Model-based Software Engineering for/with Petri nets
Behaviour Modelling (and its challenges)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science
Motivation

[Diagram showing the relationship between Petri nets, objects, nodes, places, and arcs, with an example of a Petri net model being implemented in a software tool.]

MBSE f/w Petri Nets: Challenges of Behaviour Modelling
EMF/GMF Technology

meta model

is instance of

generate an editor

model

concrete syntax

abstract syntax
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

But, all this is “standard functionality” or non-standard behaviour?
Table of Contents

- Motivation
  - Model-based Software Engineering
  - Business process modelling
- Challenges and Problems
- Some Ideas
  - modelling behaviour
  - integration and coordination of behaviour
- Discussion
Example: Business Trip

Workflow Management Systems

For business processes, the "modelling only"
idea is working now for 25 years.

SOA, web services, PAIS, ERP are close-by and more modern buzzwords!

application form

copy of af

trip documents

receipts

note

lecture determine trip data

superior support trip

dean approve trip

clerk

send trip form

clerk reimburse expenses

clerk reimburse-ment form

clerk reimburse-ment form & receipts

 lecturer

Trip documents

receipts

approval

make trip

fill in trip form

book trip

send trip form

clerk

lecture"
Where does the “modelling only” idea work?

Graphical editors
- graphics only
- standard functions only
- no business logic

Workflow management
- standard GUI only
- business logic explicitly modelled
- dedicated modelling notation
e.g. a Petri net simulator?
"There are no notations for modelling behaviour!"

This claim is actually as wrong as it can get! There are (too?) many such notations!
Challenges

- Adequate modelling methodologies
  - Coarse grain behaviour
  - Fine grain behaviour

- Mechanism for integrating and coordinating behaviour beyond invocation (calls of procedure, function, method, or service)

- Integration with
  - existing software (legacy, manually created, generated)
  - other models (structural & behavioural)

- Change mentality (change culture)
  - Stuck with thread- and invocation-based thinking
  - Software engineering is programming thinking
    (→ model interpretation vs. code generation)

Moreover: Today’s modelling technologies are not very “agile”!
Table of Contents

- Motivation
  - Model-based Software Engineering
  - Business process modelling
- Challenges and Problems
- Some Ideas
  - modelling behaviour
  - integration and coordination of behaviour (a vision: Event Coordination NOtation ECNO)
- Discussion
2. **The Event Coordination NOtation**

**Motivation**

- Given some object oriented software with (or without) explicit domain model,

- model behaviour on top of it – and make these models executable.

- Model behaviour on a high level of abstraction (domain): coordination of behaviour

→ Integrate behaviour models with structural models

→ Integrate different structural models and manually written code (or code generated by different technologies)
2.1 Example: Vending machine

Class diagram as usual

- Coin
- Slot
- Safe
- Panel
- Control
- Brewer
- Output
- Coffee
- Tea

1

*
Instance: Object Diagram

Initial configuration, current situation

Object diagram as usual
- We call objects elements now!

- Events (event types)
- Coordination annotations: event type + quantification annotation
... + Event declaration

- Event (type) declaration
- Parameters

```java
insert(Coin coin, Slot slot) coffee()
pass(Coin coin, Slot slot) tea()
return(Slot slot) cancel()
reset()
```
Interaction

:Coin → :Slot
:Coin → :Slot
:Panel → :Control
:Safe → :Control

:Slot
:Panel
:Safe

:Control
:Tea
:Output

:Coin
:Coin
:Coin

:Safe
:Safe
:Safe

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass

:Pass
:Pass
:Pass
Interaction = local behavior + coordination

We come back to local behaviour! → slide 20
Another Interaction

Interaction = local behavior + coordination

- Return
- Coin
- Safe
- Panel
- Slot
- Control
- Coffee
- Tea
- Output

Actions:
- Reset
- Return
- Cancel

Connections:
- Return: ALL
- Reset: ALL
- Cancel: ALL
Local behaviour: Coffee

Elements are objects with an explicitly modelled life-cycle

Event binding

\[ r = \text{reset}(); \]

\[ c = \text{coffee}(); \]

\[ \text{cup} = \text{cup\_in}(); \]

\[ \text{ready} \]

\[ \text{brewing} \]
Local behaviour: Coin

- Event binding
- Parameter assignment
Local behaviour: Control

- Event binding with multiple event types!
Local behaviour: Slot

\[
\text{self.getCoin().size()} < 2
\]

\[
i = \text{insert}(\text{none, self}); \quad \text{self.getCoin().add(i.coin)};
\]

\[
p = \text{pass}(\text{none, self}); \quad \text{self.getCoin().remove(p.coin)};
\]

\[
\text{res = reset();}
\]

\[
\text{r = return_(self)}; \quad \text{self.getCoin().clear();}
\]
Interaction = local behavior + coordination

Interaction:
- Slot: return, reset
- Coin: return
- Safe: return
- Panel: cancel
- Control: cancel, reset
- Coffee: reset
- Tea: reset
- Output: 

Diagram:
- Interaction nodes: Slot, Coin, Safe, Panel, Control, Coffee, Tea, Output
- Arrows: return, reset, cancel

MBSE f/w Petri Nets: Challenges of Behaviour Modelling
2.2 ECNO: Basic Concepts

- **ElementTypes (Classes)**

- **EventTypes with**
  - parameters

- **Global Behaviour: Coordination annotations for references**
  - Event type
  - Quantification (1 or ALL)

- **Local behaviour (life-cycle): ECNO nets (or something else)**
  - Event binding (with parameter assignment)
  - Condition
  - Action
ECNO with its basic concepts has some limitations, which makes modelling things in an adequate way a bit painful. ECNO has some additional concepts to make modelling more convenient. E.g.

- Inheritants on events
"Nicer Vendingmachine"

MBSE f/w Petri Nets: Challenges of Behaviour Modelling
Behaviour inheritance

Clearer separation of life-cycle of general brewer behaviour and coffee brewer specifics.

$\text{c} = \text{coffee}();$

$\text{cup} = \text{cup\_in}();$

$\text{drinker}();$

$\text{reset}();$

$\text{ready}.$

$\text{brewing}.$
2.4 Example 2: Petri nets

Transition $t$ **enabled**: for ALL incoming Arcs $a$: for ONE source Place $p$ of Arc $a$: find a token

**Fire** Transition $t$: for ALL incoming Arcs $a$: for ONE source Place $p$ of Arc $a$: find a token and remove it

for ALL outgoing arcs $a$: for ONE target Place $p$ of Arc $a$: add a new Token

How can we model that behaviour in ECNO nets?
Petri net: Abstract Syntax
Example 2: Petri nets

Transition $t$ enabled:
for \textbf{ALL} incoming Arcs $a$:
for \textbf{ONE} source Place $p$ of Arc $a$:
find a token

Fire Transition $t$:
for \textbf{ALL} incoming Arcs $a$:
for \textbf{ONE} source Place $p$ of Arc $a$:
find a token and remove it

for \textbf{ALL} outgoing arcs $a$:
for \textbf{ONE} target Place $p$ of Arc $a$:
add a \textbf{new} Token
ECNO Semantics of PN

**Transition**
- fire
  - add
  - remove
- add $\rightarrow$ ALL
- remove $\rightarrow$ ALL

**Arc**
- add
  - add $\rightarrow$ ONE
  - remove $\rightarrow$ ONE

**Place**
- add
  - add $\rightarrow$ ONE
  - remove $\rightarrow$ ONE

**Token**
- tokens
  - remove

```
f = fire(); r = remove(); a = add();
a = add();
r = remove();
r = remove();
```

```
import dk.dtu.imm.se.ecno.example.petrinets.PetrinetsFactory;
final PetrinetsFactory factory = PetrinetsFactory.eINSTANCE;

a = add();
self.getTokens().add(factory.createToken());

r = remove();
```
Petri net simulator

MBSE f/w Petri Nets: Challenges of Behaviour Modelling
Find released ECNO Tool: http://www2.compute.dtu.dk/~ekki/projects/ECNO/
ECNO nets and the code generator are probably the largest application of the ePNK!

```java
import dk.dtu.imm.se.ecno.engine.ExecutionEngine;

final ExecutionEngine engine = ExecutionEngine.getInstance();

i = insert(self, none);
self.getSlot().remove(i.slot);
engine.removeElement(self);

r = return_(none);
self.getSlot().add(r.slot);
engine.addElement(self);

p = pass(self, none);
```
Beyond Mickey Mouse
Discussion

- Models of software on higher level of abstraction → concise / adequate

- Domain model including semantics

- Coordination! Not invocation!

- Idea: Define semantics of ECNO in ECNO itself (truly "meta")
ECNO: Focus

- Software Engineers (vs. end-users)
- Domain level (vs. low-level programming)
- Coordination (vs. algorithmics/invocation/sequential flow)
- Behaviour (vs. GUI)

But, there is a project were standard GUI generators could be coupled with ECNO, to model the complete software including behaviour and GUI!
"Oh, that’s great. Then I can also develop my own software now."

My first reaction: 
Nooo?! 
ECNO is for software engineers, helping them focussing on the domain and not on technical details!

On second thought: 
Can’t we help end-users develop their own programs? At least in simple cases.