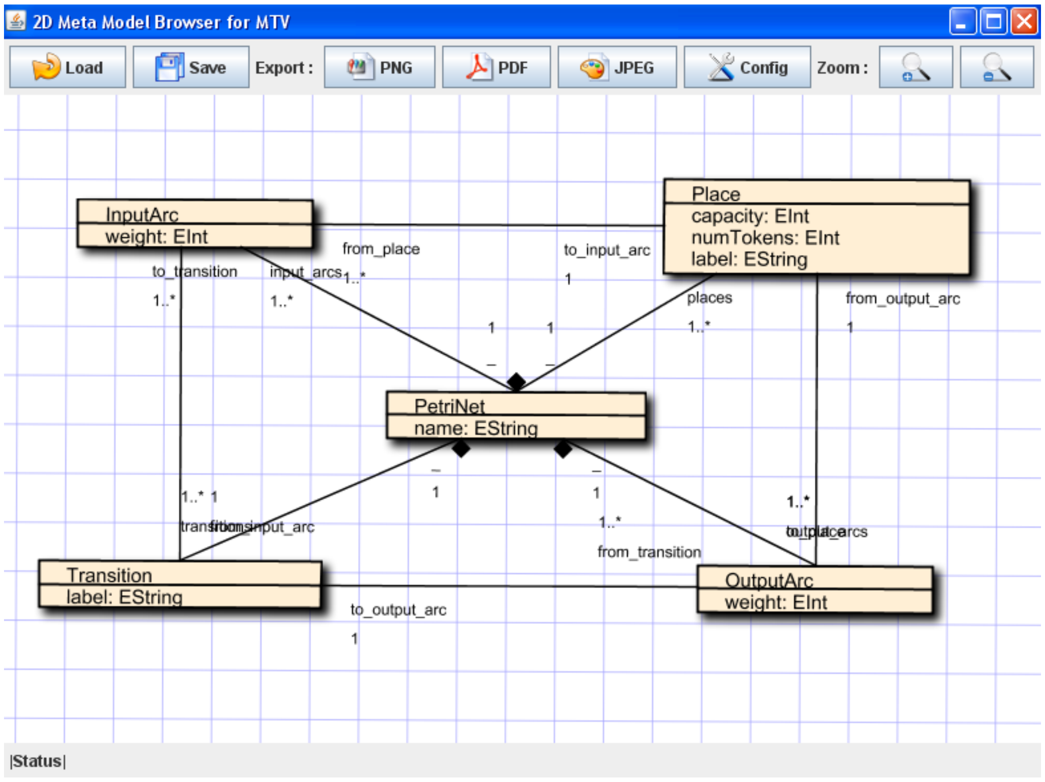


Figure 6.5. Screen shot of the 2D Meta Model Browser after moving elements.

Chapter 7

Conclusion

So, we have seen in this report most of the work done during this project. To resume this project, it's possible to say that we have done a browser that takes as input a description of a meta model without any graphical information described in a meta model formalism (Ecore, but can be extended to MOF or others). This browser produces as output a graphical 2D description of the given meta model with a UML-like style.

One important thing is to know that Batik SVG has some bugs with Mac OS X. Those bugs are presented in the Batik web site. One that has been specially problematic is that the SVG filter operation is working badly. The filter element is used in our project to render the shadow of the class elements. Lot of time has been lost with this issue. Another thing is that Mac OS X takes much more memory to work with Batik than other platforms.

Appendix A

Acronyms

API Application Programming Interface

CIM Computation Independent Model

DOM Document Object Model

EMF Eclipse Modeling Framework

FST Formal Specification Techniques

IST Informal Structured Techniques

MDA Model-Driven Architecture

MOF Meta Object Facilities

MTV Model Transformation for Verification

OCL Object Constraint Language

OMG Object Management Group

OO Object Oriented

PDAs Personal Digital Assistants

PIM Platform Independent Model

PSM Platform Specific Model

SMV	Software Modeling and Verification
SVG	Scalable Vector Graphics
SUT	System Under Test
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
XMI	XML Metadata Interchange

Bibliography

[1] Object Management Group. XML Metadata Interchange. URL: http://www.omg.org/technology/documents/formal/xmi.htm.

[2] Eclipse. Eclipse modeling framework. URL: `http://eclipse.org/emf/`.

[3] OMG. Meta-Object Facility Specification. URL: `http://www.omg.org/technology/documents/formal/mof.htm`.

[4] Stephane Heck. Model transformation and verification: building the basis for a generic tool. Bachelor thesis, Centre Universitaire D'Informatique, Universite de Genève, September 2005.

[5] Software Modeling and Verification. URL: `http://smv.unige.ch/`.

[6] Unified Modeling Language. URL: `http://www.uml.org/`.

[7] Object Management Group. Mda guide version 1.0.1. Technical report, OMG, June 2003.

[8] Object Management Group. URL: `http://www.omg.org/`.

[9] MetaCase Consulting. Metaedit+. URL: http://www.metacase.com.

[10] Vanderbilt University Institute for Software Integrated Systems. Gme 2000 and metagme 2000. URL: http://www.isis.vanderbilt.edu.

[11] Computer Associates. Paradigm plus. URL: http://ca.com/products.

[12] Honeywell International Inc. Domain modeling environment. URL: http://www.htc.honeywell.com/dome/.

[13] Didier Buchs. Software Engineering course. 2005 - 2006. URL: `http://smv.unige.ch/`.

[14] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. the eclipse series. Addison Wesley, 2004.

[15] W3C. Scalable Vector Graphics. URL: `http://www.w3.org/Graphics/SVG/`.

[16] Micah Laaker. *SAMS Teach Yourself SVG in 24 Hours*. SAMS Publishing, 2002.

[17] W3C. Extensible Markup Language (XML). URL: `http://www.w3.org/XML/`.

[18] World Wild Web Consortium. URL: `http://www.w3.org/`.

[19] W3C. Document Object Model. URL: `http://www.w3.org/DOM/`.

[20] Flash. URL: `http://www.adobe.com/products/flash/`.

[21] Apache. Batik SVG Toolkit. URL: `http://xmlgraphics.apache.org/batik/`.

[22] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.

[23] Java Graph Visualization and Layout. URL: `http://www.jgraph.com/index.html`.