Model-based Software Engineering for/with Petri nets

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
1. Motivation
Today

Technical artefacts!

Get rid of them!
Exploit conceptual artefacts instead (and generate software from them).
There are tools that partially support this idea already today (e.g. Eclipse and EMF).

Analysis

Design

Implementation

Coding

+ analysis and verification!

Code is generated
"Buzzwords"

- Model Driven Architecture® (MDA®) OMG™ software development approach for separating business logic from platform specific details
  - using models
  - transformations
  - 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.
If somebody has built a garage, we would never let him build a sky scraper, would we?

But, if somebody knows how to program, we let him do software engineering, don’t we?
**Analysis**

**Design**

**Implementation**

**Coding**
How do Petri nets come into this vision?

- How can we exploit this for developing Petri net tools?
- How can Petri nets be used for making software?

We do a bit of both in this course!
Schedule

9:00-10:30: Session 1
MBSE overview
- vision, idea, concept
- technologies

11:00-12:30: Session 2
Technology details
Modelling behaviour
- Petri nets and/in software
- Behaviour coordination

13:00-15:00: Session 3
Model-based technology in Petri nets
- PNML (overview)
- ePNK (overview)

15:30-17xx: Session 4
Project (hands-on)
Model-based Software Engineering for/with Petri nets
MBSE: Introduction and Overview

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
Table of Contents

- Motivation
- MBSE overview
  - example (and basic concepts)
  - discussion
- Principles and Concepts
  - MVC: model, view, controller
  - “meta”, “meta-meta”, MOF
- Technologies (overview)
  - Eclipse Modeling Framework (EMF) and GMF
  - Transformations (M2M, M2T)
2.1. Example: A Net Editor
A Model (Petri net)

A rule of modelling:
→ Before making a model look at examples (instances)
→ Before making a meta-model look at the models (instances of meta-models)
"Meta model" for Petri nets
(as a UML class diagram + OCL)

Petri net model

→ PNML meta model
ISO/IEC 15909-2
Syntax (abstract and concrete)

graphical / concrete syntax

abstract syntax
(as an UML object diagram)

:PetriNet

:Place

:Arc

:Transition

:Place

:Arc

:Token

:Arc

:Transition
Abstract syntax could also “look” like this (in EMF generated tree editor)!
Overview

meta model

is an instance of

model

build-time

runtime

model

is an instance of

runtime

meta model

is an instance of

model

build-time

runtime

* is an instance of *

* is an instance of *

is an instance of

is an instance of

is an instance of

is an instance of
Benefits of Modelling

- Better understanding
- Mapping of instances to XML syntax (XMI) (canonical exchange format for instances)
- 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)
    - Transactional changes (ACI)
    - Database access / persistence (D)
Class Diagrams are Models too

**Disclaimer:** The real model of UML is much more evolved (see MOF and UML standards)

Now, the term “meta” model makes more sense!
Different Meta-levels: MOF

Where does the concrete syntax come from?
EMF/GMF Technology

meta model

A Petri net editor in 15 minutes!
Not kidding!

concrete syntax

abstract syntax

generate an editor
GMF not in the focus of this course. But if you are interested, please, feel free to ask.

If I'd start the ePNK project today, I'd probably chose Graphiti: https://www.eclipse.org/graphiti/
2.2 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)
  - Editors and GUIs

*NB: All this is “standard functionality”?

No programming at all!
Purpose of (good) models

- Abstraction
- Focus
- Simplification
- Separation

- Understanding
- Communication
- Analysis and verification
- Execution (interpretation / code generation)

But isn’t code just another form of model?
From this (EMF) model for Petri nets:
Generation of (Java) code for
• all classes
• methods for changing the Petri net
• loading and saving the Petri net as XML files (→XMI)

With this and some more GMF information:
Generation of the Java code of a graphical complete editor (with many fancy features). No programming at all.

Almost all you need to say about a Petri net editor.
Conventional: We can generate parts of the code from UML class diagrams automatically (MDA, MDE, EMF, EMFT/GMF)

- Class diagrams → Java class stubs with standard access methods (see basic project)
- Implementation of standard behaviour:
  - Loading and saving models
  - Accessing and modifying the models
  - Editors and graphical user interfaces
- The actual functions is implemented by hand (e.g. simulator, state-space generator, …)

Advanced: Actual functions also „modelled“ and code generated

We come back to that in the modelling behaviour session.
3. Principles and Concepts

- Model, view, controller
- ”Meta”, ”meta-meta”, who has more?
- MOF
The **domain models** are an (the) essential part of the software

In addition to that we need

- Information about the presentation (view) of the model to the user
- The coordination / interaction with the user

**Note:** These parts of the software can also be modelled! Petri nets are probably particularly suited for this.
Model View Controller (MVC)

**View**

```
+-----+----+
|     |    |
|     |    |
|+----+----|
|     |    |
|     |    |
```

**Model**

```
- PetriNet
  - Object
    - Node
      - Transition
      - Place
      - Token
      - Arc
```

**Controller**

```
:Arc
```
Model View Controller (MVC)

**Model**
- Domain model and functions

**View**
- Representation of model and user interaction

**Controller**
- Makes changes and calls functions of the model

**Notes:**
- This is a rough idea only!
- There are many variants (e.g. GEF/GMF uses this in a slightly differently way)
MVC is a principle (pattern / architecture) according to which software should be structured.

Eclipse and GEF (as well as GMF) are based on this principle and guide (force) you in properly using it.

When using MBSE technologies, many more design patterns are relevant.
This part can be generated automatically (e.g. GMF)
### 3.2 Meta Object Facility

<table>
<thead>
<tr>
<th>M2</th>
<th>Unified Modelling Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Model</td>
</tr>
<tr>
<td>M0</td>
<td>User data</td>
</tr>
</tbody>
</table>

- **modelling notation**
- **design-time**
- **run-time**
Re: Example

MBSE f/w Petri Nets: MBSE: Introduction and Overview
What, if we would like to use another modelling notation!

<table>
<thead>
<tr>
<th>M2</th>
<th>Unified Modelling Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Model</td>
</tr>
<tr>
<td>M0</td>
<td>User data</td>
</tr>
</tbody>
</table>

= conforms to / is instance of
<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>MOF notation</td>
</tr>
<tr>
<td>M2</td>
<td>Any modelling notation</td>
</tr>
<tr>
<td>M1</td>
<td>Model</td>
</tr>
<tr>
<td>M0</td>
<td>User data</td>
</tr>
</tbody>
</table>

Beware a myth: Though “suggested” by the first versions of MOF and related standards, the number of levels is NOT fixed! There can be any number of levels!
MOF: Discussion

- Is that any good?
  - There is one level that we did not have before! So, this seems to be more complicated!

- If UML can be defined in terms of itself, why should we define it in terms of something else?

What do you think?
## Meta Object Facility (MOF)

<table>
<thead>
<tr>
<th>M3</th>
<th>MOF notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>Any modelling notation</td>
</tr>
<tr>
<td>M1</td>
<td>Model</td>
</tr>
<tr>
<td>M0</td>
<td>User data</td>
</tr>
</tbody>
</table>

= conforms to / is instance of

The MOF model(s)

MOF instance model and reflection
Actually, this comes from the UML infrastructure Core::Basic.

The MOF standard refers to and uses concepts and notations from the UML standard (infrastructure).

Ecore is an implementation of MOF (actually) EMOF.
EMOF Classes

**Type**

**TypedElement**

**Class**

- `isAbstract`: Boolean = false

**Property**

- `isReadOnly`: Boolean = false
- `default`: String [0..1]
- `isComposite`: Boolean = false
- `isDerived`: Boolean = false
- `isID`: Boolean

**MultiplicityElement**

- `isOrdered`: Boolean = false
- `isUnique`: Boolean = true
- `lower`: Integer = 1
- `upper`: UnlimitedNatural = 1

**Operation**

- `0..1 {ordered} 0..*`

**superClass**

There is a bit more, but we do not go into it.
Meaning of "Meta-

Meta (from Greek: μετά = "after", "beyond", "with", "adjacent", "self"), is a prefix used in English in order to indicate a concept which is an abstraction from another concept, used to complete or add to the latter.

In epistemology, the prefix meta- is used to mean about (its own category). For example, metadata are data about data, something about something (who has produced them, when, what format the data are in and so on). Similarly, metamemory in psychology means an individual's knowledge about whether or not they would remember something if they concentrated on recalling it. Furthermore, metaemotion in psychology means an individual's emotion about his/her own basic emotion, or somebody else's basic emotion.

Another, slightly different interpretation of this term is "about" but not "on" (exactly its own category). For example, in linguistics a grammar is considered as being expressed in a metalanguage, or a sort of language for describing another language (and not itself). A meta-answer is not a real answer but a reply, such as: "this is not a good question", "I suggest you ask your professor". Here, we have such concepts as meta-reasoning and meta-knowledge.

…

From: http://en.wikipedia.org/wiki/Meta
"The Meaning of Meta"

Connotations and meaning of "meta" in Software Engineering:

- beyond, "one level higher"
- possibly self-referential
  (with all the problems of self-referentiality)

Often also:

- a UML model
- a class diagram

Self-references are at the core of all paradoxes. Example: “This statement is wrong”!

Abuse of language! I guess, introduced by people working only or just too much on the meta-level.
Analysis
Design
Implementation
Coding

"Standard code" is generated
If we want to get software automatically from models,

- we need to have a technology for making models
- transforming between models and into code

EMF provides specific mechanisms for making models and transforming them into (Java) code.

What if we want other modelling notations (DSL) or other transformations?
4.1. EMF/GMF for DSL
4.2. Transformations

Basically, two kinds of transformations in MBSE
- model to model transformation (M2M)
- model to text (M2T)

In this course, we consider Java Emitter Templates (JET) as an example of M2T

Characteristics of M2T:
- No restrictions of output format
- No need to define output syntax
- Flexibility
Dear <Name>,

we are pleased to inform you that you will be refunded <amount> in income tax.

The reason is that <reason>.

Best regards,
<clerkname>

Today, this is much better known from web programming: PHP or JSP, ASP, ...

Standard text in which some "specifics" will be filled in (attributes/parameters/fields).
TBC after break
Appendix
EMOF Classes

Type

TypedElement

Class

ownedAttribute

Property

MultiplicityElement

Operation

isAbstract: Boolean = false

class

isReadOnly: Boolean = false
default: String [0..1]
isComposite: Boolean = false
isDerived: Boolean = false

ownedOperation

isUnique: Boolean = true

upper: UnlimitedNatural = 1

opposite

0..1

To be continued on next slides!
EMOF Classes (cntd.)

**TypedElement**

**MultiplicityElement**

**TypedElement**

**MultiplicityElement**

**Operation**

operation ownedParameter

{ordered} 0..* 0..*

Type

raisedException
EMOF Data Types

Type

DataType

PrimitiveType

Enumeration

NamedElement

Enumeration

Literal

0..1 {ordered} 0..*

ownedLiteral

enumeration
EMOF Packages

- **NamedElement**

- **Package**
  - uri: String
  - package
  - ownedType
    - 0..1
    - nestingPackage
  - 0..1
  - nestedPackage
  - 0..*

- **Type**
Reflection package

For properties with more than one value, there exist ReflexiveCollection and ReflexiveSequence (similar to Java Collections)!

MBSE f/w Petri Nets: MBSE: Introduction and Overview