Model-based Software Engineering (02341, spring 2016)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
Recapitulation of lecture 1 and some parts, which were skipped last week.
Models are the “floor plans” of software engineers, and are the key to the success of software projects.
Analysis

Design

Implementation

Coding

Code is generated
Example of a Petri net
Stages

- Examples
- Taxonomy (done on blackboard)
- Glossary
- Model (developed on blackboard)

**Rule:** Never ever start making a UML model without having looked at some examples first and naming the main concepts (taxonomy)!
Models (Meta Models)

Meta model for Petri nets

PetriNet

context Arc inv:
( self.source.oclIsKindOf(Place) and self.target.oclIsKindOf(Transition) )
or
( self.source.oclIsKindOf(Transition) and self.target.oclIsKindOf(Place) )

Object

Node

Arc

Transition

Place

Token

Meta model for Petri nets

Place

Transition

Token
Don’t think models as Java

**Rule:** Don’t think of programming for now! These models are on concepts only: the concepts of "our" example domain: Petri nets!

```
context Arc inv:
( self.source.oclIsKindOf(Place) and self.target.oclIsKindOf(Transition) )
or
( self.source.oclIsKindOf(Transition) and self.target.oclIsKindOf(Place) )
```
Syntax (abstract and concrete)

**Graphical / Concrete Syntax**

**Abstract Syntax** (as an UML object diagram)
Benefits of Modelling

- Better understanding

- Mapping of instances to XML syntax (XMI)

- Automatic code generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
generate an editor
Benefits of Modelling (cntd.)

- Better Understanding

- Mapping of instances to XML syntax (XMI)

- Automatic Code Generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
  - Editors and GUIs
Model Driven Architecture® (MDA®)
OMG™ software development approach for separating business logic from platform specific details
- using models
- automatic generators (for code and other models)

Model-based Software Engineering (MBSE)
General term for making “better” use of models for easing the software development

Ultimately: Getting rid of programming resp. technical artefacts.
We will always have programming and programmers!

We should always teach programming!

But, software engineers should be trained in their engineering and modelling skills!

And this is where they should be at their best!

Most of the rest can be automated!

Eventually, programming will be for software engineers as assembler is today for programmers.
3. Modelling with a Purpose

Analogies:

- Models as floor plans (see earlier slides)
  - Architects and construction engineers use quite different kind of plans – driven by the purpose
  - They even use models (miniatures)

- Models as maps
  - Understand the world (→ domain)
  - Find your way round in the software
Which of them is the best?
Software vs Programming

- For programs (small software), models are often not needed, and making them might be a waste of time.

- For software, they are essential for building something which works out and the different pieces fit to each other.
Tutorial 1: Q & A / Wrap up (BBD)
II. Modelling with a Purpose
1. Models to which end

- Blackboard Discussion (BBD):

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Kind of model</th>
</tr>
</thead>
</table>
Petri net example revisited (see next two slides)

Discussion:

- Should in/out (opposites of target and source) be in domain model?
- What makes it a domain model?
- What is the difference to a data model or data base schema?
Petri net: Domain model

PetriNet

context Arc inv:
  ( self.source.oclIsKindOf(Place) and 
    self.target.oclIsKindOf(Transition) ) 
or 
  ( self.source.oclIsKindOf(Transition) 
    and 
    self.target.oclIsKindOf(Place) )

Object

Node

Arc

Transition

Place

Token

1 source

1 target

*
Ecore model
Representations of model

Same model can have different representations:

- Graphical / tree (as of Tutorial 1)
- Java
- Ecore
- XML Schema (XSD)

Different representation might serve different purposes and have a different focus!

What would the focus for XSDs, Java and Ecore be?

Actually, in our EMF technology, Ecore models can be imported from XML Schema and from annotated Java classes (see Java example on the next slides).

Also JPA can be considered a model represented in Java (with annotations mapping it to a database schema).
/** @model */

public interface Petrinet {

    /** @model opposite="petrinet" containment="true" */
    List<Node> getNodes();

    /** @model opposite="petrinet" containment="true" */
    List<Arc> getArcs();

    /** @model */
    String getName();

}
/** @model */
public interface Arc {

    /** @model opposite="out" required="true" */
    Node getSource();

    /** @model opposite="in" required="true" */
    Node getTarget();

    /** @model opposite="arcs" transient="false" */
    Petrinet getPetrinet();

}
/** @model abstract="true" */
public interface Node {

    /** @model opposite="nodes" transient="false" */
    Petrinet getPetrinet();

    /** @model opposite="target" */
    List<Arc> getIn();

    /** @model opposite="source" */
    List<Arc> getOut();

    /** @model */
    String getName();
}

/**
 * @model
 */

public interface Transition extends Node {

}
/**
 * @model
 */

public interface Place extends Node {

/**
 * @model opposite="place" containment="true"
 */
 List<Token> getTokens();
}


/**
 * @model
 */

public interface Token {

/**
 * @model opposite="tokens" transient="false"
 */

Place getPlace();

}
/** @model */
public interface Petrinet {

    /** @model opposite="petrinet" containment="true" */
    List<Node> getNodes();

    /** @model opposite="petrinet" containment="true" */
    List<Arc> getArcs();

    /** @model */
    String getName();
}

3. Software Models

Petri net example (cntd.): Models for

- (small part of) the generated code
- framework the generated code uses

Two objectives:

- Understand (a bit) the generated code and the framework behind it
- See how models can be used for that purpose
3.1. Eclipse: JFace

- “JFace is a UI toolkit with classes for handling many common UI programming tasks.”
  [https://wiki.eclipse.org/JFace]

- Viewers are a core part of editors (there are different kinds of viewers), which are generic.

- Here, we discuss the TreeViewer, which is the basis for the automatically generated tree editor for Petri nets.
Assuming that the input object (model) is a Petri net
TreeViewer

TreeViewer

1

input

Object

Shows the input as a tree (with all the features of a tree view like opening and closing sub-trees, etc)

Root object of the tree which is to be shown in the TreeViewer
How could the TreeViewer, which does not know anything about Petri nets (and the classes representing the concepts of Petri nets), know how this tree should be shown?
TreeViewer

ILabelProvider
- getText(Object) : String
- getImage(Object): Image

Provides the label and the icon for each object

ITreeContentProvider
- getChildren(Object) : List
- getImage(Object): Image

For each object, provides the current list of children.

TreeViewer

Object

input

1

1

labelProvider

1

contentProvider

1
In the tutorial, you will change the item provider for Arcs for changing the labels for arcs.

Will come from the generated code (ItemProviders for each kind of object in the model): edit code
Similarly for Properties

![Petri net diagram]

IPropertySourceProvider (not discussed here)
3.2 EMF: Use of TreeViewer

- In EMF, this is even more complicated: using a generic ContentProvider, which creates the respective ItemProviders and delegates to them

Idea is discussed on blackboard (BBD)
3.3 Update Viewer on Changes

- In order to make sure that the viewer properly updates, whenever changes occur, it registers itself as listener to the respective elements (actually to their ItemProviders.

Idea is discussed on blackboard (BBD); more details next time