Lecture notes
Model-Driven Engineering with Eclipse

practical examples for the Lab

Courses on Model-Driven Engineering and Domain Specific (Modelling) Languages

2021/2022

Vasco Amaral

November 2021

1. Summary

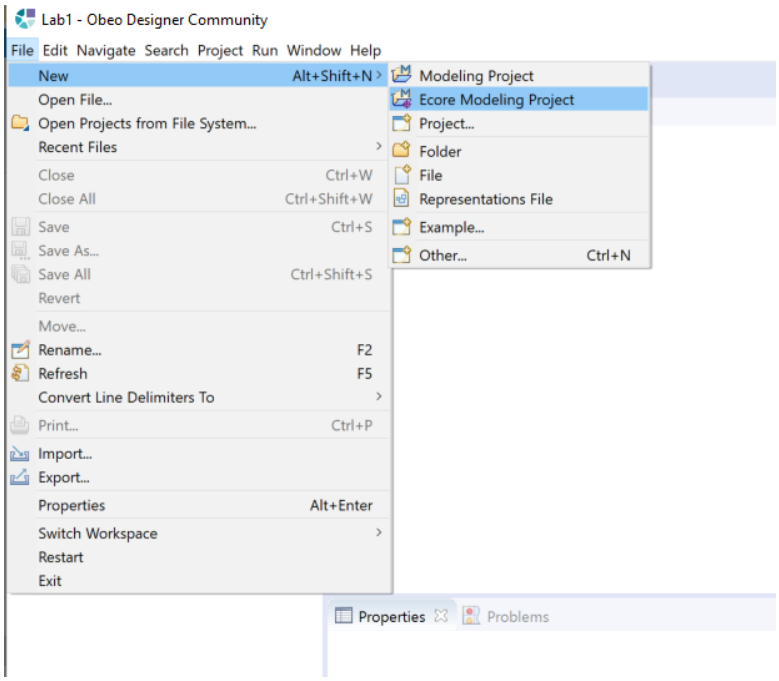
This document contains the code artefacts for the implementation of graphical DSLs using a Model-Driven Approach with Eugenia and EMF/GMF in the Eclipse framework as well as with Epsilon. The text is to be used as a collection of practical examples to support for the lab lectures of the course Model-Driven Engineering at Faculdade de Ciências e Tecnologia at Universidade Nova de Lisboa (FCT/UNL) in Portugal.

2. Setup

Note: The detailed descriptions in this and the following sections are using eclipse-epsilon-1.5-win32-x86_64 downloaded from https://www.eclipse.org/epsilon/download/

Creating a Domain Model

Create an Ecore project:

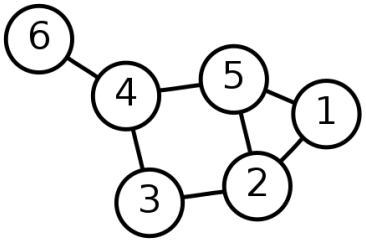


Specify the Ns URI as a universally unique identifier as an absolute URL for this package

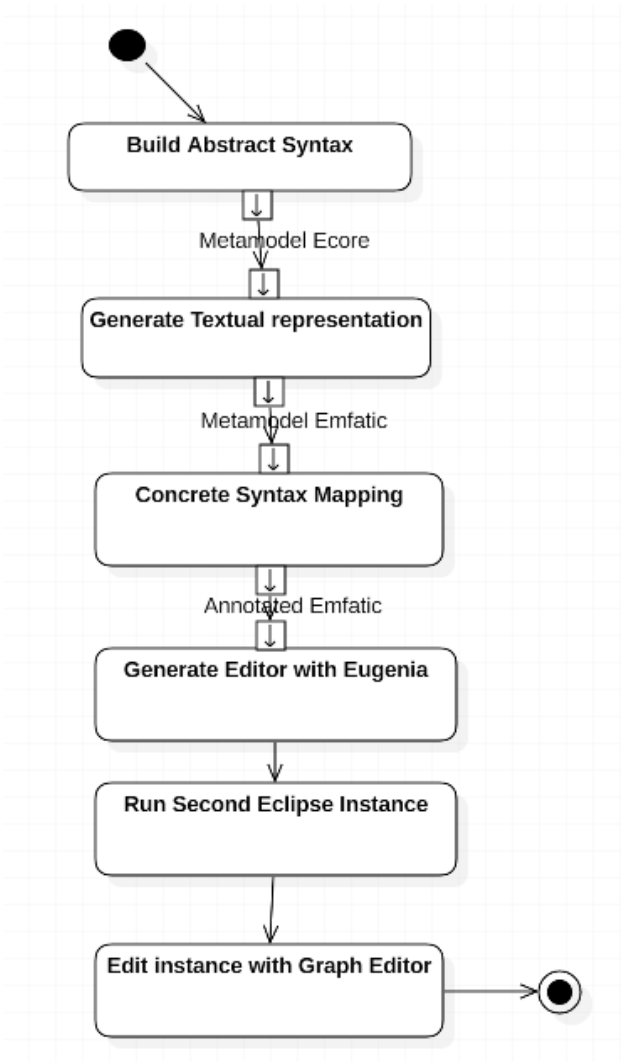
2. A Simple Undirected Labelled Graph Case Study

In graph theory, a graph is a set of related objects. The objects are called vertices and each related pair of vertices is called an edge. We can define graph as an ordered pair G = (V, E) where V is the set of vertices and E id s set of sets of two vertices.

We can draw a diagram as a schematic (visual) representation of a graph. For instance, consider the following Graph G={G,V} where the vertices are V={1,2,3,4,5,6} and the edges are E={{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}. We could represent that example as:

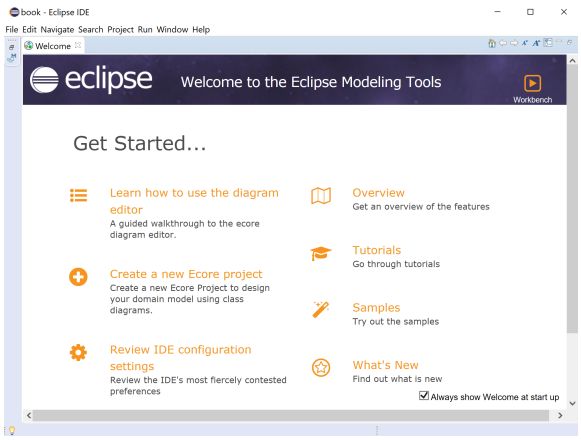


Suppose we want to design the visual editor of diagrams to represent the previous kind of graphs. We can represent undirected graphs in a diagrammatic form using a Model-Driven approach with the following general process:

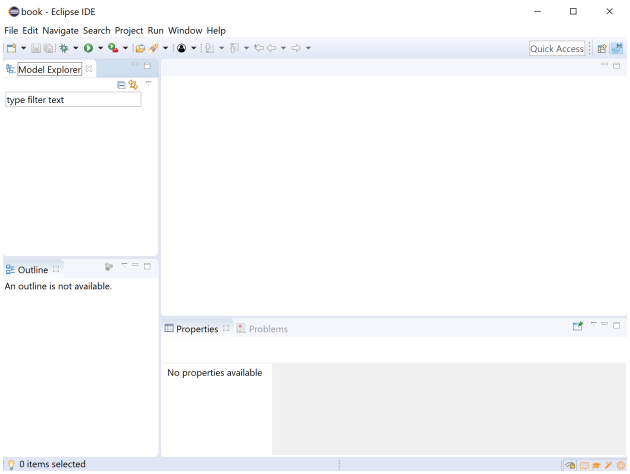


Let’s follow the following steps:

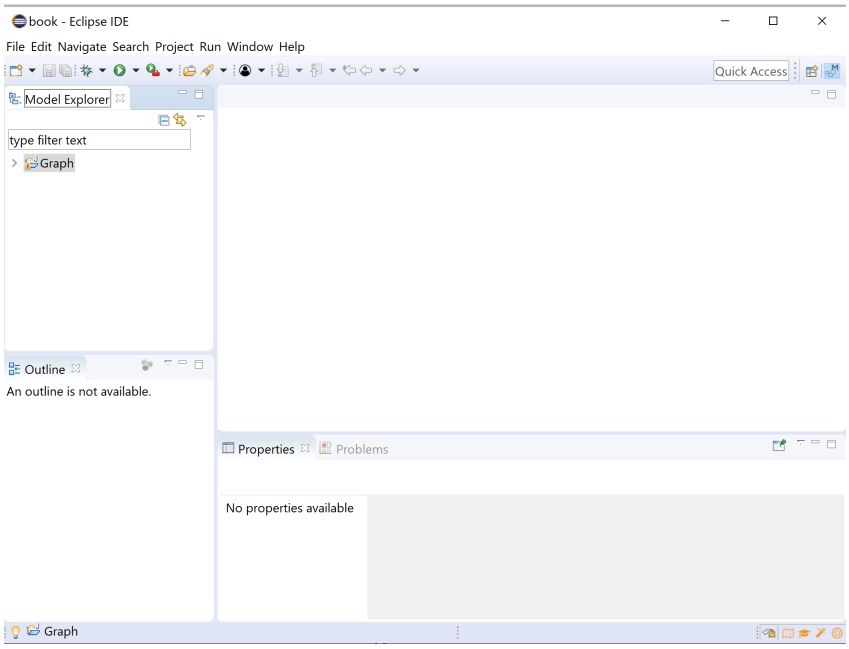
Step 1 - Start Eclipse



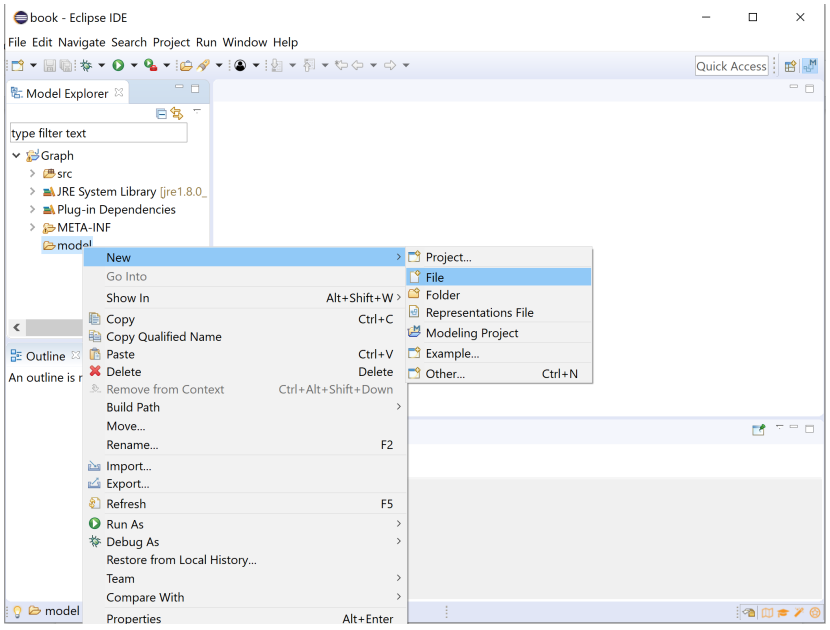
Step 2 - choose your workbench



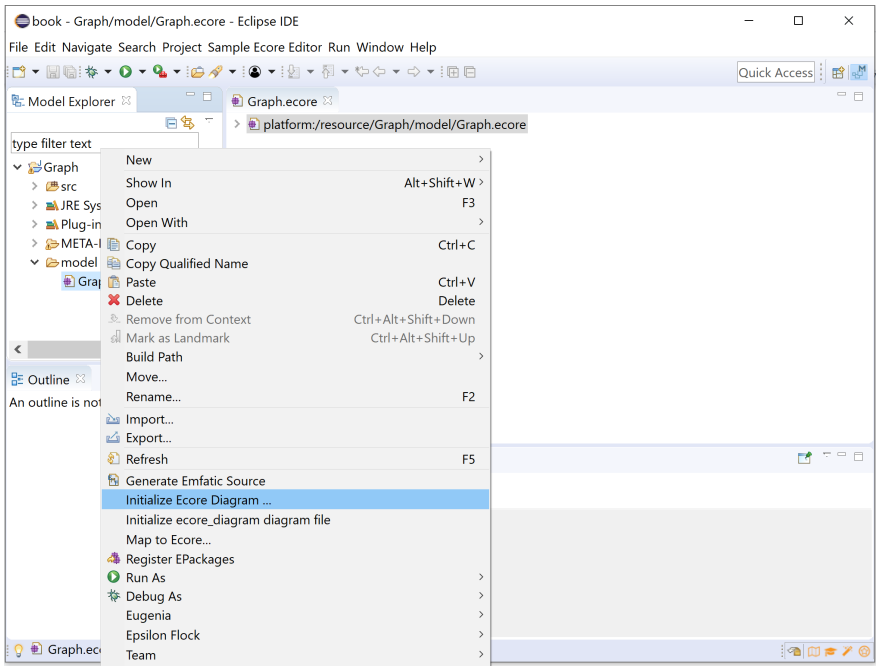
Step 3 - create a new empty EMF project (File->New...->New Empty EMF Project)



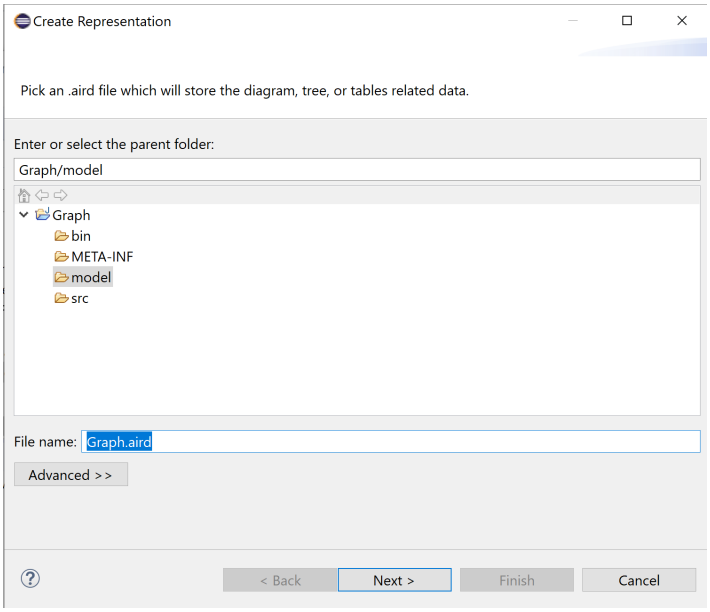
Step 4 - inside the Model folder in the Graph project create an ecore file (File->New...->New Empty EMF Project)



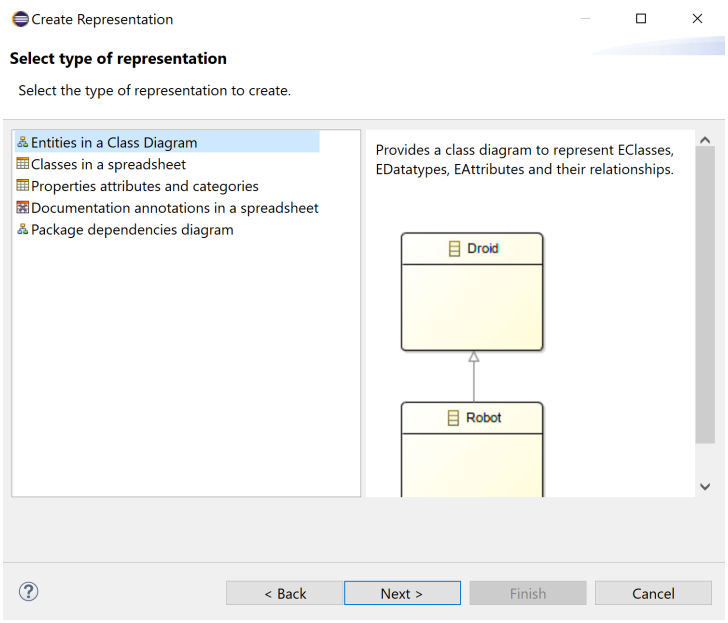
Step 5 - Right-Click on the Graph.ecore file and select initialize Ecore Diagram



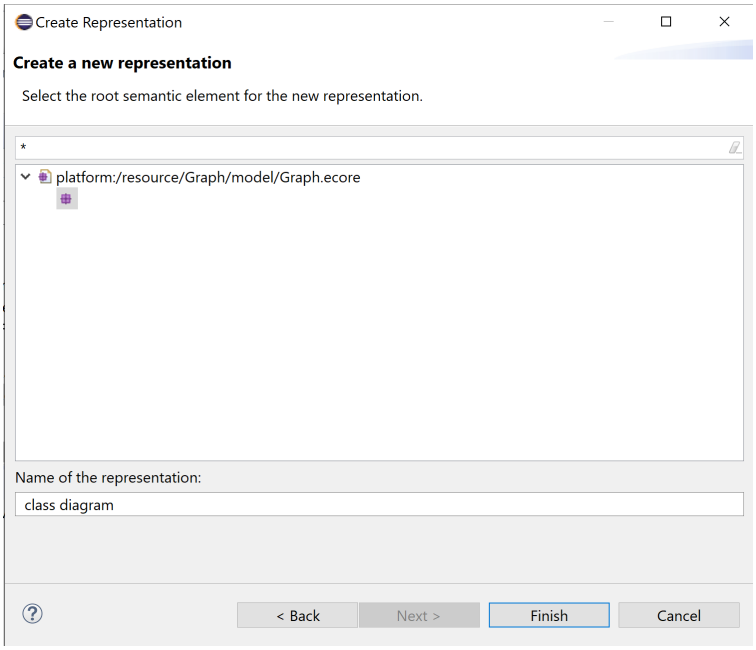
Step 6 - press Next



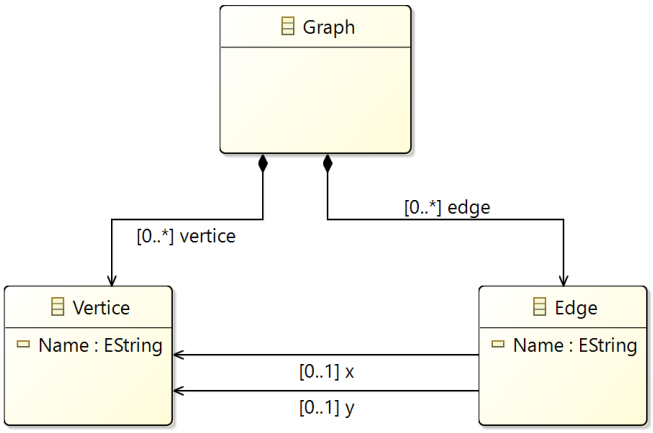
Step 7 - Select Entities in a Class Diagram



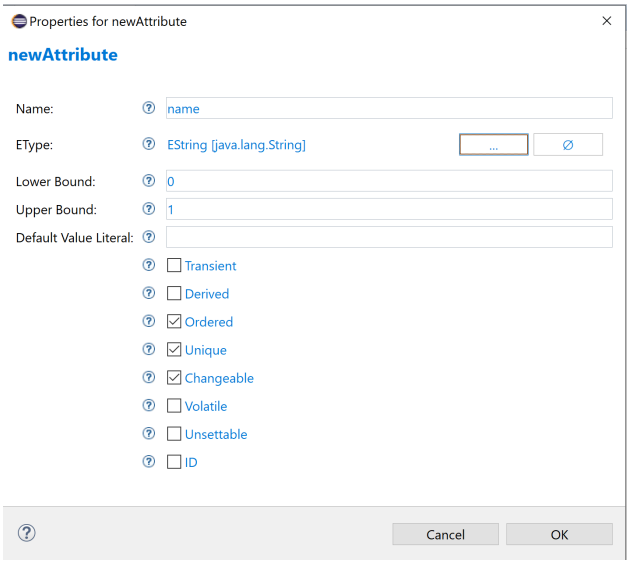
Step 8 - Press Finish



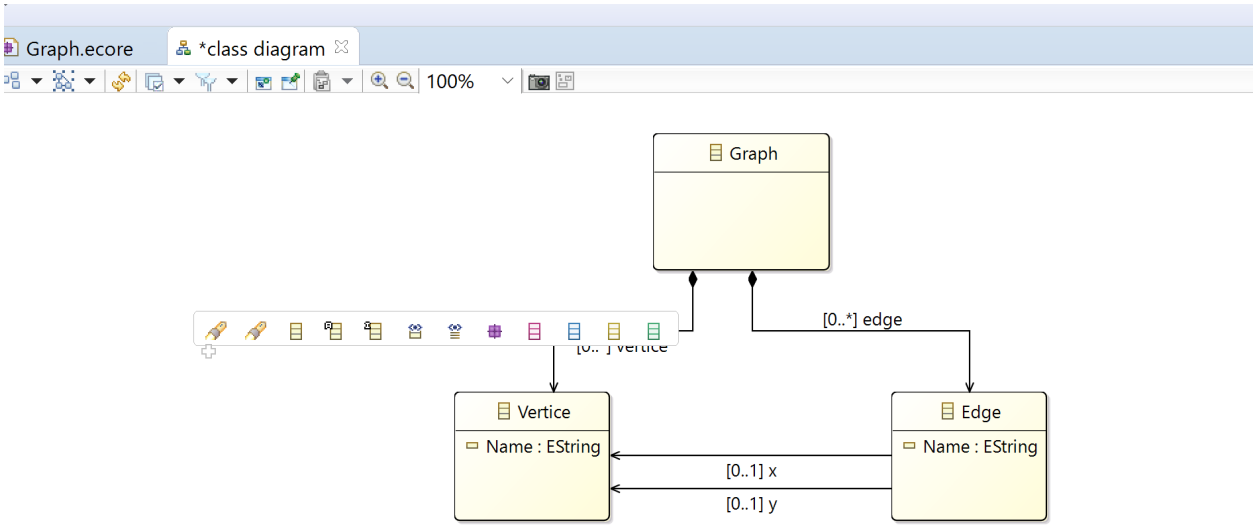
Step 9 - Create the following metamodel

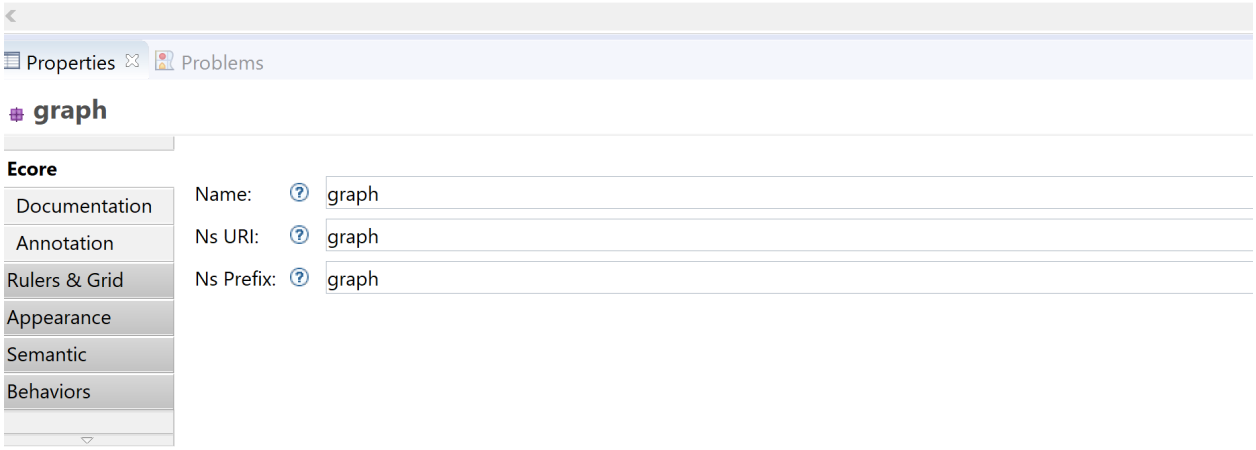


Step 10 - Insert an attribute “Name” in the classes (defining the EType as EString)

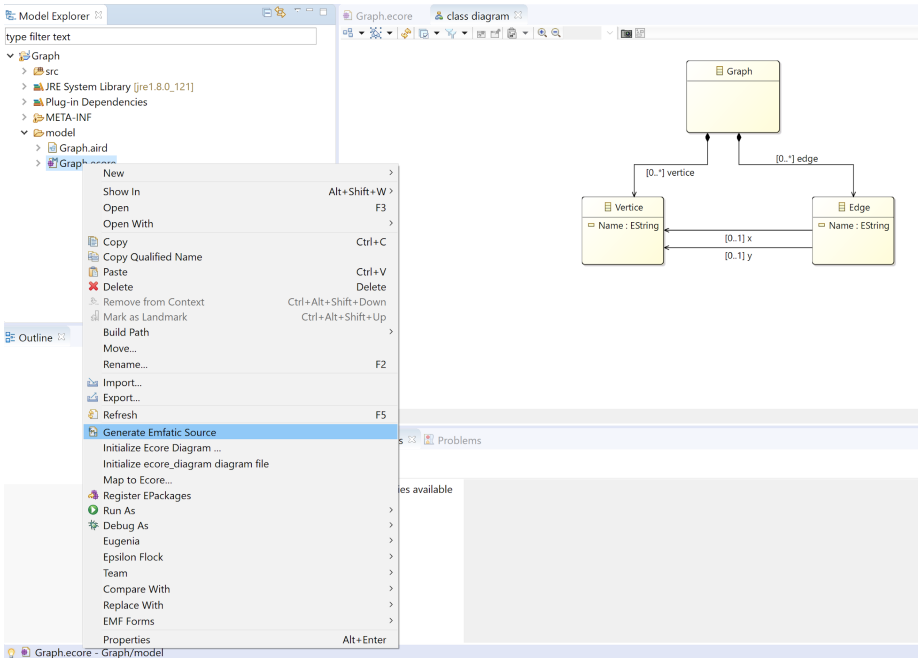


Step 11 - Click on the white canvas behind the elements and complete the information in the properties window

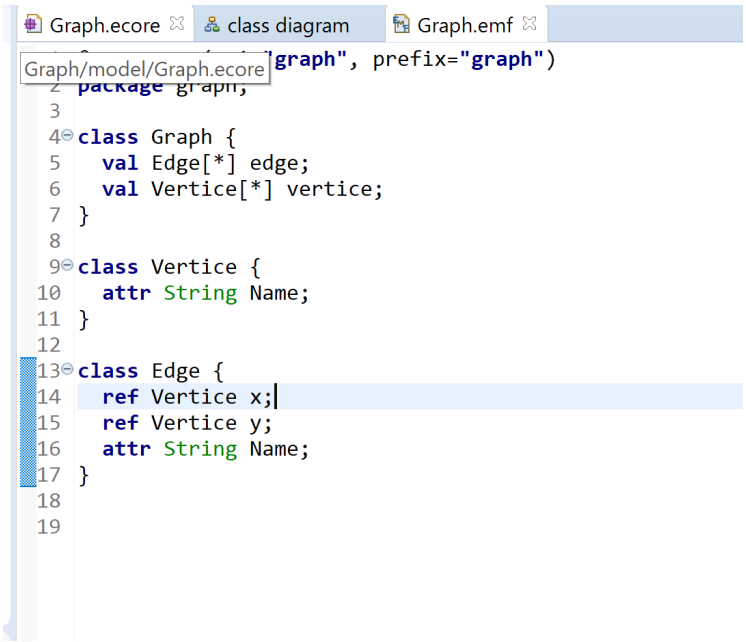




Step 12 - Save the model and right-click on the file to select “Generate Emfatic Source”



You should get the following textual representation of the metamodel:



Step 13 - Complete the model with the following concrete syntax mapping instructions (for more details about the notations please consult the Eugenia website https://www.eclipse.org/epsilon/doc/eugenia/):

a) The root node (here called Graph as referring, which can be seen as the node that contains the Graph language elements and relations) should have the following instruction before:

```
@gmf.diagram
```

b) The relation elements that are to be seen as lines in the diagram should be mapped with:

```
@gmf.link(source="x", target="y", style="dot", width="2")
```

c) The element to be represented as a box (more formats are possible)

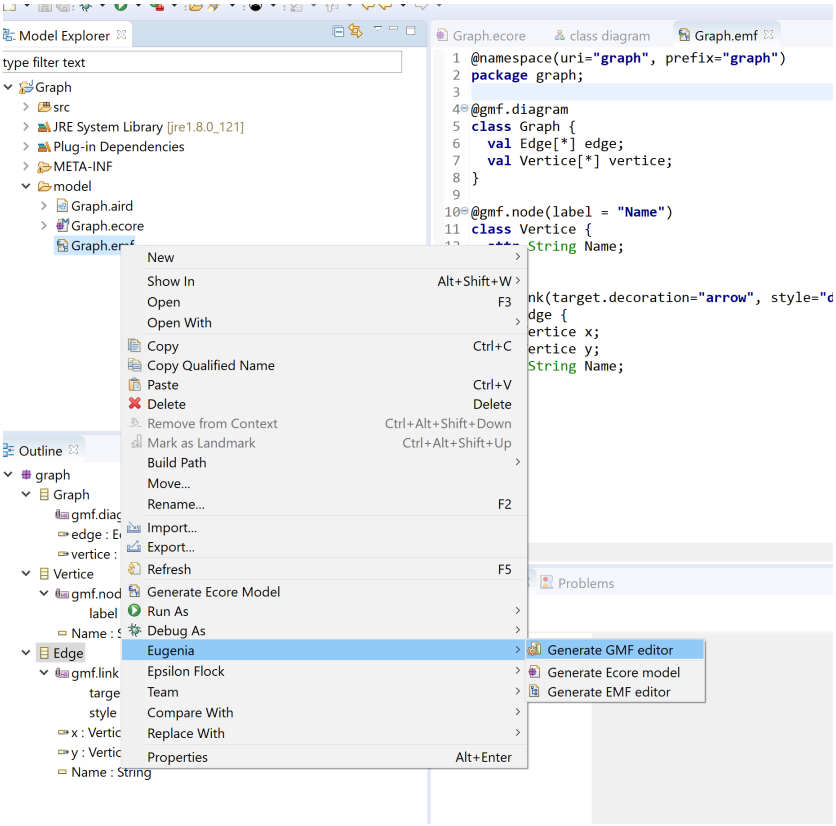
```
@gmf.node(label = "name")
```

The result should be similar to:

Graph/model/Graph.ecore

```
1  @...("graph", prefix="graph")
2  package graph;
3
4  @gmf.diagram
5  class Graph {
6    val Edge[*] edge;
7    val Vertice[*] vertice;
8  }
9
10 @gmf.node(label = "Name")
11 class Vertice {
12   attr String Name;
13 }
14
15 @gmf.link(source="x", target="y", style="dot", width="2")
16 class Edge {
17   ref Vertice x;
18   ref Vertice y;
19   attr String Name;
20 }
21
22
```

Step 14 - Generate the diagram editor by right-clicking in the Graph.emf file and choose to generate the gmf editor:

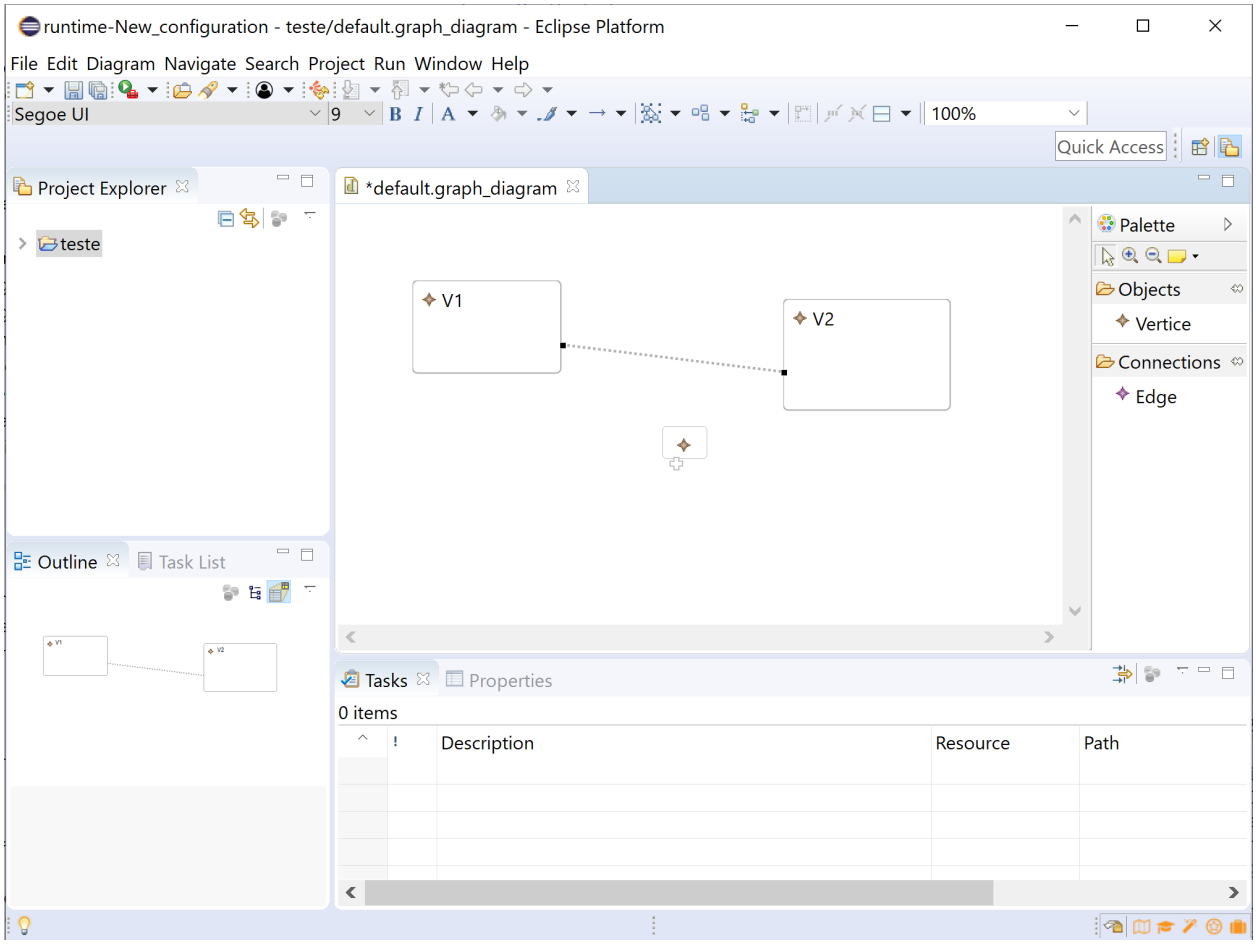


Step 15 - Right-Click in the Graph.emf file and choose “Run as”->”Run Configurations” and choose to run as “Eclipse Application”

Step 16 - A second instance of Eclipse will run

Step 17 - Create a General Project.

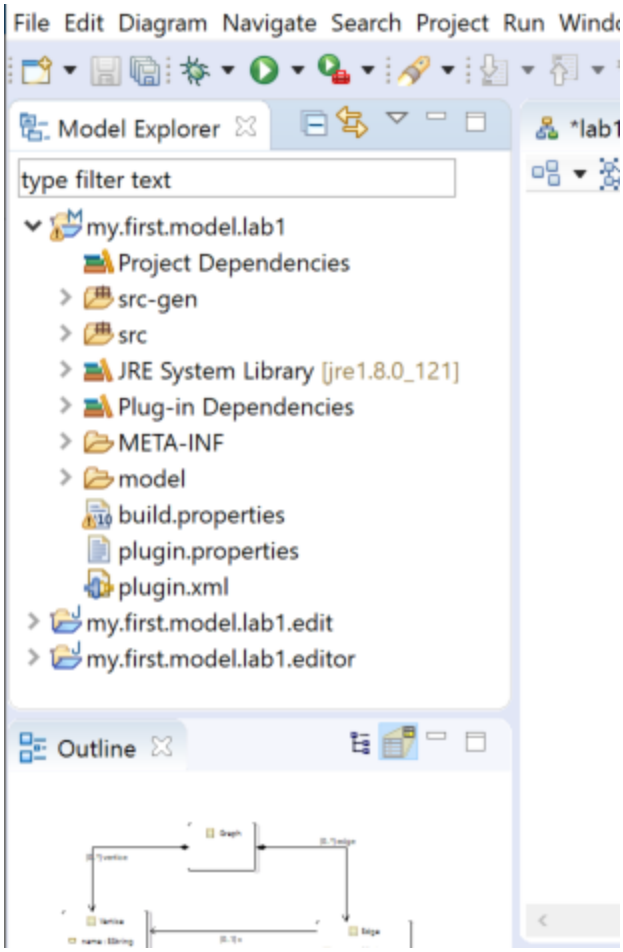
Step 18 - Create inside the Project the file you want to edit File->Create->Other (look for Examples and there you should find “Graph Diagram”)

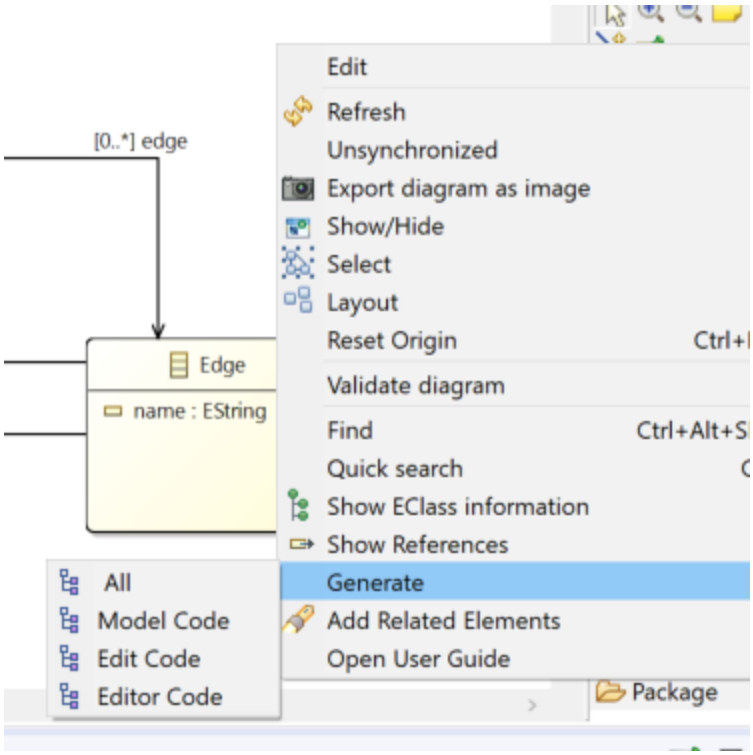


Save the model

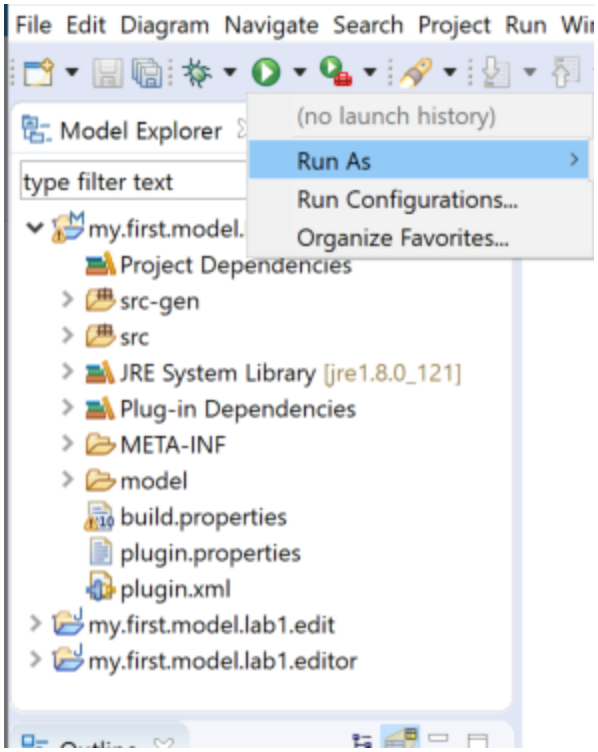
Right-click on the canvas and select Generate->All

On the left side you will see new packages being generated:

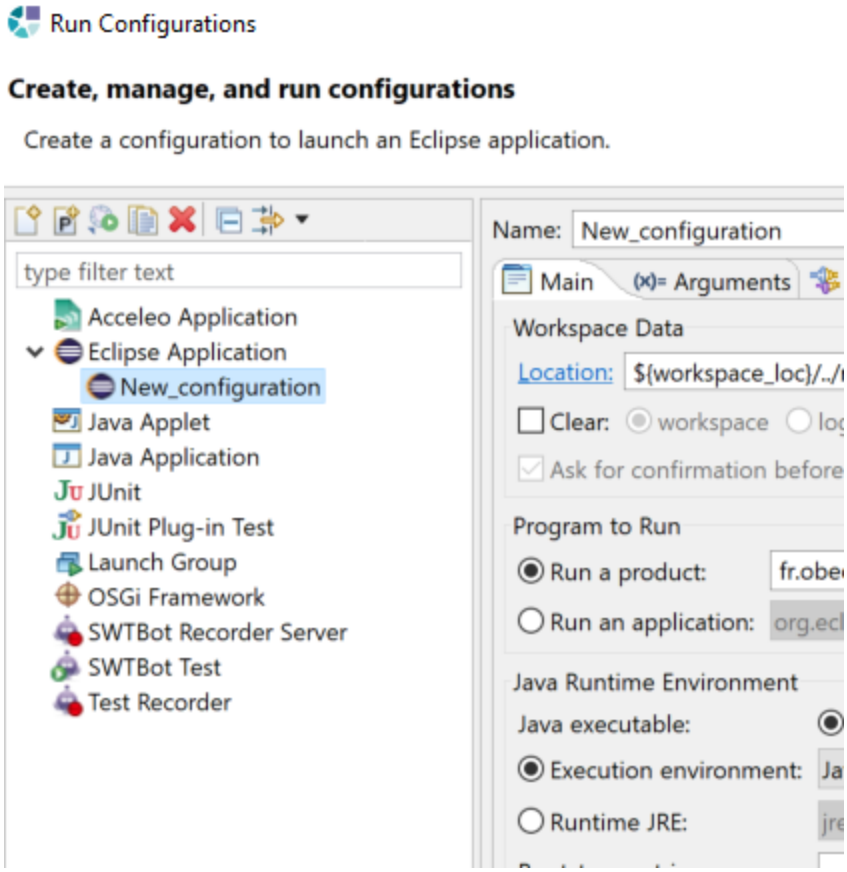

File Edit Diagram Navigate Search Project Run Windo
Model Explorer
type filter text
my.first.model.lab1
Project Dependencies
src-gen
src
JRE System Library [jre1.8.0_121]
Plug-in Dependencies
META-INF
model
build.properties
plugin.properties
plugin.xml
my.first.model.lab1.edit
my.first.model.lab1.editor
Outline
*lab1


[0..*] edge
Edge
name : EString
Edit
Refresh
Unsynchronized
Export diagram as image
Show/Hide
Select
Layout
Reset Origin
Validate diagram
Find
Quick search
Show EClass information
Show References
Generate
Add Related Elements
Open User Guide
All
Model Code
Edit Code
Editor Code
Package

To test the metamodel we can launch a simple editor, by selecting runtime configurations:



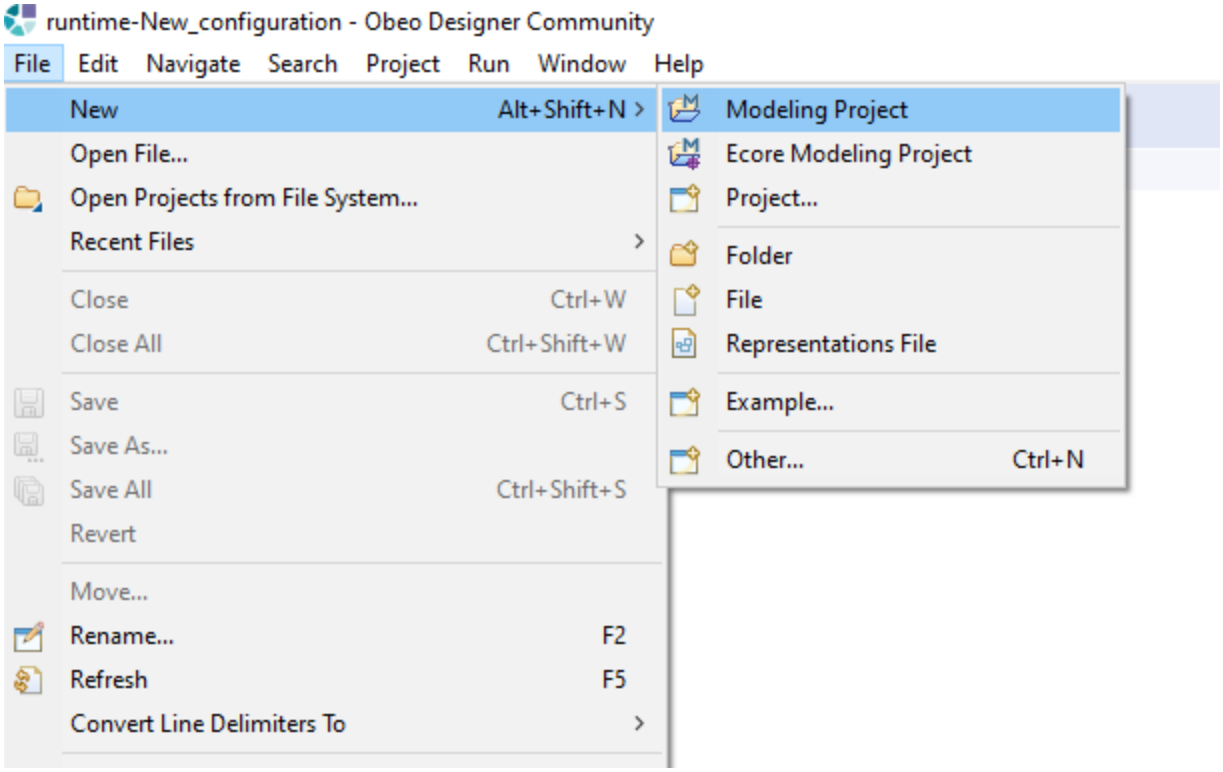
Select “Eclipse Application” for a new configuration:



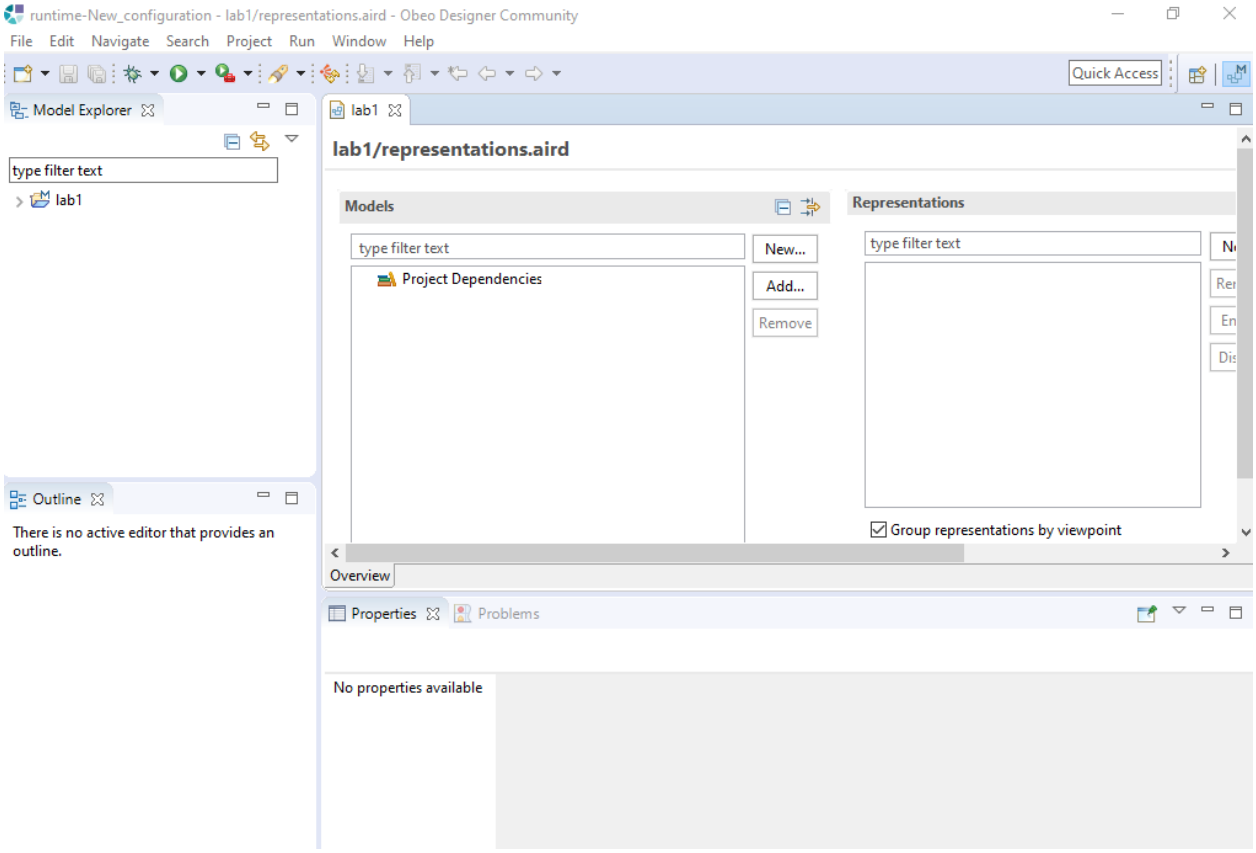
And you can rename at will (for instance to “test” instead of “New_Configuration”).

A second instance of Eclipse should run.

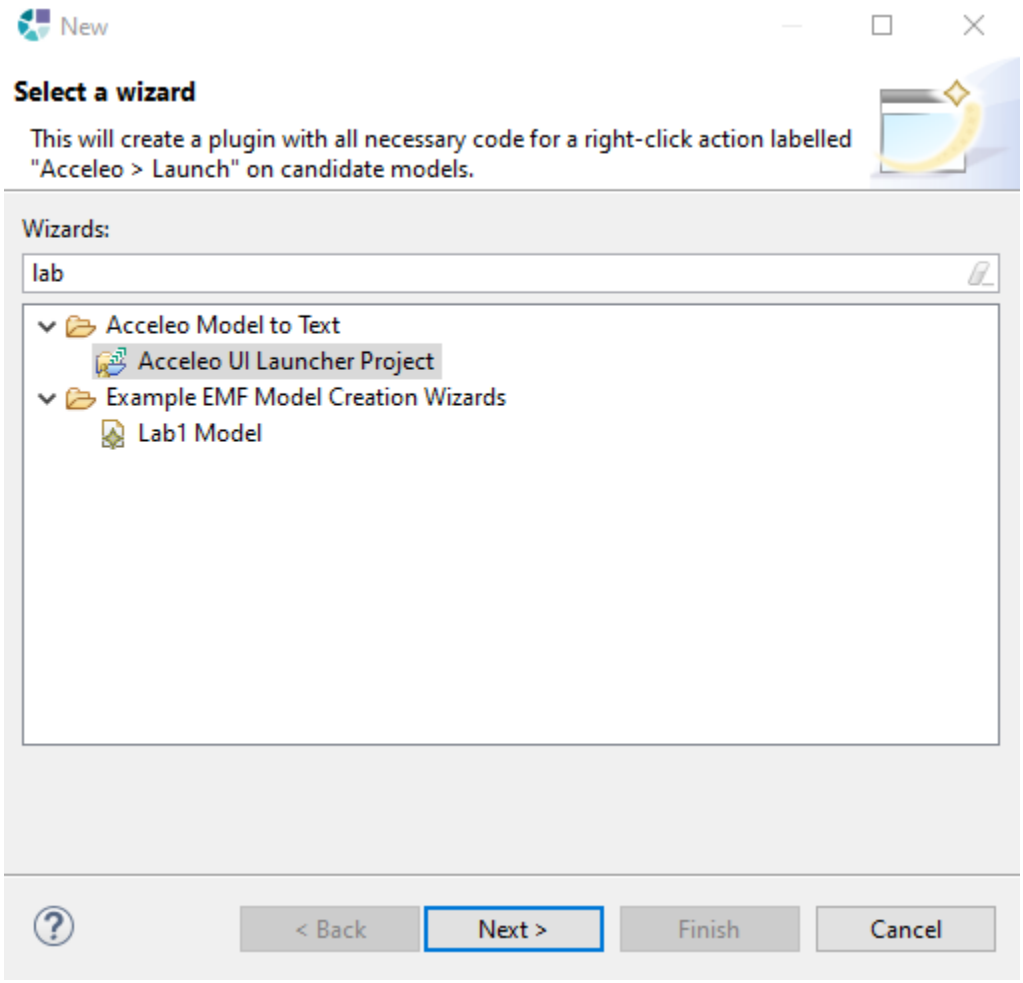
Create a new modelling project



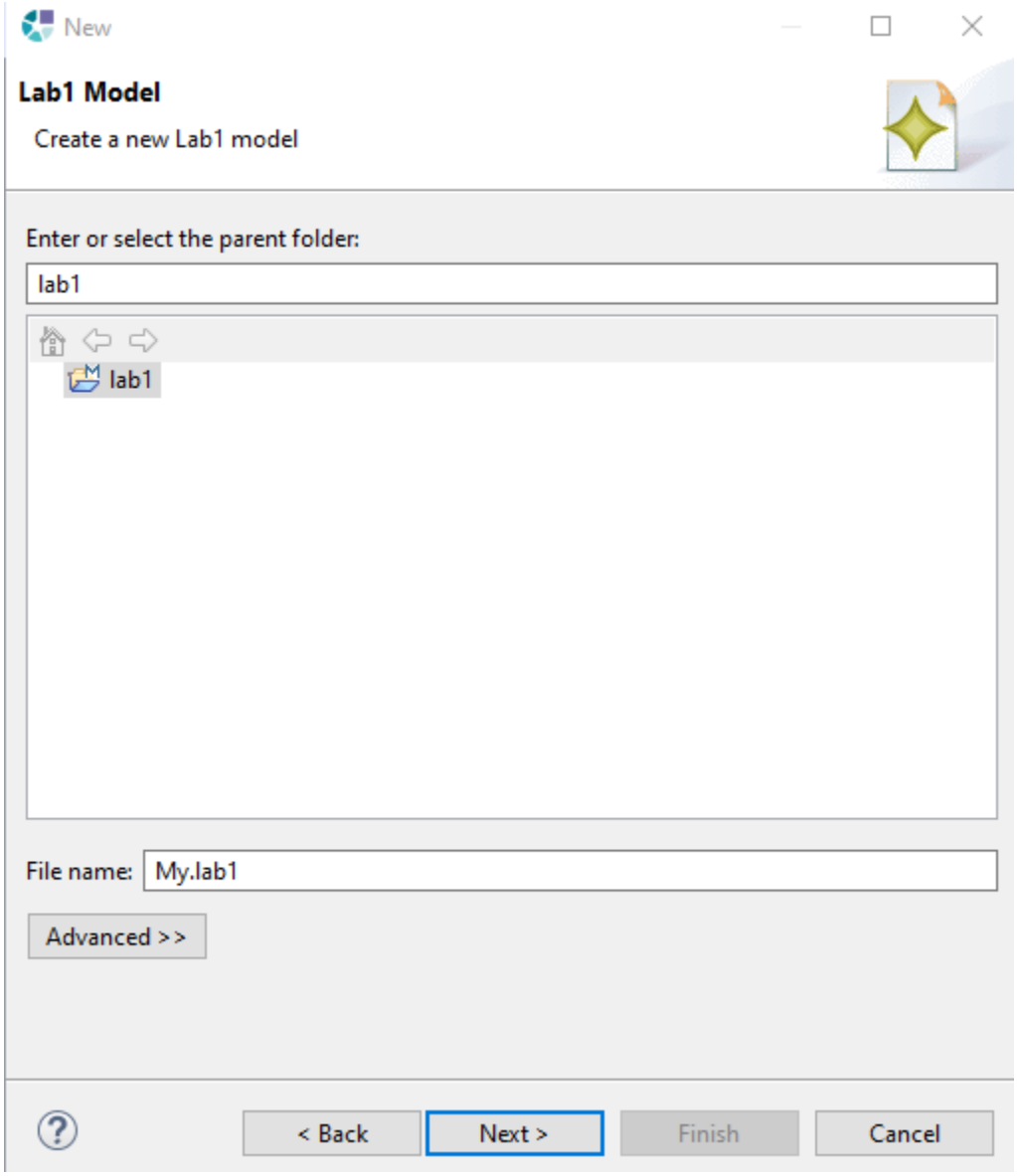
Give it a name, for instance lab1



right -click on lab1 and select new->other and write lab, you will see this:

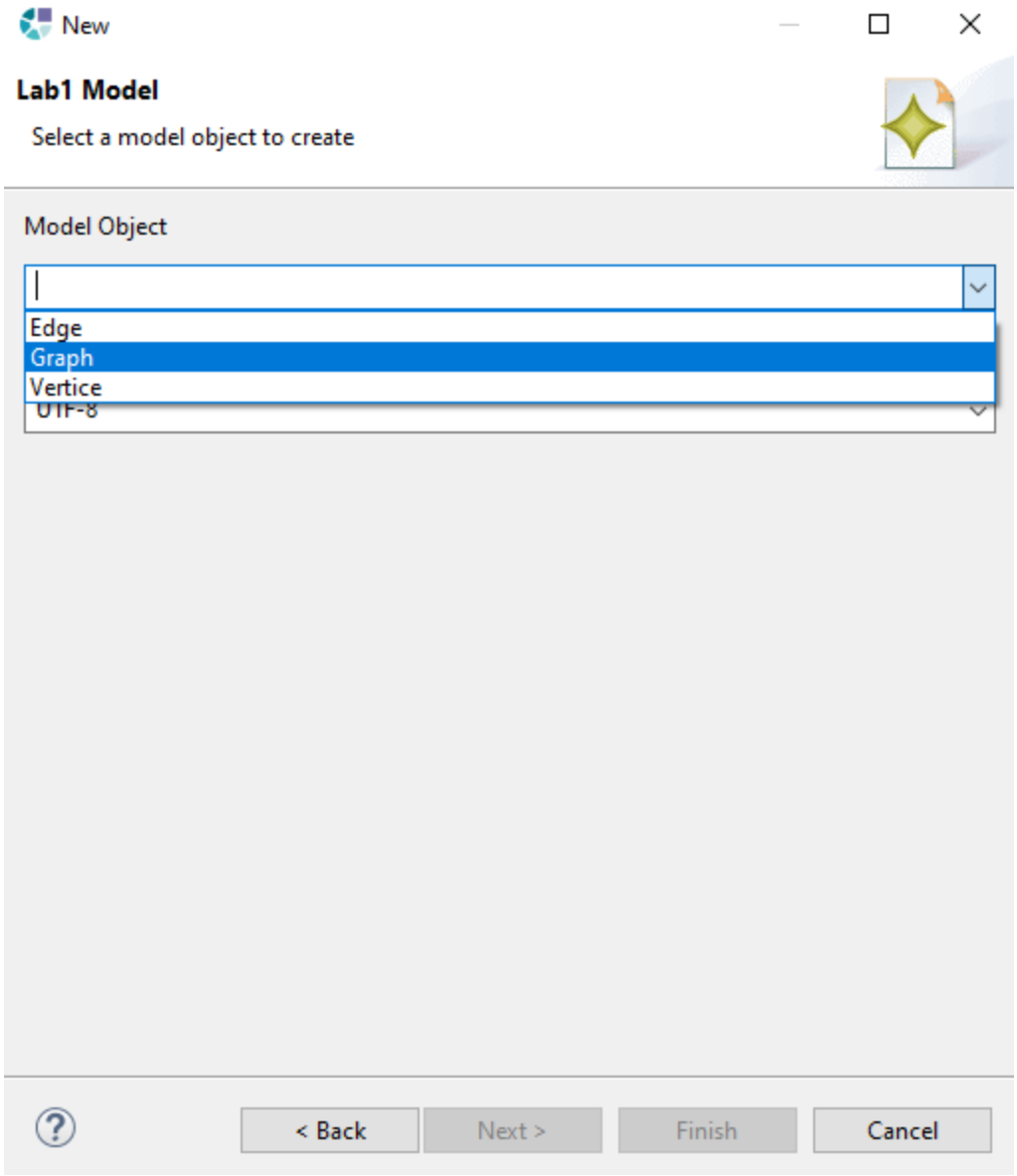


Select lab 1 Model

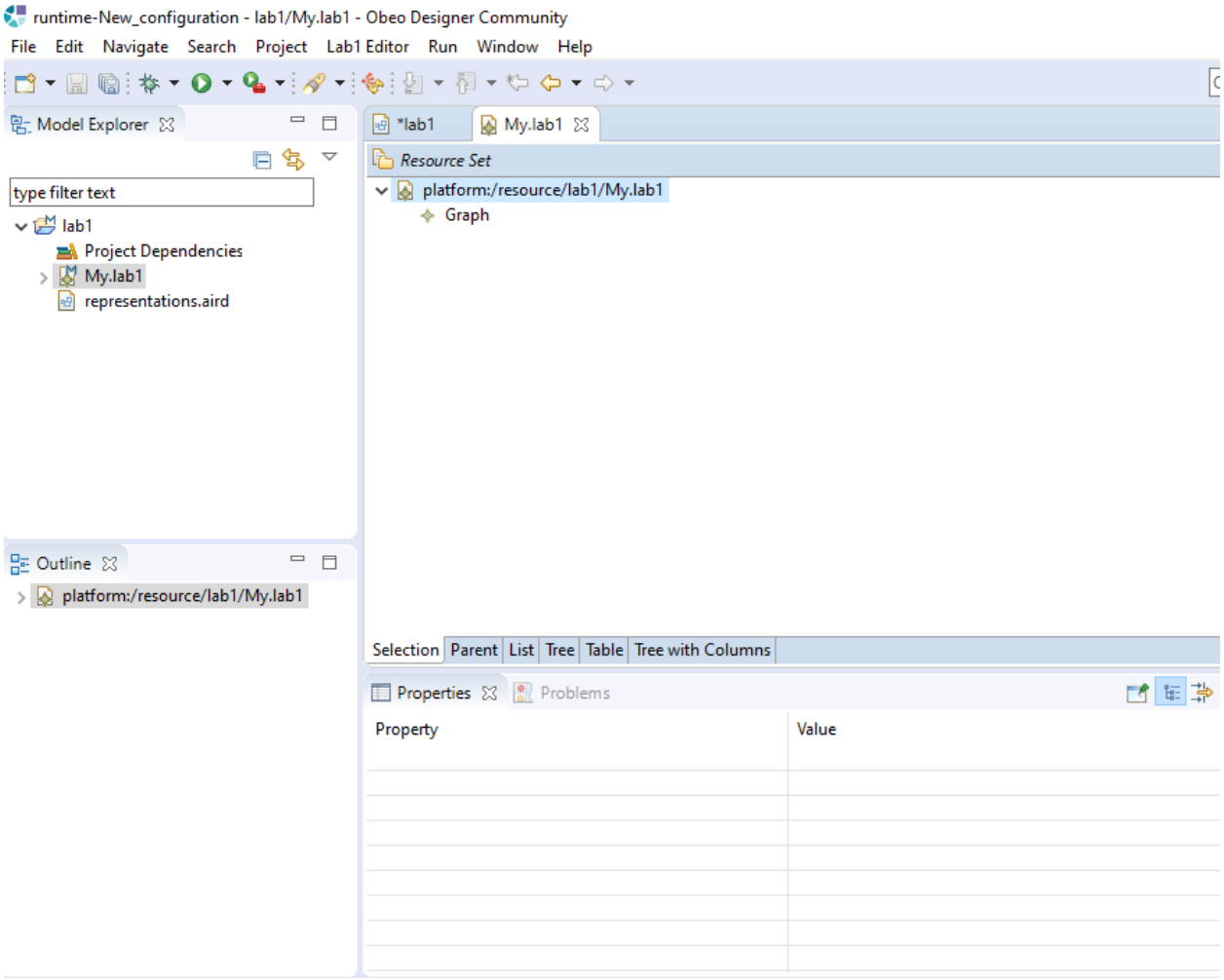


Change the name, if you prefer, and click next

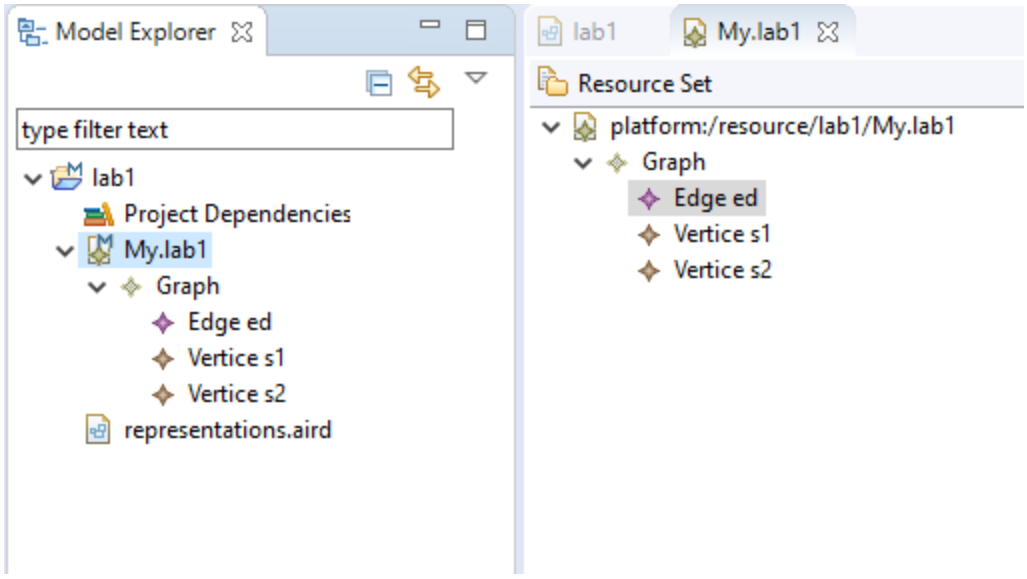
Select Graph as Model Object (model root element)



You will see the following:

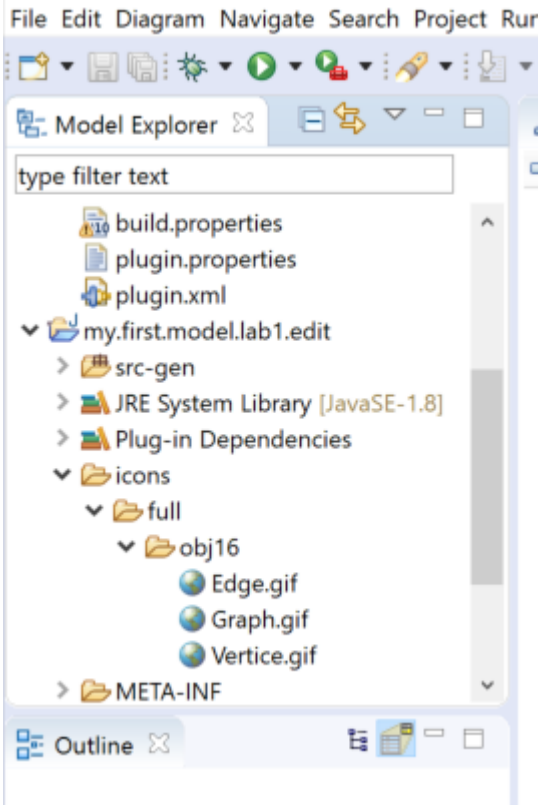


You can now add elements to the model in the right side:



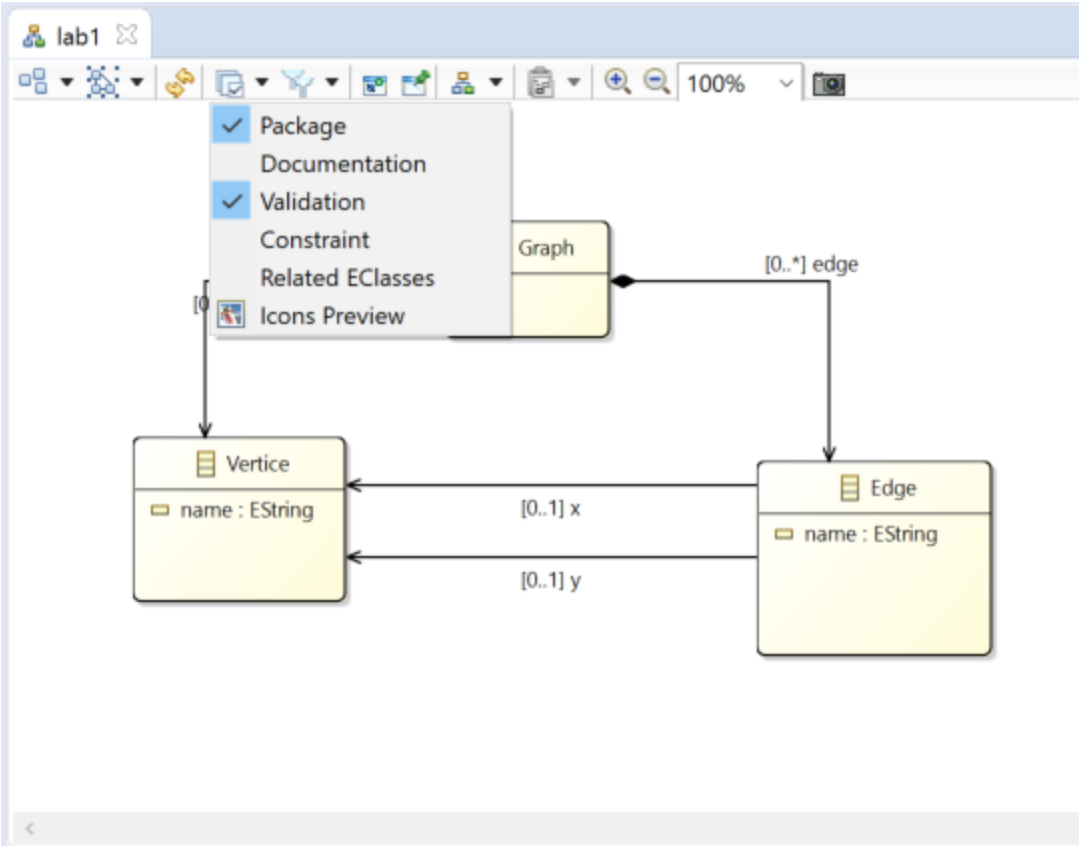
The tool will only allow you to insert instances that are valid according to the metamodel.

In case you want to customize the icons, you can change them just by going back to the first instance of eclipse and in the edit package look for the icons folder:

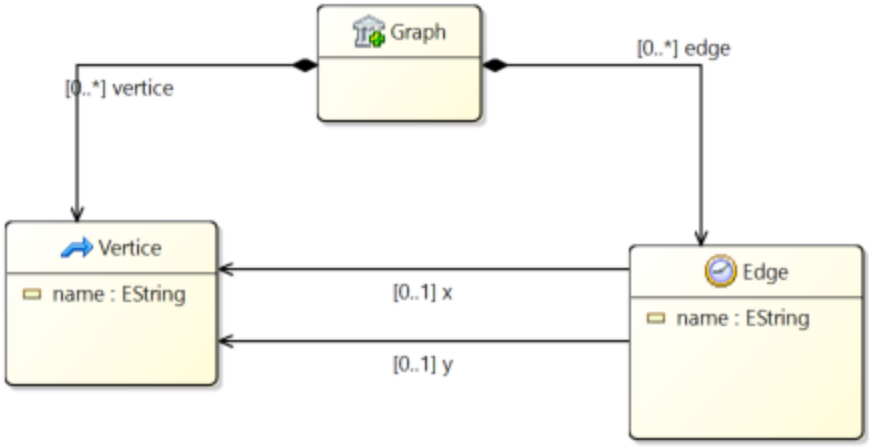


Select the icons you want to overwrite (should have the same names) and copy them there.

To preview, select in the metamodel on the right the following “Icons Preview” option on the ecore tools:

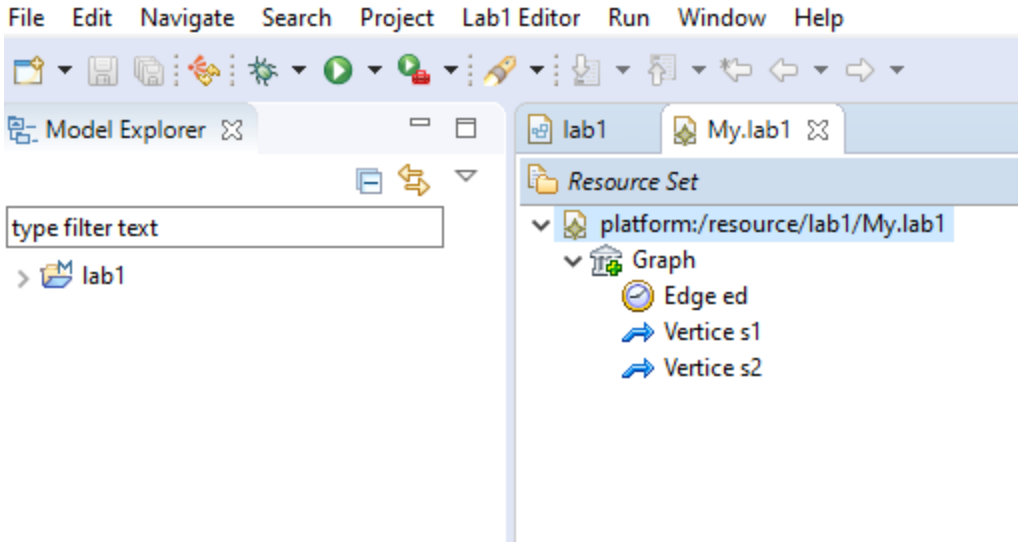


And there it goes:



So that the simple ecore editor is also updated we have to save this model and regenerate All

If we restart the workplace in the second eclipse instance, we will update the editor as well.



3. Turing Machines graphical language

The Turing machine was invented in 1936 by Alan Turing and is a mathematical model of computation of an abstract machine that operates on an unlimited tape (extending infinitely in one direction) according to programmed control logic. Can be seen as a finite-state machine in which a transition prints a symbol on the tape. The head of the tape may move to either left or right, allowing the machine to manipulate by reading and writing at will.

It can be defined as a 6-tuple **M= { Q,** $\sum$**,** Γ**, δ, q0, F }**, where:

- **Q** is a finite set of states,
- Γ is a finite set called the tape alphabet with a special symbol **B** that represents a blank cell in the tape,
- $\sum$ is a subset of Γ except B,
- **δ** is the transition function (a partial function from Q x Γ to Q x Γ x {L,R}),
- **q0** ∈ Q is the initial/start state,
- **F**⊆ Q is a set of final states.

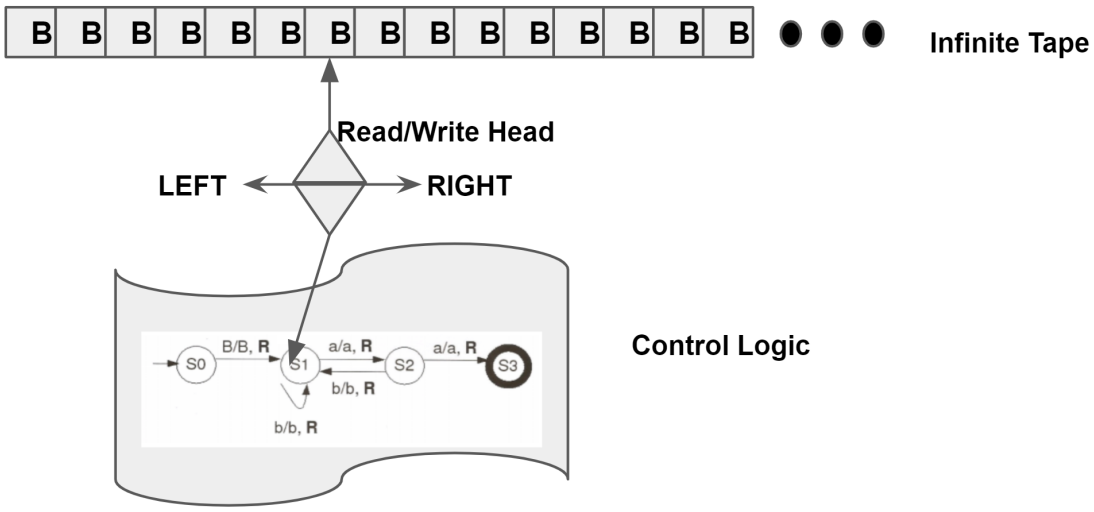
The transition function of a standard Turing machine with the input alphabet Γ={a,b} U {B} and the states Q = { s0,s1,s2,s3 }, where q0 = s0 and F = { s3 } with the program to accept strings with the following regular expression **`(a+b)*aa`** can be described as follows:

δ	B	a	b

>	s0	S1, B, R		
	s1		S1, a, R	S2, a, R
	s2		S3, b, R	S1, b, R
*	s3			

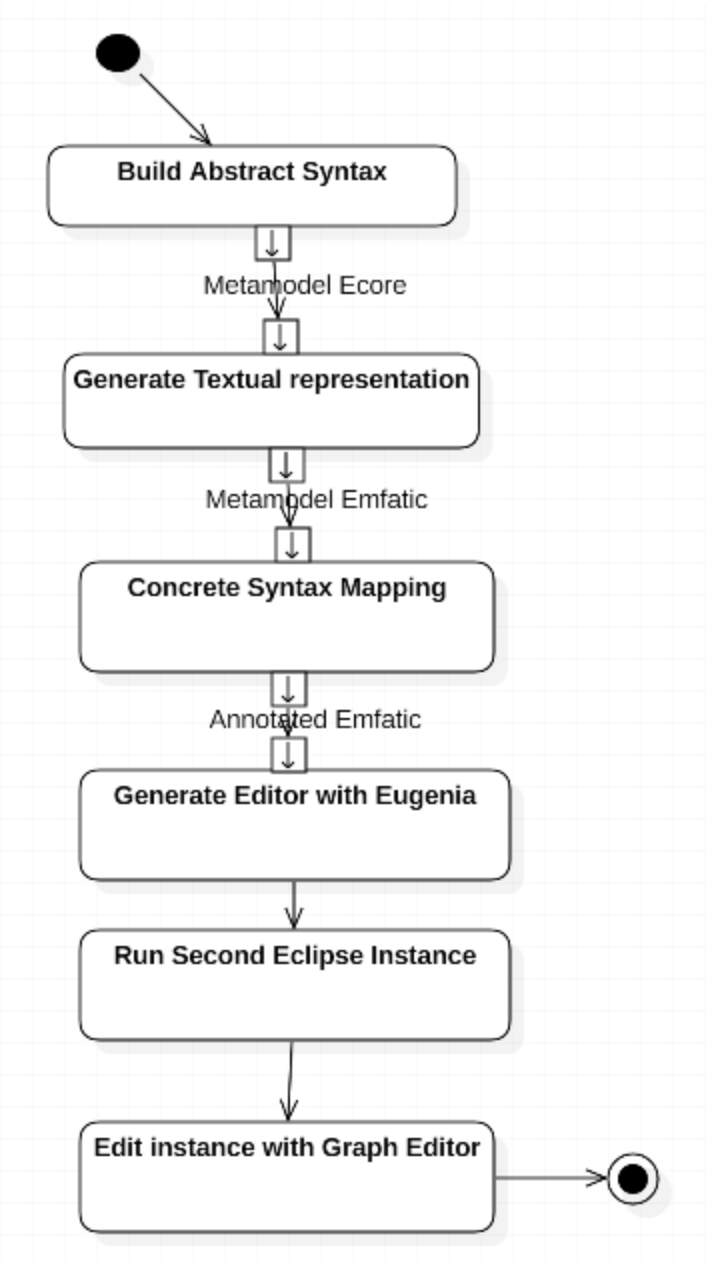
(* identifies a final state, while > tags the start state)

Consider the following visual notation to describe Turing machines

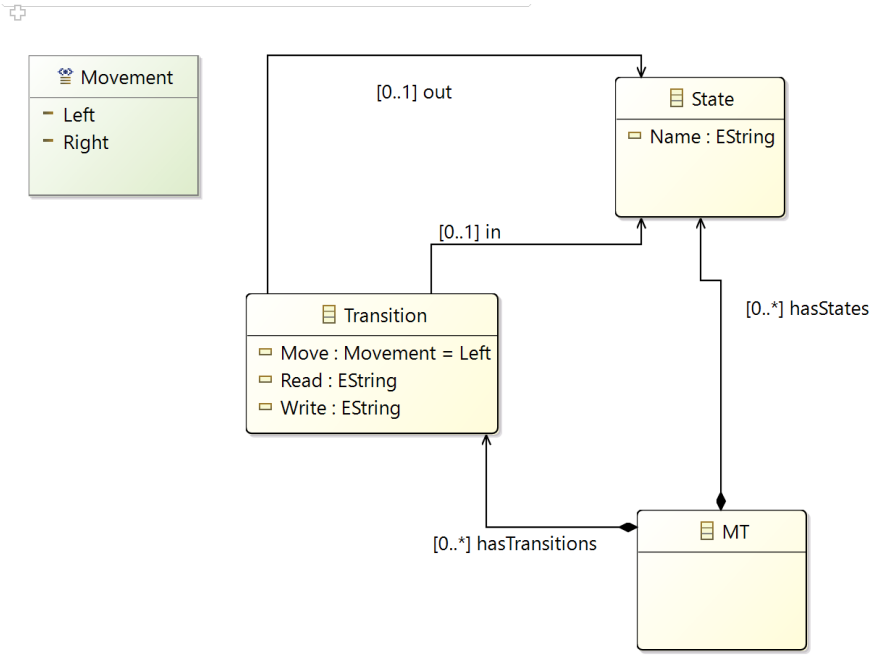


The transition function can be graphically represented by a state diagram:

Now, following the same process as the section before, we have:



Please follow the steps used to create the Graph editor for the Graph language. The metamodel in ecore can be the following.

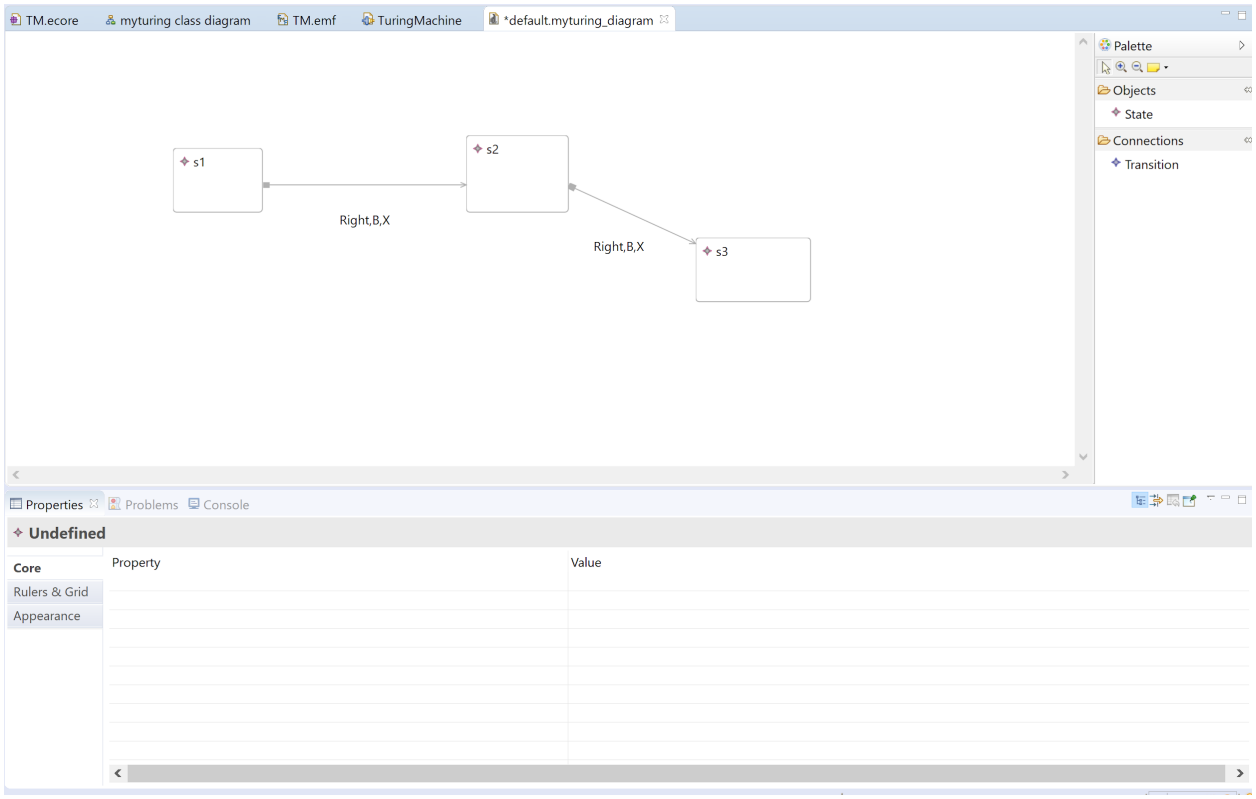


After generating the emfatic file with the textual representation of the metamodel, you can annotate (syntax mapping) in the following way:

```
@namespace(uri="http://myturing/1.0", prefix="myturing")
package myturing;

@gmf.diagram(foo="bar")
class MT {
  val State[*] hasStates;
  val Transition[*] hasTransitions;
}


@gmf.node(label = "Name", tool.name= "State")
class State {
  attr String Name;
}

@gmf.link(source="in", target="out", label="Move, Read, Write",
label.pattern="{0},{1},{2}",source.decoration="filledsquare")
class Transition {
  attr Movement Move;
  attr String Read;
  attr String Write;
  ref State in;
  ref State out;
}

enum Movement {
  Left = 0;
```

```
  Right = 1;
}
```

After generating the editor and running the second instance of Eclipse, you can build an instance model like the one in the following example:



4. The Traffic Lights Control language - Different target technology platforms

Traffic-lights are a common device of our daily life. We pass by them unaware that their behaviour varies more widely than we might expect. Different countries have different rules, equipment and standards (have a glimpse at this variety for instance in https://en.wikipedia.org/wiki/Traffic_light). We will next design a naive language for describing the behavior of a traffic-light like the one in the figure above. In the process we will introduce the notion of well-formed instance models and how to specify well formedness rules using EVL. We will also show how to generate code from our models to two different target technologies: an arduino controller and a simulator implemented in Java.

4.1. The control language editor and examples of well-formedness rules using EVL

The emfatic code for the editor:

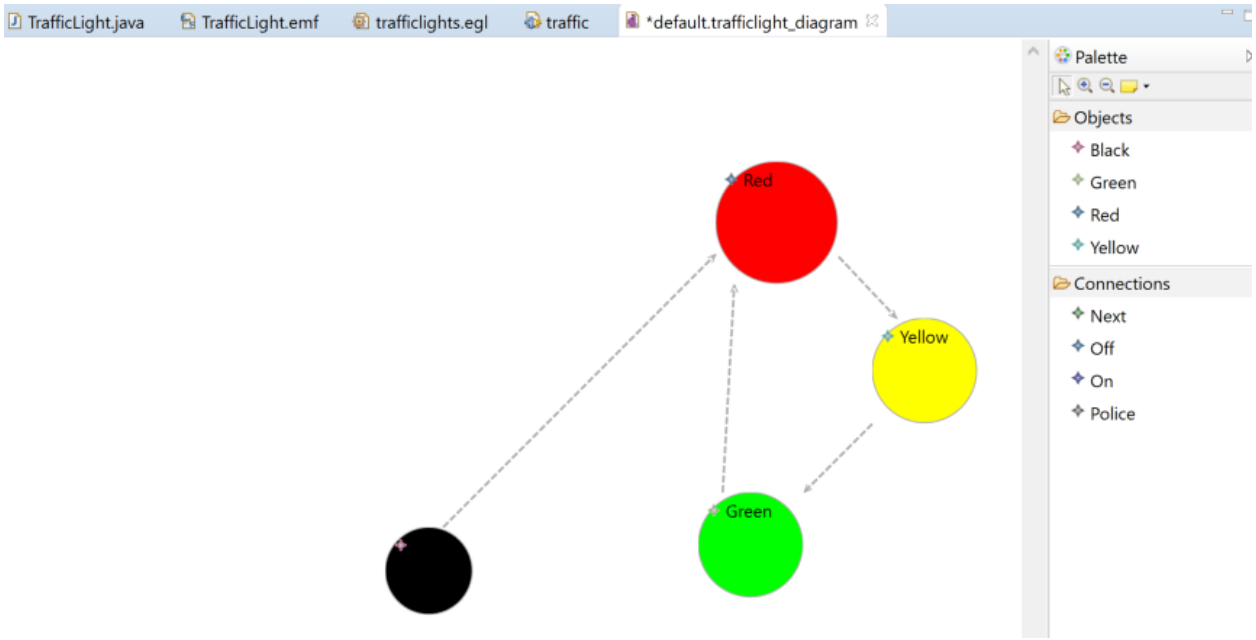
```
@namespace(uri="http://trafficlight/1.0", prefix="trafficlight")
package trafficlight;

@gmf.diagram
class Traffic {
  val Green hasGreen;
  val Red hasRed;
  val Yellow hasYellow;
  val Black hasBlack;
  val Off[*] hasOff;
  val On[*] hasOn;
  val Police[*] hasPolice;
  val Next[*] hasNext;
}

abstract class Light {
```

```
  attr String name;
  attr boolean IsStart;
}

@gmf.node(label="name", color="255,255,0", figure="ellipse")
class Yellow extends Light {
}

@gmf.node(label="name", color="0,255,0", figure="ellipse")
class Green extends Light {
}

@gmf.node(label="name", color="255,0,0", figure="ellipse")
class Red extends Light {
}

@gmf.node(label="name", color="0,0,0", figure="ellipse")
class Black extends Light {
}

class Change {
  ref Light hasInputLightState;
  ref Light hasOutputLightState;
  attr String name;
}

@gmf.link(source="hasInputLightState", target="hasOutputLightState", style="dot",
width="1", target.decoration="arrow", source.decoration="filledrhomb")
class On extends Change {
```

Example of Eclipse Validation Language (EVL) rules:

```
context Light {

constraint  AtLeastTwo {

       check: Light.allInstances.size() >=2

       message : 'There should be at least two lights'

       }

}
```

Example of an instance model:



4.2. Model-to-Text - Java Code Synthesis

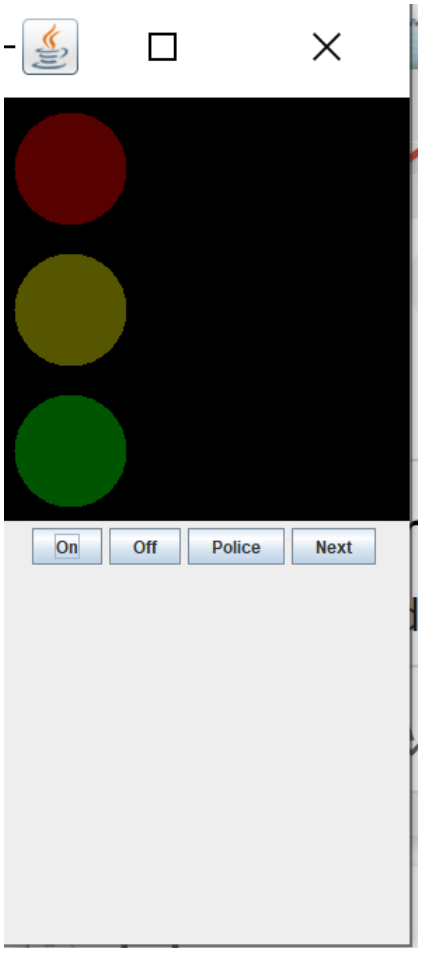
Below we can find an example of an egl code to generate Java code of a simulator of the traffic light control.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;

public class TrafficLight extends JFrame implements ActionListener {
    JButton bon, boff, bpolice, bnext;

    public static final int BLACK   = 0;
    public static final int GREEN   = 1;
    public static final int YELLOW  = 2;
    public static final int RED     = 3;

      Signal green = new Signal(Color.green);
      Signal yellow = new Signal(Color.yellow);
      Signal red = new Signal(Color.red);

      Integer presentState=BLACK;

    public TrafficLight(){
        super("Traffic Light");
```

```
        getContentPane().setLayout(new GridLayout(2, 1));

        bon = new JButton("On");
        boff = new JButton("Off");
        bpolice = new JButton("Police");
        bnext = new JButton("Next");

        bon.addActionListener(this);
        boff.addActionListener(this);
        bpolice.addActionListener(this);
        bnext.addActionListener(this);

        green.turnOn(false);
        yellow.turnOn(false);
        red.turnOn(false);

        JPanel p1 = new JPanel(new GridLayout(3,1));
        p1.add(red);
        p1.add(yellow);
        p1.add(green);

        JPanel p2 = new JPanel(new FlowLayout());
        p2.add(bon);
        p2.add(boff);
        p2.add(bpolice);
        p2.add(bnext);

        getContentPane().add(p1);
        getContentPane().add(p2);
        pack();
        }


    public static void main(String[] args){
        TrafficLight tl = new TrafficLight();
        tl.setVisible(true);
    }

    public void turnblack(){
        green.turnOn(false);
        yellow.turnOn(false);
        red.turnOn(false);
        presentState=BLACK;
    }

    public void turngreen(){
        green.turnOn(true);
        yellow.turnOn(false);
        red.turnOn(false);
        presentState=GREEN;
    }
    public void turnyellow(){
        green.turnOn(false);
```

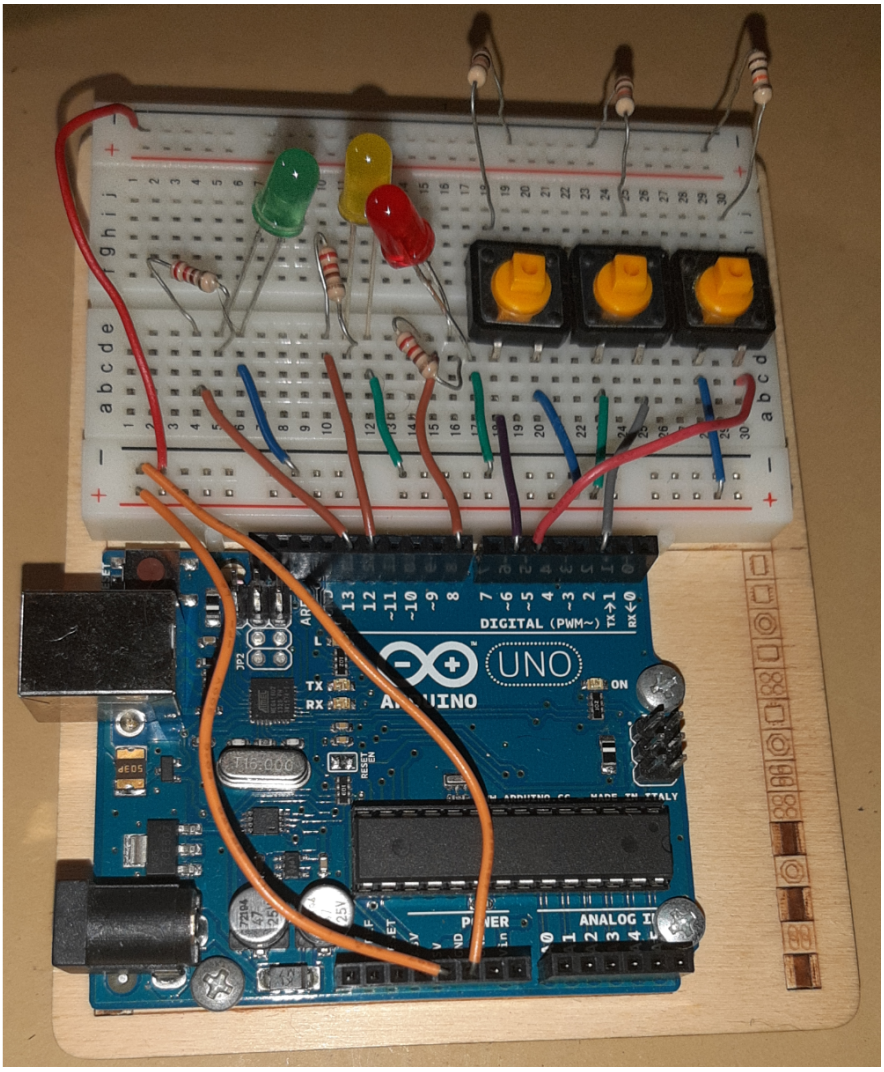
```
        yellow.turnOn(true);
        red.turnOn(false);
        presentState=YELLOW;
    }
    public void turnred(){
        green.turnOn(false);
        yellow.turnOn(false);
        red.turnOn(true);
        presentState=RED;
    }


    public void actionPerformed(ActionEvent e){

        if (e.getSource() == bon){

             switch(presentState){

 [% for (o in On.all) {%]
      case [%=o.hasInputLightState.type.name.toUpperCase()%]:
turn[%=o.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
             default: break;
             }

         } else if (e.getSource() == boff){
             switch(presentState){
[% for (o in Off.all) {%]
      case [%=o.hasInputLightState.type.name.toUpperCase()%]:
turn[%=o.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
               default: break;
             }

         } else if (e.getSource() == bpolice){
             switch(presentState){
[% for (p in Police.all) {%]
      case [%=p.hasInputLightState.type.name.toUpperCase()%]:
turn[%=p.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
             default: break;
             }

        } else if (e.getSource() == bnext){
             switch(presentState){
[% for (n in Next.all) {%]
      case [%=n.hasInputLightState.type.name.toUpperCase()%]:
turn[%=n.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
             default: break;
             }
        }
    }
```

```
}


class Signal extends JPanel{

    Color on;
    int radius = 40;
    int border = 10;
    boolean change;

    Signal(Color color){
        on = color;
        change = true;
    }

    public void turnOn(boolean a){
        change = a;
        repaint();
    }

    public Dimension getPreferredSize(){
        int size = (radius+border)*2;
        return new Dimension( size, size );
    }

    public void paintComponent(Graphics g){
        g.setColor( Color.black );
        g.fillRect(0,0,getWidth(),getHeight());

        if (change){
            g.setColor( on );
        } else {
            g.setColor( on.darker().darker().darker() );
        }
        g.fillOval( border,border,2*radius,2*radius );
    }
```



4.2. Model-to-Text - generating to Arduino as target platform

Suppose we have an arduino implementation of the traffic-lights controller with the following hardware setup:



Below we can find an example of an egl code to generate the controller for an arduino setup of the traffic light control.

```
int next=0;
int onoff=0;
int police=0;

const int BLACK   = 0;
const int GREEN   = 1;
const int YELLOW  = 2;
const int RED     = 3;

const int EONOFF = 1;
const int EPOLICE = 2;
const int ENEXT = 3;

int presentState=BLACK;
```

```
void currentState() {
             switch(presentState){

      case BLACK: black(); break;
      case GREEN: green(); break;
      case YELLOW: yellow(); break;
      case RED: red(); break;

               default: break;
             }
}

void black(){
  digitalWrite(11,LOW);
  digitalWrite(12,LOW);
  digitalWrite(13,LOW);
    }

void green(){
  digitalWrite(11,HIGH);
  digitalWrite(12,LOW);
  digitalWrite(13,LOW);
    }

void yellow(){
  digitalWrite(11,LOW);
  digitalWrite(12,HIGH);
  digitalWrite(13,LOW);
    }

void red(){
  digitalWrite(11,LOW);
  digitalWrite(12,LOW);
  digitalWrite(13,HIGH);
    }


void turnblack(){
      presentState=BLACK;
  }

void turngreen(){
        presentState=GREEN;

    }

void turnyellow(){
        presentState=YELLOW;
    }

void turnred(){
        presentState=RED;
```

```
    }

void actionPerformed(int e){

if (e==EONOFF){

             switch(presentState){
[% for (o in On.all) {%]
      case [%=o.hasInputLightState.type.name.toUpperCase()%]:
turn[%=o.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
[% for (o in Off.all) {%]
      case [%=o.hasInputLightState.type.name.toUpperCase()%]:
turn[%=o.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
              default: break;

         };
} else if (e == EPOLICE){
             switch(presentState){
[% for (p in Police.all) {%]
      case [%=p.hasInputLightState.type.name.toUpperCase()%]:
turn[%=p.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
             default: break;
             }

        } else if (e == ENEXT){
             switch(presentState){

 [% for (n in Next.all) {%]
      case [%=n.hasInputLightState.type.name.toUpperCase()%]:
turn[%=n.hasOutputLightState.type.name.toLowerCase()%](); break;
 [%}%]
             default: break;
             }
        };
        delay(250);
    }


void setup() {
  // put your setup code here, to run once:
pinMode(13, OUTPUT);
pinMode(12, OUTPUT);
pinMode(11, OUTPUT);
pinMode(4, INPUT);
pinMode(3, INPUT);
pinMode(2, INPUT);

}

void loop() {
```

```
next=digitalRead(2);
police=digitalRead(3);
onoff=digitalRead(4);


  int tmp=0;
  if (onoff==HIGH) tmp=EONOFF;
  else if (police==HIGH) tmp=EPOLICE;
  else if (next==HIGH) tmp=ENEXT;

  if(tmp>0) actionPerformed(tmp);
  currentState();
}
```

The output TraffiLight.java file will be:

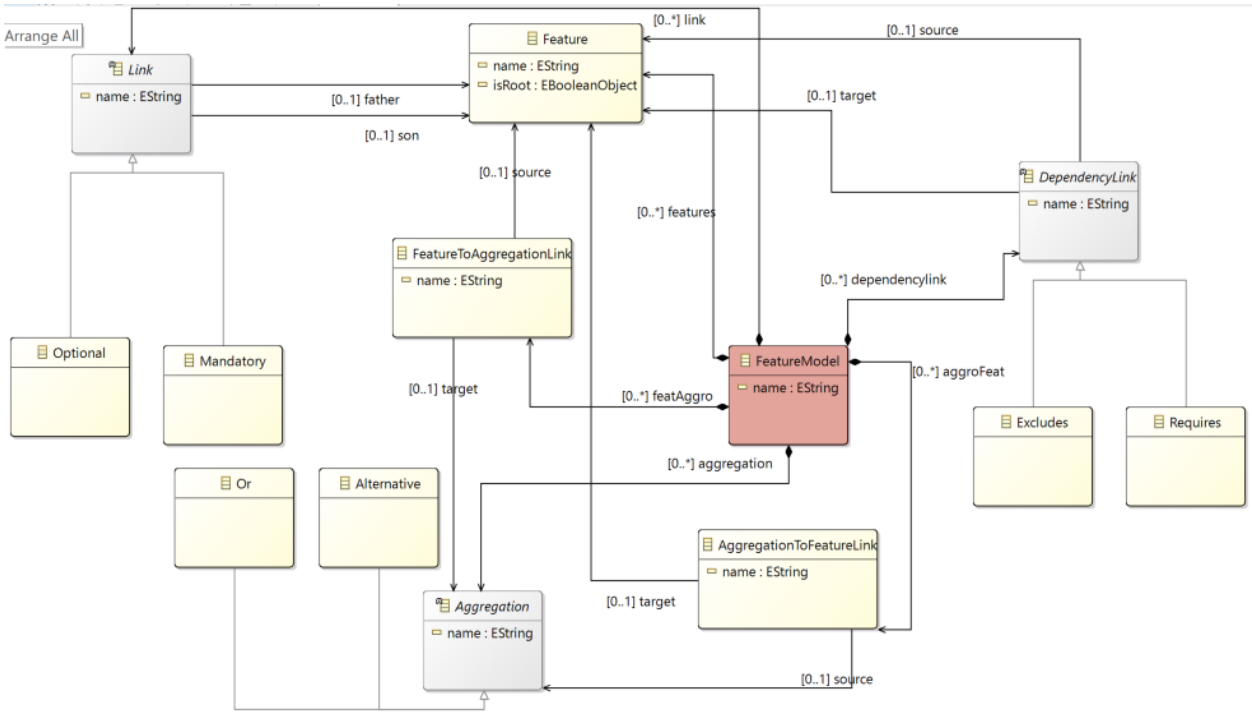
```
import java.awt.*;

import javax.swing.*;

import java.awt.event.*;

import javax.swing.border.*;


public class TrafficLight extends JFrame implements ActionListener {
   JButton bon, boff, bpolice, bnext;

   public static final int BLACK   = 0;
   public static final int GREEN   = 1;
   public static final int YELLOW  = 2;
   public static final int RED     = 3;

     Signal green = new Signal(Color.green);
     Signal yellow = new Signal(Color.yellow);
     Signal red = new Signal(Color.red);

     Integer presentState=BLACK;

   public TrafficLight(){
      super("Traffic Light");
```

```
    getContentPane().setLayout(new GridLayout(2, 1));

    bon = new JButton("On");
    boff = new JButton("Off");
    bpolice = new JButton("Police");
    bnext = new JButton("Next");

    bon.addActionListener(this);
    boff.addActionListener(this);
    bpolice.addActionListener(this);
    bnext.addActionListener(this);

    green.turnOn(false);
    yellow.turnOn(false);
    red.turnOn(false);

    JPanel p1 = new JPanel(new GridLayout(3,1));
    p1.add(red);
    p1.add(yellow);
    p1.add(green);

    JPanel p2 = new JPanel(new FlowLayout());
    p2.add(bon);
    p2.add(boff);
    p2.add(bpolice);
    p2.add(bnext);

    getContentPane().add(p1);
    getContentPane().add(p2);
    pack();
    }


public static void main(String[] args){
    TrafficLight tl = new TrafficLight();
    tl.setVisible(true);
}

public void turnblack(){
        green.turnOn(false);
    yellow.turnOn(false);
    red.turnOn(false);
    presentState=BLACK;
}

public void turngreen(){
        green.turnOn(true);
```

```
    yellow.turnOn(false);
    red.turnOn(false);
    presentState=GREEN;
}
public void turnyellow(){
        green.turnOn(false);
    yellow.turnOn(true);
    red.turnOn(false);
    presentState=YELLOW;
}
public void turnred(){
        green.turnOn(false);
    yellow.turnOn(false);
    red.turnOn(true);
    presentState=RED;
}


public void actionPerformed(ActionEvent e){

    if (e.getSource() == bon){

        switch(presentState){


        default: break;
        }

     } else if (e.getSource() == boff){
        switch(presentState){

           default: break;
        }

     } else if (e.getSource() == bpolice){
        switch(presentState){

        default: break;
        }

    } else if (e.getSource() == bnext){
        switch(presentState){
  case BLACK: turnred(); break;
  case RED: turnyellow(); break;
  case YELLOW: turngreen(); break;
  case GREEN: turnred(); break;
```

```
            default: break;
            }
        }
    }

}


class Signal extends JPanel{

    Color on;
    int radius = 40;
    int border = 10;
    boolean change;

    Signal(Color color){
        on = color;
        change = true;
    }

    public void turnOn(boolean a){
        change = a;
        repaint();
    }

    public Dimension getPreferredSize(){
        int size = (radius+border)*2;
        return new Dimension( size, size );
    }

    public void paintComponent(Graphics g){
        g.setColor( Color.black );
        g.fillRect(0,0,getWidth(),getHeight());

        if (change){
            g.setColor( on );
        } else {
            g.setColor( on.darker().darker().darker() );
        }
        g.fillOval( border,border,2*radius,2*radius );
    }
}
```

5. The Feature Models Language

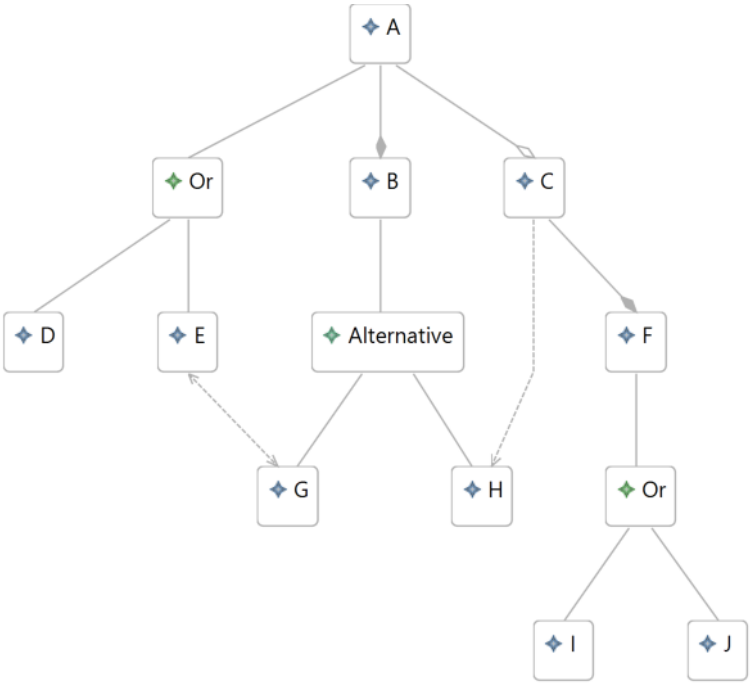


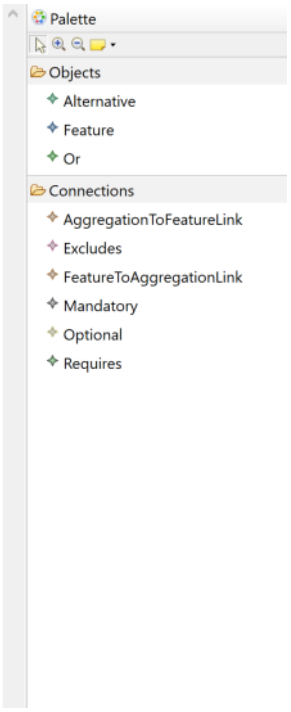
Next, we can see a short implementation of a feature models editor without much concerns related to the usability (naive and simplistic concrete syntax):

```
@namespace(uri="FeatureDiagram", prefix="FeatureDiagram")
package FeatureDiagram;

@gmf.diagram
class FeatureModel {
  attr String name;
  val Link[*] link;
  val Feature[*] features;
  val Aggregation[*] aggregation;
  val FeatureToAggregationLink[*] featAggro;
  val AggregationToFeatureLink[*] aggroFeat;
  val DependencyLink[*] dependencylink;
}

@gmf.node(label="name")
class Feature {
  attr String name;
  attr Boolean isRoot;
```

```
}

abstract class Link {
  attr String name;
  ref Feature father;
  ref Feature son;
}

@gmf.link(source="father", target="son", target.decoration="filledrhomb")
class Mandatory extends Link {
}

@gmf.link(source="father", target="son", target.decoration="rhomb")
class Optional extends Link {
}

abstract class Aggregation {
  attr String name;
}

@gmf.node(label="name")
class Alternative extends Aggregation {
}

@gmf.node(label="name")
class Or extends Aggregation {
}

@gmf.link(source="source", target="target")
class FeatureToAggregationLink{
  attr String name;
  ref Feature source;
  ref Aggregation target;
}

@gmf.link(source="source", target="target")
class AggregationToFeatureLink{
  attr String name;
  ref Aggregation source;
  ref Feature target;
}


abstract class DependencyLink {
  attr String name;
  ref Feature source;
  ref Feature target;
}

@gmf.link(source="source", target="target", source.decoration="arrow", target.decoration="arrow",
style="dash")
class Excludes extends DependencyLink {
}

@gmf.link(source="source", target="target", target.decoration="arrow", style="dash")
class Requires extends DependencyLink {
}
```

A snapshot of the editor looks like (an example of usability problems related to the chosen concrete syntax):





The corresponding validation rules are:

```
context FeatureModel{
	constraint mustHaveName {
		check: self.name <> ''

		message: 'A Feature Diagram must have a name'

		fix {
			title: 'Insert ' + self.eClass().name + ' Name'

			do {
				var name := UserInput.prompt('Insert ' + self.eClass().name + ' Name',
'');

				self.name = name;
			}
		}
	}
}

context Feature {
	constraint noFatherOfSelf{
		check: Link.allInstances.select(l | l.father.name = l.son.name and self.name =
l.son.name).size() = 0

		message: 'Features cant be father of themselves'
	}

	constraint noNameRepetions {
		check: Feature.allInstances.excluding(self)->forAll(i | i.name <> self.name)

		message: 'Two Features cant have the same name'
	}

	constraint noRootFeatures {
		guard: Feature.allInstances.select(f | f.isRoot = true).size() = 0
```

```
                check {
                        return Feature.allInstances.select(f | f.isRoot = true).size() <> 0;
                        }

                message: 'There are no Root Features. This can be fixed by Quick Fix'

                fix {
                        title: 'Root ' + self.eClass().name + ' is needed'

                        do {
                                var rootFeatureSequence = Feature.allInstances.select(f |
Link.allInstances.forAll(l | l.son.name <> f.name)
                                        and AggregationToFeatureLink.allInstances.forAll(l |
l.target.name <> f.name));

                                if(rootFeatureSequence.size() > 1) {
                                        rootFeatureSequence.size().println('Size do If: ');
                                        var sonFeature := UserInput.choose('Select feature',
_Model.getAllOfType(self.eClass().name));

                                sonFeature.isRoot = true;
                                } else {
                                        rootFeatureSequence[0].isRoot = true;
                                }
                        }
                }
        }

        constraint onlyOneRootFeature {
                guard: Feature.allInstances.select(f | f.isRoot = true).size() > 0
                check: Feature.allInstances.select(f | f.isRoot = true).size() = 1 or not (self.isRoot
= true)

                message: 'There can only be one Root Feature'
        }

        constraint nonRootFeatureMustHaveAFather {
                check: Link.allInstances.select(l | l.son.name = self.name).size() +
AggregationToFeatureLink.allInstances.select(l | l.target.name = self.name).size() = 1 or self.isRoot =
true

                message: 'Non-Root Features can only have one father feature'
        }
}

context Aggregation {
        constraint atLeastTwoFeatures{
                check: AggregationToFeatureLink.allInstances.select(l | l.source = self).size() > 1

                message: 'Aggregation Nodes must have at least two child Features'
        }
}

context Link {
        constraint notFatherOfRoot {
                check: self.son.isRoot <> true

                message: 'The Root Feature cant have fathers'
        }

}

//Excludes
operation DependencyLink noExcludesMandatories() : Boolean {
        var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
```

```
        var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
        return excludesSourceLinks.eClass().name <> 'Mandatory' and excludesSourceLinks.eClass().name
<> 'Mandatory'
         or excludesSourceLinks.father.name <> excludesTargetLinks.father.name;
}

operation DependencyLink noExcludesAlternative() : Boolean {
        var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
        var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return excludesSourceLinks.source.name <> excludesTargetLinks.source.name or
excludesSourceLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noExcludesBetweenMandatoryAndAlternative() : Boolean {
        var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return excludesSourceLinks.eClass().name <> 'Mandatory' or
excludesTargetLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noExcludesBetweenAlternativeAndMandatory() : Boolean {
        var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
        var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
        return excludesSourceLinks.source.eClass().name <> 'Alternative' or
excludesTargetLinks.eClass().name <> 'Mandatory';
}

operation DependencyLink noExcludesBetweenMandatoryAndOr() : Boolean {
        var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return excludesSourceLinks.eClass().name <> 'Mandatory' or
excludesTargetLinks.source.eClass().name <> 'Or'
                or AggregationToFeatureLink.allInstances.select(l | l.source =
excludesTargetLinks.source).size() > 2;
}

operation DependencyLink noExcludesBetweenOrAndMandatory() : Boolean {
        var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
        var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
        return excludesSourceLinks.source.eClass().name <> 'Or' or excludesTargetLinks.eClass().name <>
'Mandatory'
                or AggregationToFeatureLink.allInstances.select(l | l.source =
excludesSourceLinks.source).size() > 2;
}

//Requires
operation DependencyLink noRequiresMandatory() : Boolean {
        var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var requiresTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
        return requiresTargetLinks.eClass().name <> 'Mandatory' or requiresSourceLinks.father.name <>
requiresTargetLinks.father.name;
}

operation DependencyLink noRequiresAlternative() : Boolean {
        var requiresSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
        var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return requiresSourceLinks.source <> requiresTargetLinks.source or
```

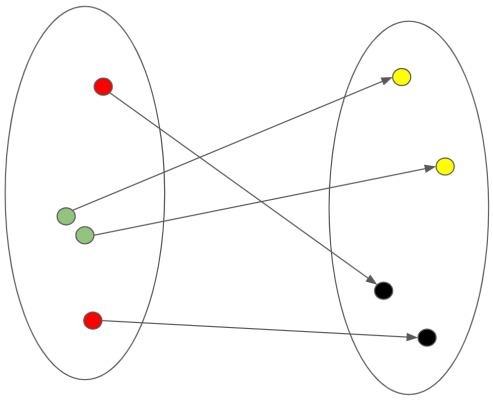
```
requiresSourceLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noRequiresBetweenMandatoryAndAlternative() : Boolean {
        var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noRequiresBetweenMandatoryAndOptionalAlternative() : Boolean {
        var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        var featureToAggregation = FeatureToAggregationLink.allInstances.select(l | l.target =
requiresTargetLinks.source)[0];
        var featureLink = Link.allInstances.select(l | l.son.name =
featureToAggregation.source.name)[0];
        return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Alternative'
                or featureLink.eClass().name <> 'Optional';
}

operation DependencyLink noRequiresBetweenMandatoryAndOr() : Boolean {
        var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
        return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Or';
}

operation DependencyLink noRequiresMandatoryAndOptional() : Boolean {
        var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
        var requiresTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
        return requiresSourceLinks.eClass().name <> 'Mandatory' or requiresTargetLinks.eClass().name <>
'Optional'
         or requiresSourceLinks.father.name <> requiresTargetLinks.father.name;
}

context Excludes {
        constraint noExcludesBetweenMandatoryAndOpcional {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
                check: self.noExcludesMandatories()

                message: 'Mandatory features cant be the source or target of an Excludes dependency
with other feature that shares the same father'
        }

        constraint noExcludesAndAlternative {
                guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noExcludesAlternative()

                message: 'This Excludes Dependency is Redundant'
        }

        constraint noExcludesBetweenMandatoryAndAlternative {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noExcludesBetweenMandatoryAndAlternative()
```

```
                message: 'The feature ' + self.target.name + ' is a Dead Feature'
        }

        constraint noExcludesBetweenAlternativeAndMandatory{
                guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
                        and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
                check: self.noExcludesBetweenAlternativeAndMandatory()

                message: 'The feature ' + self.source.name + ' is a Dead Feature'
        }

        constraint noExcludesBetweenMandatoryAndOr {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noExcludesBetweenMandatoryAndOr()

                message: 'The Feature ' + self.target.name + ' is a False Optional Feature'
        }

        constraint noExcludesBetweenOrAndMandatory{
                guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
                        and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
                check: self.noExcludesBetweenOrAndMandatory()

                message: 'The Feature ' + self.source.name + ' is a False Optional Feature'
        }
}

context Requires{
        constraint noRequiresAndAlternative{
                guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noRequiresAlternative()

                message: 'The Feature ' + self.source.name + ' is a Dead Feature'
        }

        constraint noRequiresAndMandatory {
                guard: Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
                        and Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                check: self.noRequiresMandatory()

                message: 'Mandatory Features being the target of a Requires Dependency whose source
shares the same father is Redundant'
        }

        constraint noRequiresBetweenMandatoryAndOptional {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0

                check: self.noRequiresMandatoryAndOptional()

                message: 'This Requires Dependency makes the non-target Feature (from the same Or node)
a false optional feature'
        }

        constraint onlyOneRequires {
                check: Requires.allInstances.select(i | i.target.name = self.target.name).size() = 1
```

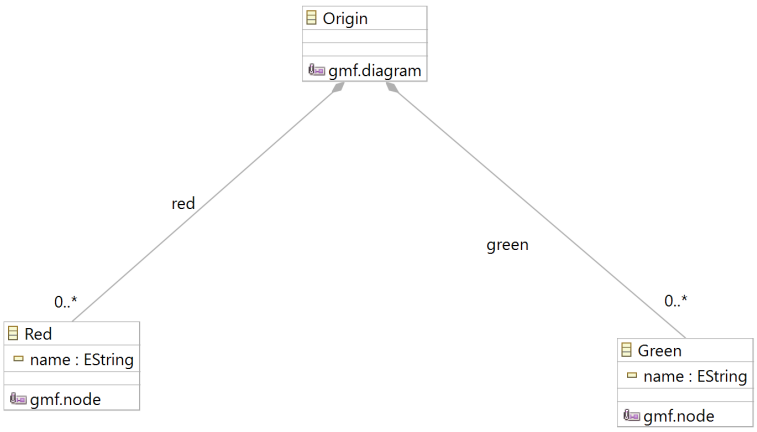
```
                message: 'More than one Requires Dependency is Redundant'
        }

        constraint noRequiresBetweenMandatoryAndAlternative {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noRequiresBetweenMandatoryAndAlternative()

                message: 'This Requires Dependency makes the non-target Features (from the same
Alternative node) Dead Features'
        }

        constraint noRequiresBetweenMandatoryAndOptionalAlternative {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noRequiresBetweenMandatoryAndOptionalAlternative()

                message: 'This Requires Dependency makes ' + self.target.name + ' a false optional
feature'
        }

                constraint noRequiresBetweenMandatoryAndOr {
                guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                        and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
                check: self.noRequiresBetweenMandatoryAndOr()

                message: 'This Requires Dependency makes ' + self.target.name + ' a false optional
feature'
        }

        constraint noRequiresBetweenFatherAndSon {
                check: Link.allInstances().select(l | l.father.name = self.source.name and l.son.name =
self.target.name).size() = 0

                message: 'Father and son features cant be related by a dependency relationship'
        }
}

context Alternative {
        constraint eachAlternativeHasOnlyOneFatherFeature {
                check: FeatureToAggregationLink.allInstances.select(l | l.target = self).size() = 1

                message: 'An Alternative Node can only have one Father Feature'
        }
}

context Or {
        constraint eachOrHasOnlyOneFatherFeature {
                check: FeatureToAggregationLink.allInstances.select(l | l.target = self).size() = 1

                message: 'An Or Node can only have one Father Feature'
        }
}
```

6. Translating colours - a simple example of model transformations

As a kind of “Hello World” example of model transformations, let us consider the case of two trivial visual languages, the one that describes red and green elements, and the one that describes black and yellow elements. Let's suppose that we want to simply translate the elements of one to the other as exemplified in the following illustration:



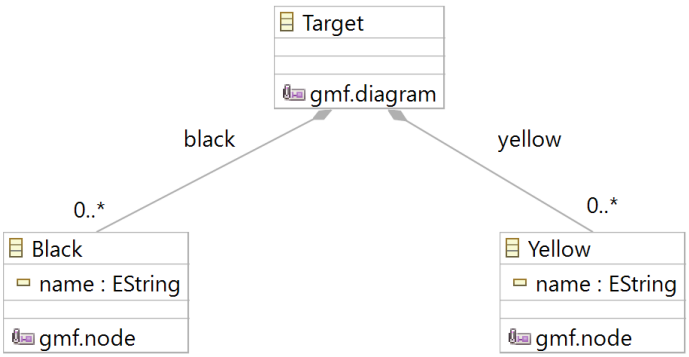
The source metamodel language:



The source emfatic file:

```
@namespace(uri="Origin", prefix="Origin")
package Origin;

@gmf.node(label = "name",color="255,0,0", figure="ellipse")
class Red {
  attr String name;
}

@gmf.diagram
class Origin {
  val Red[*] red;
  val Green[*] green;
}

@gmf.node(label = "name",color="0,255,0", figure="ellipse")
class Green {
  attr String name;
}
```

The target metamodel language:



The target emfatic file:

```
@namespace(uri="Target", prefix="Target")
package Target;

@gmf.diagram
class Target {
  val Black[*] black;
  val Yellow[*] yellow;
}
```

```
@gmf.node(label = "name",color="0,0,0", figure="ellipse")
class Black {
  attr String name;
}

@gmf.node(label = "name",color="255,255,0", figure="ellipse")
class Yellow {
  attr String name;
}
```

The model transformations rules (ETL file):

```
rule Red2Black
transform r: Origin!Red
to b: Target!Black
{
   b.name=r.name;
}

rule Green2Yellow
transform g: Origin!Green
to y: Target!Yellow {
    y.name=g.name;
}
```

The target XMI file can be transformed to be edited using the target diagram editor by following steps:

Bibliography

[1] Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, Richard Paige, **“The Epsilon Book”**, Eclipse Public Licence, 2018

[2] Marco Brambilla, Jordi Cabot, Manuel Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool, US, 2012

[2] **NuSMV**, a symbolic model checker: http://nusmv.fbk.eu/

[3] **Tina**, a Petri Nets simulator: http://projects.laas.fr/tina/