Engenharia de Domínio

Mestrado em Engenharia Informática

PL 4

Introduction to the Eclipse Modeling Framework

Exercise 2: DSL abstract syntaxes for "Mindmap" and "Requirements" models

Alexandre Bragança, Isabel Azeved, Nuno Bettencourt
atb@isep.ipp.pt, ifp@isep.ipp.pt, nmb@isep.ipp.pt

Dep. de Engenharia Informática – ISEP

2017/2018

Terminology

Some key terminology used in this course:

- A **Domain-Specific Language (DSL)** is a language designed to be useful for a specific set of tasks - the domain is captured by the designer of the language
- A **metamodel** captures the structure of a DSL and its usually referred to as its **abstract syntax**
- Abstract sintaxes are defined using **Ecore models** in EMF
- **XML Metadata Interchange (XMI)** is the format used to persist Ecore models

Simplified subset of the Ecore metamodel

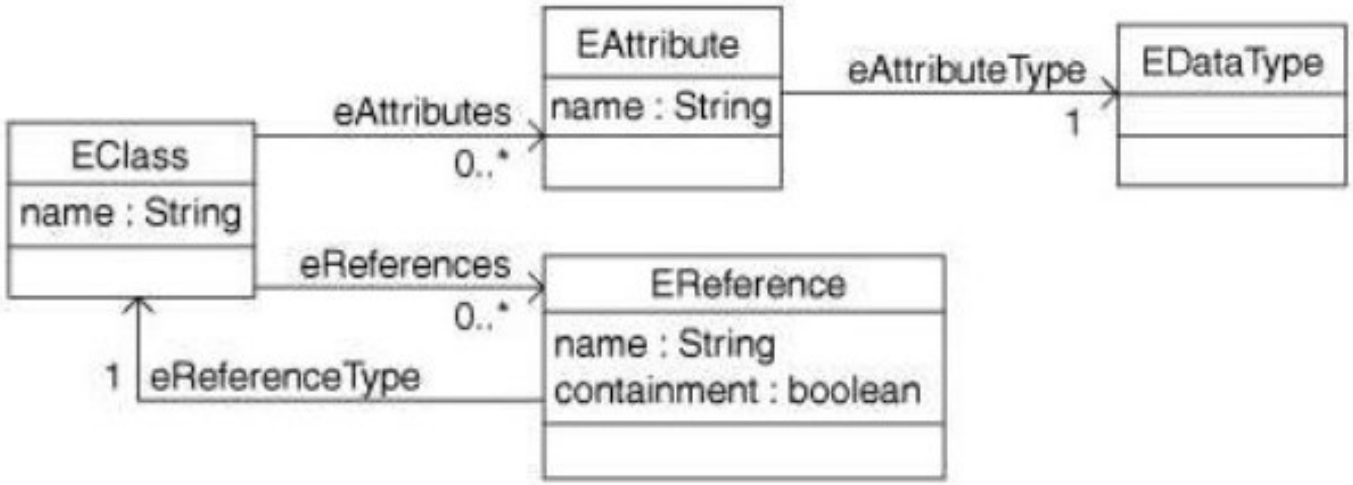


Figure 1: A (very) simplified Ecore metamodel (Source: book "EMF: Eclipse Modeling Framework", page 34)

Exercise 2 (1): DSL abstract syntax for Mindmap model - all solutions

Follow the instructions in the next slides. You will:

1. Create a DSL abstract syntax for mindmaps models;
2. Specify the metamodel for the DSL in Ecore;
3. Generate supporting code for creating and manipulating instances of mindmaps;
4. Create a mindmap instance for the college problem;
5. Add several constraints (i.e., validations) to the metamodel with the OCL language.

Very Important:

- Do not forget to create all projects inside the repository (inside the folders "edom/exercise2/solution1"and "edom/exercise2/solution2"). Also, commit all your work and do not forget to reference the bitbucket issue!
- Please locate and download the **.gitignore** file from moodle and place it in the root folder of all your eclipse projects. It will instruct git to ignore some files of eclipse projects that should not be included in the repository.

DSL abstract syntax for Mindmap model

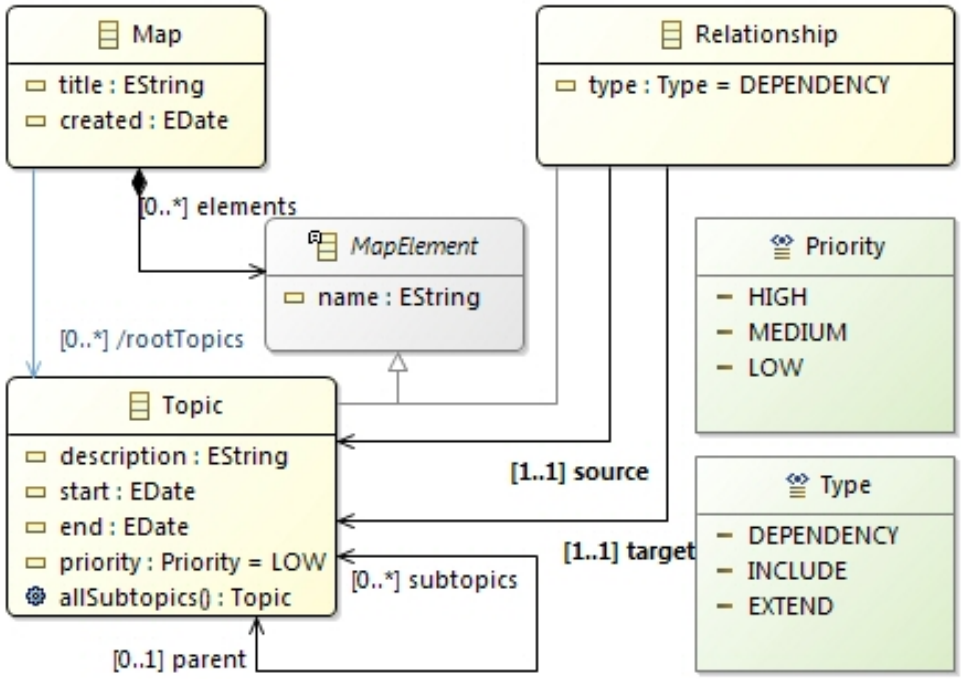


Figure 2: Mindmap model (adapted from Chapter 3 of book "Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit")

To create a mindmap model, consider that a Map contains an arbitrary number of MapElements (Topic or Relationship). A Topic may be related to many subtopics but one subtopic has at most one Parent. A Relationship has exactly one Topic as source and one Topic as target. Finally, a Topic has an Enumeration that determines its Priority and a Relationship has an Enumeration that determines its Type (default values specified).

Mindmap model - create an empty project

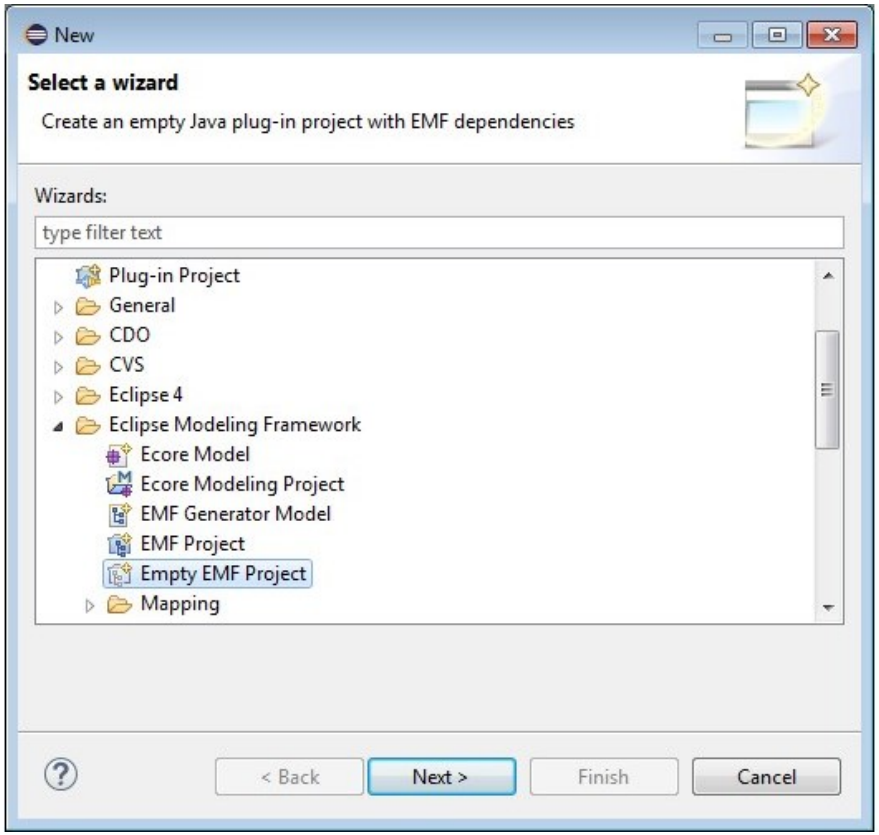


Figure 3: Creating a new project

Create a project:

- In the IDE select "File"from the toolbar menu –> "New--> "Other. . . "and choose "Empty EMF Project"
- Click "Next", enter a name for the project, e.g., "org.eclipse.dsl.mindmap", and hit "Finish"

Mindmap model - create an Ecore model

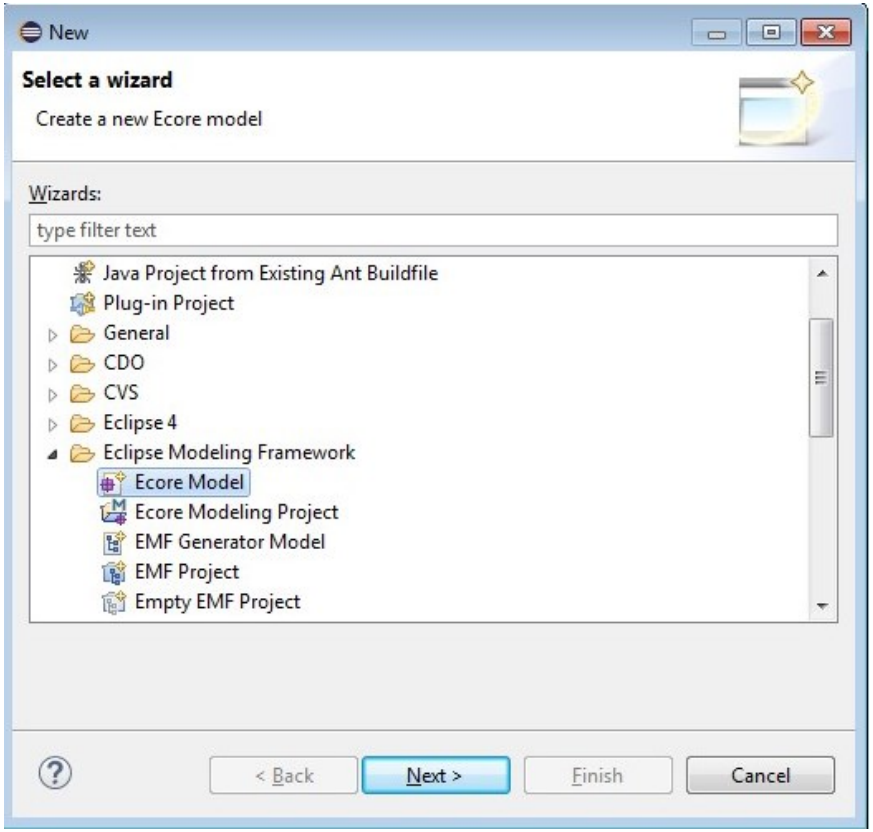


Figure 4: Creating a new ECORE model

The model itself needs to be defined:

- Right click the model folder in the new modeling project –> "New--> "Other. . .--> "Ecore Model--> "Next"and give the ecore file the name "mindmap.ecore"
- Click "Finish"to create the model. The default Ecore editor will be opened. There are additional options for defining Ecore models, including graphical modeling, textual modeling, Java annotations and importing from UML tools, but for now the default editor and its tree-based view will be used

Properties view

Give the package of the new model a name and a URI. This will be done in the properties view. Properties of model elements can be modified in a second view, which opens up on double-click or right-click –> "Show Properties View". The URI is used to identify the model later on. Name the package "mindmap", set the Ns Prefix to "org.eclipse.dsl.mindmap"and the Ns URI to "http://org/eclipse/dsl/mindmap".

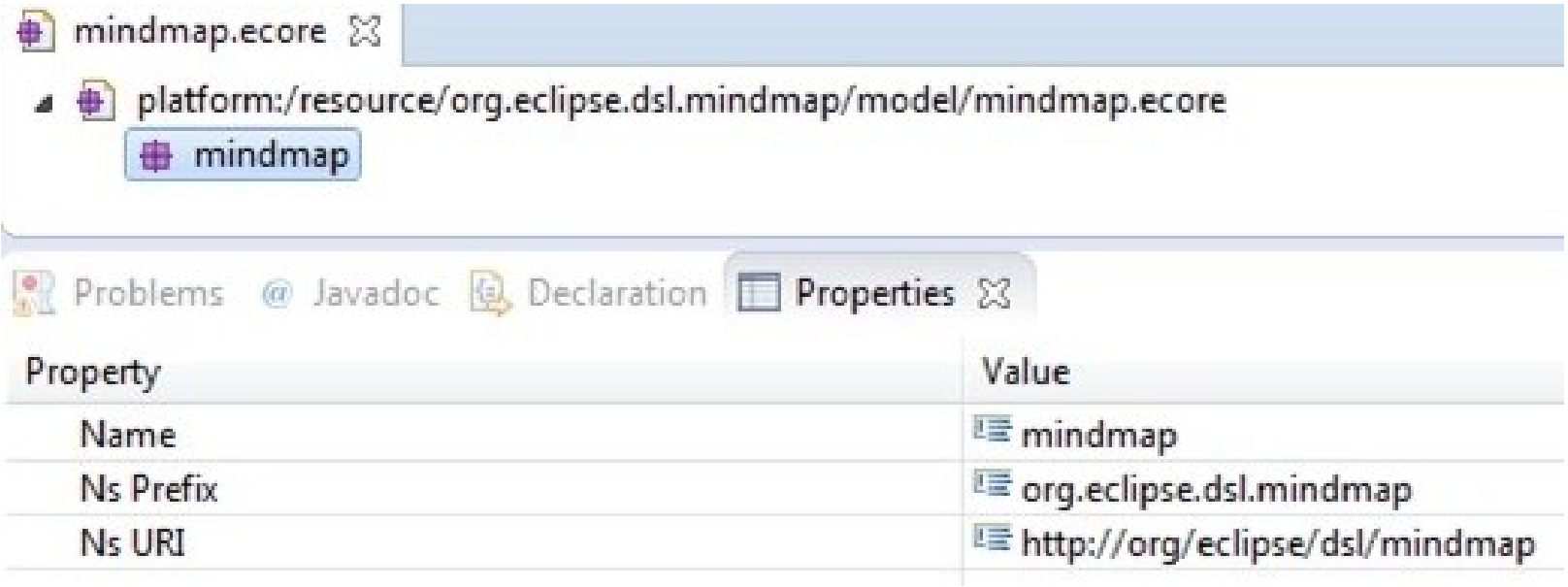


Figure 5: Changing the package properties

Model elements

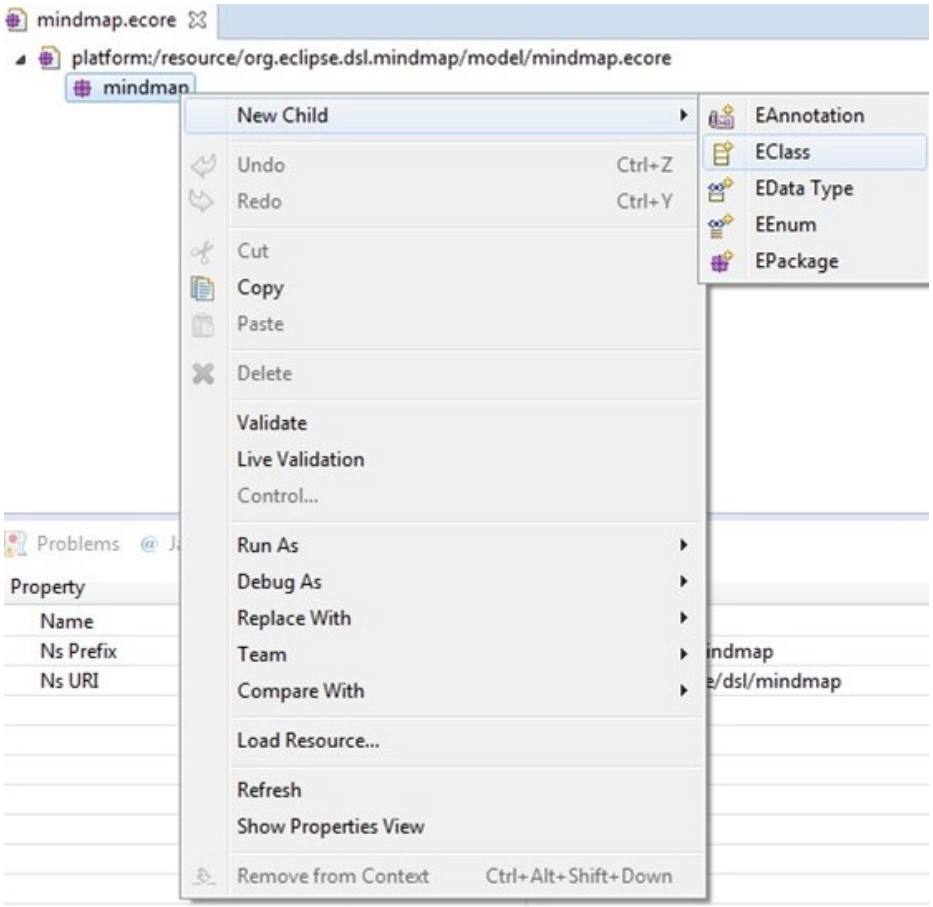


Figure 6: Adding children of the root package

Model elements have to be defined:

- Create an EClass by right clicking on the package name –> "New Child--> "EClass" and set its name in the properties view of the newly created EClass
- From the context menu of an EClass, EAttributes and EReferences can be added as children. Properties can be set using the Property view. The MapElement class is to be defined as abstract (use the Property view)
- Convention: Class names should start with an uppercase letter and attribute and reference names start with a lowercase letter

Containment properties

Create an EReference between Map and MapElements by right clicking on the Map model element. Name the reference "elements" and set its EType to "MapElement". Since a Map can contain an arbitrary number of MapElements, set the upper bound to -1 ("many"). Finally, set the property Containment to "true", defining the EReference to be a containment reference.

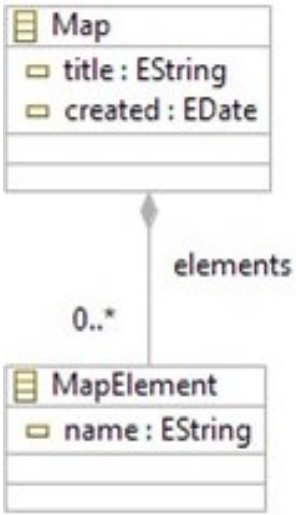


Figure 7: A Map contains 0 to many MapElements

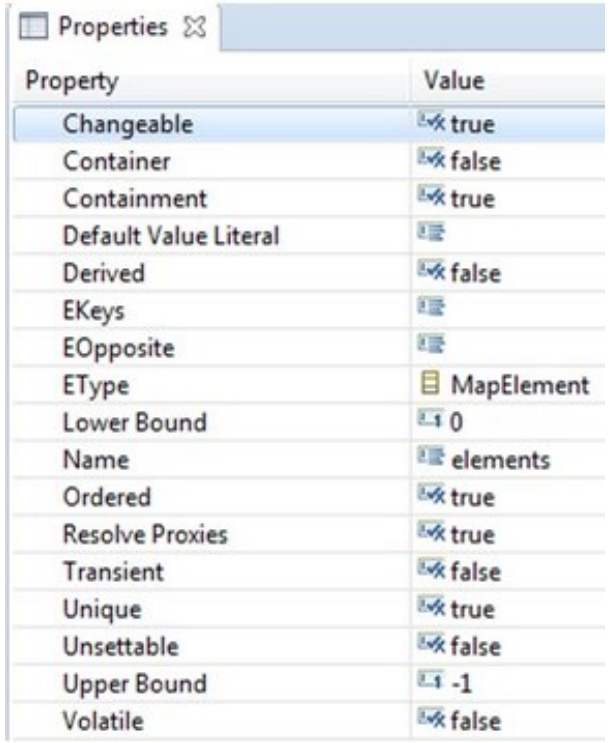


Figure 8: Properties view of EReference "elements"

Derived references

Create an EReference between Map and Topic by right clicking on the Map model element. Name the reference "rootTopics". This relationship is derived, transient, and volatile. Later it will be implemented using OCL.

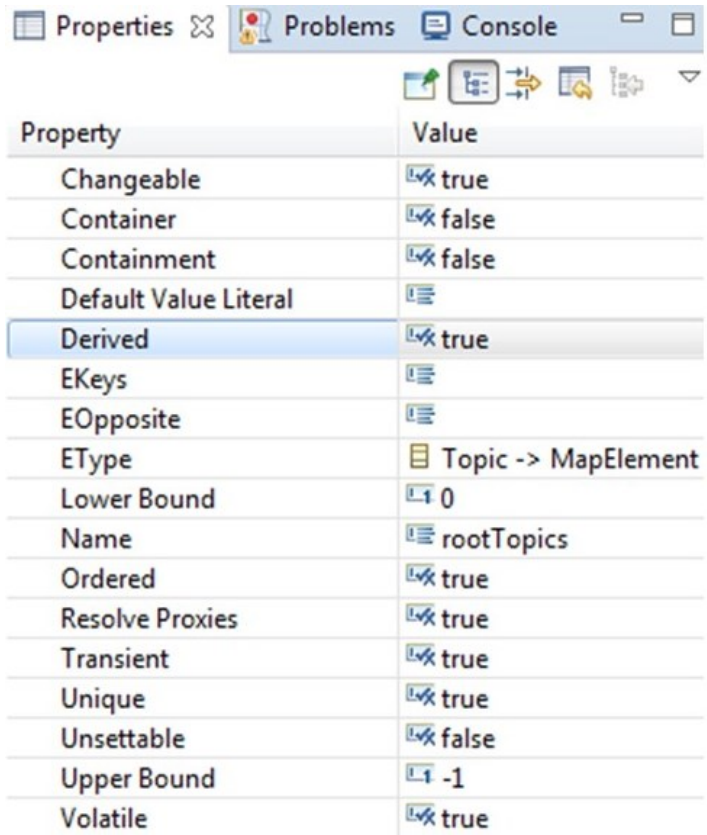


Figure 9: Defining the EReference "rootTopics"

Operations

Add the EOperation "allSubTopics" to the EClass "Topic". Change some properties: ordered="false", upperBound=-1" and eType to "Topic". Latter it will be implemented using OCL.

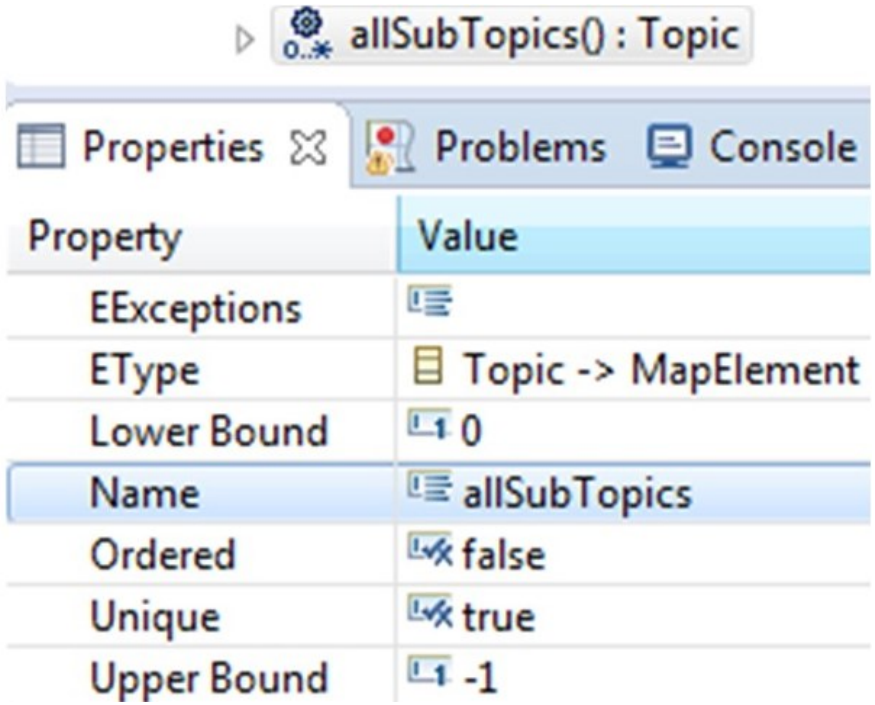


Figure 10: Defining the EOperation "allSubTopics"

Enumerations

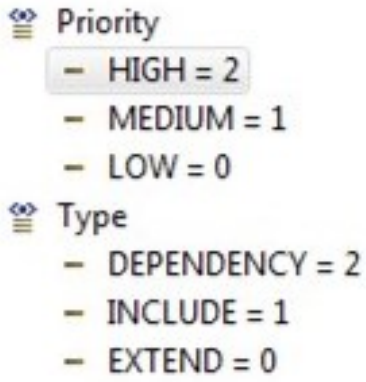


Figure 11: Enumeration children of the root package

To define enumerations:

- Create an EEnum by right-clicking on the root mindmap model package, in a similar way to how it was made to create a class. EEnum Priority, for instance, needs three EEnum Literals (HIGH, MEDIUM, LOW)
- Add an EAttribute to the related EClass, name it and set the EType to the right EEnum. Set a Default Literal Value

Validating the model

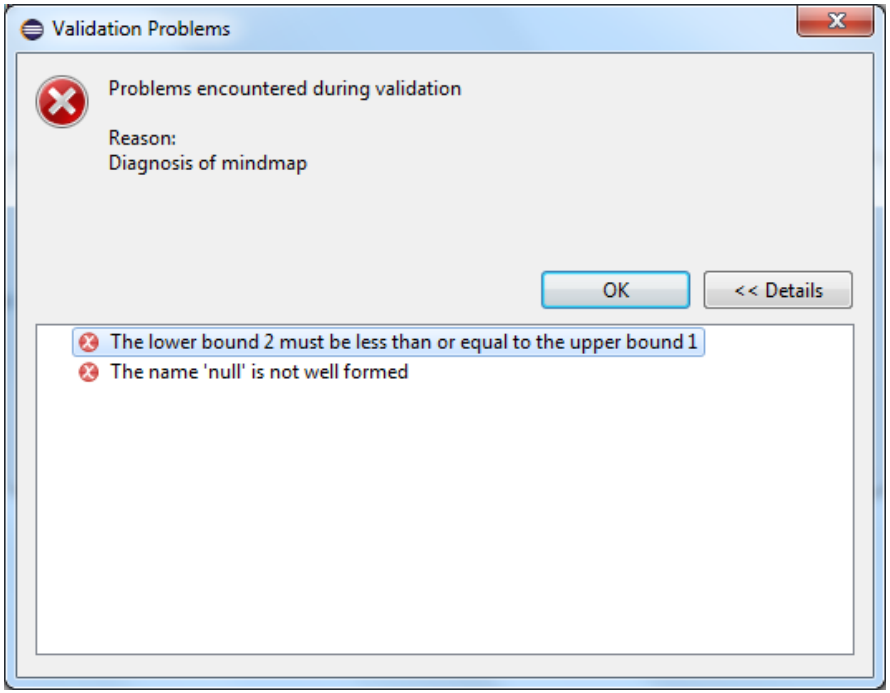


Figure 12: List of problems found during validation

Validate the model with a right-click on the model root in the Ecore editor. For instance, if the upper bound of the EAttribute "source" (in Relationship) is lower than the lower bound, this problem will be detected by the model validation.

EMF files

The domain model is stored in EMF **.ecore** and **.genmodel** files. What is only relevant for code generation resides in a .genmodel file, which has not yet been created. Both of them are serialised as XMI files.

Listing 1: an extract from EMF .ecore

```
<?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="mindmap"
         nsURI="http://org/eclipse/dsl/mindmap"
         nsPrefix="org.eclipse.dsl.mindmap">
  <eClassifiers xsi:type="ecore:EClass" name="Map">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
         eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="created"
         eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EDate"/>
...
```

Creating the Mindmap Generator Model

Figure 13: Defining an EMF Generator Model

To generate entities from the Ecore file, a generator model has to be created (a .genmodel file):

- Right-click the mindmap.ecore file in Explorer view and select "New"-> "Other. . . "-> "EMF Generator Model"-> "Next" and enter mindmap.genmodel as the file name
- Proceed to the next page and select "Ecore model" as the model importer. After clicking "Next", select "Browse Workspace. . . " and select the previously created mindmap.ecore. Go to the next wizard page and select "Finish"

At this point only a generator model was created, and no code at all.

Plugins that can be generated

EMF allows the generation of up to four different plugins based on the generator model:

- **Model**: This plugin contains all entities and packages of the model as well as factories to create instances of the model
- **Edit**: The edit plugin is useful to display a model in a UI. For example, the providers offer a label for every model element, which can be used to display an entity showing an icon and a name
- **Editor**: The editor plugin is a generated example editor to create and modify instances of a model
- **Test**: It contains test skeletons

Properties for the code generation

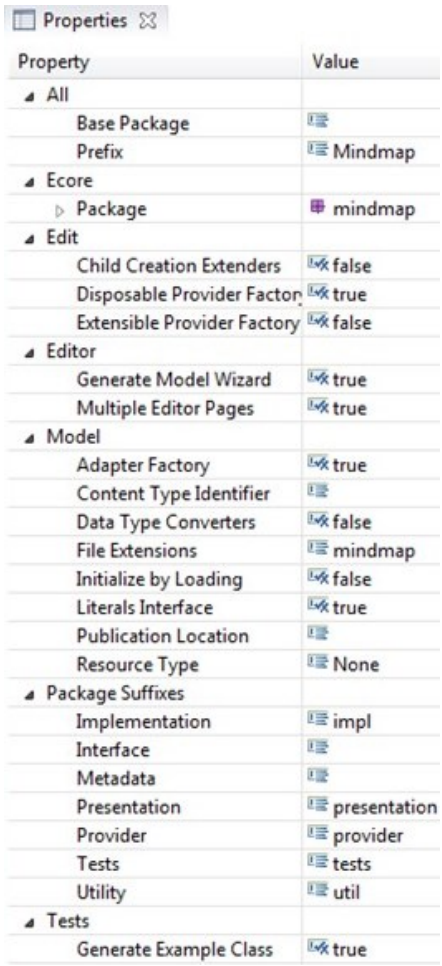
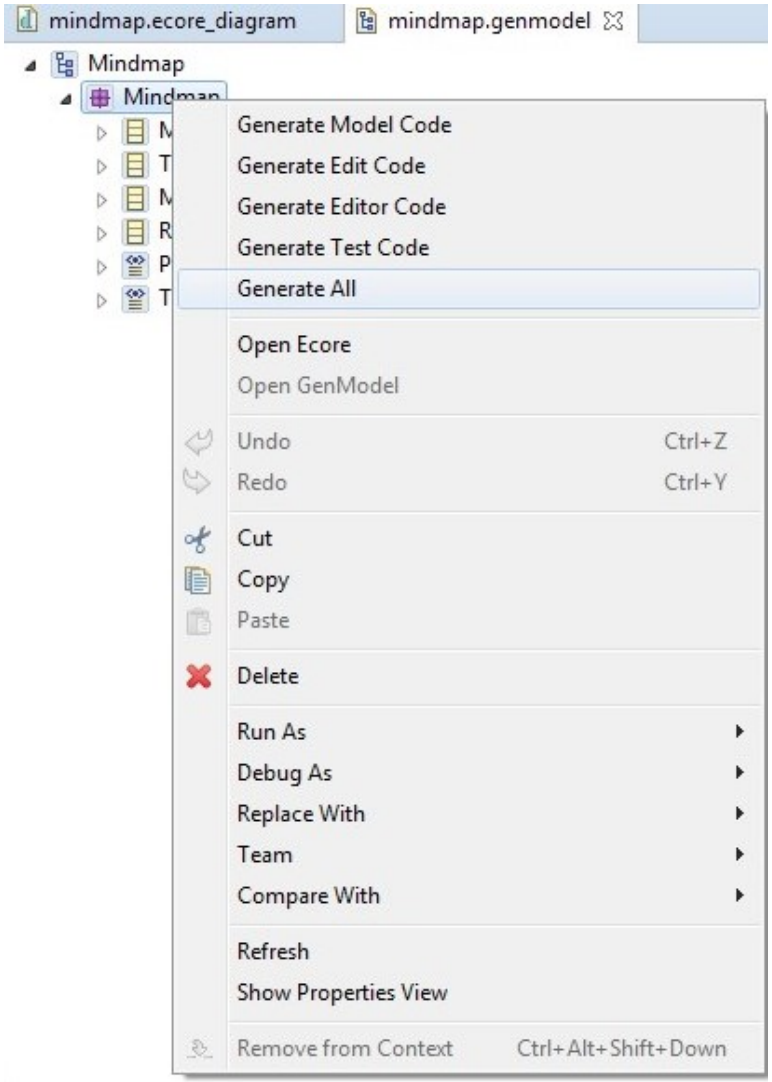


Figure 14: Other properties considered during code generation

Properties for the code generation that are not part of the model itself can be configured through the Properties View. It is not necessary to change anything at this moment.

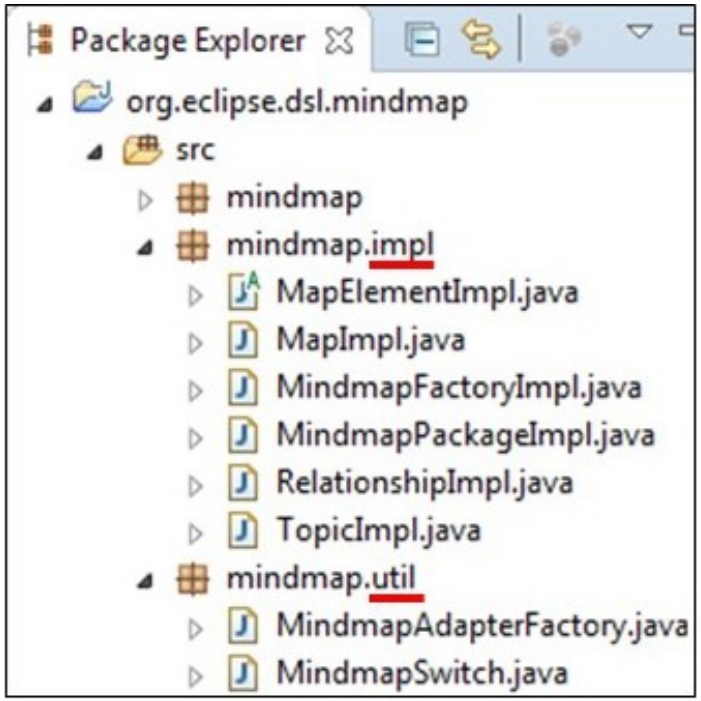
Generating code



To generate the plugins, right-click on the root node of the generator model and select the desired ones. However, for now, select "generate all".

Figure 15: Defining an EMF Generator Model

Property effects in the code



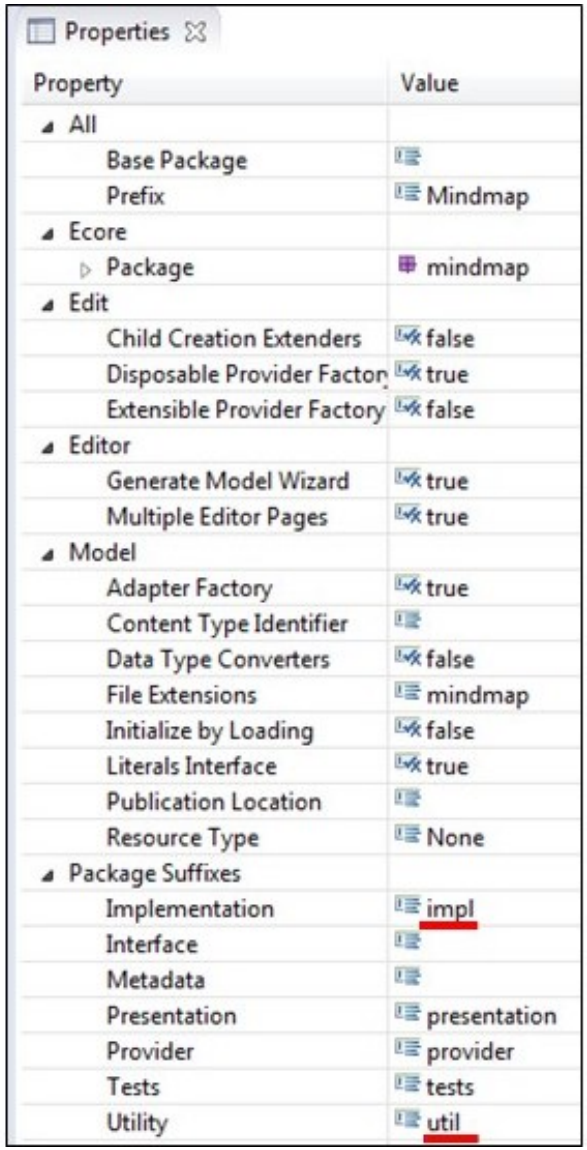


Figure 16: Defining an EMF Generator Model

Running the plug-ins

To run the plug-ins and test the functionality of the editor, a new configuration is to be created. Toolbar menu -> "Run"-> "Run Configurations" and create a new Eclipse Application run configuration. Run the configuration.
An alternative is to right click the mindmap.ecore file, select "Run As"-> "Run Configurations" and follow the same process.

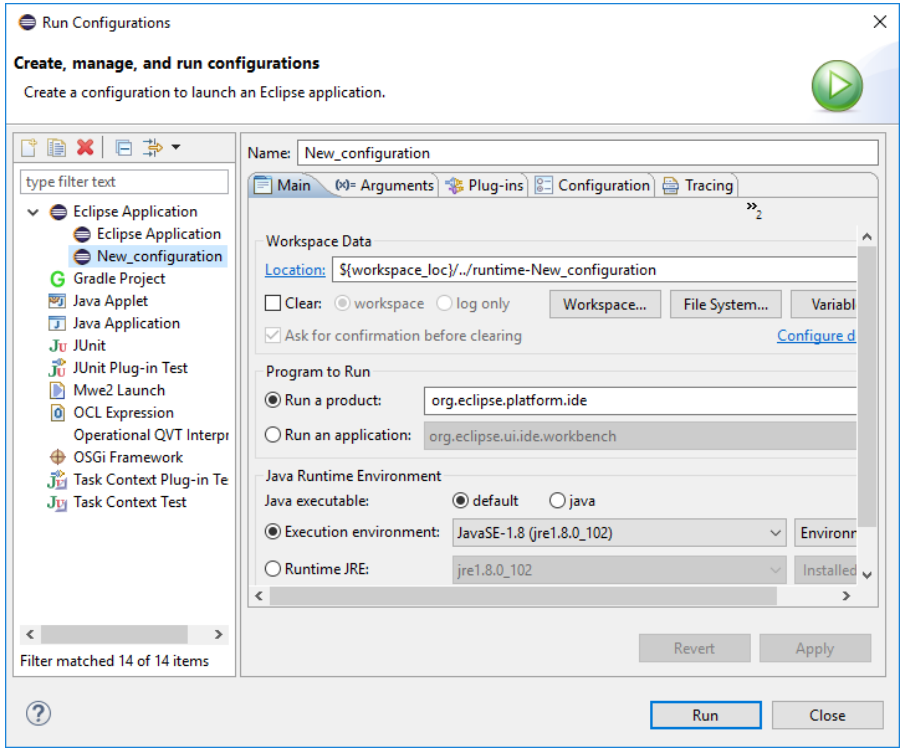


Figure 17: Creating a configuration (partial view)

Creating a mindmap instance

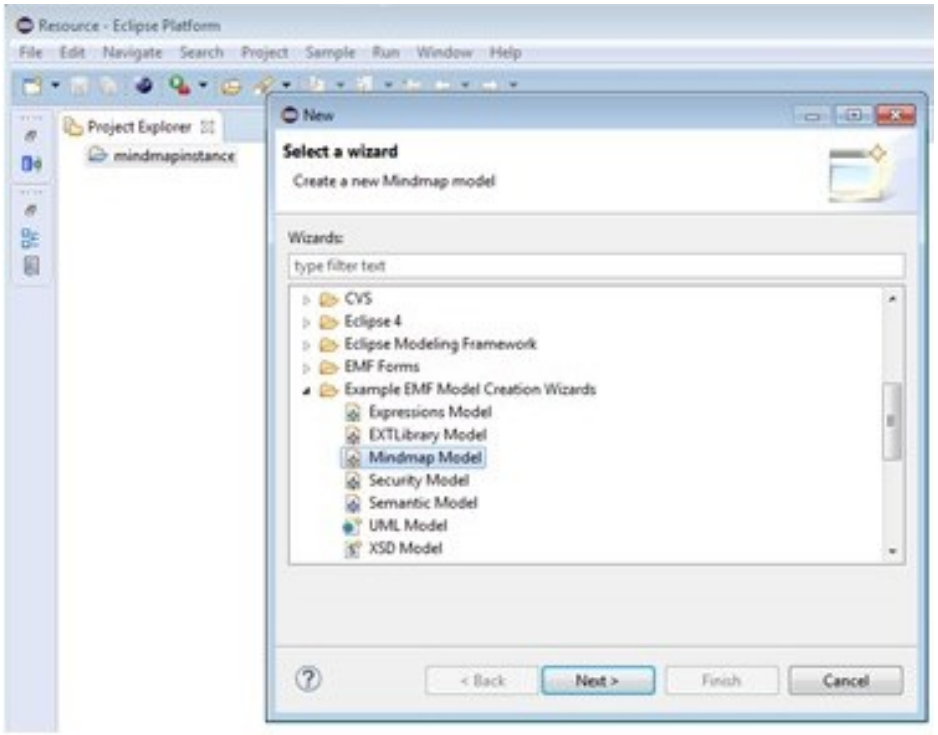


Figure 18: Creating a instance (partial view)

To create a mindmap instance:

- In the new runtime Eclipse create a new empty project (Toolbar menu -> "File"-> "New"-> "Other. . . "-> "General"-> "Project") named mindmapinstance
- Right click the created project -> "New"-> "Other. . . "-> "Example EMF Model Creation Wizards"-> "Mindmap Model"-> "Next" and enter "map.mindmap"as the name. This file will contain a serialized version of the model instance. Select Map as the model object to set the root object of the model instance

Using the editor for model instances

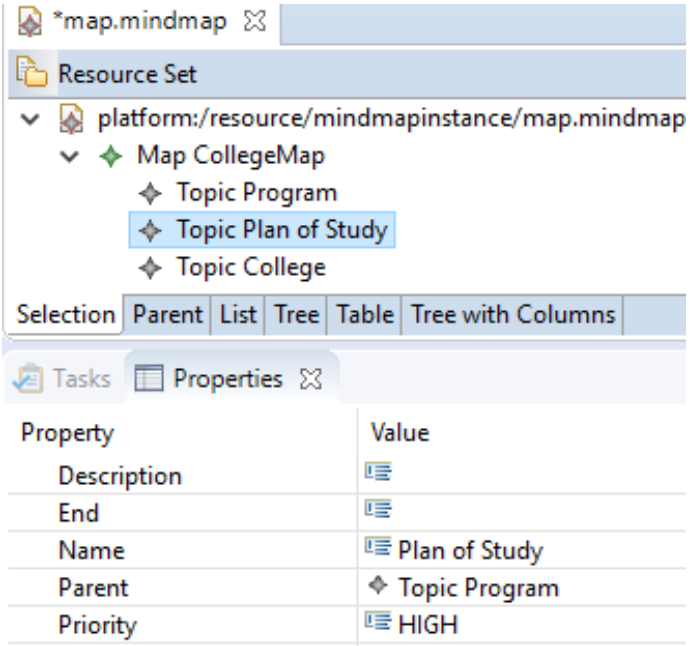
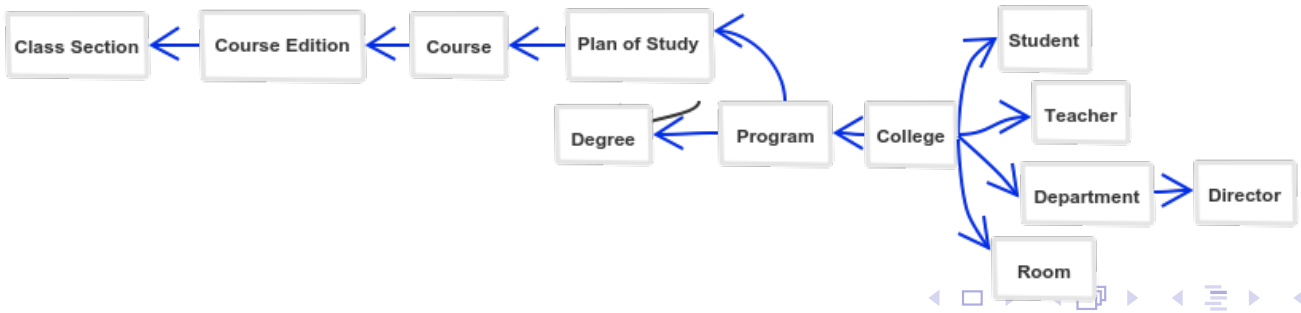


Figure 19: Adding elements to the mindmap instance (partial view)

The generated editor for model instances works similarly to the Ecore editor. Model element instances can be created via a right-click and EAttributes can be modified in the properties view. After saved, the created instance is serialized in the XMI file "map.mindmap".
Create an instance with a `CollegeMap`, the twelve topics and the corresponding parent-child relationships, as shown in the image. Consider a DEPENDENCY relationship between Class Section (target) and Director (source) - not shown in the image.



Other editors

Models can be built with different tools. Another option is the EcoreTools (http://www.eclipse.org/ecoretools/).

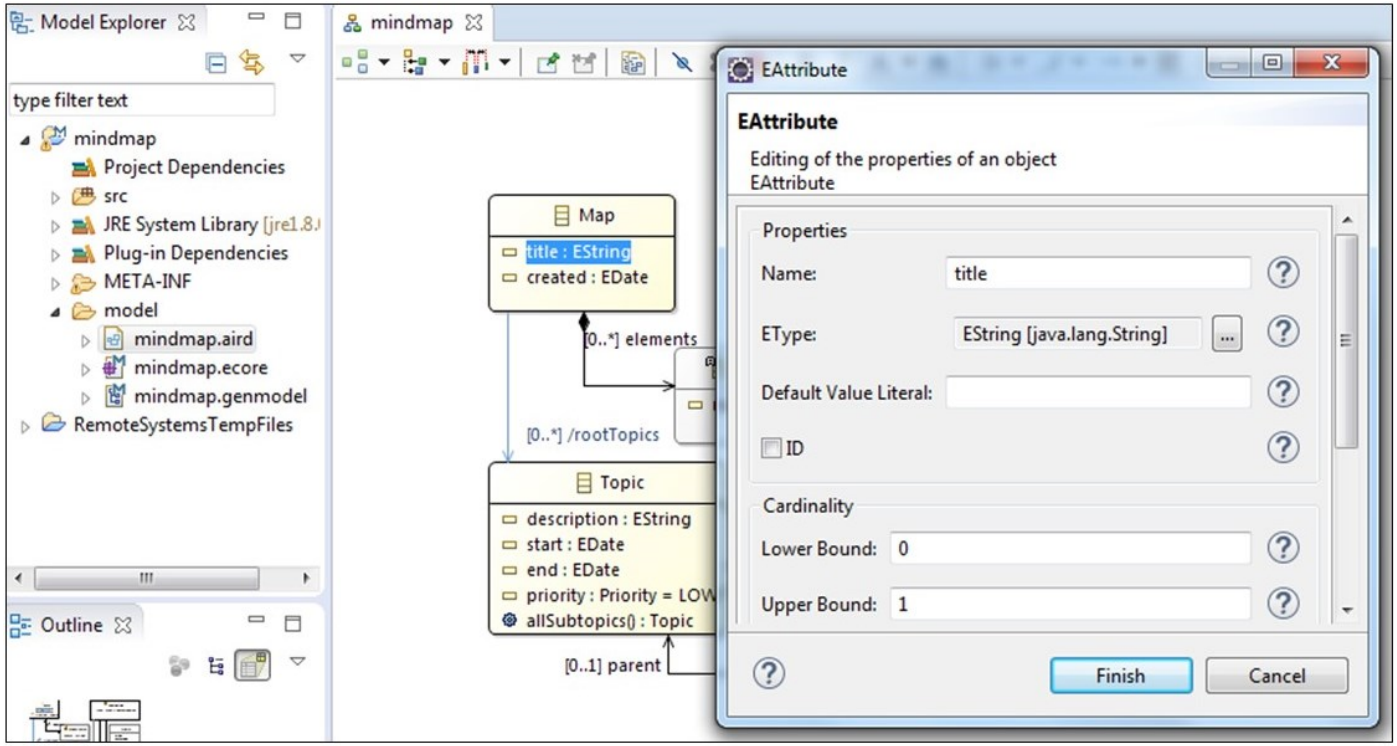


Figure 21: Using EcoreTools

The result is just a model. Other editors can be used as well.

EcoreTools installation

If not already installed, install EcoreTools.

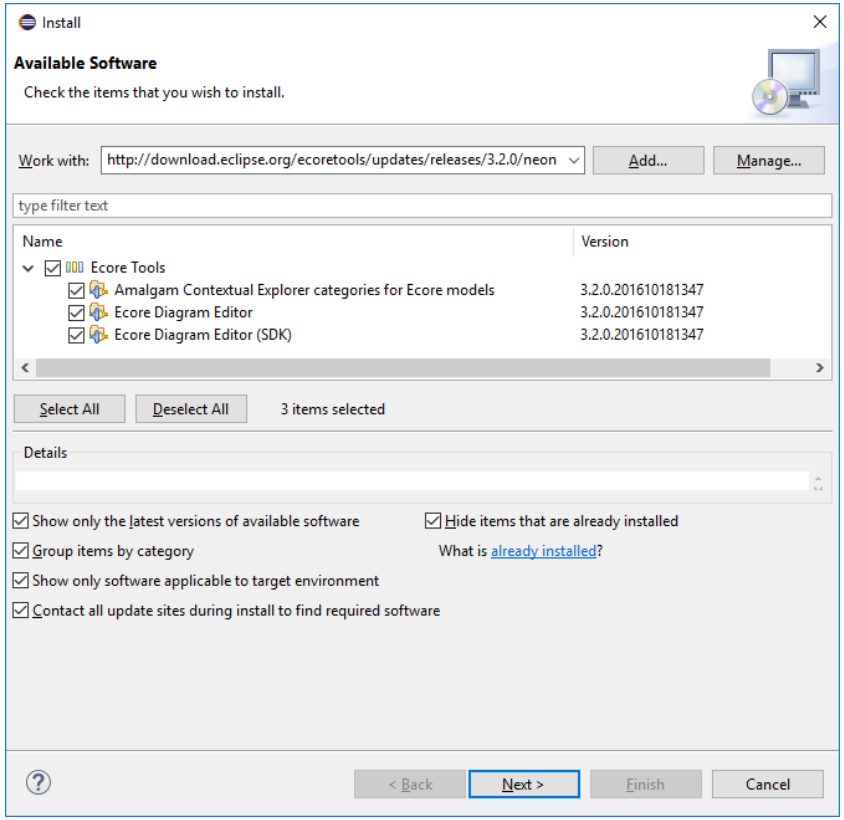


Figure 22: Installing EcoreTools

Object Constraint Language

The structure and semantics of our DSL can be refined using Object Constraint Language (OCL). OMG's OCL 2.4 is a specification with many implementations, including Eclipse OCL for use with Ecore and UML metamodels. The Eclipse OCL project provides four OCL languages:

- Essential OCL provides the core extensible capability of specifying expressions for models
- Complete OCL provides the ability to use OCL in a self-standing document to complement an existing meta-model with invariants, and additional features
- OCLinEcore enables OCL to be embedded within an Ecore meta-model to add invariants for classifiers, bodies for operations and computed values for properties
- OCLstdlib supports the definition of the standard and custom OCL Standard Libraries

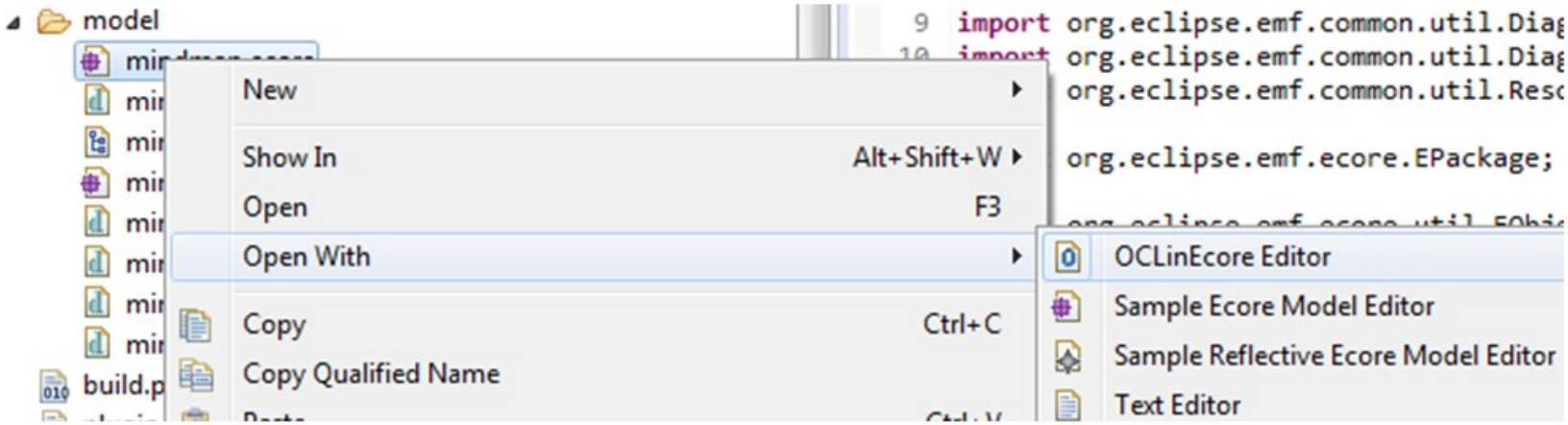
Object Constraint Language

Diferent kinds of expressions can be defined with OCL:

- Invariants to define conditions that must be satisfied in each possible instantiation of the model
- Initialization of class properties
- Derivation rules that define the value of derived model elements at all times
- Query operations
- Preconditiion and postconditions, i.e., constraints that must be satisfied before/after operation execution starts/finishes.

Other OCL editors

The Sample Ecore Editor has its limitations. For instance, it does not detect syntactic and semantic errors in OCL. The OCLinEcore Editor can be used instead. With a Ecore file selected in the Model Explorer, select "Open With"-> "OCLinEcore Editor" from the context dependent right button menu.



Using OCL to provide validations: mandatory title

Define title as mandatory:

Listing 2: Validating the title

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore' ;
package mindmap : _'org.eclipse.dsl.mindmap' = 'http://org/eclipse/dsl/mindmap'
{
   class _'Map'
   {
      attribute title : String[?];
      attribute created : ecore::EDate[?];
      ...
      invariant mustHaveTitle: not title.oclIsUndefined();
   }
...
```

`oclIsUndefined()` is an operation that results in true if its argument is null or invalid and false otherwise.

Using OCL to provide validations for date properties

Indicate that the end date needs to be greater or equal to the start date:

Listing 3: Validating dates

```
class Topic extends MapElement
   {
      ...
      attribute description : String[?];
      attribute start : ecore::EDate[?];
      attribute end : ecore::EDate[?];
      attribute priority : Priority[?];
      ...
      invariant EndAfterStart: self.end >= self.start;
   }
```

In an OCL expression, the reserved word `self` is used to refer to the contextual instance. For example, if the context is Topic, then self refers to an instance of Topic.

Using OCL to define an operation

Add the following OCL statement using the editor OCLinECORE:

Listing 4: Defining allSubTopics

```
class Topic extends MapElement
   {
      operation allSubTopics() : Topic[*]
      {
         body: self->closure(subtopics);
      }
      attribute description : String[?] { id };
      ...
```

For a simple parent-children relationship and known parents **parents->closure(children)** computes the set of parents, parents.children, parents.children.children, etc.

Using OCL to define a derived reference

Add the following OCL statement about the EReference rootTopics using the editor OCLinECORE:

Listing 5: Defining the derived EReference rootTopics

```
class _'Map'
   {
      ...
      property rootTopics : Topic[*] { ordered derived transient volatile }
      {
         initial: let topics : Set(mindmap::Topic) =
              self.elements->select(oclIsKindOf(mindmap::Topic))->
             collect(oclAsType(mindmap::Topic))->asSet() in
                 topics->asOrderedSet()->
               symmetricDifference(topics.subtopics->asSet())->asOrderedSet();
      }
      ...
   }
```

The result of the operation **symmetricDifference(s : Set(T)) : Set(T)** is a set containing all the elements that are in self or s, but not in both. The operation **asSet() : Set(T)** provides a set containing all the elements from self, with duplicates removed. asOrderedSet() is similar to asSet but the resulting Set is ordered.

Using a simple text editor to enter OCL statements or others

The same OCL statement about the EReference rootTopics but using a text editor:

Listing 6: Defining the derived EReference rootTopics

```
<eClassifiers xsi:type="ecore:EClass" name="Map">
...
    <eStructuralFeatures xsi:type="ecore:EReference" name="rootTopics"
        upperBound="-1"
      eType="#//Topic" volatile="true" transient="true" derived="true">
      <eAnnotations source="http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot">
        <details key="derivation" value="let topics : Set(mindmap::Topic) =
            self.elements->
           select(oclIsKindOf(mindmap::Topic))->
               collect(oclAsType(mindmap::Topic))->asSet() in
             topics->asOrderedSet()->symmetricDifference(topics.subtopics->
              asSet())->asOrderedSet()"/>
      </eAnnotations>
    </eStructuralFeatures>
...
```

Checking the validations

Working with a mindmap instance, it is possible to validate it. For instance, if a title is not provided to a map, the validation is supposed to fail.

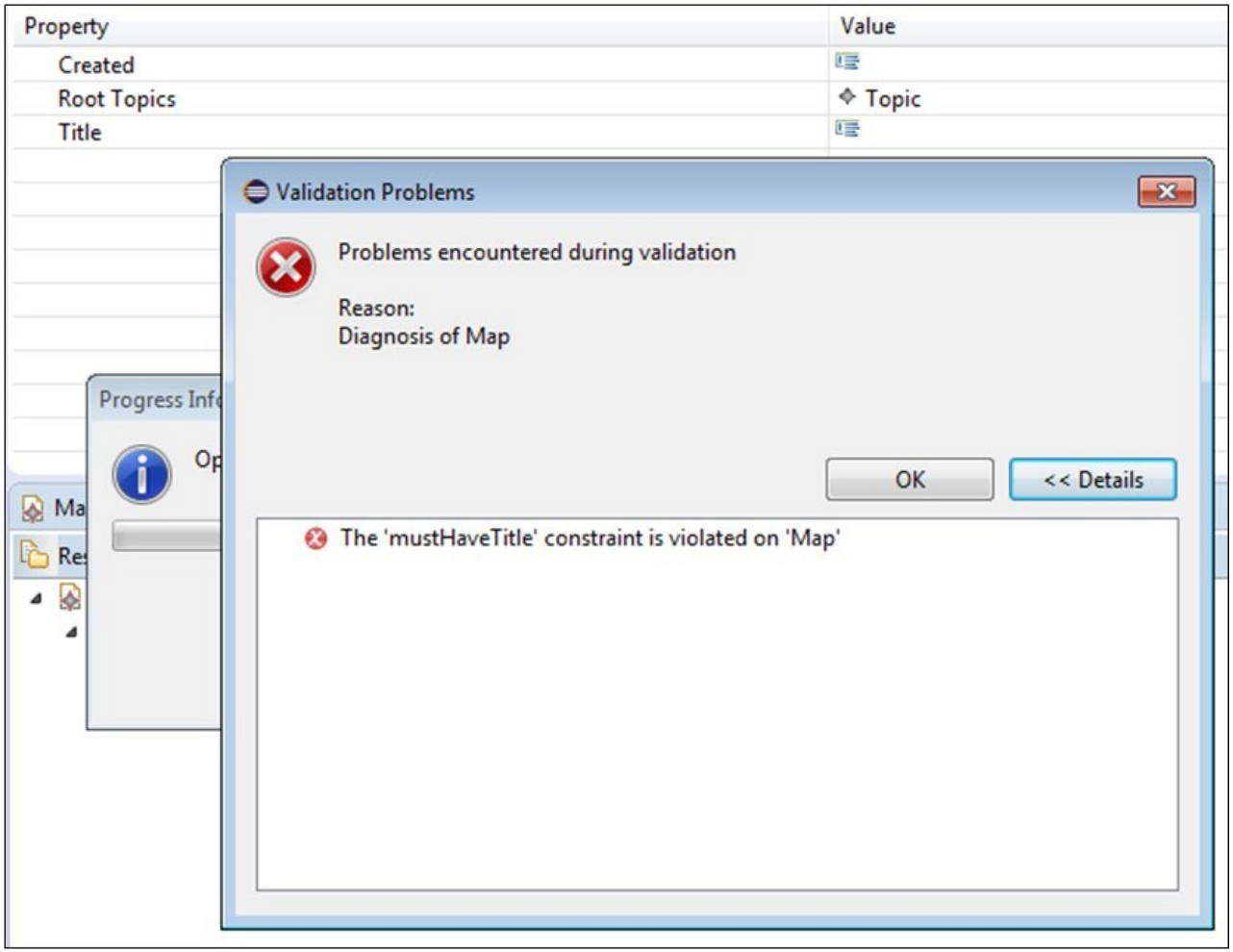


Figure 24: Validating an instance

Xtext OCL Console

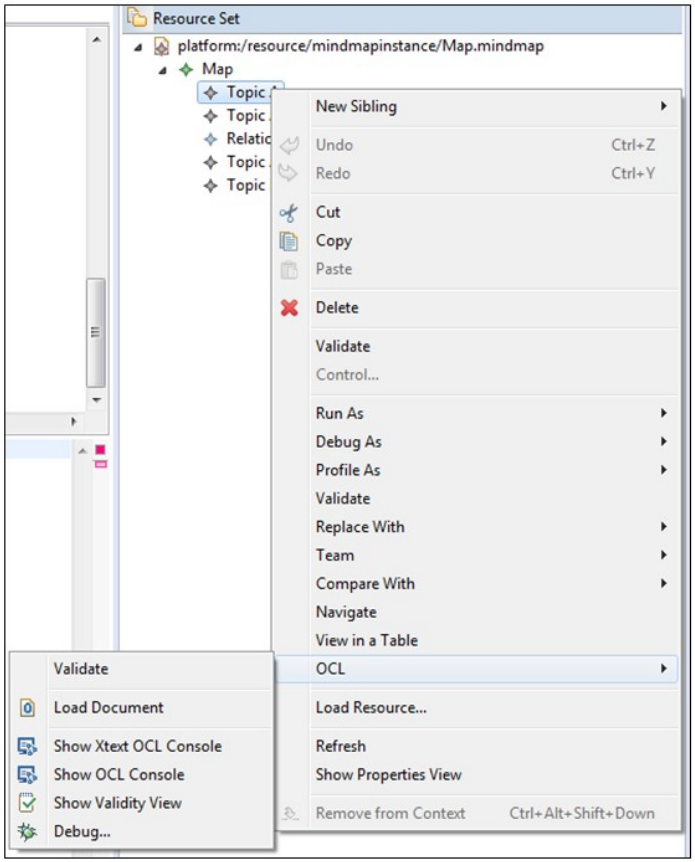


Figure 25: Opening the Xtext OCL Console

Working with a model instance, for instance, a mindmap instance, it is possible to evaluate constraints and queries. An useful tool is the Xtext OCL Console. Invoke OCL->Show Xtext OCL Console from the context menu.

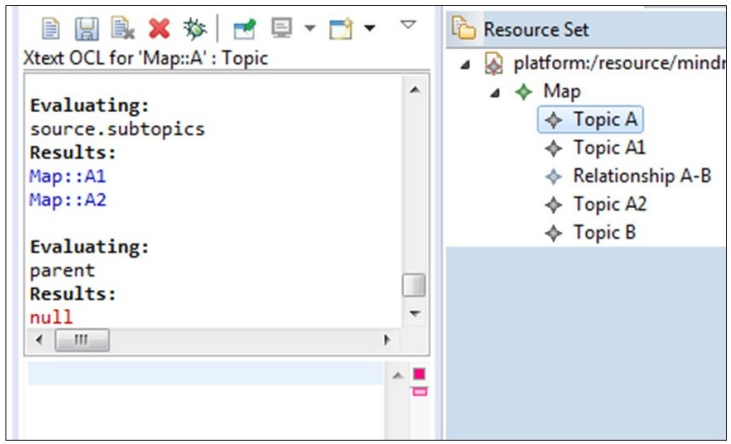


Figure 26: The Xtext OCL Console

Evaluating OCL expressions

In the bottom line of the console enter OCL expressions and then results are shown in the panel. Some expressions to try:

- `parent.description` when Program is selected
- `subtopics.name` when Program is selected
- `rootTopics->size()` when CollegeMap is selected
- `self.target.parent.name` when the DEPENDENCY relationship is selected

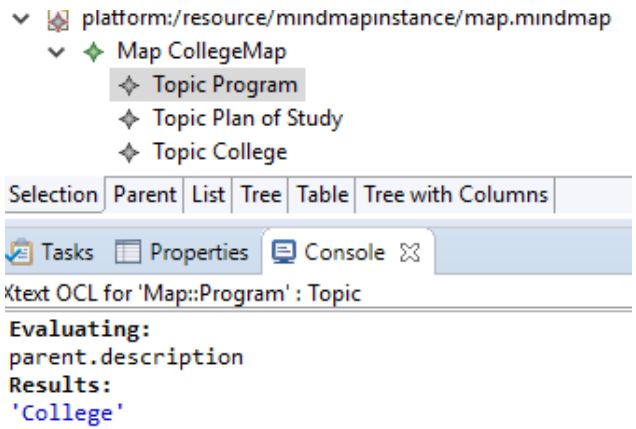


Figure 27: A Mindmap instance

Exercise 2 (2): DSL abstract syntax for Requirements model - all solutions

You need to:

1. Create a DSL abstract syntax for requirements models;
2. Specify the metamodel for the DSL in Ecore;
3. Generate supporting code for creating and manipulating instances of requirements;
4. Alter the code as it will be indicated;
5. Create a instance for the college problem;
6. Constraint title attributes to be mandatory without using the OCL language.

Very Important:

- Do not forget to create all projects inside the repository (inside the folders "edom/exercise2/solution1"and "edom/exercise2/solution2"). Also, commit all your work and do not forget to reference the bitbucket issue!
- Please locate and download the **.gitignore** file from moodle at place it in the root folder of all your eclipse projects. It will instruct git to ignore some files of eclipse projects that should not be included in the repository.

Exercise 2: DSL abstract syntax for Requirements model

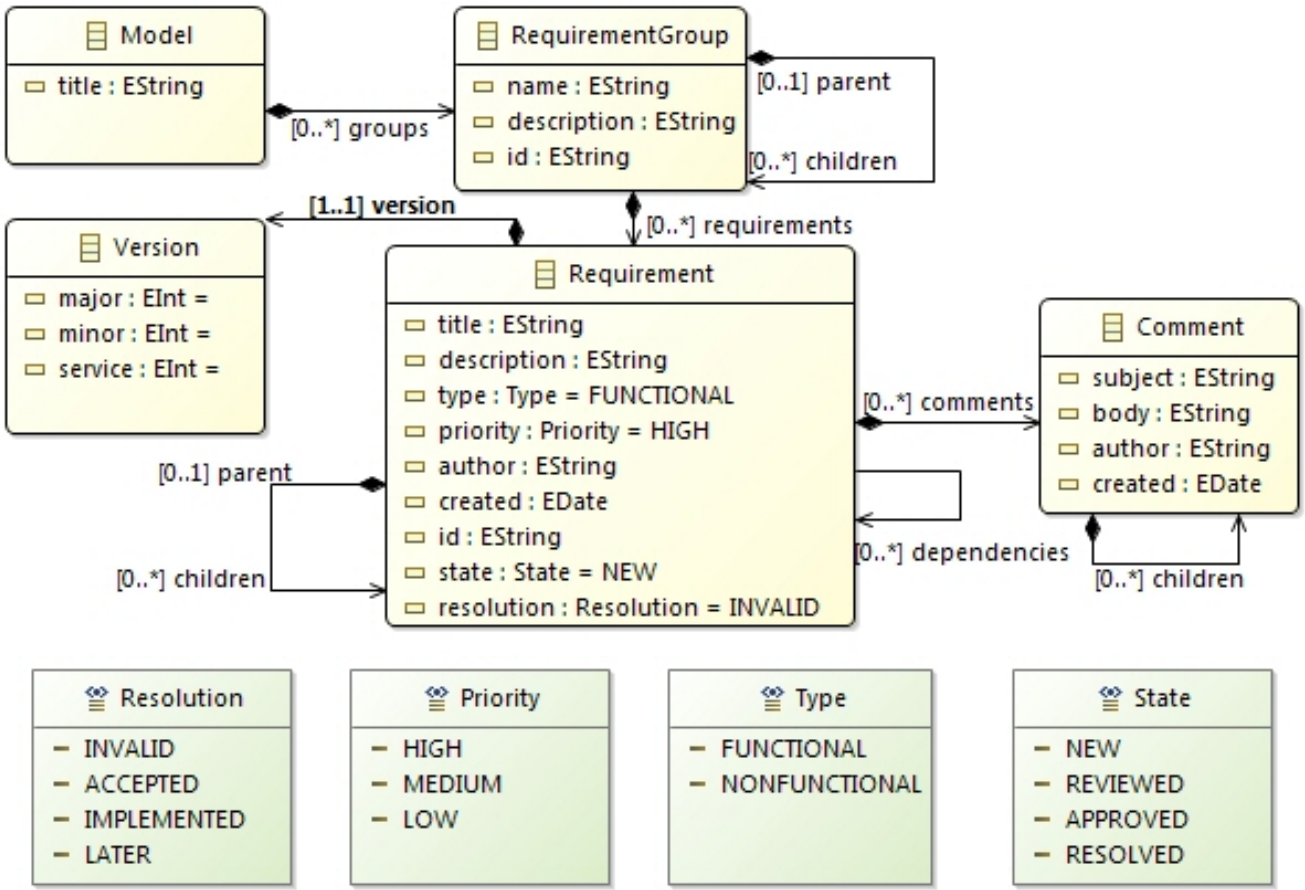


Figure 28: Requirements model (adapted from chapter 3 of book "Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit")

Create a requirements model and generate the Java code as before.

Requirements model: updating a class

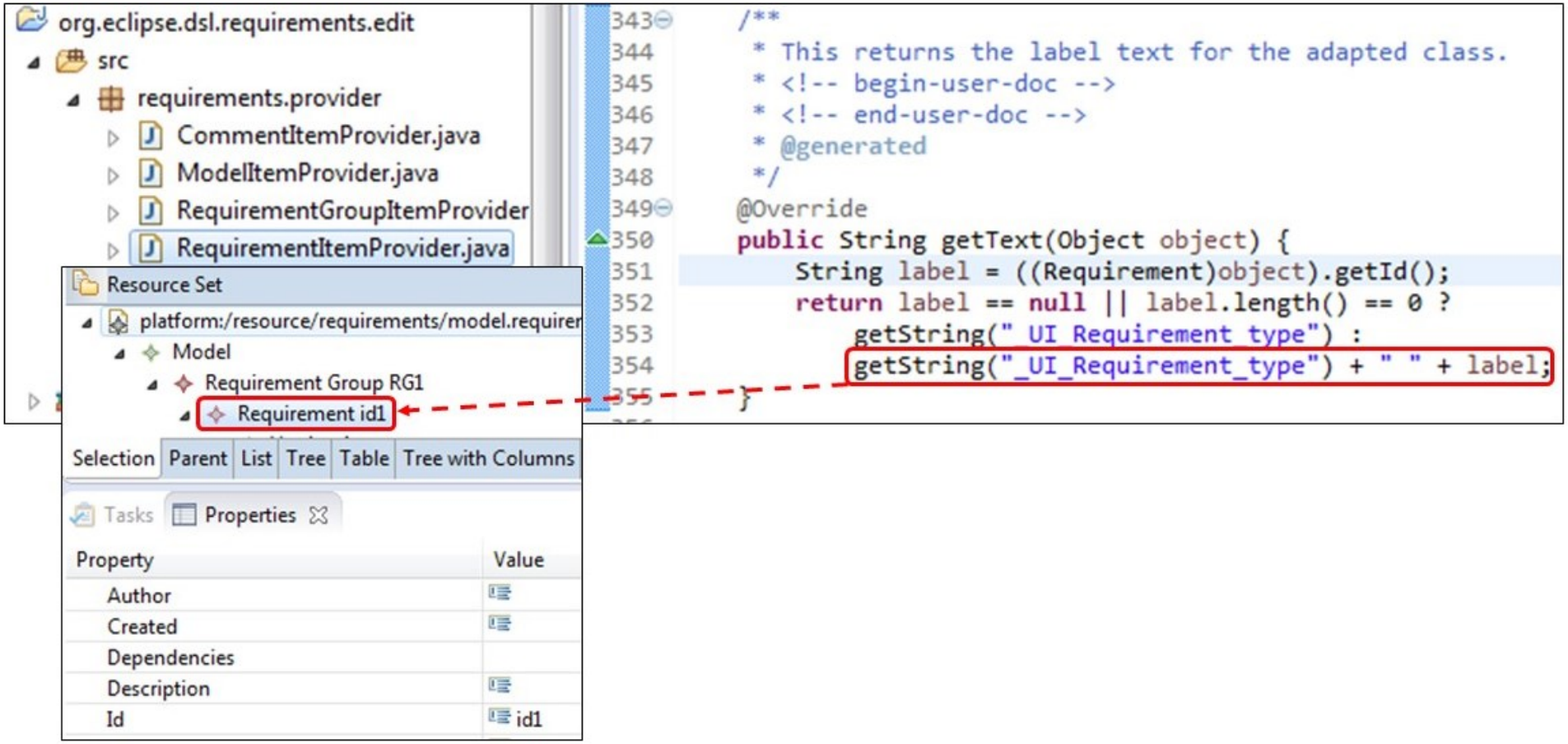


Figure 29: Method getText

When working with instances, only the id is shown for each requirement. The Java method getText is to be changed to include other attributes.

Updating method getText

Edit the method getText:

Listing 7: method getText

```
public String getText(Object object) {
   StringBuilder sb = new StringBuilder();
   sb.append(((Requirement)object).getId());
   sb.append(" (");
   requirements.Version version = ((Requirement)object).getVersion();
   if (version != null) {
      sb.append(((Requirement)object).getVersion().getMajor());
      sb.append(".");
      sb.append(((Requirement)object).getVersion().getMinor());
      sb.append(".");
      sb.append(((Requirement)object).getVersion().getService());
   } else {
      sb.append("0.0.0");
   }
   sb.append(") : ");
   sb.append(((Requirement)object).getTitle());
   String label = sb.toString();
   return label == null || label.length() == 0 ?
         getString("_UI_Requirement_type") : label;
}
```

Working with instances after changes

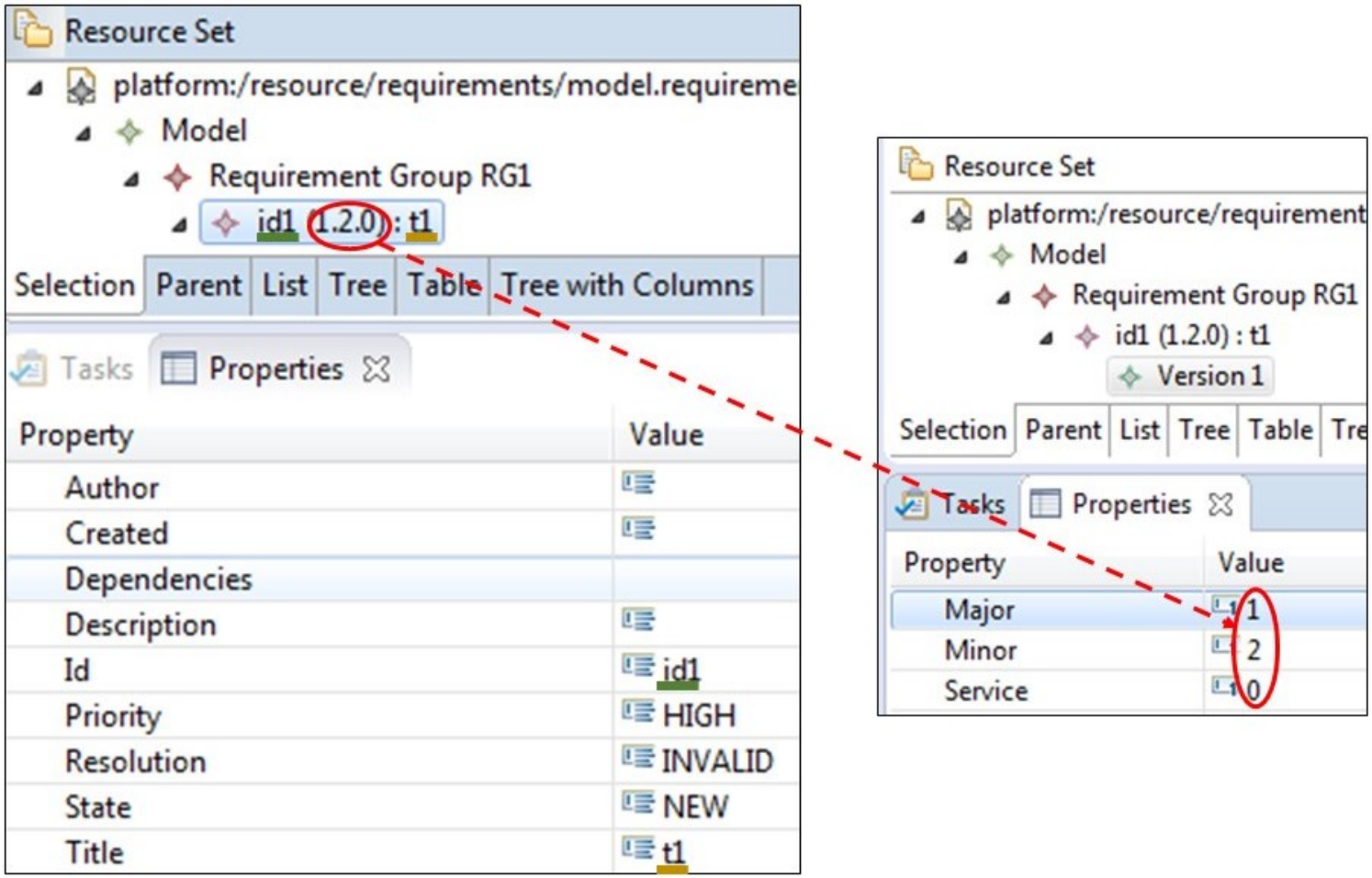


Figure 30: Editing instances after the update of getText

An instance for the college requirements

Mindmap is a technique also used to elicit the main concepts for requirements engineering [Pescador & Lara, 2016].

Create an instance for the requirements of the college system. All topics should be modelled as RequirementGroup items with the same hierarchy represented in the mindmap.

Choose one RequirementGroup to have two requirements related to creation and change. These requirements must have child references:

- `Performance Requirement`, non-functional requirement, with the following description: *It is to be done in less than 10 seconds*

Solutions

1. The previous steps should be followed in both solutions.
2. In addition, diagram class like representations of the models as provided by EcoreTools are to be included, similar to the ones presented in this document (see Figure 4 and Figure 28).
3. Each proponent of a solution must provide an alternative scenario for the mindmap or the requirement models, respectively.
   - **The alternatives to be considered are only the ones presented on the next two slides**.
   - The student must base his analysis on **a comparison between the alternative models he includes in his solution**.

Alternative mindmap model

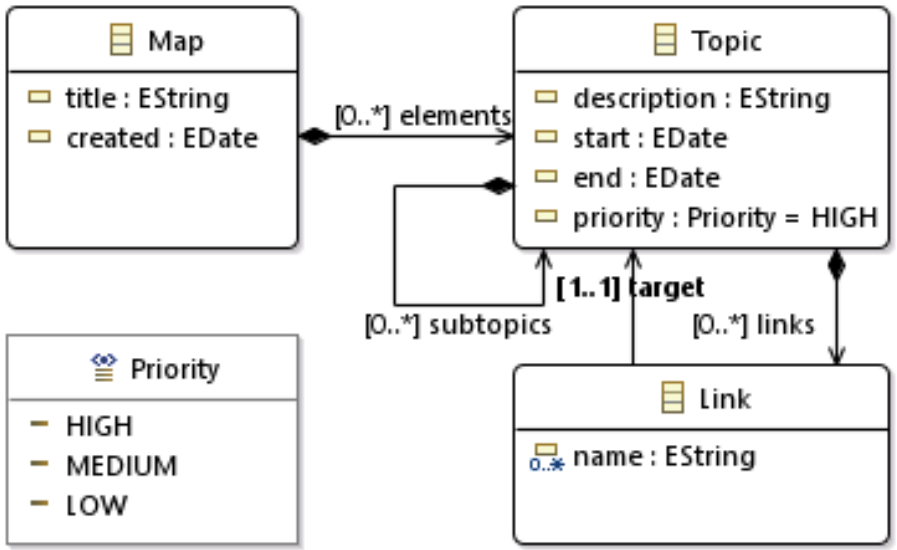


Figure 31: Another mindmap model

You should provide the new model represented in the image and the mindmap instance as previously explained (*How to represent DEPENDENCY relationships?*). Include a diagram class like representation of the model as provided by EcoreTools, similar to the one shown in Figure 31. Analize how this alternative compares with the original one.

Alternative requirements model

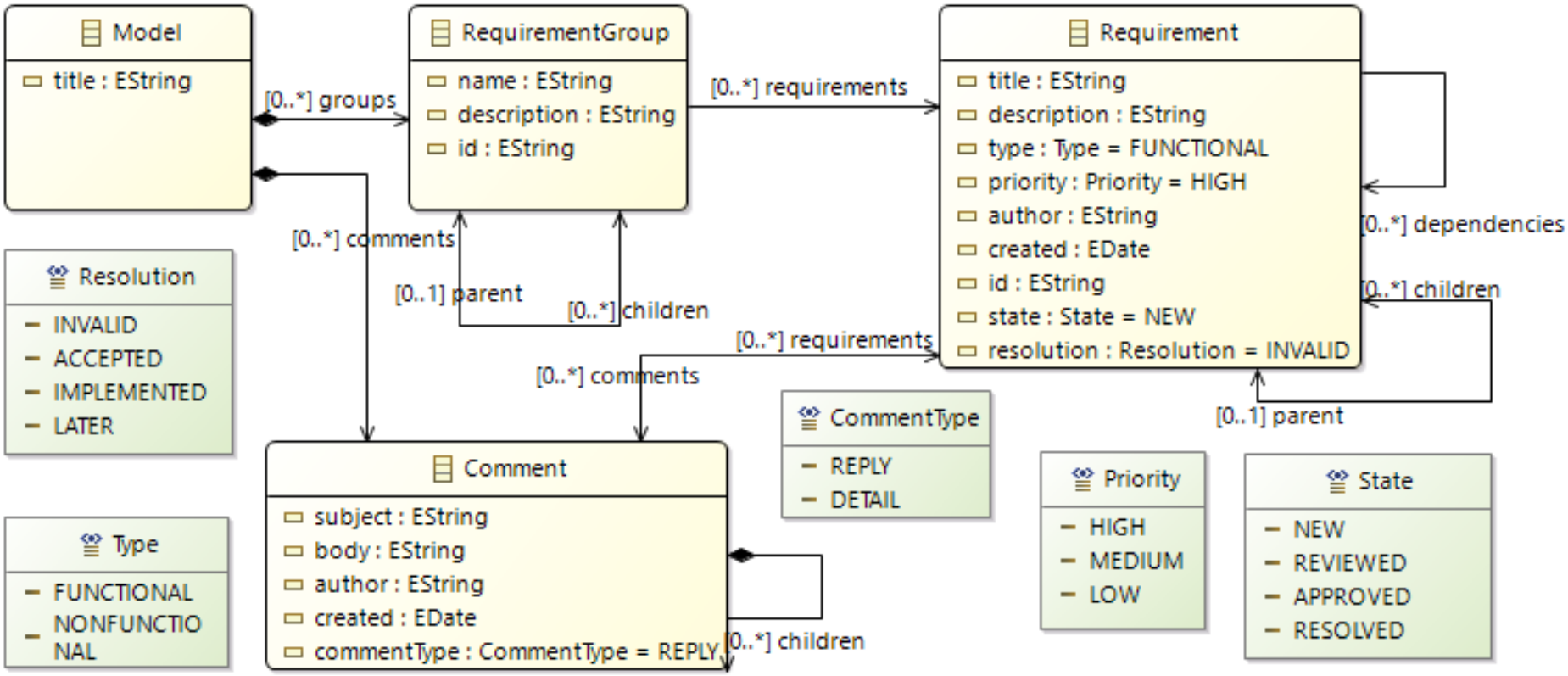


Figure 32: Another requirements model

You should provide the new model represented in the image and the requirements instance as previously explained. Include a diagram class like representation of the model as provided by EcoreTools, similar to the one shown in Figure 32. Analize how this alternative compares with the original one.

Some Useful Recipes...

Avoid Running a Second Instance of Eclipse

Sometimes running a second instance of eclipse may require computing resources not available in our system.

One way to avoid this situation is to install our plugins in the "default" eclipse.

- Locate in the installation folder of eclipse a subfolder named **dropins**.
- We can copy/install to this folder our plugins so that they are automatically loaded by eclipse when it starts.
- For that:
  1. Select "File/Export..." and then "Deployable plug-ins and fragments". Click "Next".
  2. Select the three plugins that were generated for the model: the project with the model; the "...edit" project; and the "...editor" project.
  3. In "Directory" browse and select the **dropins** directory of your eclipse installation.
- That's it. The next time you start eclipse the plugins of the model should be loaded and ready for use. Simple select "File/Restart" to see that happen.

Attention: If you make changes in you model you need to repeat the process!

Creating Dynamic Instances

Sometimes it may be useful to create instances of our metamodels without having to run a new eclipse instance our installing our plugins in eclipse (as described before)

- With your metamodel open, select the root element for the instance.
- Right click on it and select "Create Dynamic Instance..."
- Enter a name for the instance file (usually ending in ".xmi")
- The new file should open in an xml editor.
- Close the file and open again by:
  - Right click in the new file and select "Open With.../Sample Reflective Ecore Model Editor".
  - That's it. You can can edit your model!
- It is also possible to use the OCL console described earlier to query the model.
- OCL rule validations should also be available.

Attention: If you make changes in the metamodel it is always god practice to reopen your instance models.

Further Reading

Maximilian Koegel and Jonas Helming
What every Eclipse developer should know about EMF.
Jul 13th, 2015,
http://eclipsesource.com/blogs/tutorials/emf-tutorial/.

Christian Damus, Adolfo Sánchez-Barbudo Herrera, Axel Uhl and Edward Willink
OCL Documentation.
2015,
http://download.eclipse.org/ocl/doc/6.0.0/ocl.pdf.

OMG
Object Constraint Language. Version 2.4.
2015,
http://www.omg.org/spec/OCL/2.4.

Ana Pescador and Juan de Lara
ODSL-maps: from requirements to design of domain-specific languages.
2016,
Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 438-443. ACM.