



# Software Engineering 2

A practical course on software engineering

Ekkart Kindler

Technical University of Denmark

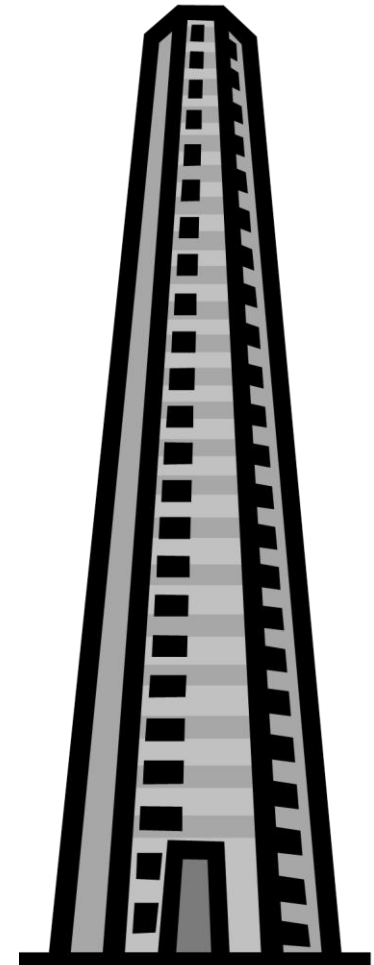
Informatics and Mathematical Modelling



The course in  
hindsight

Conclusion, Outlook, Discussion

# Motivation



group. Every participant of the course is expected to be an active part of at least one of these presentations.

The (preliminary) schedule for the different parts of course can be found at <http://www2.imm.dtu.dk/courses/02162/e12/material.html> along with the deadlines for the deliverables and the slots for the presentations. All the material the lectures and tutorials will be made available via these pages. On this web page, you will also find a rough work for the project. The exact details will be fixed in the first week of the semester (when we know the exact number of participants).

Please be aware that, in addition to the above slots, each participant is expected to invest about 12 hours per week on the project and the tutorials (10 ECTS points correspond to an overall workload of 270 hours). This work, however, is more flexible and a matter of your individual and your group's work plan.

## Objective

Sometimes, we are tempted to believe that making software is programming—just bigger. But, this is not at all true. For developing software, we need good skills in programming, of course. But, this is not enough for successfully completing a software project. Other skills are not less important:

- social interaction and communication (orally as well as in writing),
- soliciting and defining the exact requirements,
- modelling the domain,
- making architecture and design decisions,
- analysing the models,
- implementing the designed system,
- testing it,
- using state-of-the-art technologies (or to acquire new ones), and
- project management.

The course on Software Engineering 2 (02162) will help acquiring these skills.

## Structure

In order to acquire these skills, the course consists of three main parts: *lectures*, *tutorials*, and the *project*, where the

The screenshot shows a web browser window with the following content:

- Browser title: Software Engineering 2 (02162)
- Address bar: www2.imm.dtu.dk/cours
- Page content includes a schedule for Tuesday 8<sup>15</sup>-12<sup>00</sup>: Lec, Friday 13<sup>00</sup>-ca. 14<sup>45</sup>: L, and Friday ca. 15<sup>00</sup>-17<sup>00</sup>: G.
- Text: "In addition to these slots, th time will be discussed once th"
- Text: "Note that, during this course, presentation, the presentatio group. Every participant of th"
- Text: "The (preliminary) schedule for /e12/material.html along with the lectures and tutorials will for the project. The exact det participants)."
- Text: "Please be aware that, in add the project and the tutorials ( more flexible and a matter of"
- Section header: "Objective"
- Text: "Sometimes, we are tempted t For developing software, we completing a software projec"
- List of skills (identical to the one in the main image).
- Text: "The course on Software Engineering 2 (02162) will help acquiring these skills."
- Section header: "Structure"
- Text: "In order to acquire these skills, the course consists of three main parts: lectures, tutorials, and the project, where the"

# 1. What did you learn?

- Objectives of this course:  
Basic skills in software engineering!

What did you learn?

What is important?

- ... much more than programming!
- ... listening and understanding!
- ... analytic and conceptual work!
- ... communication!
- ... a social process!
- ...
- ... acquiring new technologies!
- ... a discipline with proven concepts, methods, notations, and tools!
- ... and ever new technologies emerging!

Software Engineering requires much experience!

This experience

- can not be taught theoretically!
- will be provided in this course!

→ tutorial

→ project

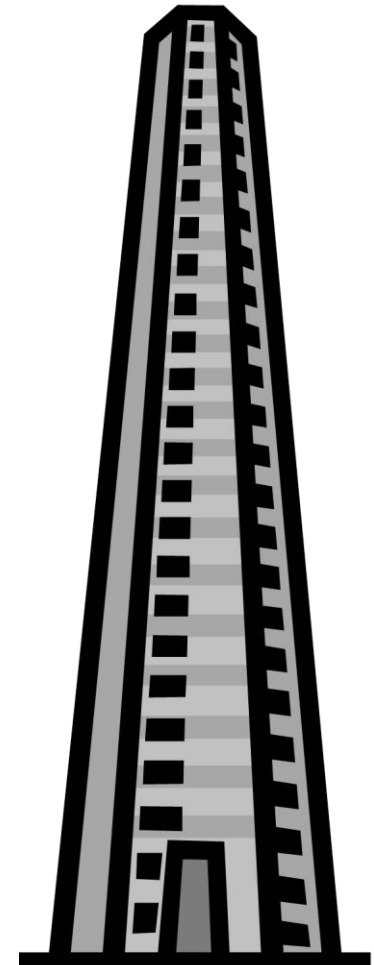
→ and (only) backed by the lectures

## Practice the concepts, methods, notations and Tools for software engineering

- (improve programming skills)
- understanding of the software engineering process
- experiences with problems and concepts for solving them
- writing documents and creating models
- use of methods and tools
- practice communication and presentation skills
- capability of teamwork and leadership
- acquire new technologies
- ...



# Motivation

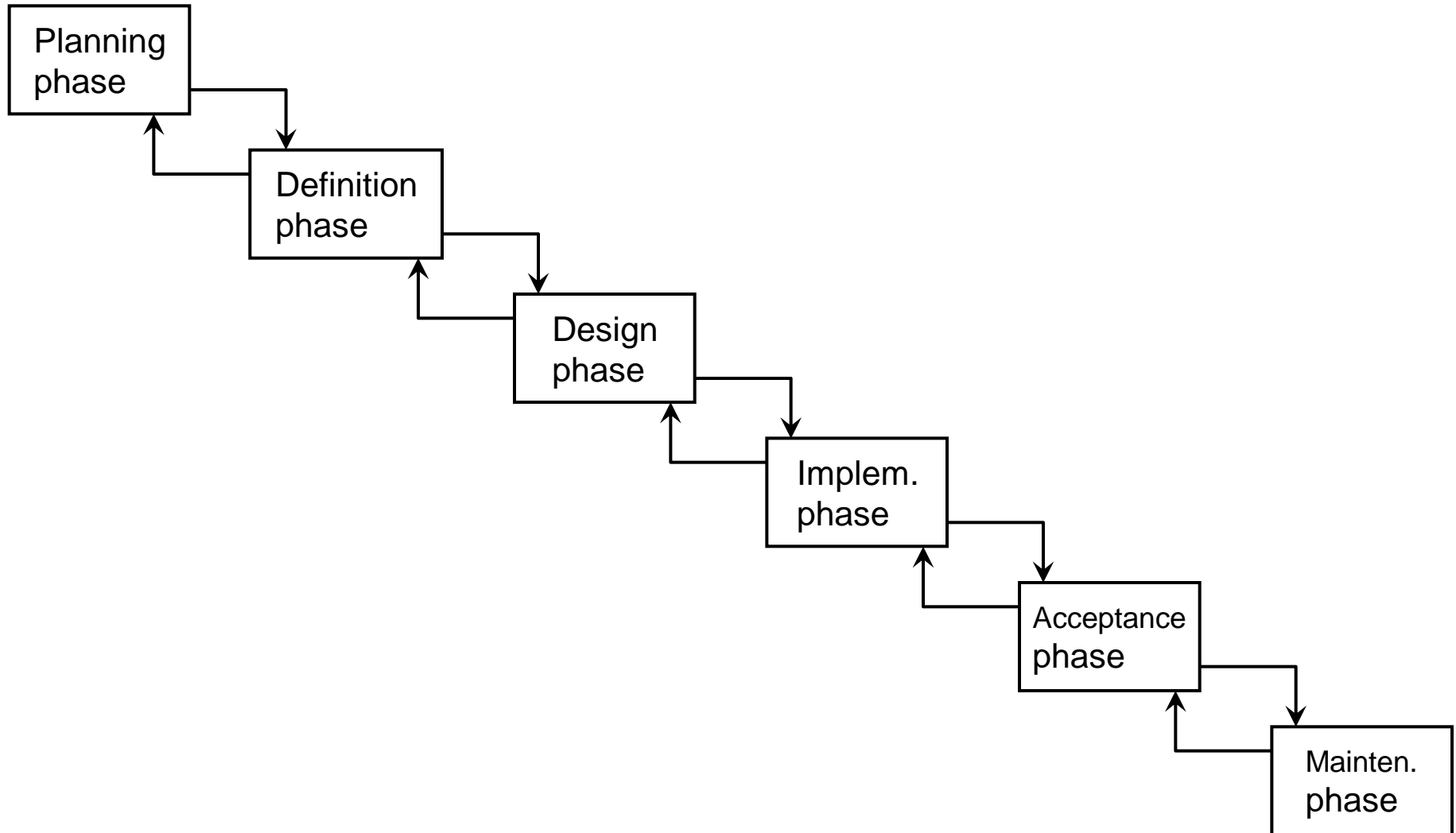


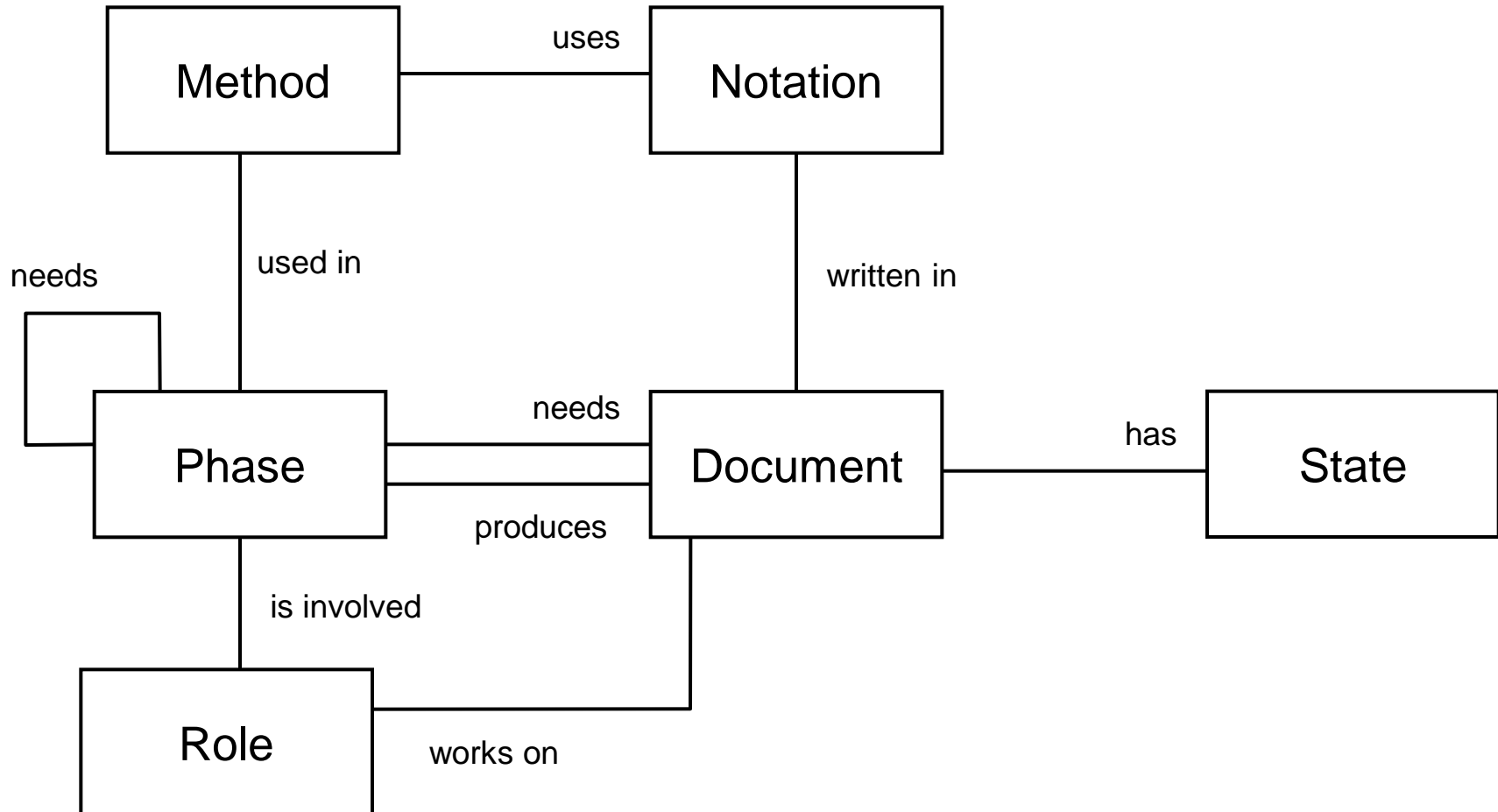
- imprecise requirements
- mistakable and unclear requirements
- inconsistent requirements
- changing requirements
  
- changing environments (software / hardware)
- different versions and configurations
- changing tools, notations, languages, methods, concepts, technologies
  
- collective knowledge only
- communication
- ...

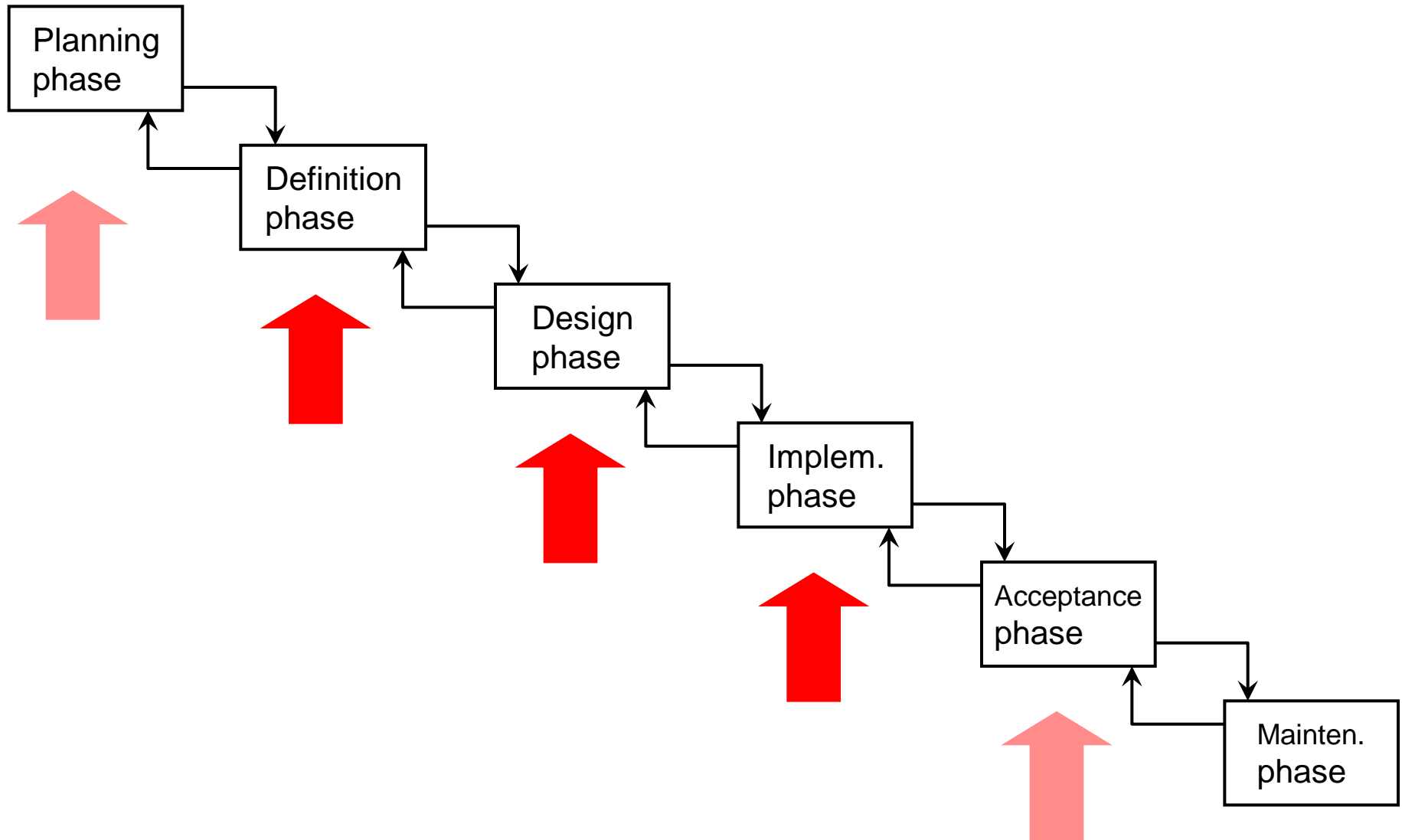
**Explicit** and  
concrete  
communication!

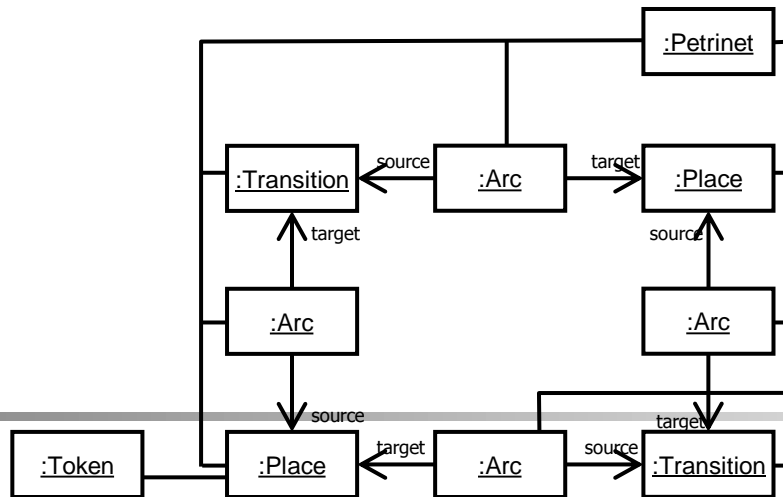
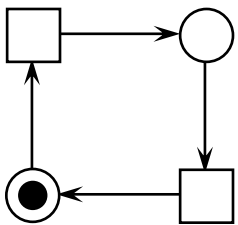
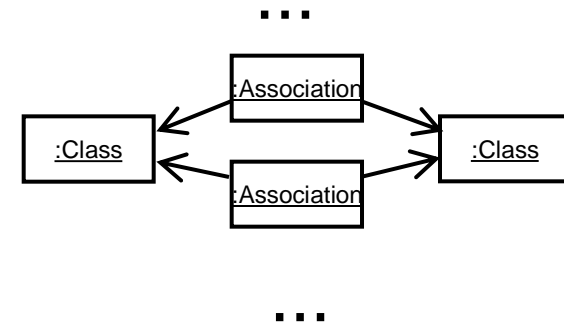
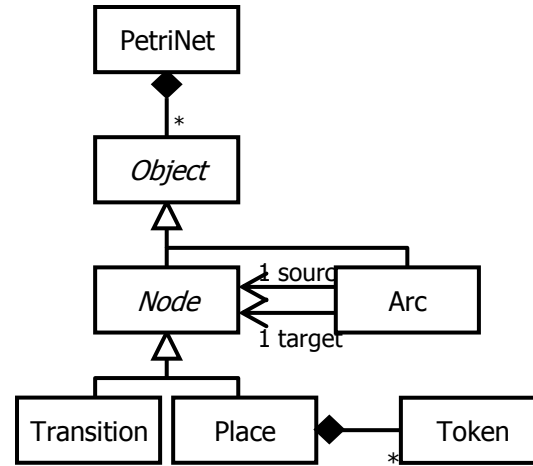
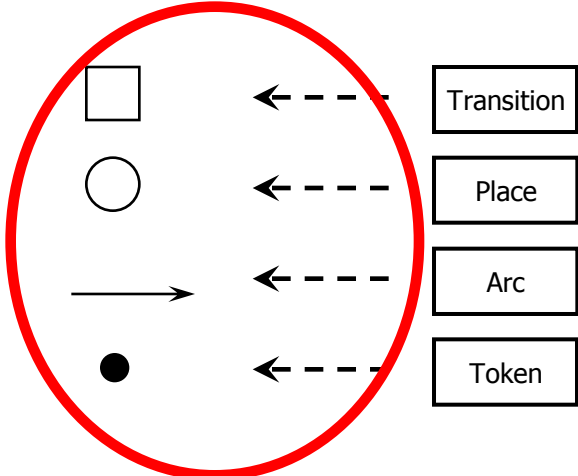
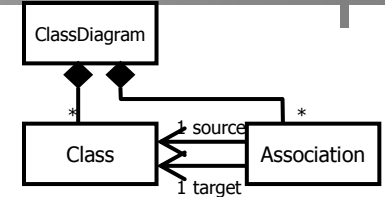
## Branches:

- **Development:**  
actual development of the software product
- **Management:**  
Manage (control and improve) the development process
- **Quality management:**  
Planning and implementing measures that guarantee that the software meets the required quality
- **Software maintenance:**  
Remove faults occurring in operation, adapt software to changing requirements and environments

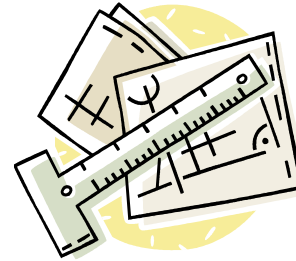








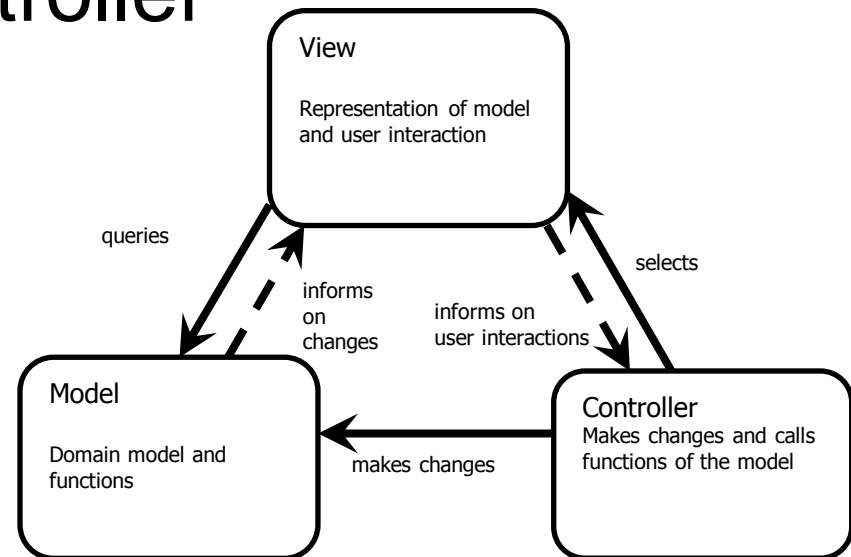
## ■ Purpose and roles of models in SE



Making adequate/helpful models!

## ■ Model / View / Controller

Principles behind!





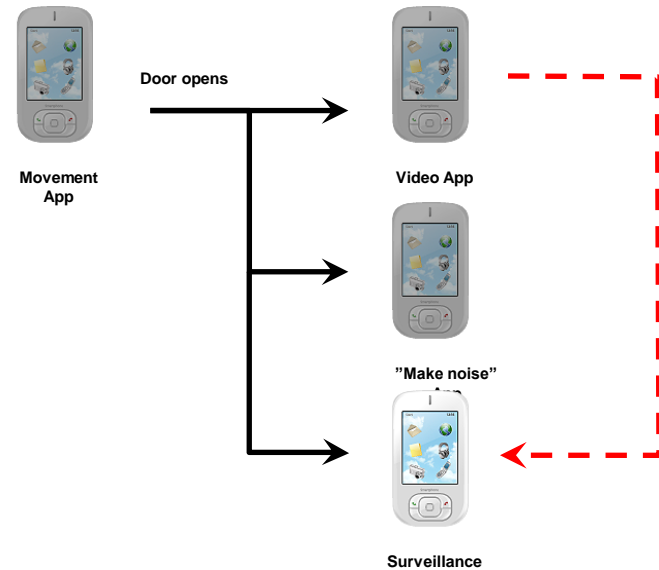
## How to make useful UML models

- Domain model OOA
  - Implementation OOD
- 
- Learn to use new technologies  
**YOURSELF!**

Eclipse / GlassFish / Webservices /  
Android / Derby / JavaCC / ...

You did it!?

- Get requirements straight



- Write a systems specification

**Important:** Have another look at your documents now and ask yourself: What should have been in there?

And why!  
(objective)

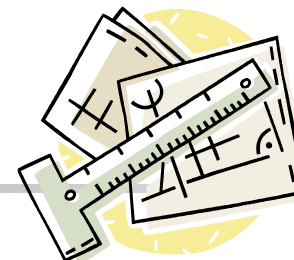
- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook



what



how



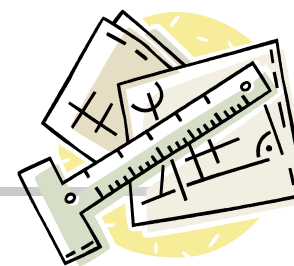
- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook



**rough**



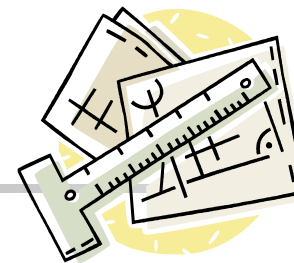
**detailed**



- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook



**low cost**



**high cost**

- writing,
  - talking,
  - communicating, and
  - organizing yourself
- 
- work together
  - version management (other/better tools)

**Explicit** and  
concrete  
communication!

- Quality
  - Management
  - Testing
  - Reviews
- Many practical issues on programming and solving problems

JavaCC  
jar file  
Eclipse  
Java  
Jersey  
EMF  
debugger  
Derby  
Plugin persistence  
DTO/Entities  
JPA  
class path

- Integration and extension
  - Integrating features in existing software (PlugIn Mechanisms, ...)
  - Developing parts in parallel (based on a common model)
  - Separating concerns
  - **Stepwise extension** (prototyping, agile)
  - ...



- Software Specifications (incl. writing)
- Modelling & Meta modelling
- Quality mangament (incl. testing)
- Code generation
- Working together
- Management

The main point of this course is NOT on the acquired knowledge!

It is on **APPLYING** it (in a meaningful way): acquiring **SKILLS!**

## 2. What did you not learn?



Concerning (Android, JPA, DB, RESTfull, ...) technology and complexity of real software, you have just seen the tip of the iceberg!

- MOF (Meta Object Facility)
- Software without Programming (EMF and more) / code generation technologies
- Other technology: other application servers, databases, service technologies
- Analyse, validate, verify these models, ...
- Other programming and modelling paradigms: e.g. Aspect oriented Modelling
- ...



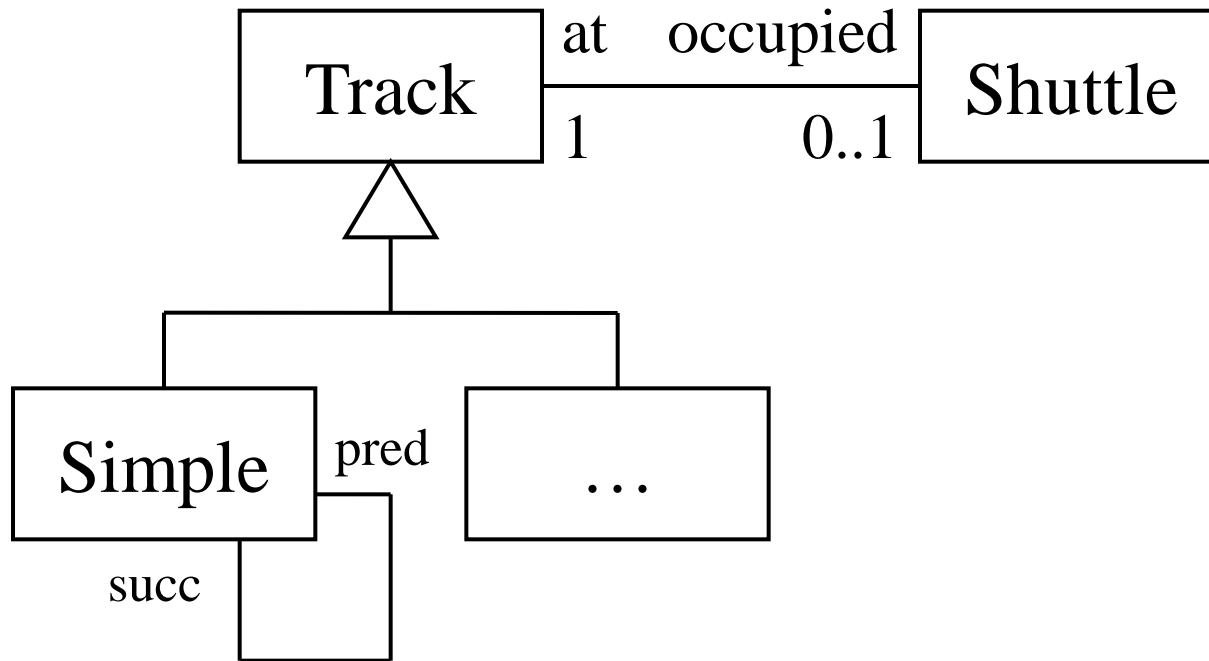
---

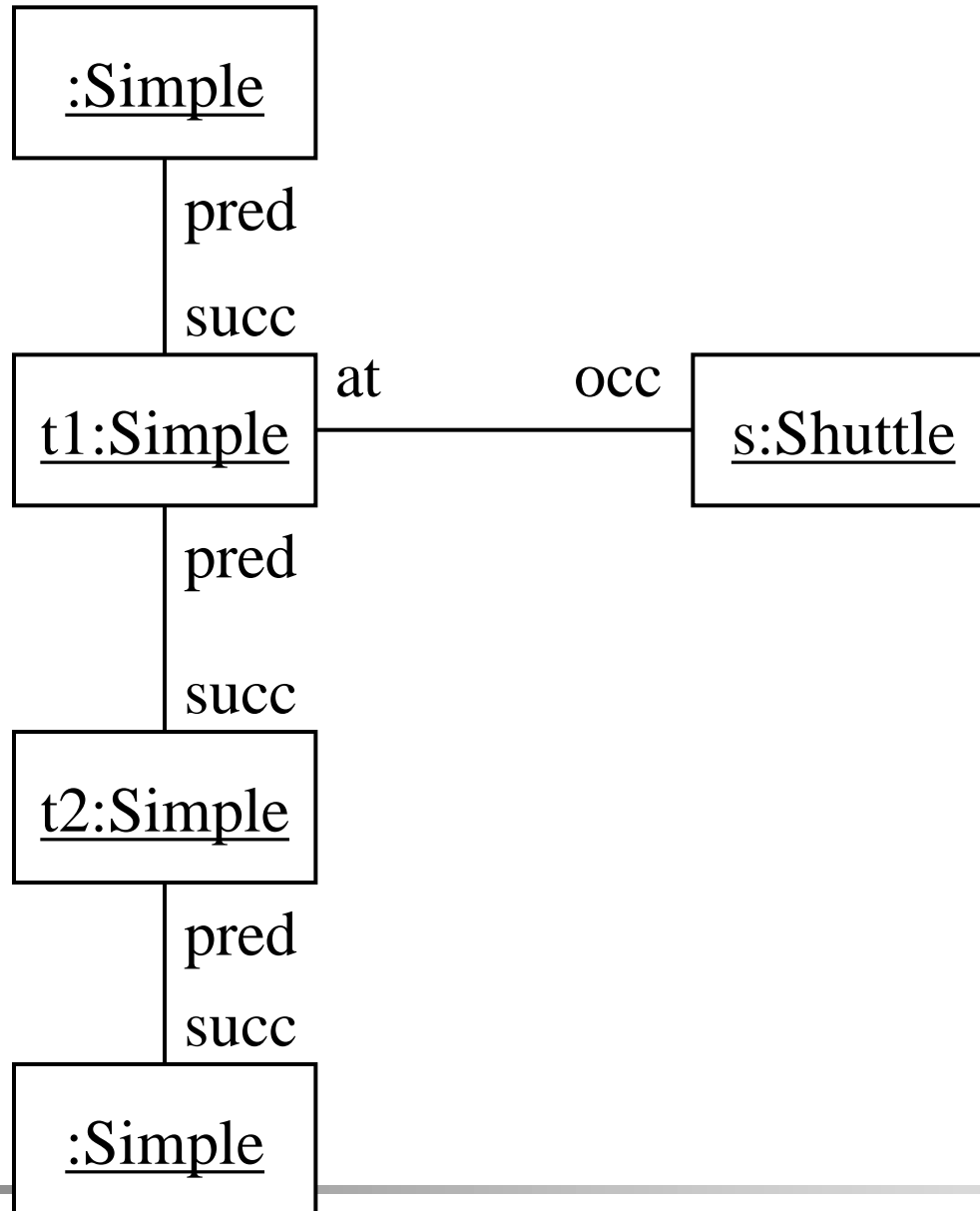
# Advanced Software Engineering

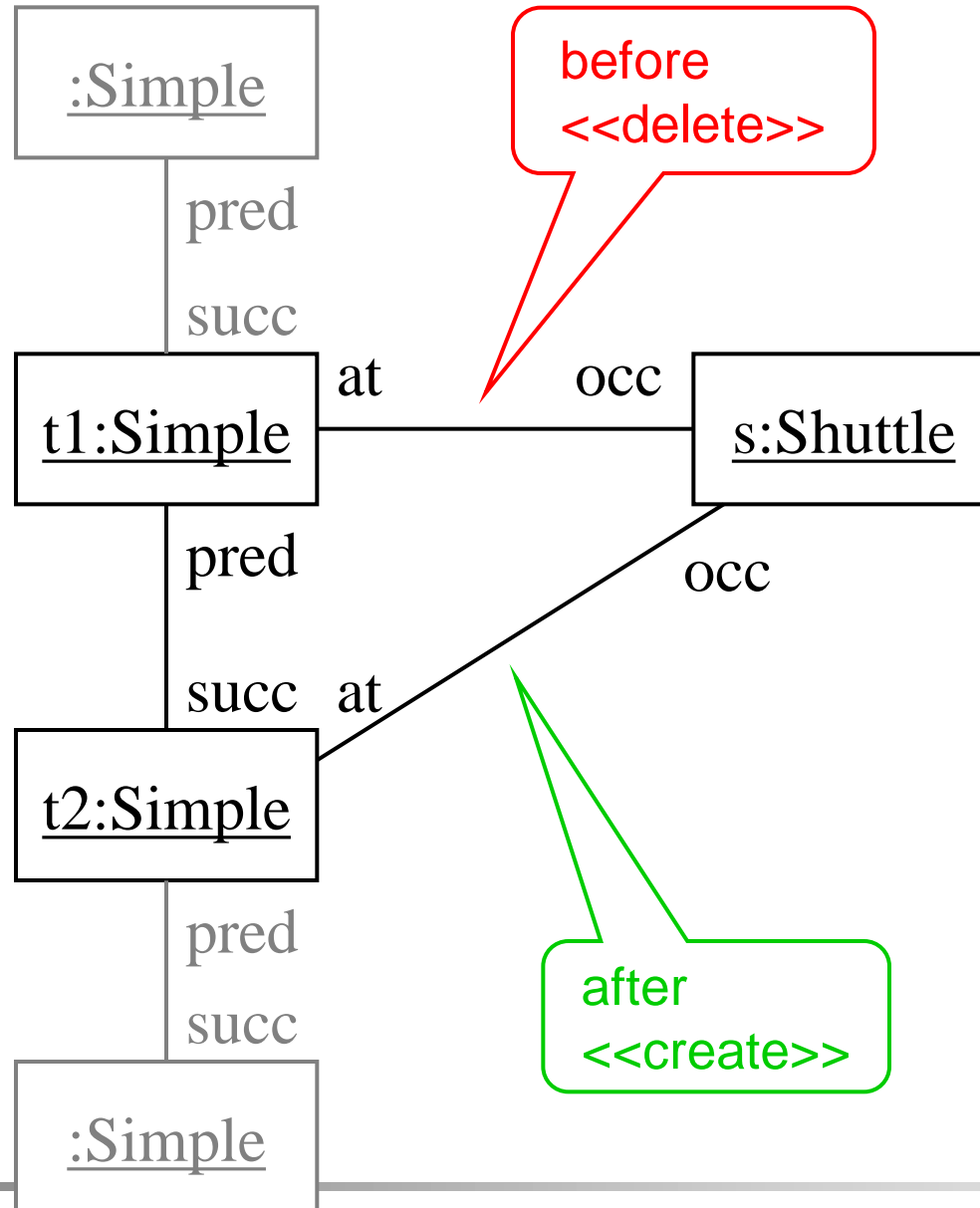
Just to give you some idea

---

- Modelling dynamic behaviour  
(and generating code from that)
- Defining transformations
- Get completely rid of programming?!

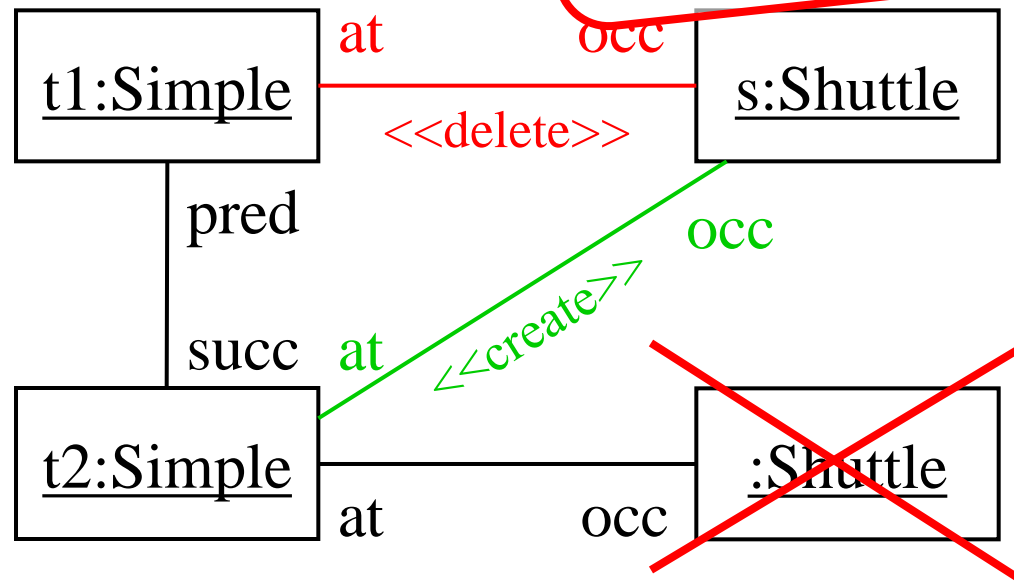








→ Story Diagrams:  
FUJABA



- Master courses
  - Systems integration (H.B.)
  - Web Services (H.B.)
  - Formal methods (A.H.)
  - Advanced topics in SE (E.K., f15)
  
- Bachelor and master projects  
(next week)



# Coordinating Interactions

## The Event Coordination Notation

DTU Compute

Department of Applied Mathematics and Computer Science

**Ekkart Kindler: Coordinating Interactions:  
The Event Coordination Notation.  
DTU Compute Technical Report 2014-05,  
May 2014.**

**ECNO home page:**

**<http://www2.imm.dtu.dk/~ekki/projects/ECNO/>**

# From lecture 1: Example

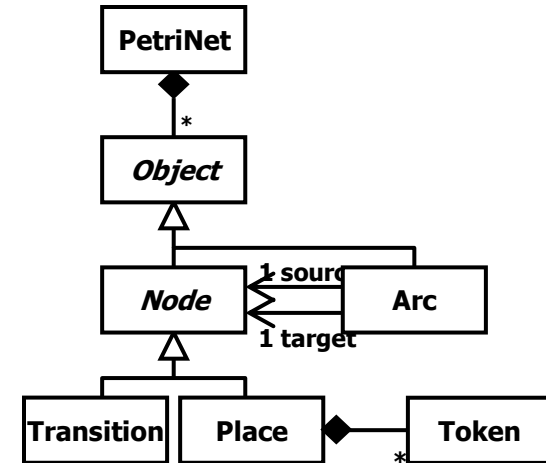
The screenshot displays the APetriNetEditor interface within the Eclipse SDK. The main workspace shows a Petri net diagram with two transitions, t1 and t2, and two places. Transition t1 is a square, and transition t2 is a square with a dashed border. Place 1 is a circle containing two tokens, and Place 2 is an empty circle. The diagram is connected as follows: t1 → Place 1, Place 1 → t2, t2 → Place 2, and Place 2 → t1.

The Properties view at the bottom right shows the properties for the selected transition t2:

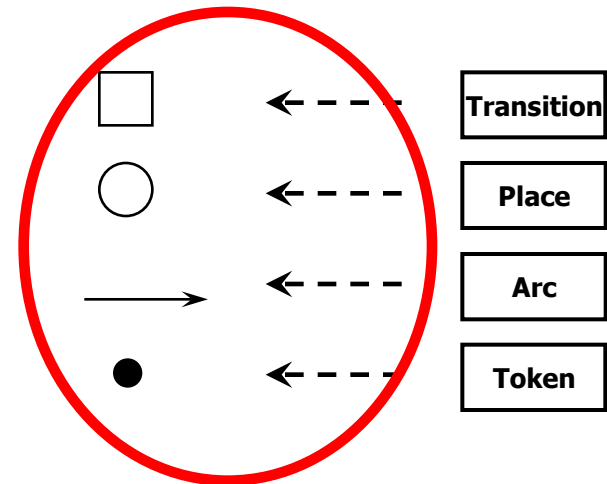
Core	Property	Value
Appearance	Name	t2

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)

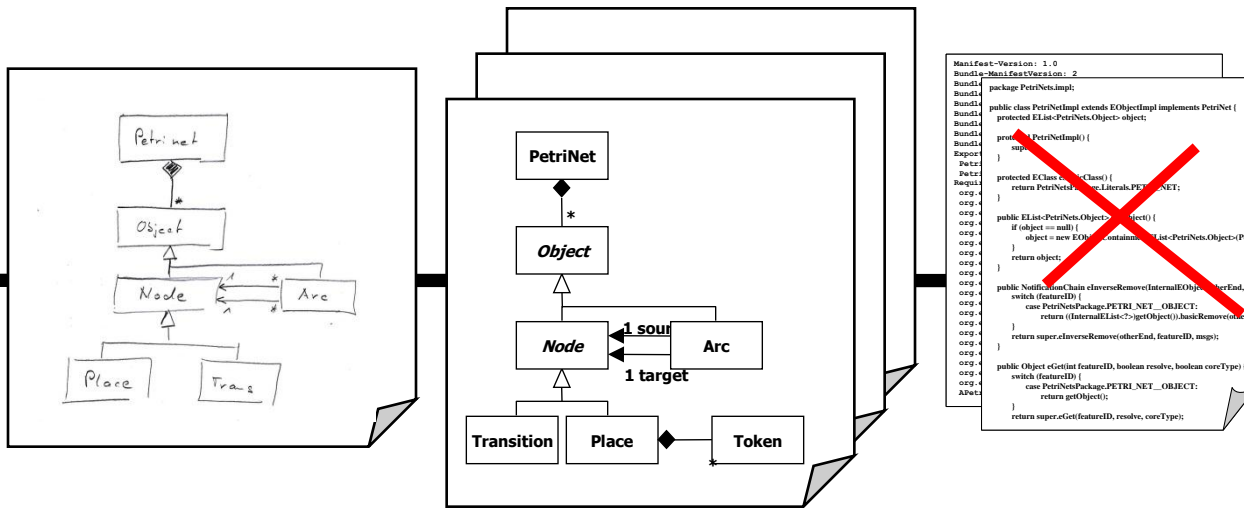


With this and some more GMF information:  
Generation of the Java code of a graphical complete editor (with many fancy features). No programming at all.



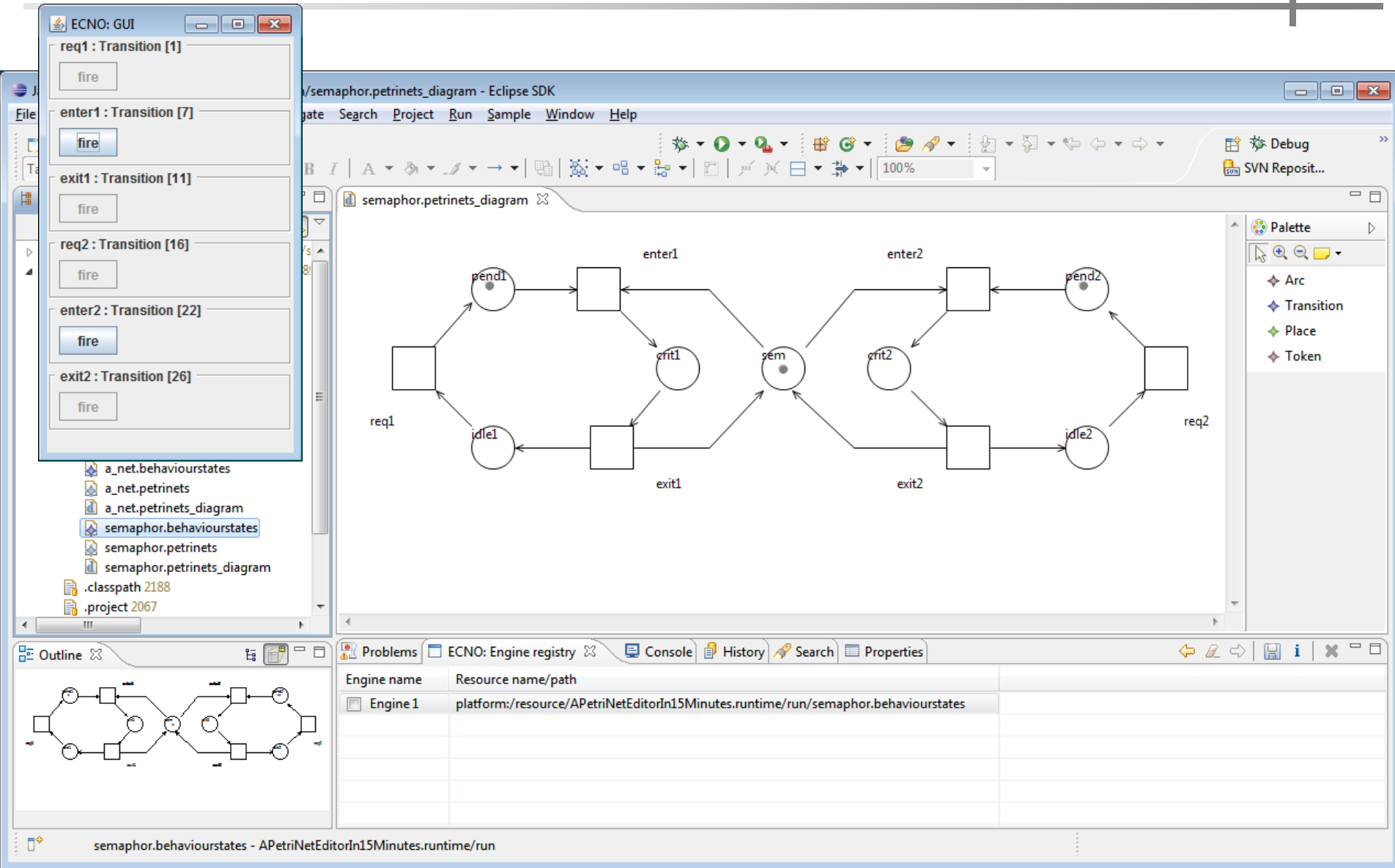
Almost all you need to say about a Petri net editor.

How about behaviour?  
(non-standard behaviour)



Analysis  
Design  
Implementation  
~~Coding~~

# e.g. a Petri net simulator?



The screenshot displays the ECNO GUI (Eclipse-based Petri Net Simulator) interface. The main window shows a Petri net diagram with the following components:

- Places:** req1, idle1, pend1, sem, pend2, idle2, req2.
- Transitions:** enter1, enter2, exit1, exit2.
- Initial State:** A token is present in the 'sem' place.

The diagram illustrates a mutual exclusion protocol for two processes (1 and 2). Each process has a request place (req1, req2), an idle place (idle1, idle2), and a pending place (pend1, pend2). Transitions enter1 and enter2 allow a process to enter its critical section (moving a token from req to pend). Transitions exit1 and exit2 allow a process to leave its critical section (moving a token from pend to idle). The central 'sem' place acts as a semaphore, ensuring mutual exclusion by allowing only one process to be in its critical section at a time.

The GUI includes a palette on the right with elements: Arc, Transition, Place, and Token. The bottom panel shows the 'ECNO: Engine registry' table:

Engine name	Resource name/path
<input type="checkbox"/> Engine 1	platform:/resource/APetriNetEditorIn15Minutes.runtime/run/semaphor.behaviourstates

The status bar at the bottom indicates the current engine: `semaphor.behaviourstates - APetriNetEditorIn15Minutes.runtime/run`.

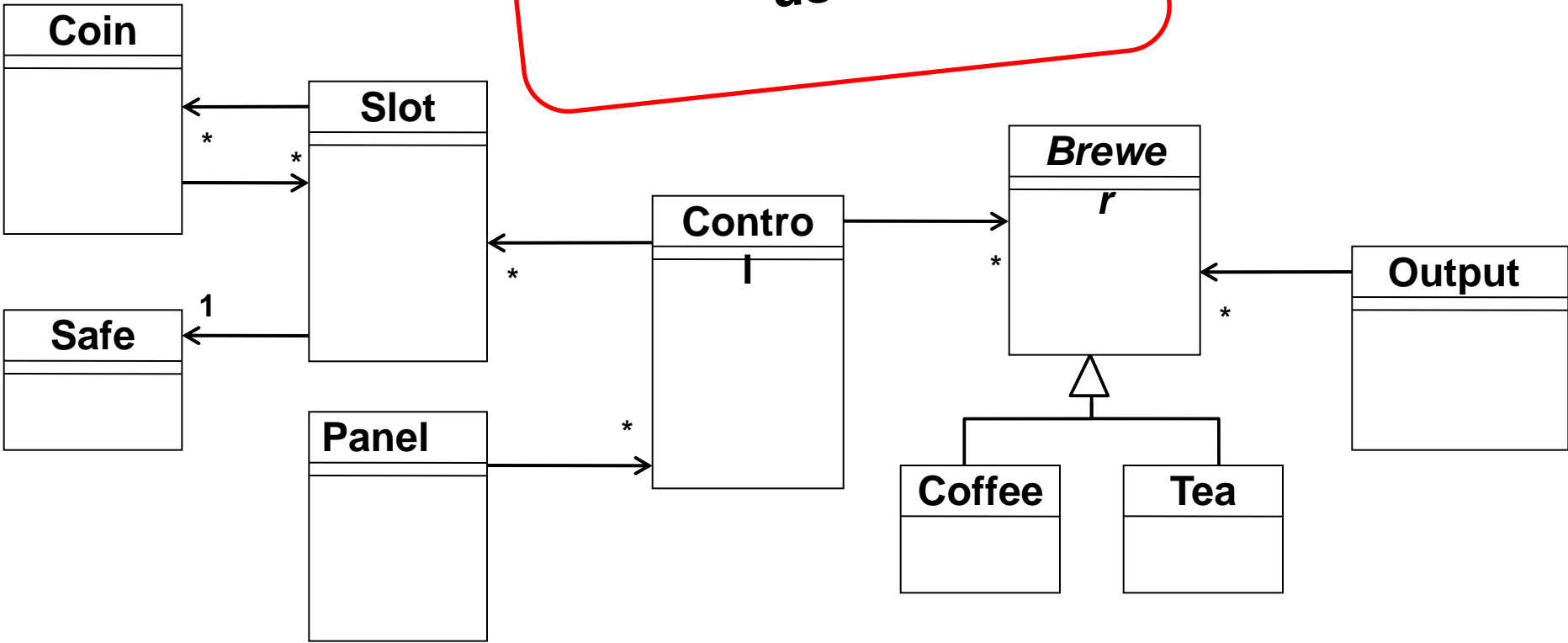
# Motivation

- 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)



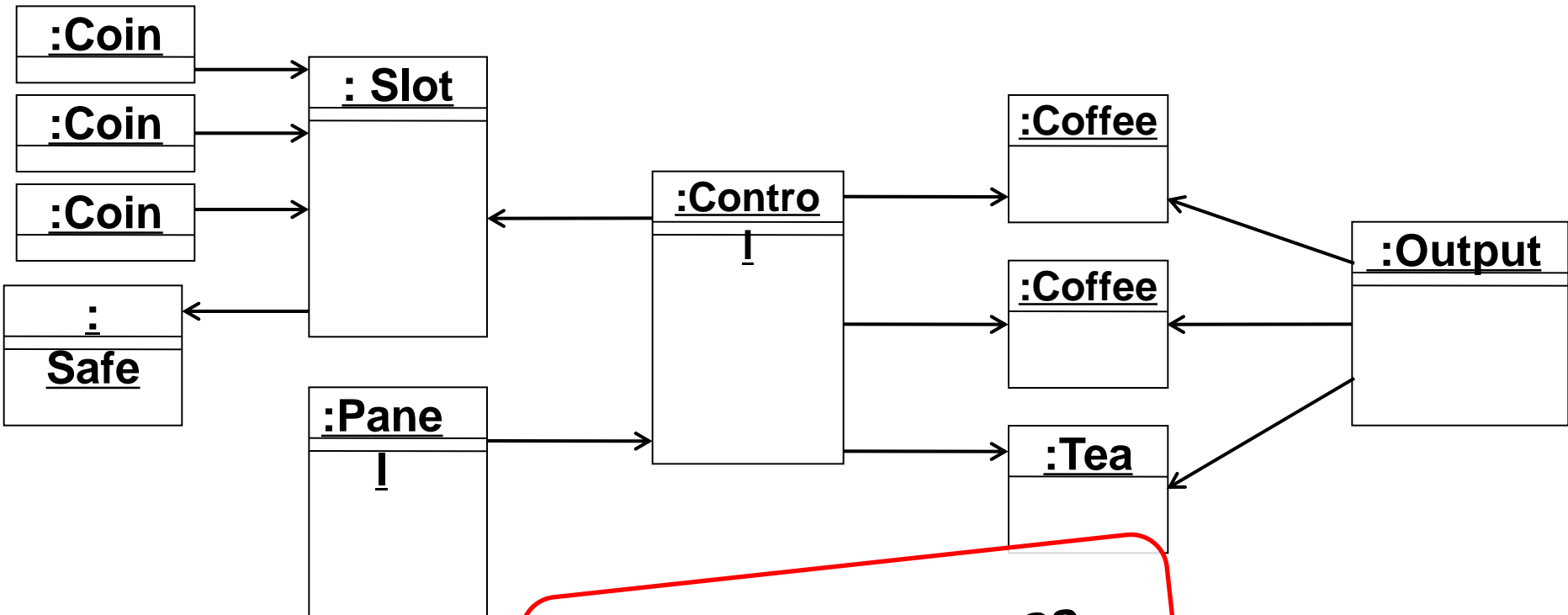
# 2.1 Example: Vending machine

Class diagram as usual



# Instance: Object Diagram

Initial configuration,  
current situation

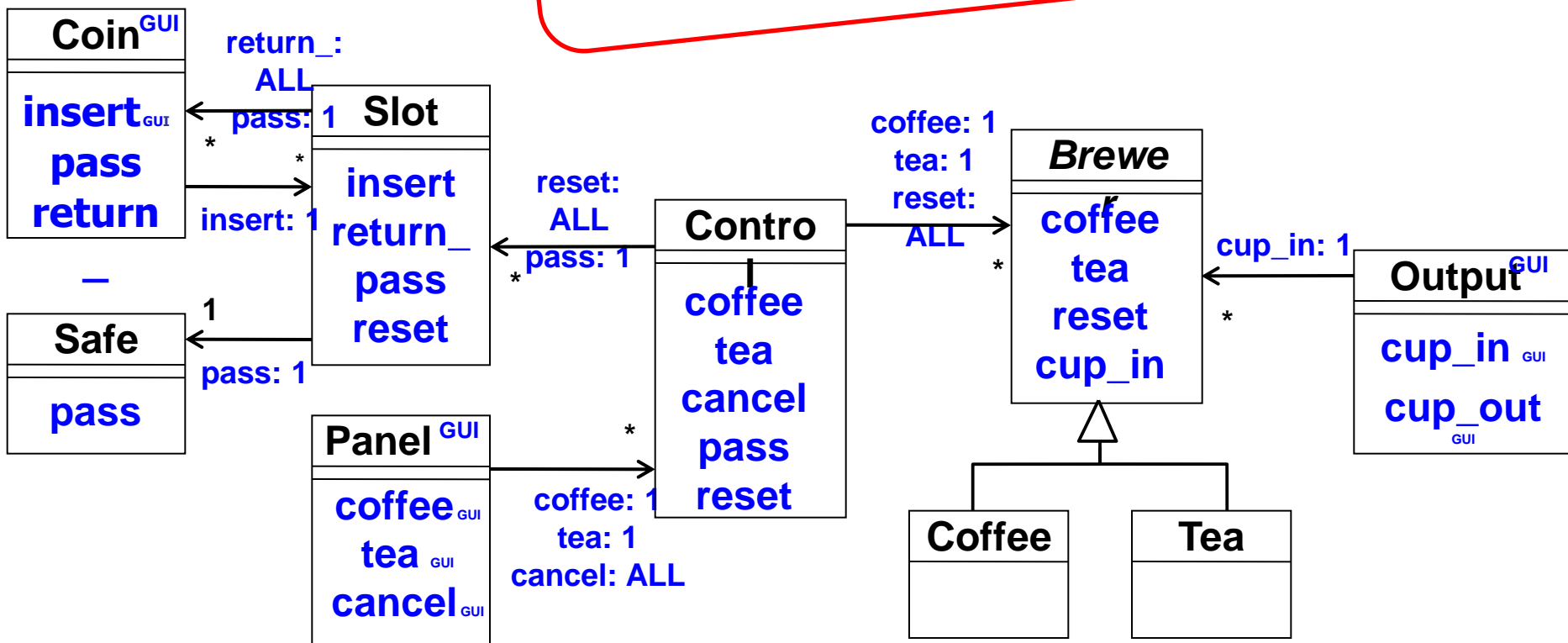


Object diagram as  
usual

# Coordination Diagram

▪ We call objects elements now!

▪ Events (event types)  
▪ Coordination annotations:  
event type + quantification  
annotation



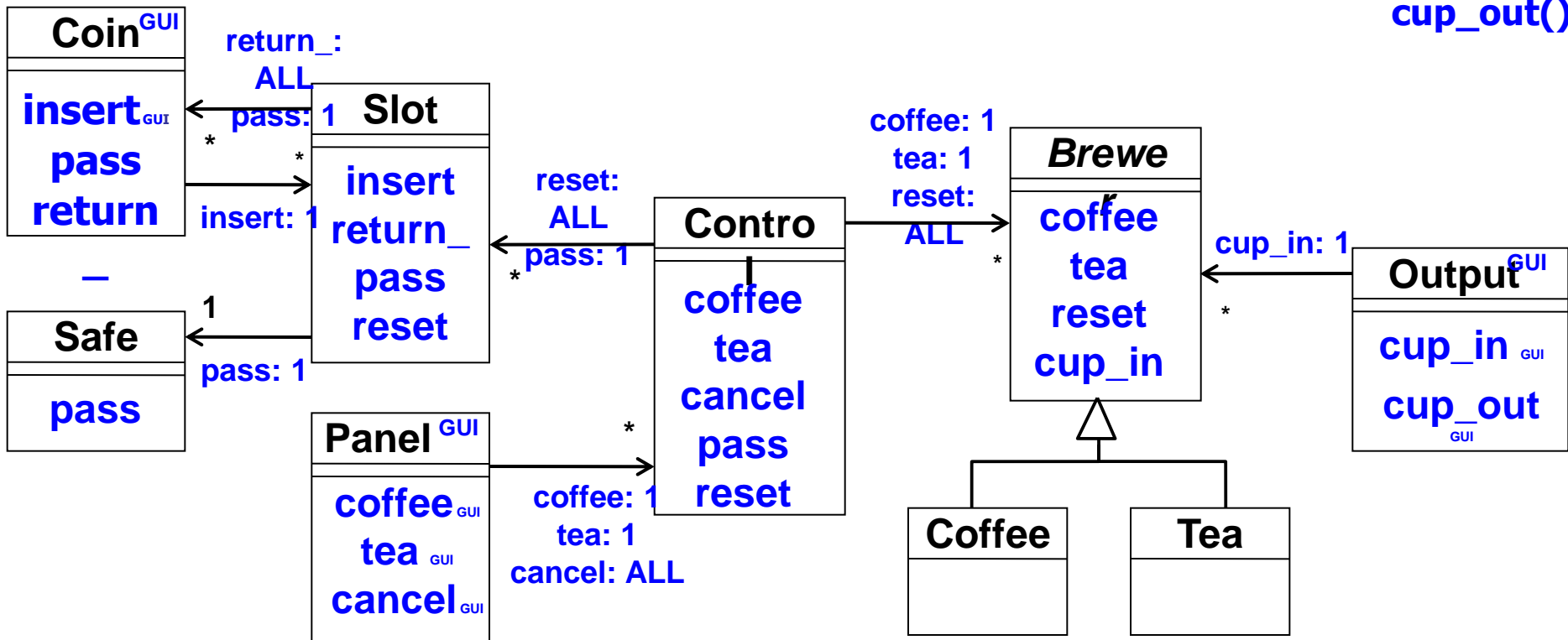
# ... + Event declaration

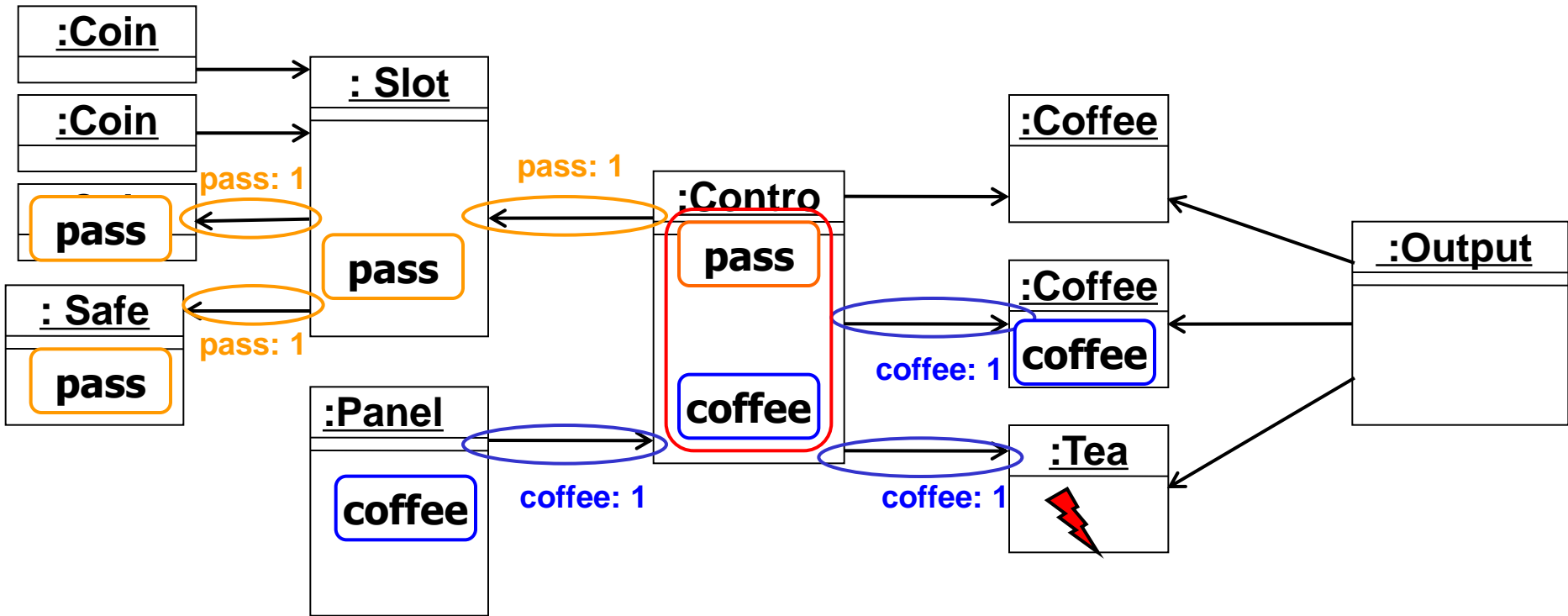
- Event (type) declaration
- Parameters

```
insert(Coin coin, Slot slot)
pass(Coin coin, Slot slot)
return(Slot slot)
reset_()
```

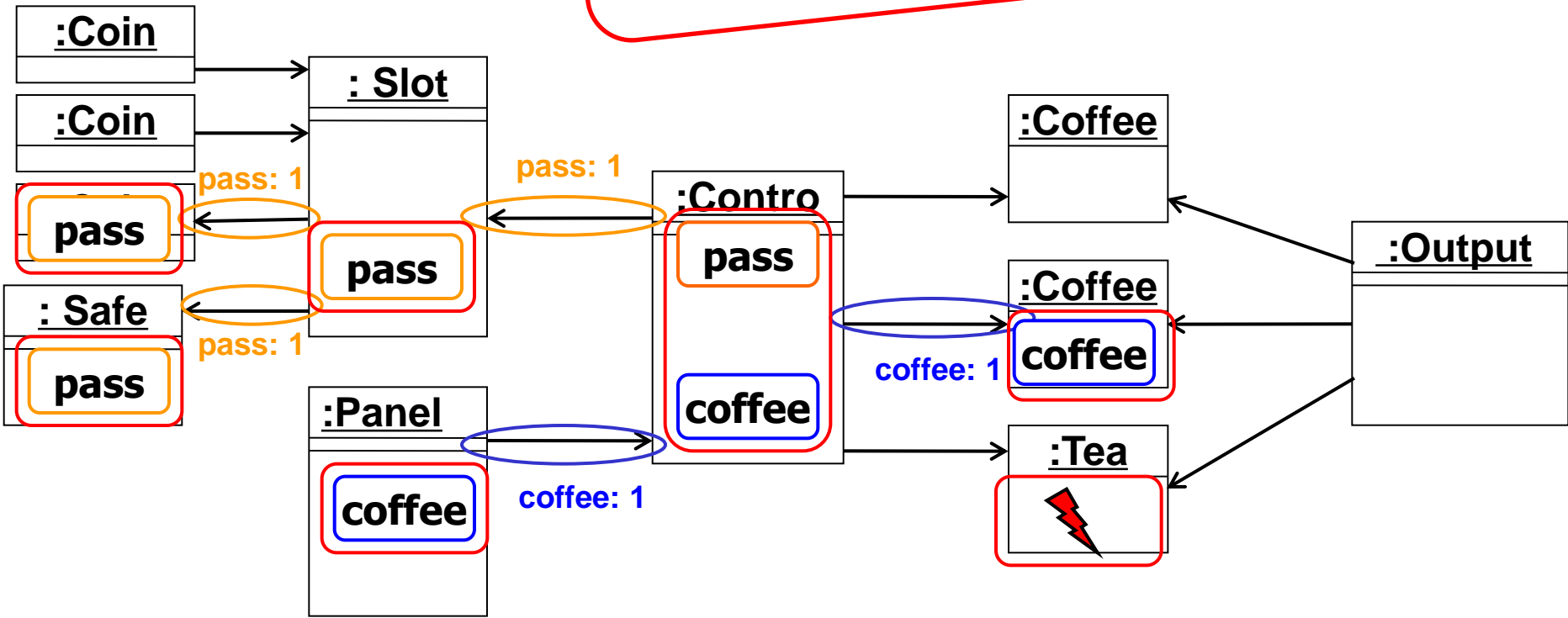
```
coffee()
tea()
cancel()
```

```
cup_in()
cup_out()
```

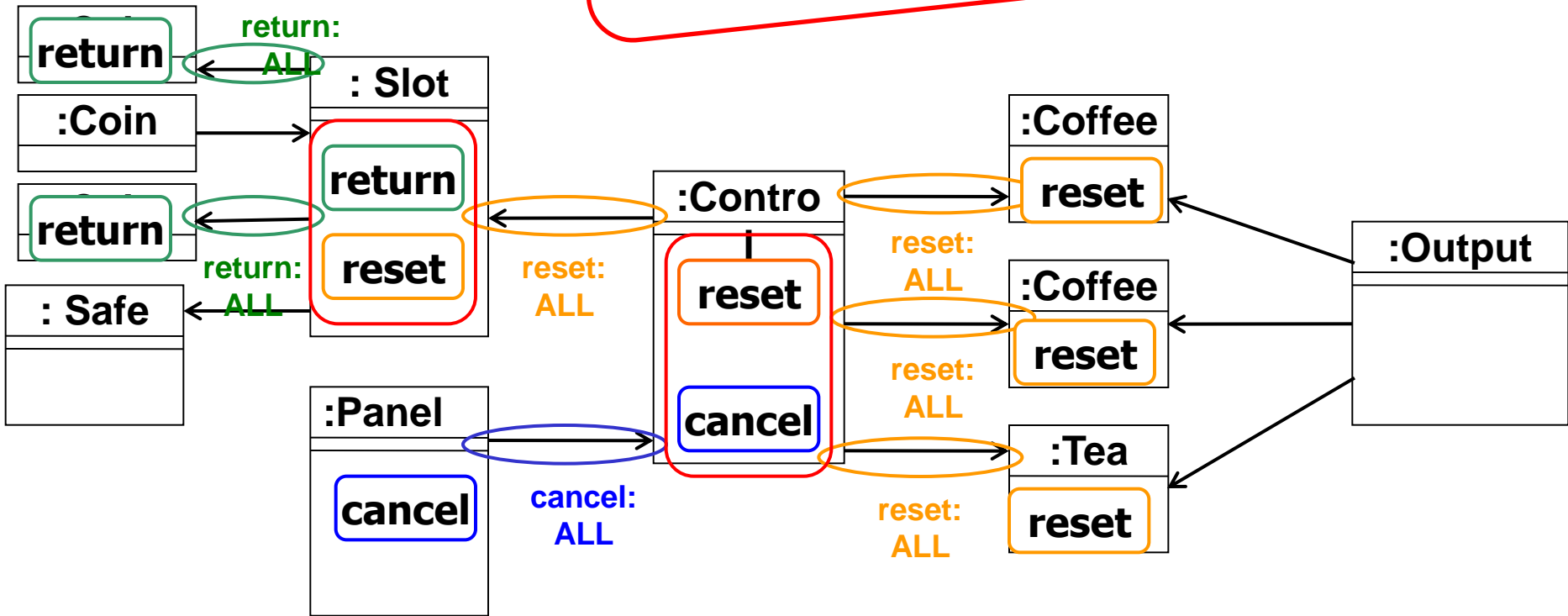




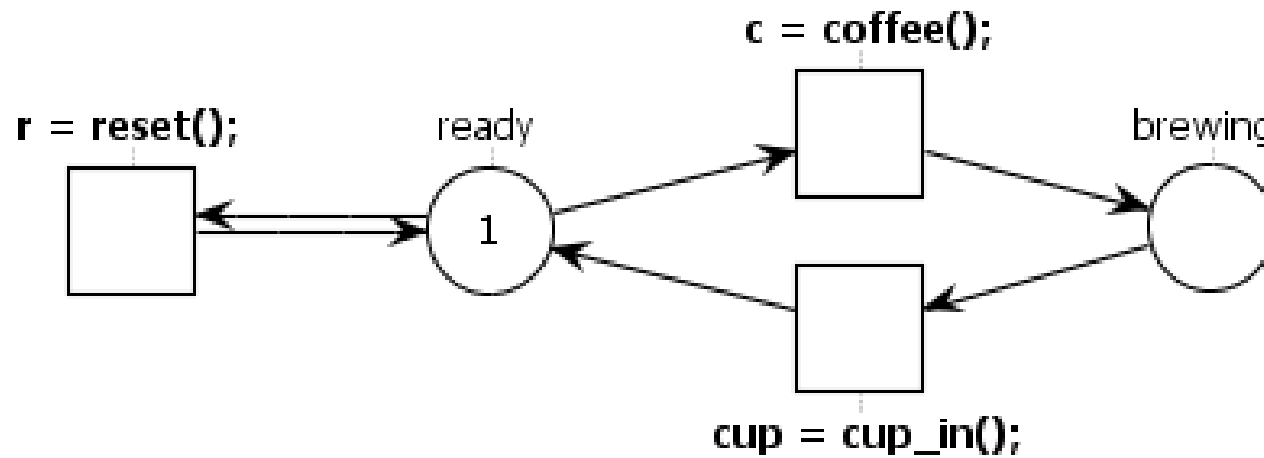
Interaction =  
local behavior +  
coordination



**Interaction =  
local behavior +  
coordination**



**Event binding**

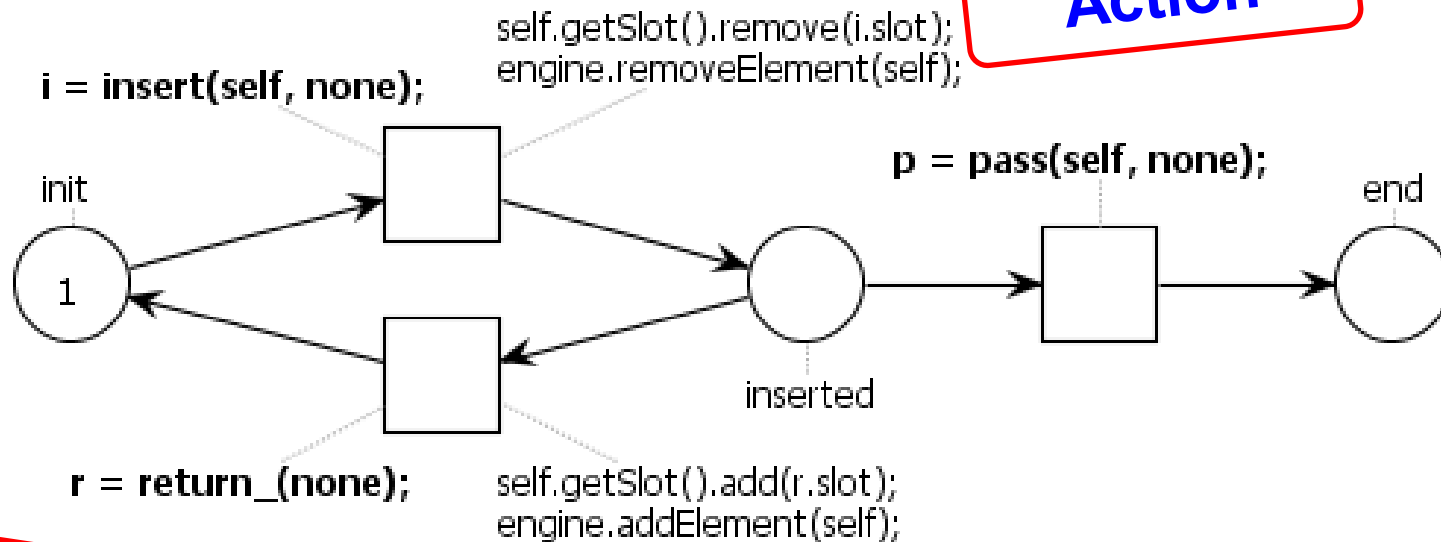




# Local behaviour: Coin

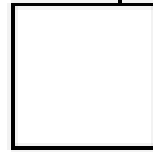
```
import dk.dtu.imm.se.ecno.engine.ExecutionEngine;
```

```
final ExecutionEngine engine = ExecutionEngine.getInstance();
```

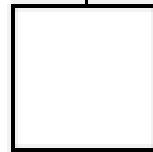


- Event binding
- Parameter assignment

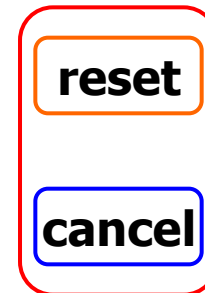
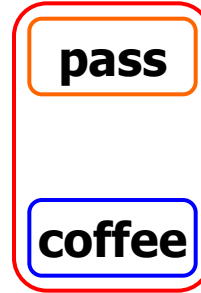
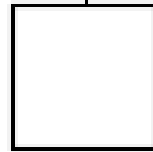
`p = pass(none,none); c = coffee();`



`p = pass(none,none); t = tea();`



`c = cancel(); r = reset();`

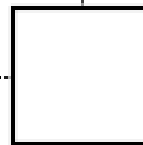


▪ Event binding with multiple event types!

**Condition**

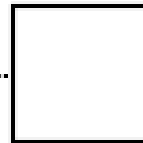
`self.getCoin().size() < 2`

`i = insert(none, self);`



`self.getCoin().add(i.coin);`

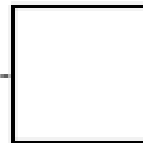
`p = pass(none, self);`



`self.getCoin().remove(p.coin);`

`res = reset();`

`r = return_(self);`

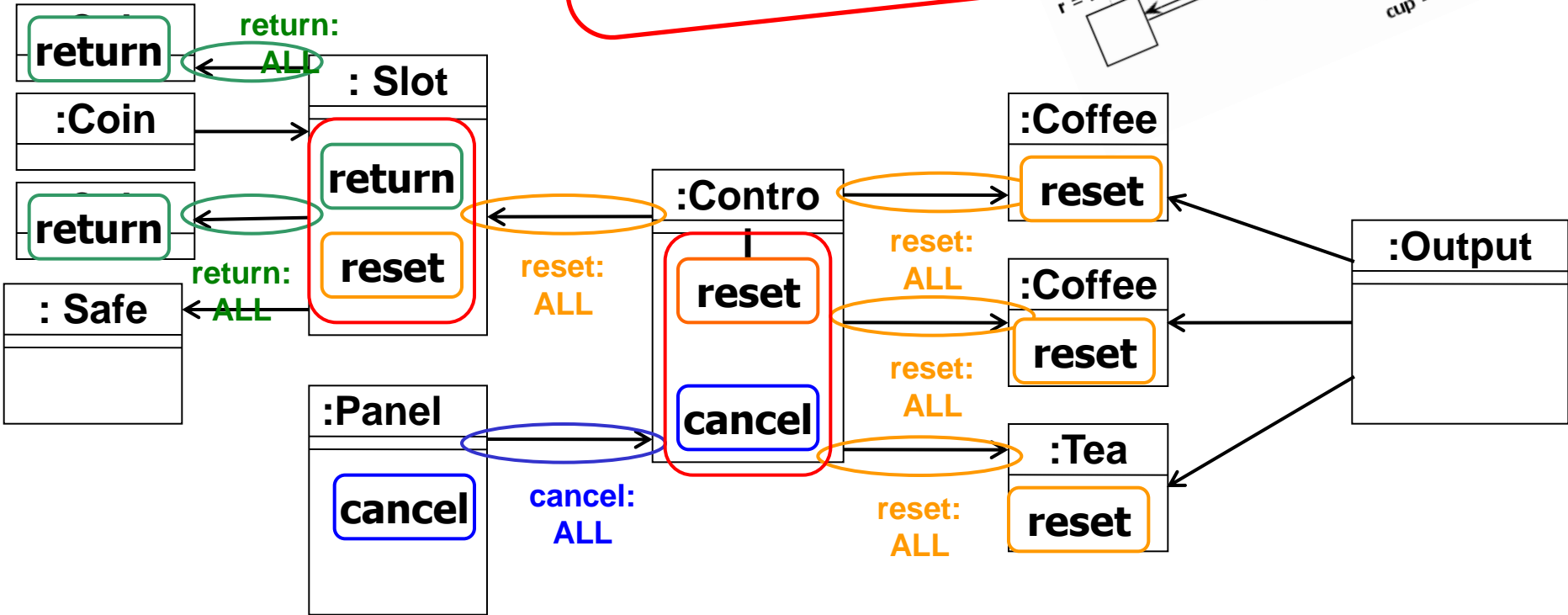
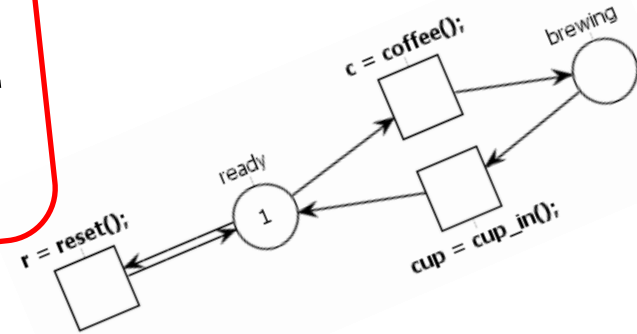


`self.getCoin().clear();`

**return**

**reset**

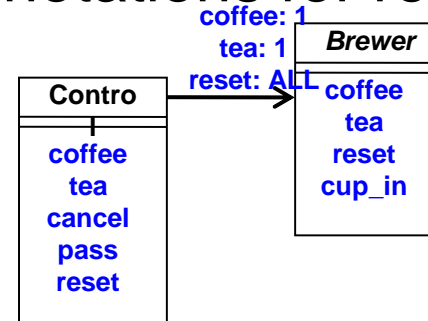
**Interaction =  
 local behavior +  
 coordination**



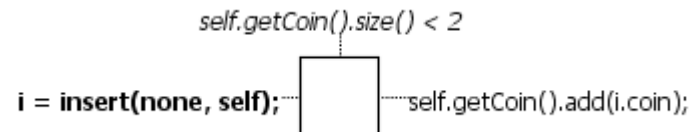
- ElementTypes (Classes)
- EventTypes with
  - parameters

**insert(Coin coin, Slot slot)**

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

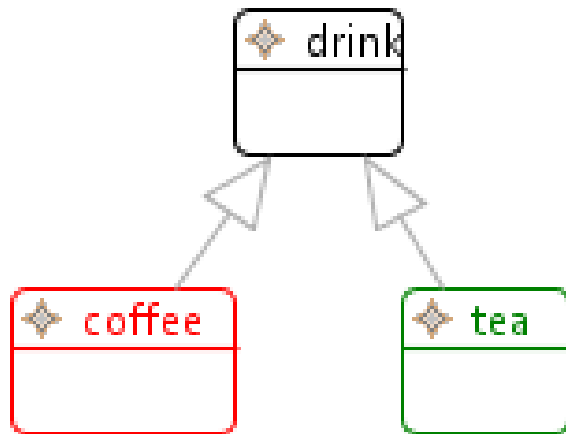


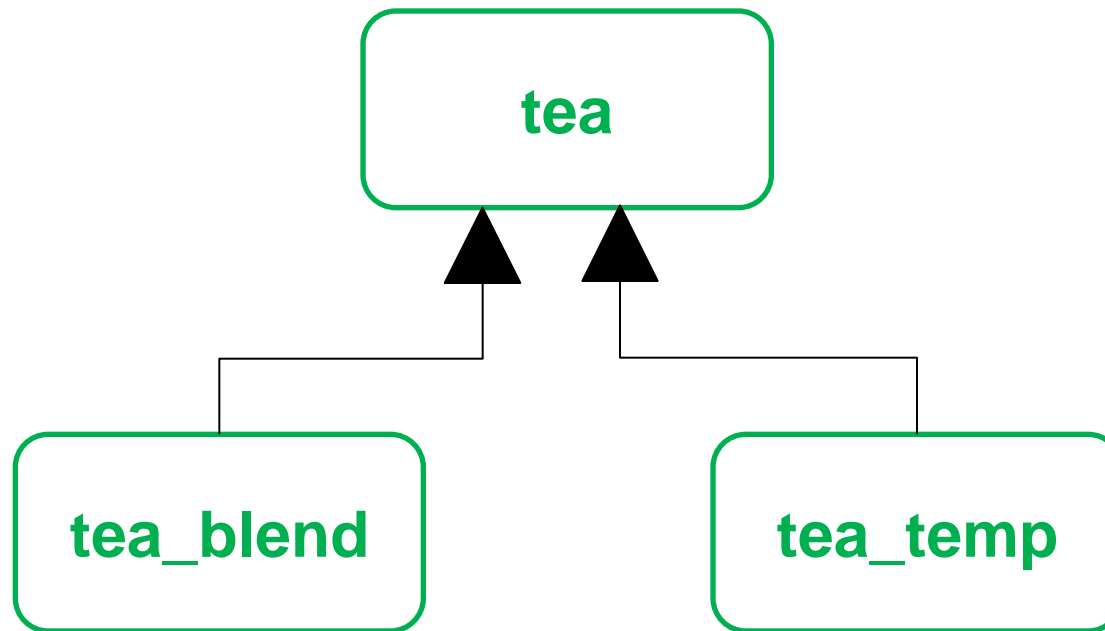
- Local behaviour: ECNO nets (or something else)
  - Event binding (with parameter assignment)
  - Condition
  - Action



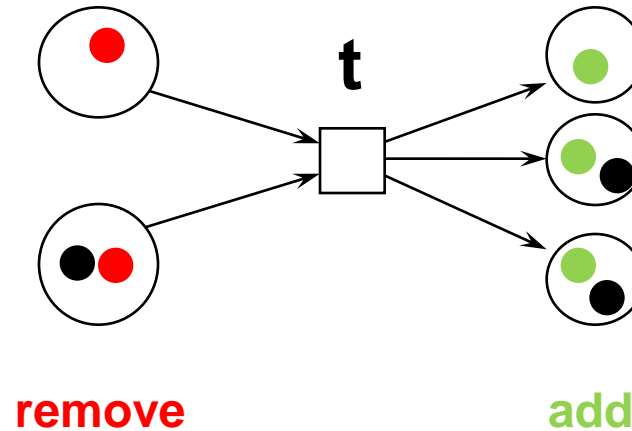
ECNO with its basic concepts has some limitations, which makes modelling things **in an adequate way** a bit painful.

- Sometimes, we want to extend event types later





**=> Two forms of inheritance  
on event types!**



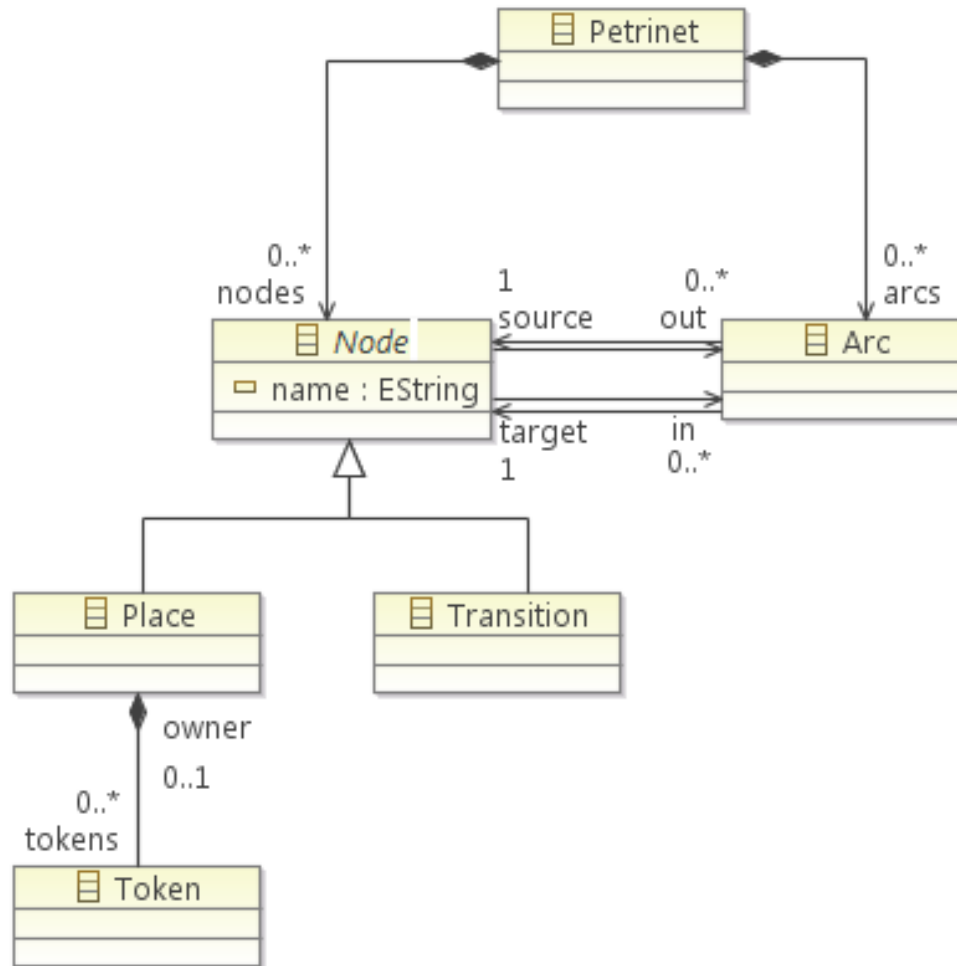
How can we model that  
behaviour in ECNO  
nets?

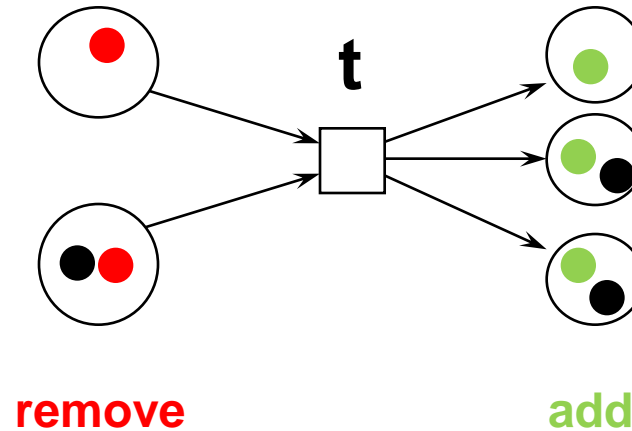
**Transition t enabled:**  
for ALL incoming Arcs a:  
for ONE source Place p of Arc a:  
find a token

**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



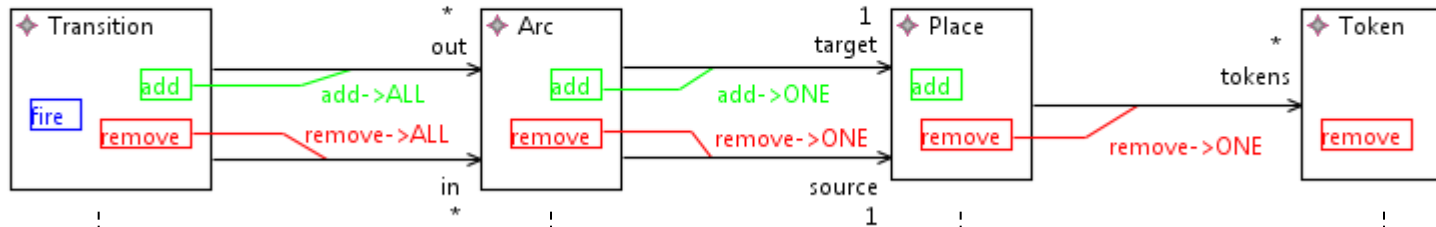




**Transition  $t$  enabled:**  
for ALL incoming Arcs  $a$ :  
for ONE source Place  $p$  of Arc  $a$ :  
find a token

**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



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

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

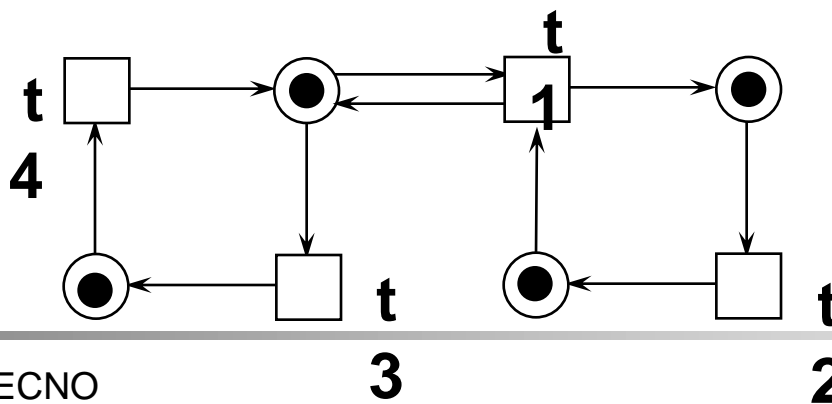
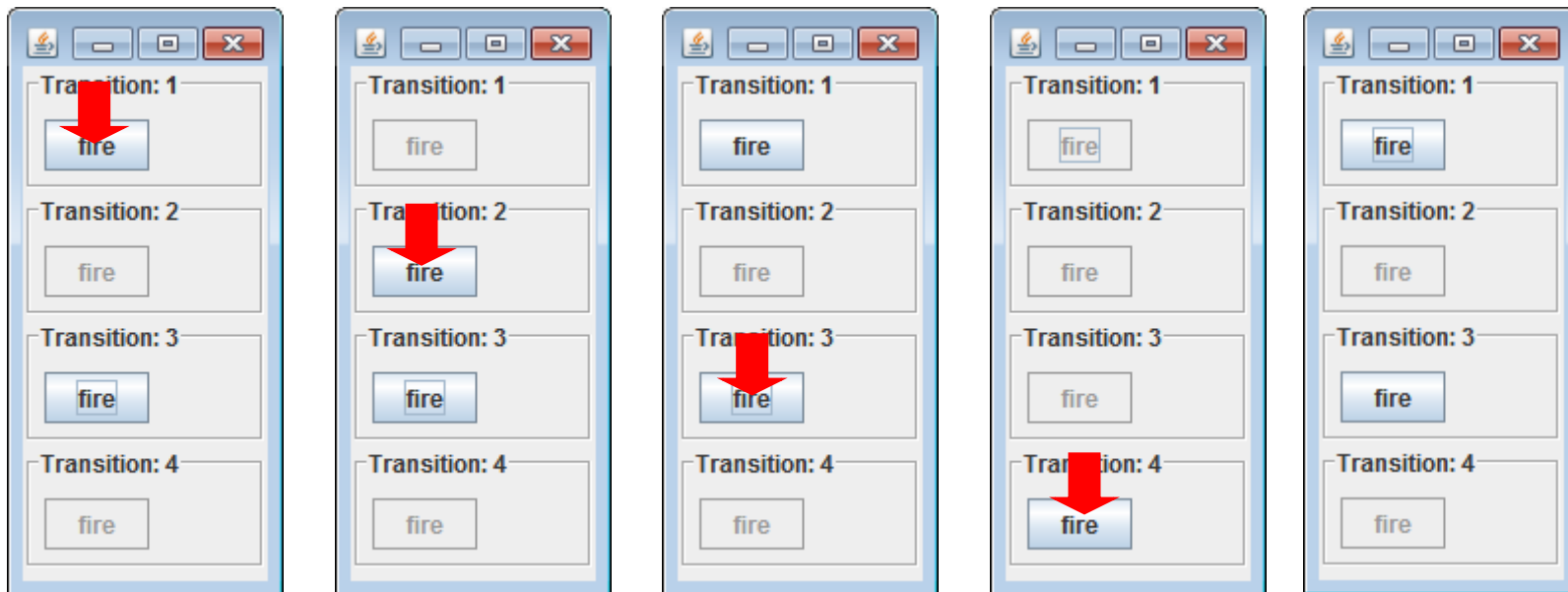
```
r = remove(); self.setOwner(null);
```

```
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

The screenshot displays the ECNO GUI for a Petri net simulator. The main window shows a Petri net diagram for a semaphore. The diagram includes a central place labeled 'sem' containing one token. It is connected to two critical sections, each with its own semaphore. The left critical section has transitions 'enter1' and 'exit1', and places 'pend1' and 'idle1'. The right critical section has transitions 'enter2' and 'exit2', and places 'pend2' and 'idle2'. The diagram is rendered in the Eclipse IDE.

On the left, a panel titled 'ECNO: GUI' lists several transitions, each with a 'fire' button:

- req1 : Transition [1]
- enter1 : Transition [7]
- exit1 : Transition [11]
- req2 : Transition [16]
- enter2 : Transition [22]
- exit2 : Transition [26]

The bottom of the interface shows an 'Outline' view with a small diagram of the Petri net and a 'Problems' view with the following table:

Engine name	Resource name/path
<input type="checkbox"/> Engine 1	platform:/resource/APetriNetEditorIn15Minutes.runtime/run/semaphor.behaviourstates

# 3. Conclusion

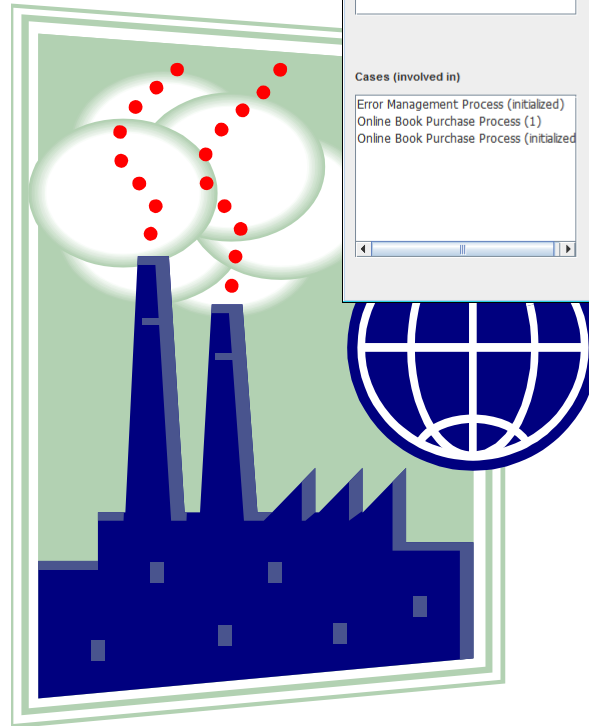
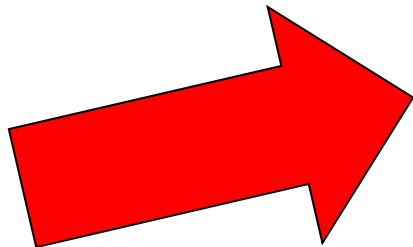
The screenshot shows the Eclipse IDE interface with several views:

- Package Explorer:** Shows the project structure for 'test.ecno.model' and 'VendingMachine'.
- UML Class Diagram:** Displays classes like Coin, Panel, Output, Safe, Slot, Control, Brewer, CoffeeBrewer, and TeaBrewer with their relationships.
- VendingMachineCoordination.ecno\_diagram:** Shows a state transition diagram with states like Coin, Slot, Control, Brewer, and Output, and transitions labeled with actions like 'pass', 'return', 'insert', 'reset', 'coffee', 'tea', 'cancel', 'cup\_in', and 'cup\_out'.
- VendingMachineBehavi Page: Control:** Contains code snippets for state transitions:

```
p = pass(none,none); c = coffee();  
p = pass(none,none); t = tea();  
c = cancel(); r = reset();
```
- Properties View:** Shows the 'ElementType' property for the 'Panel' class, with values like 'VendingMachineComponent', 'true', and 'Panel'.

**Release of version 0.3.2 (including examples):**  
<http://www2.imm.dtu.dk/~ekki/projects/ECNO/>

# Next Steps



Demonstrator for an ECNO based Workflow Engine

**Worklist Viewer**

Username: jack  
Password: \*\*  
Log in Log out

Debug

Logged in as: Jack (Customer)

Process Library	Inbox
Error Management Process Online Book Purchase Process	Start a new Error Management Process (1) Start a new Online Book Purchase Process (2)

Start

Cases (involved in)	Work in Progress
Error Management Process (initialized) Online Book Purchase Process (1) Online Book Purchase Process (initialized)	Place New Book Order -- in Online Book Purchase Process (1)

Finish Open...