Software Engineering 2

A practical course on software engineering

Ekkart Kindler

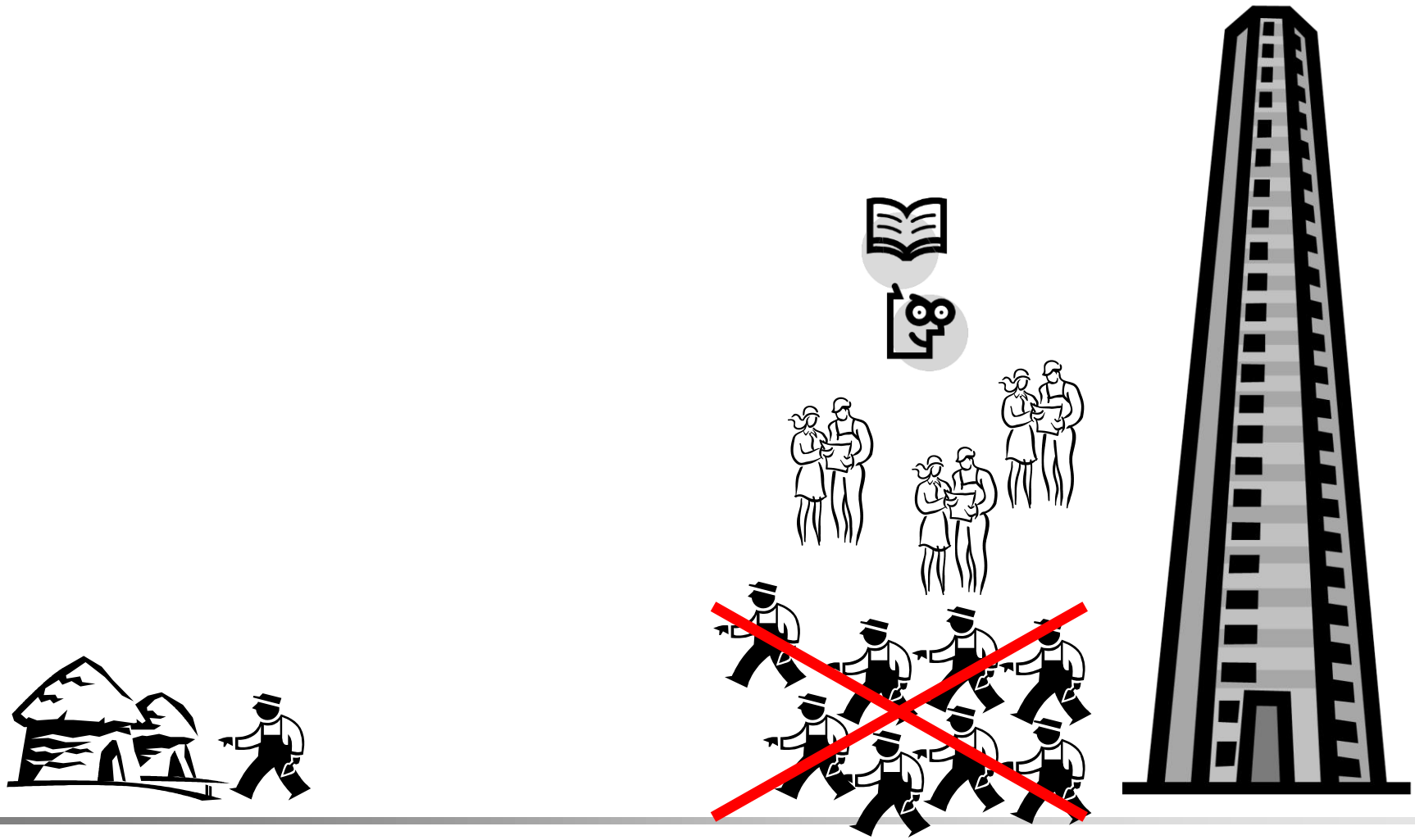
Technical University of Denmark

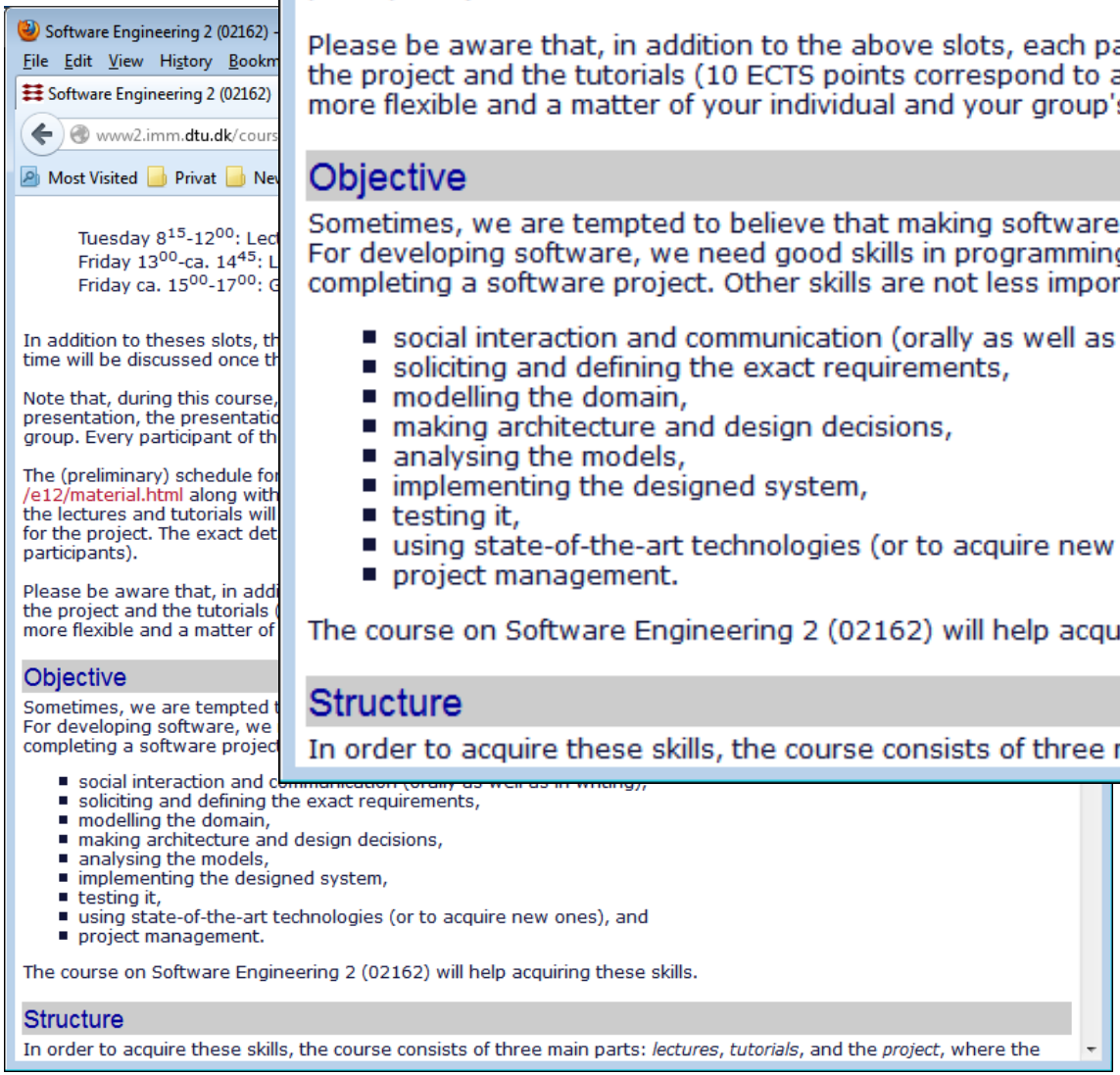
Informatics and Mathematical Modelling

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 wee the project and the tutorials (10 ECTS points correspond to an overall workload of 270 hours). This work, however, i 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 th

1. What did you learn?

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

What did you learn?

What is important?

Software Engineering is

… 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!

Experience

Software Engineering requires much experience!

This experience

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

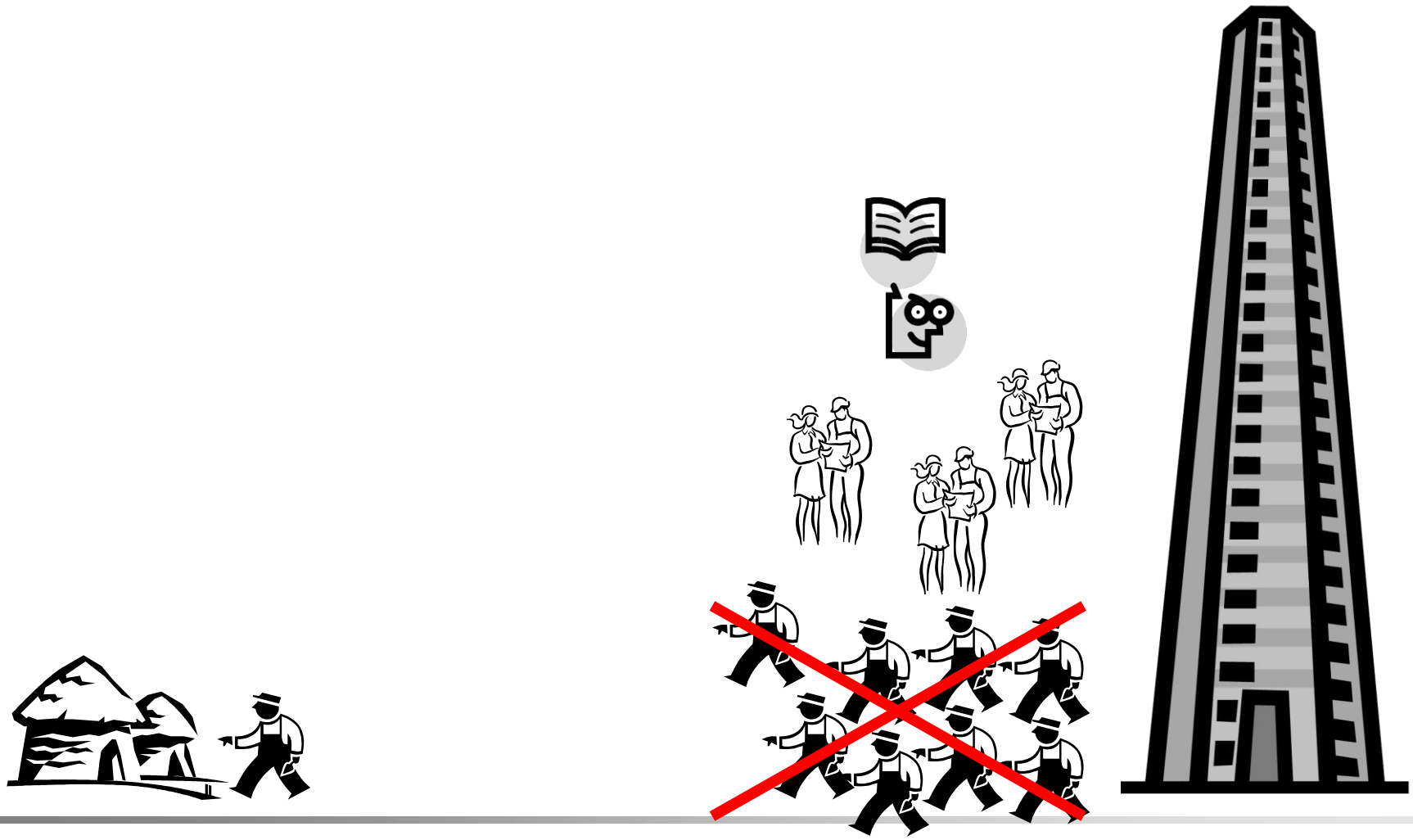
→ tutorial
→ project
→ and (only) backed by the lectures

Objective

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



Problems

- 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
- …

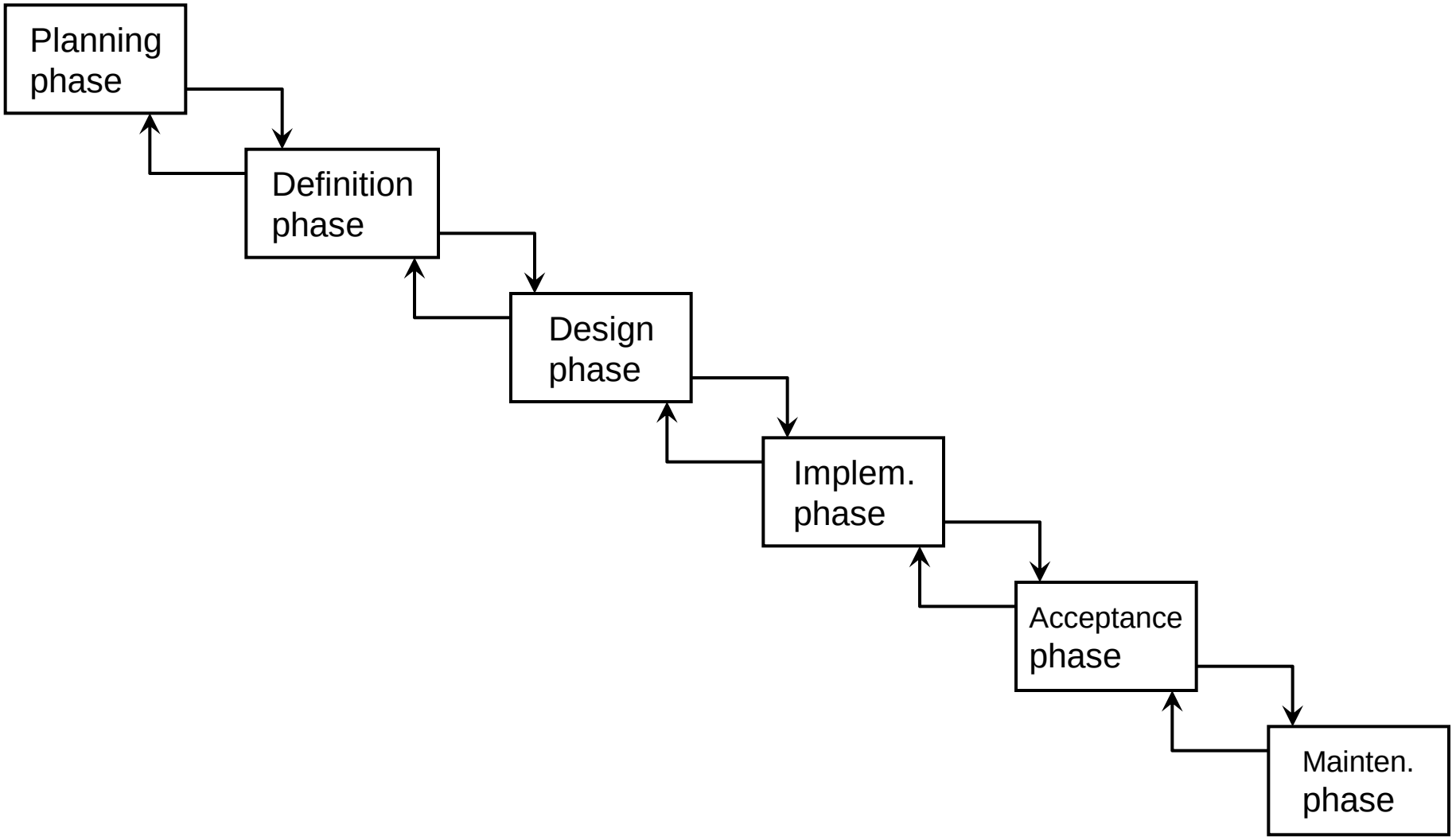
Explicit and concrete communication!

Software engineering

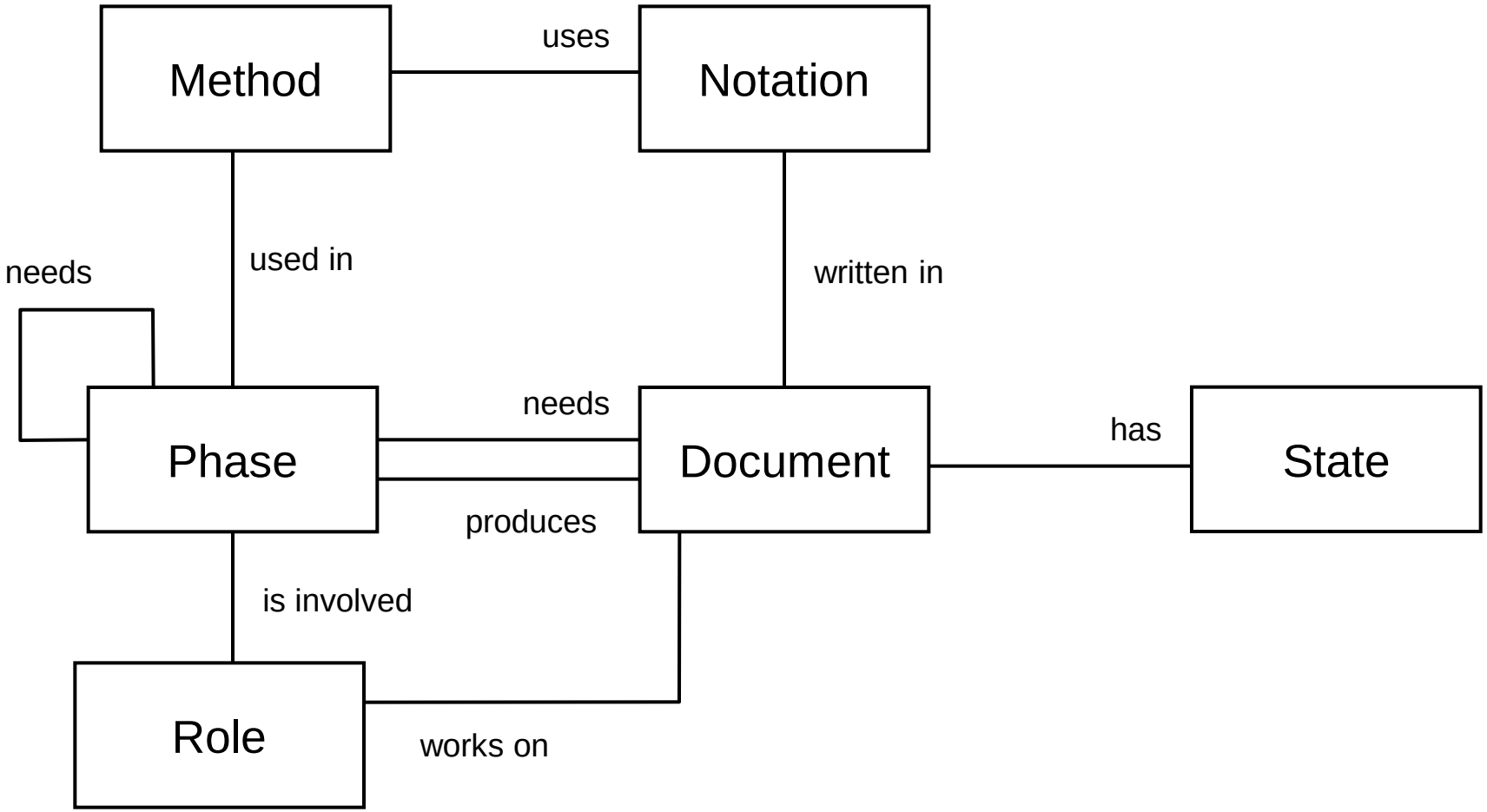
Branches:

- **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

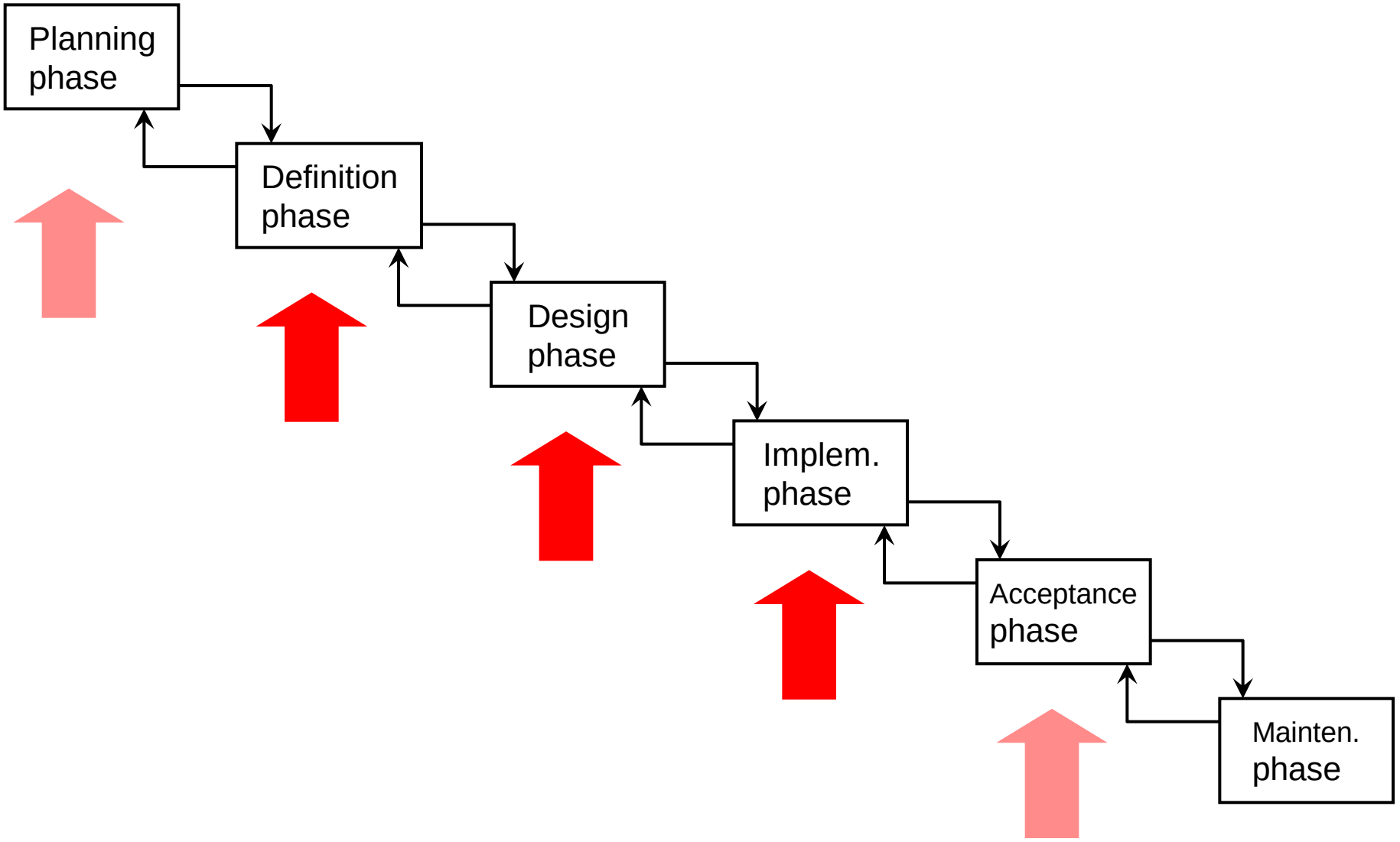
Process


Planning phase
Definition phase
Design phase
Implem. phase
Acceptance phase
Mainten. phase

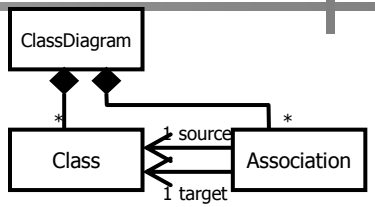
Process Meta model

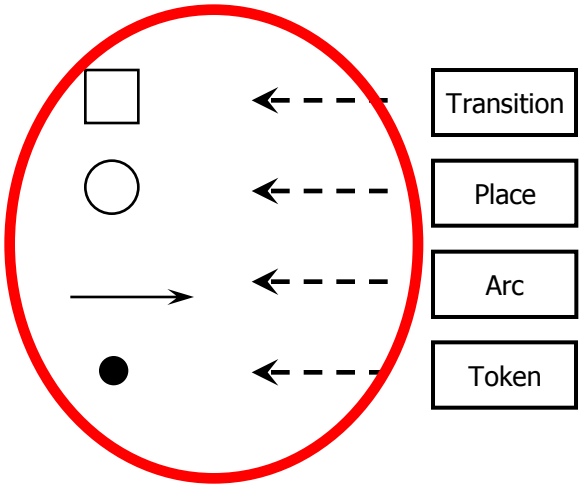


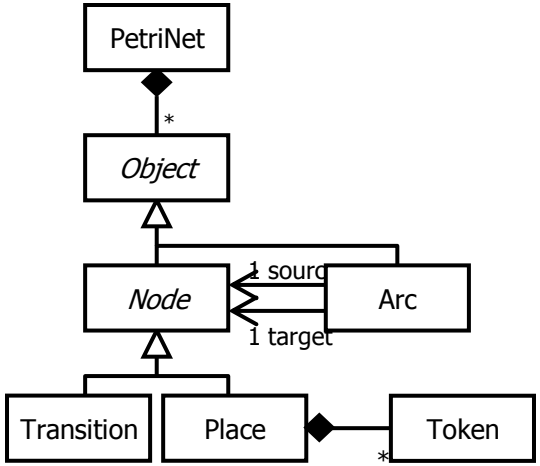
Process

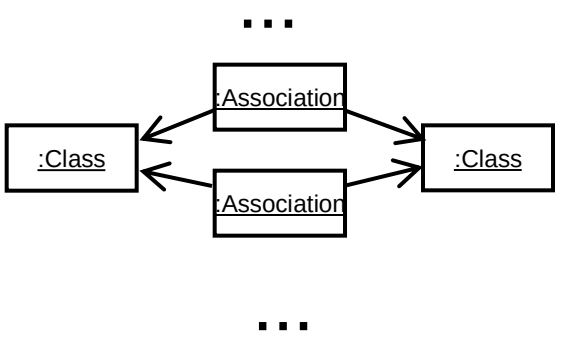

Planning phase
Definition phase
Design phase
Implem. phase
Acceptance phase
Mainten. phase

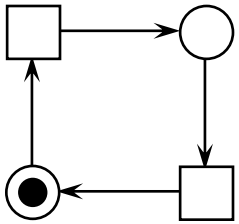
Models

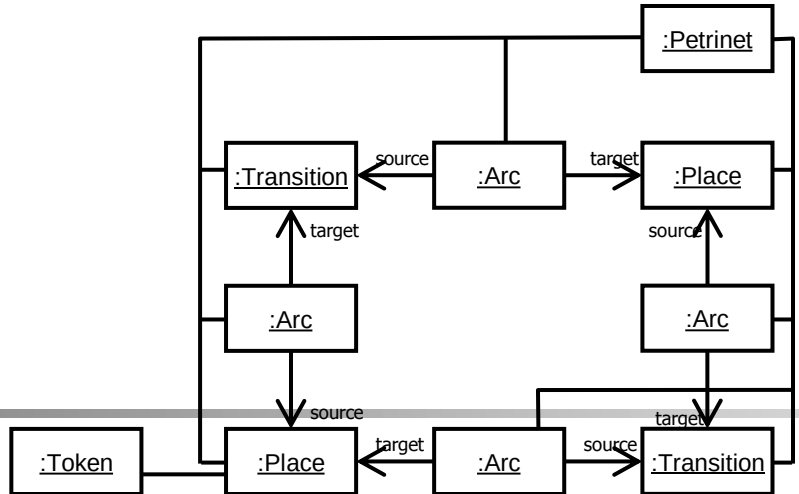

ClassDiagram
*
*
Class
1 source
1 target
Association


Transition
Place
Arc
Token


PetriNet
*
Object
Node
1 source
1 target
Arc
Transition
Place
Token
*

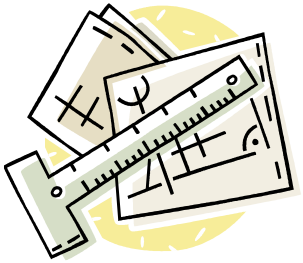

...
:Association
:Class
:Class
:Association
...




:Petrinet
:Transition
source
:Arc
target
:Place
target
source
:Arc
:Arc
source
target
:Token
:Place
target
:Arc
source
:Transition

Issues

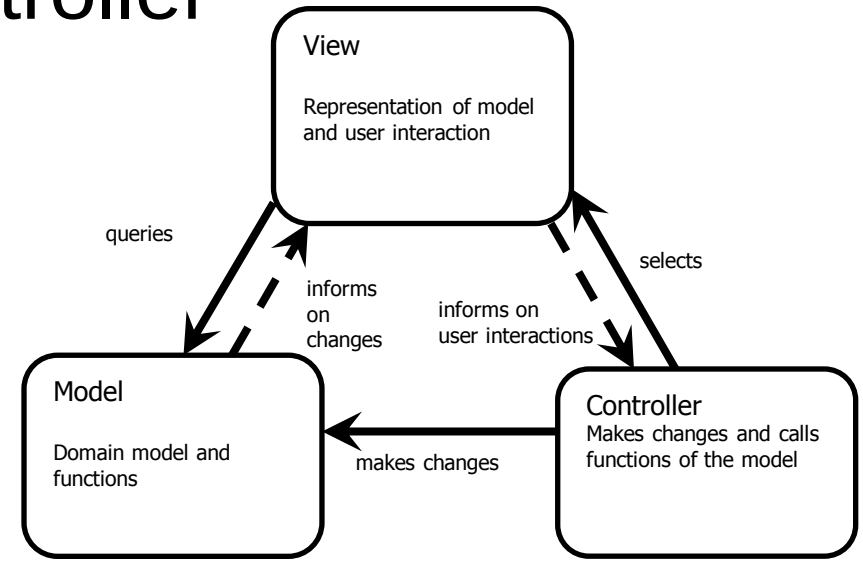
- Purpose and roles of models in SE



Making adequate/helpful models!

- Model / View / Controller

Principles behind!



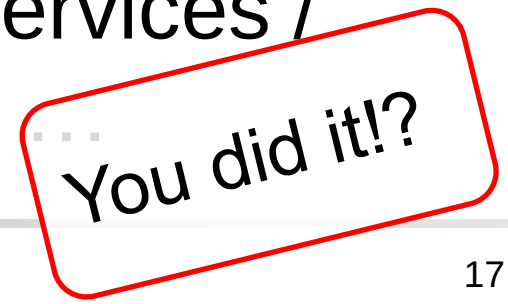
Issues

How to make useful UML models

- Domain model OOA
- Implementation OOD

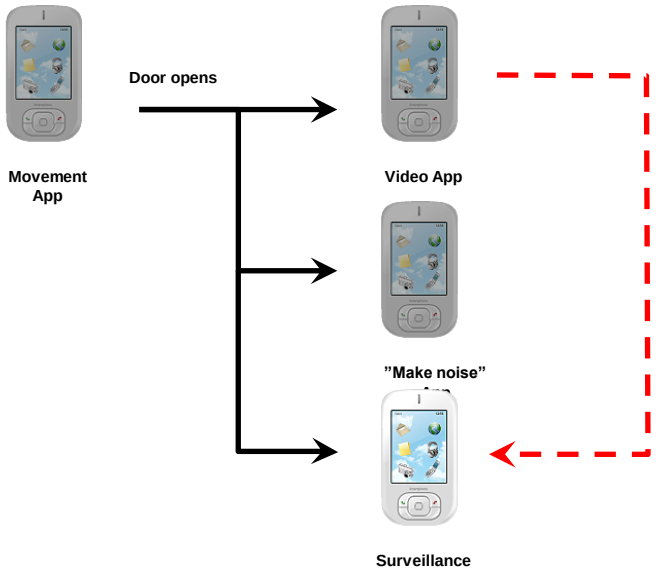
- Learn to use new technologies **YOURSELF**!

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



Issues

- Get requirements straight



- Write a systems specification

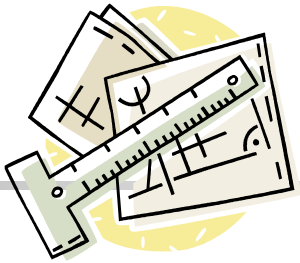
Important: Have another look at your documents now and ask yourself: What should have been in there?

Specifying Software

And why!
(objective)

what

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

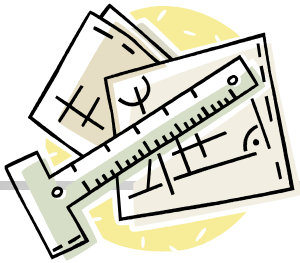


how

Specifying Software

rough

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

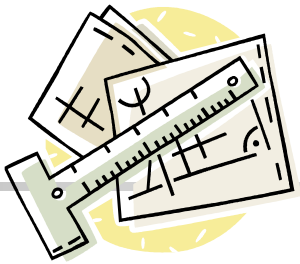


detailed

Specifying Software

low cost

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



high cost

Issues

- writing,
- talking,
- communicating, and
- organizing yourself

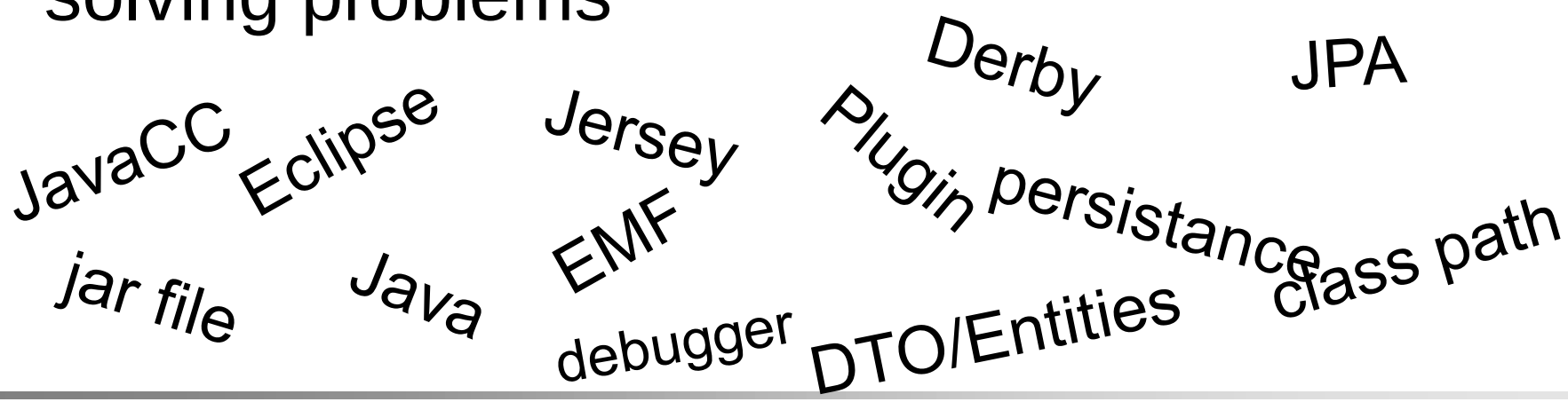
- work together
- version management (other/better tools)

Explicit and concrete communication!

Issues

- Quality
  - Management
  - Testing
  - Reviews

- Many practical issues on programming and solving problems



Issues

- 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)
  - …

Topics

- 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!

Technology issues

- 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
- ...

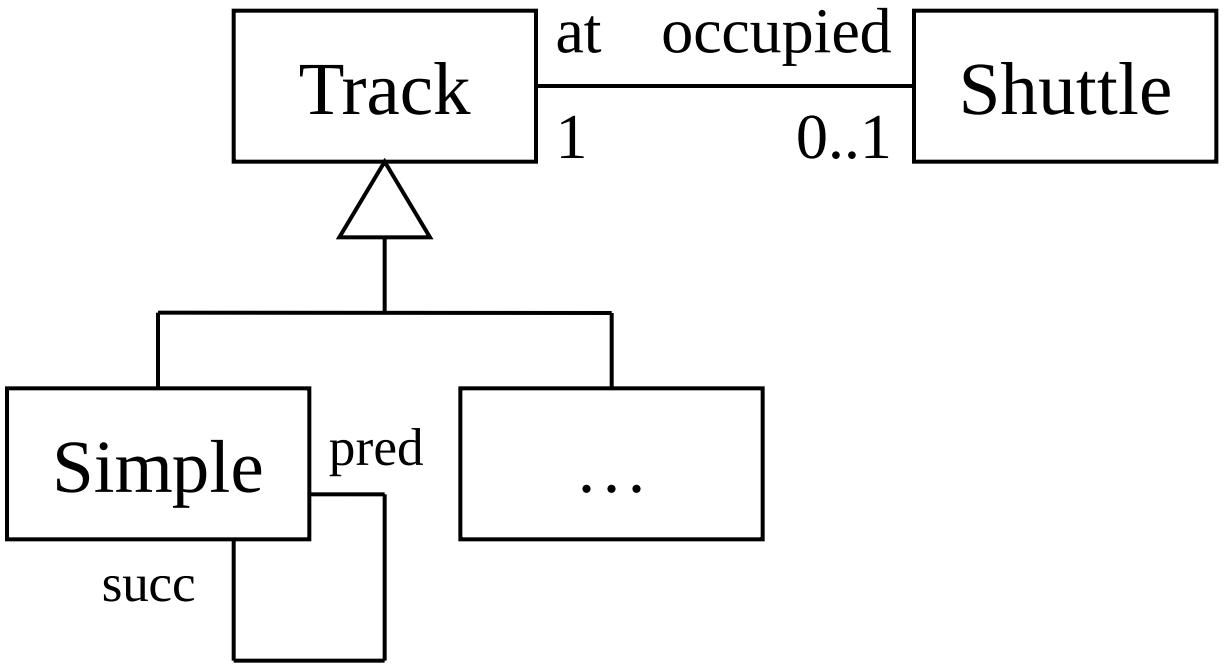
Advanced Software Engineering

Just to give you some idea

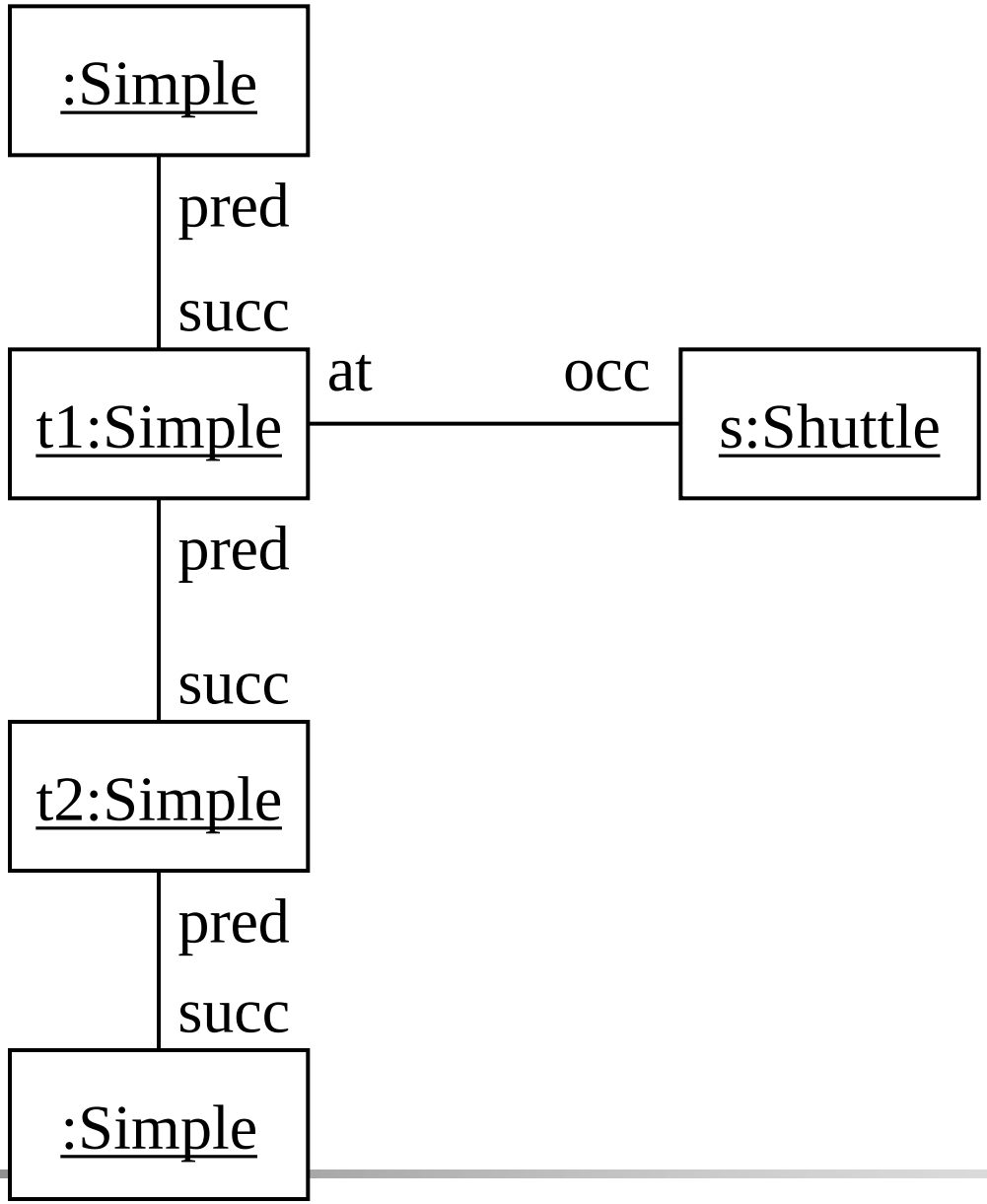
Future technologies

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

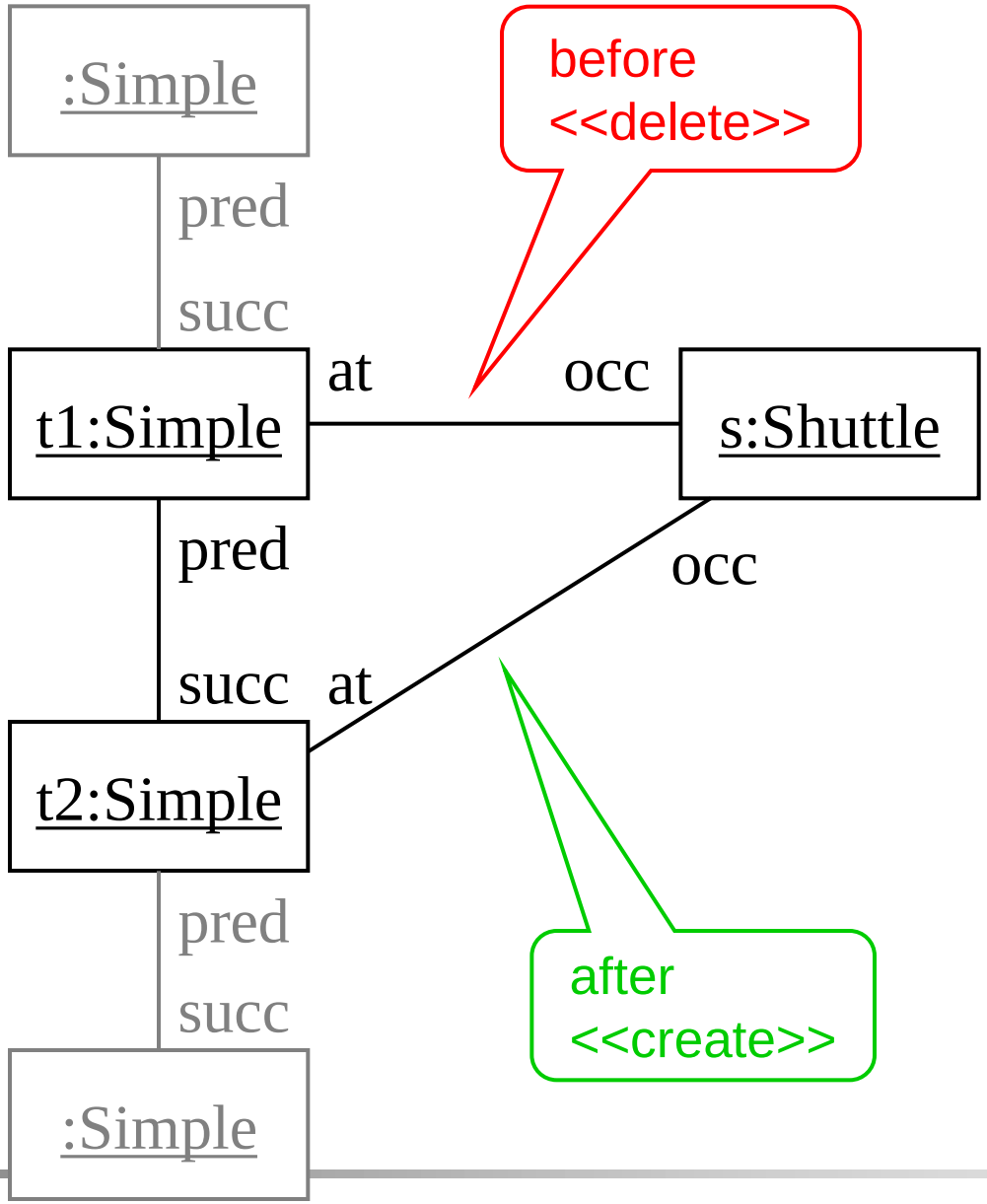
Example: Moving Shuttles



An Instance

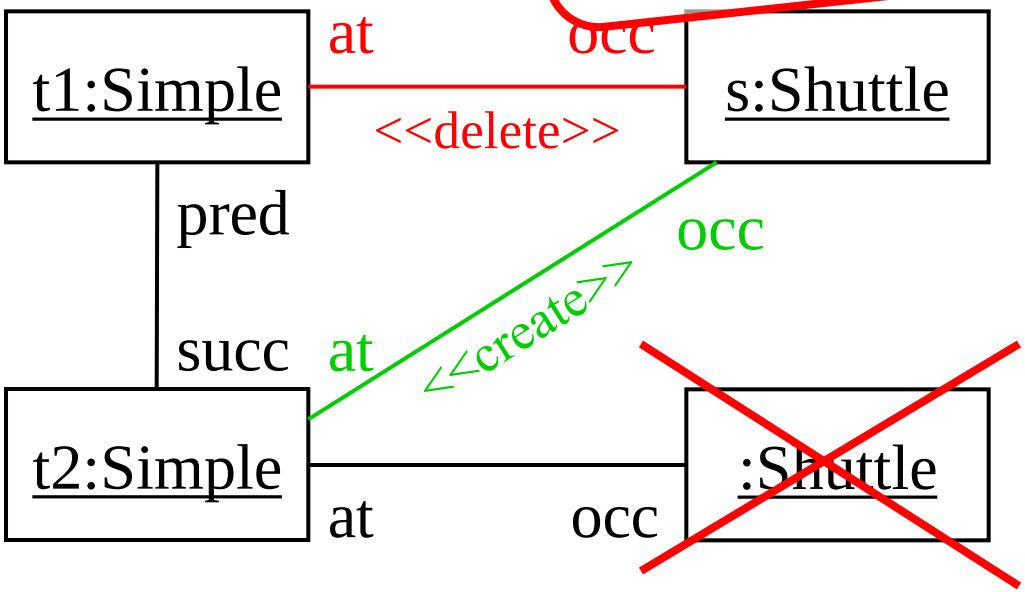

:Simple
pred
succ
t1:Simple
at
occ
s:Shuttle
pred
succ
t2:Simple
pred
succ
:Simple

Behaviour


:Simple
pred
succ
t1:Simple
at
occ
s:Shuttle
before
<<delete>>
pred
occ
succ
at
t2:Simple
pred
succ
:Simple
after
<<create>>

Behaviour as Story Pattern

→ Story Diagrams: FUJABA



More details & experience

- 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)

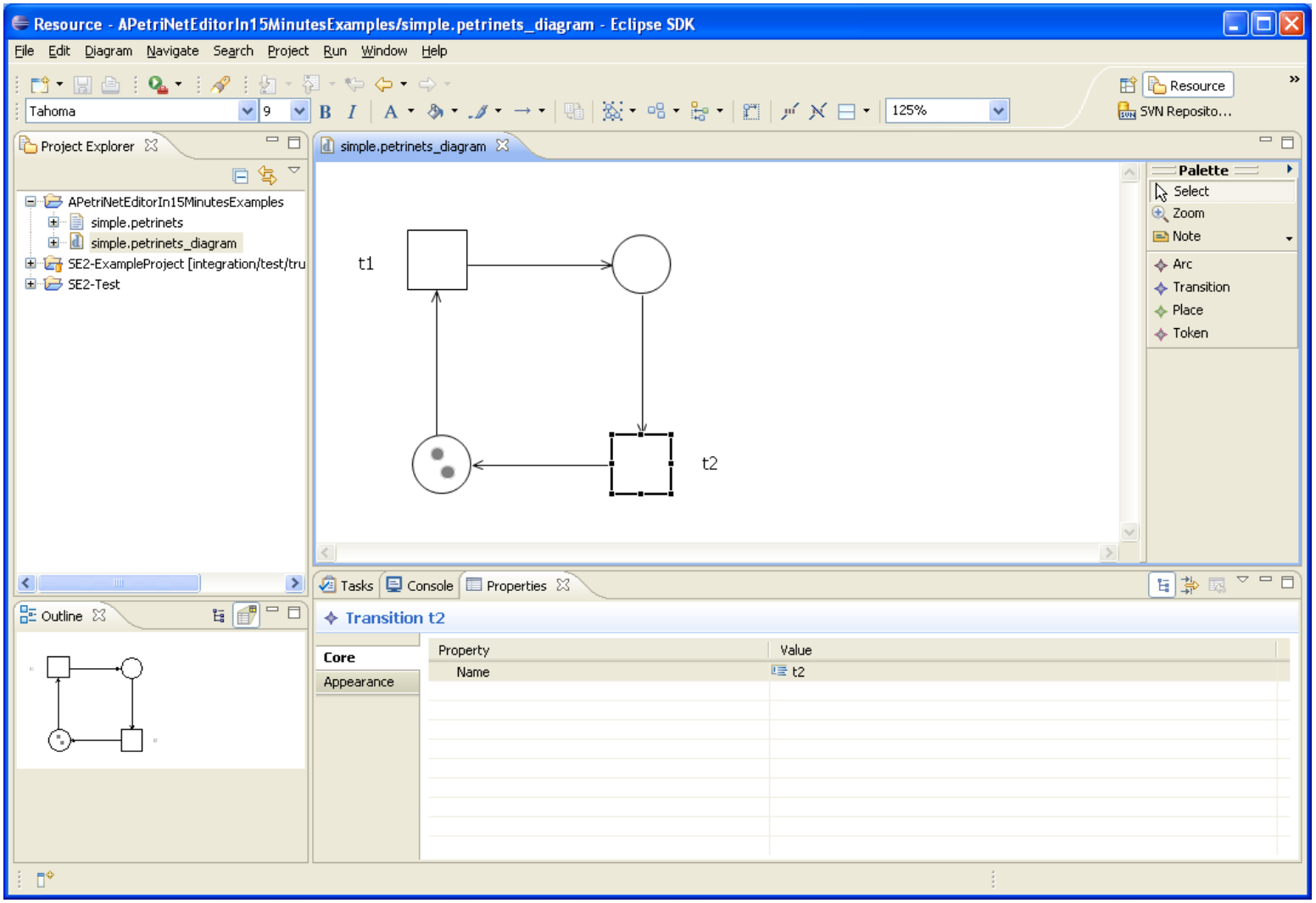
Coordinating Interactions
The Event Coordination Notation

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/

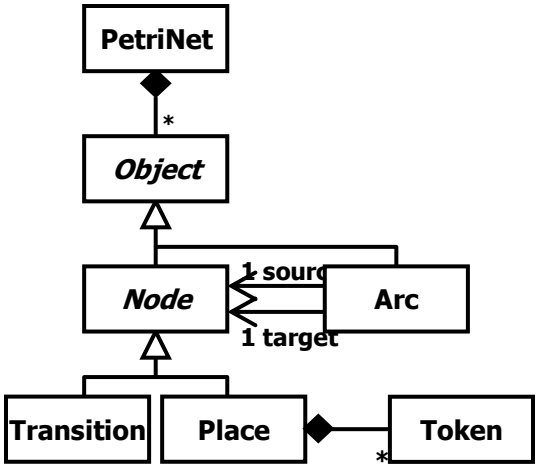
From lecture 1: Example



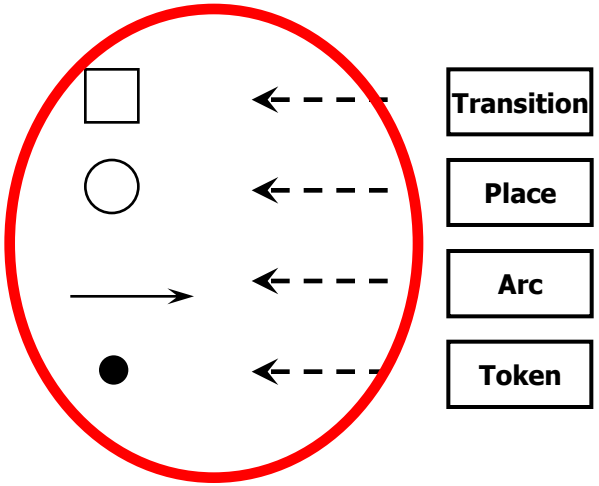
From lecture 1: Example

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)



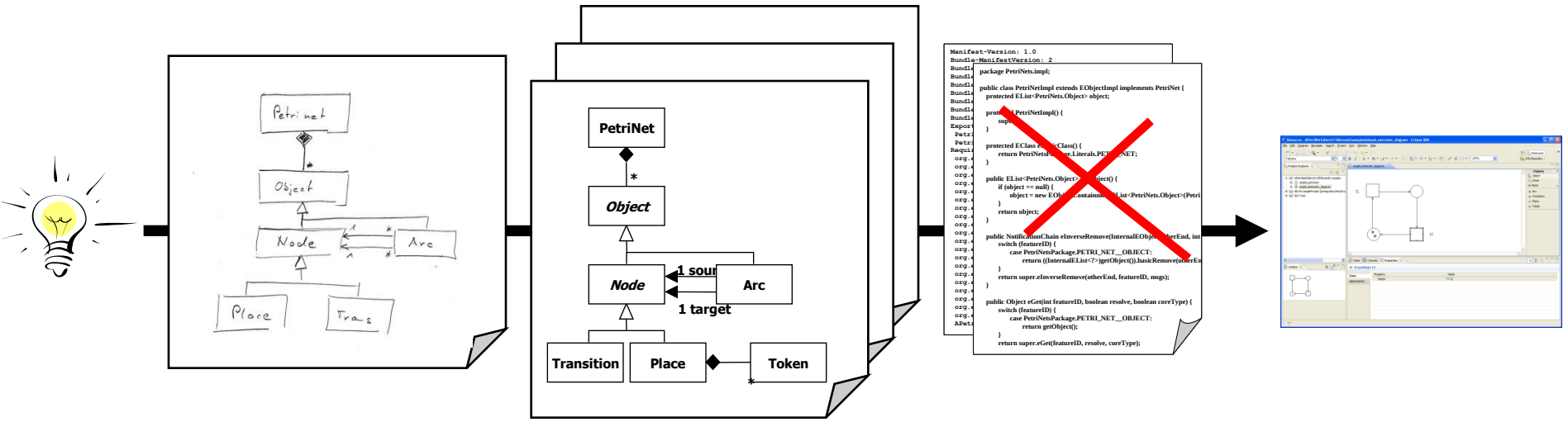
With this and some more GMF information:
Generation of the Java code of a graphical complete editor (with many fancy features). No programming at all.



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

From lecture 1

How about behaviour ?
(non-standard behaviour)



Analysis

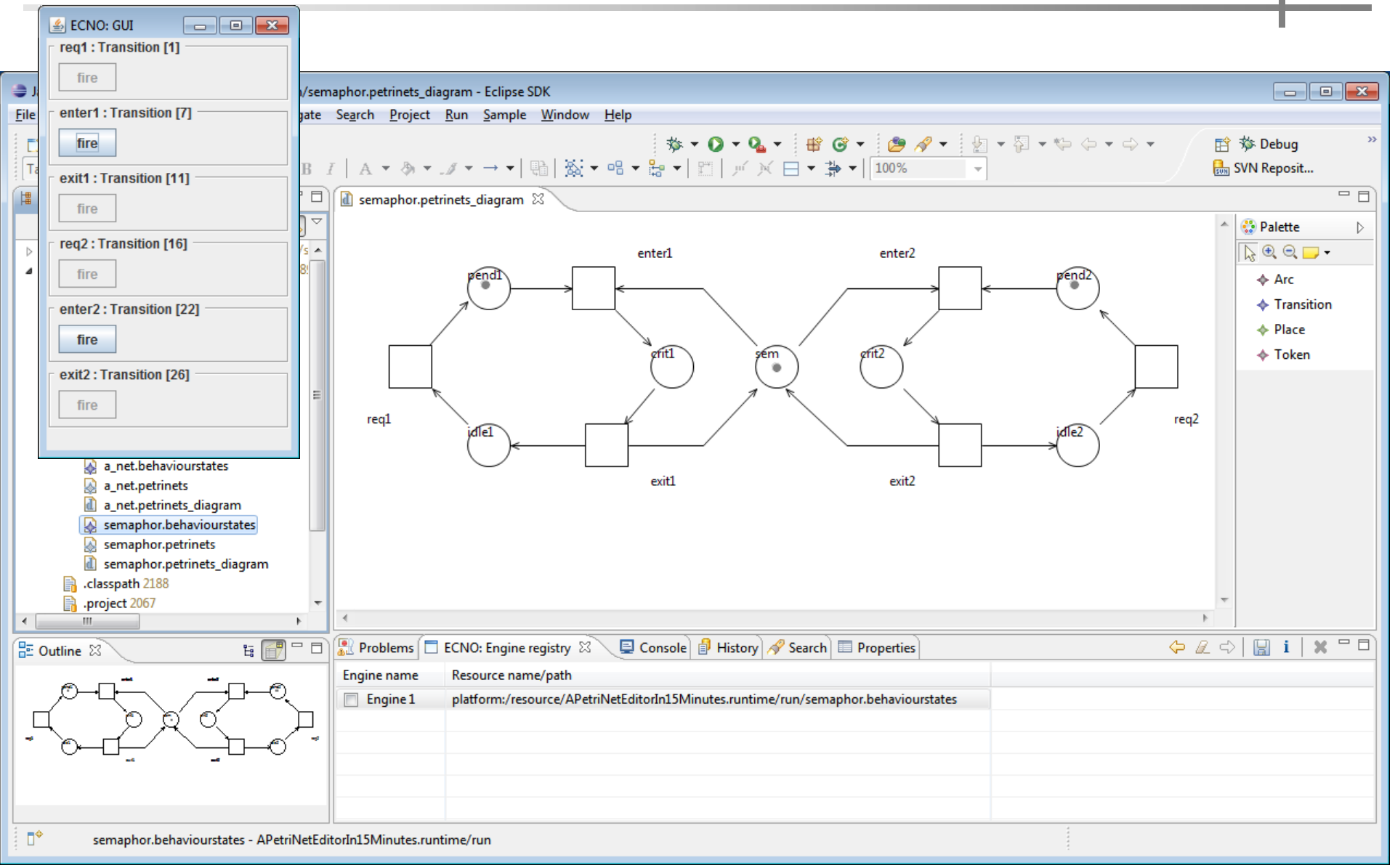
Design

Implementation

Coding

e.g. a Petri net simulator?

Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler



Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler

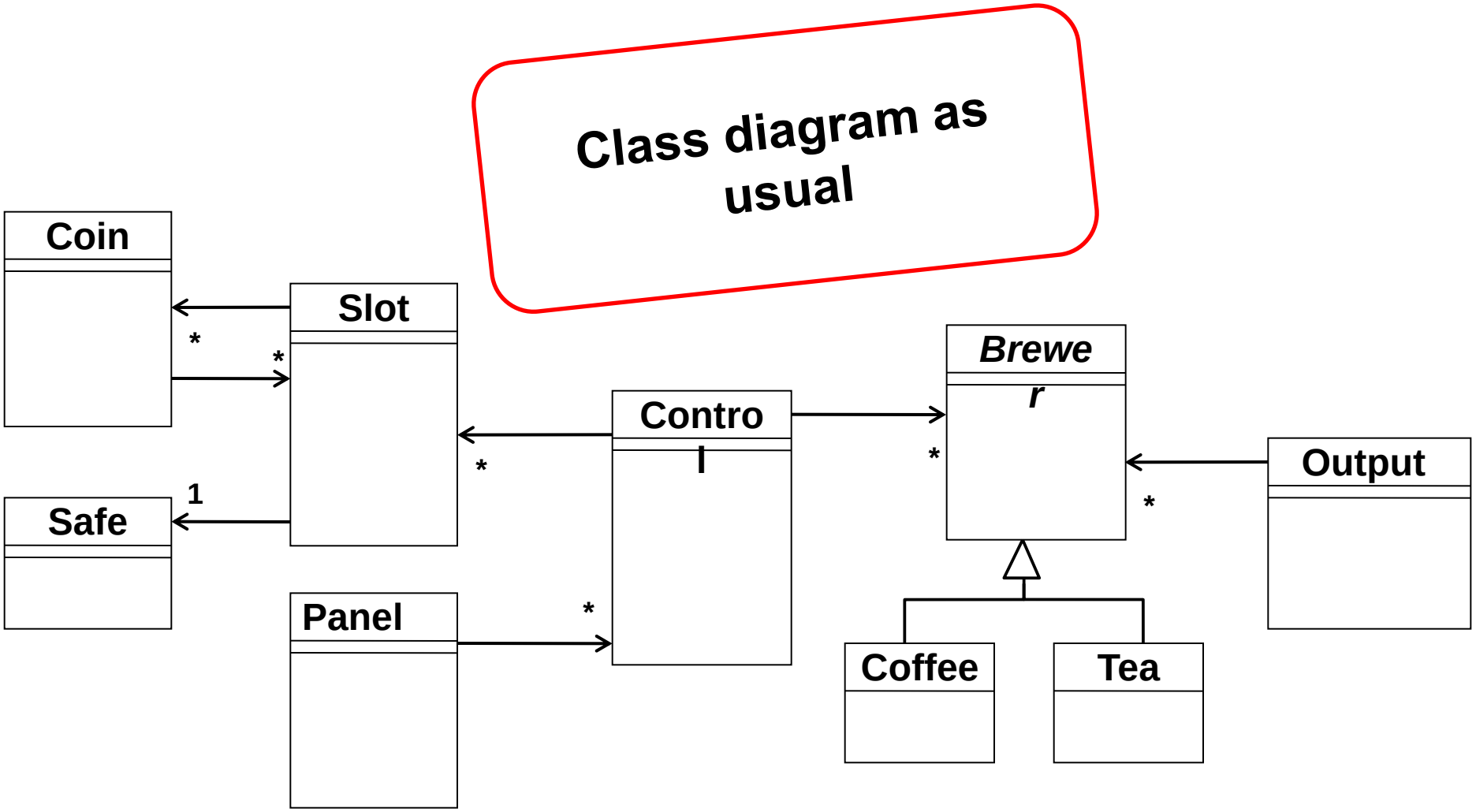
Motivation

- 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

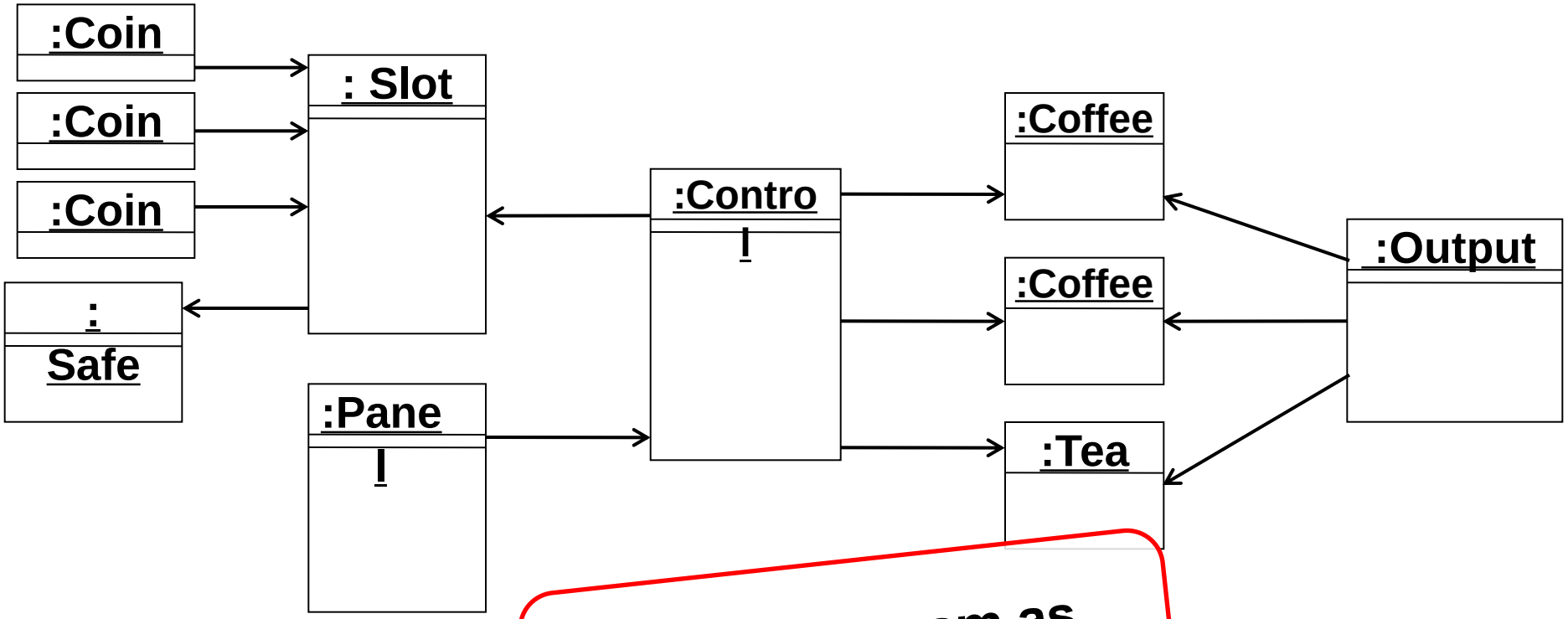
→ Integrate different structural models and manually written code (or code generated by different technologies)

2.1 Example: Vending machine



Instance: Object Diagram

Initial configuration, current situation

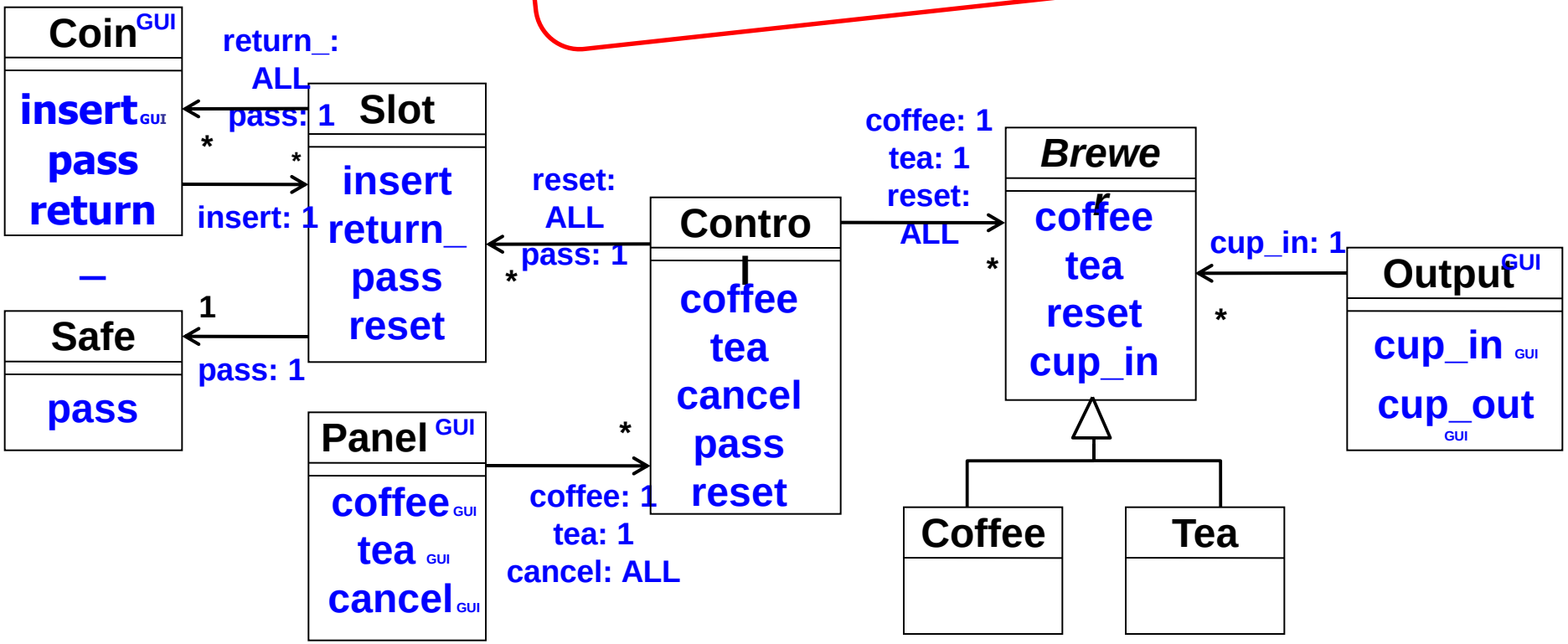


Object diagram as usual

Coordination Diagram

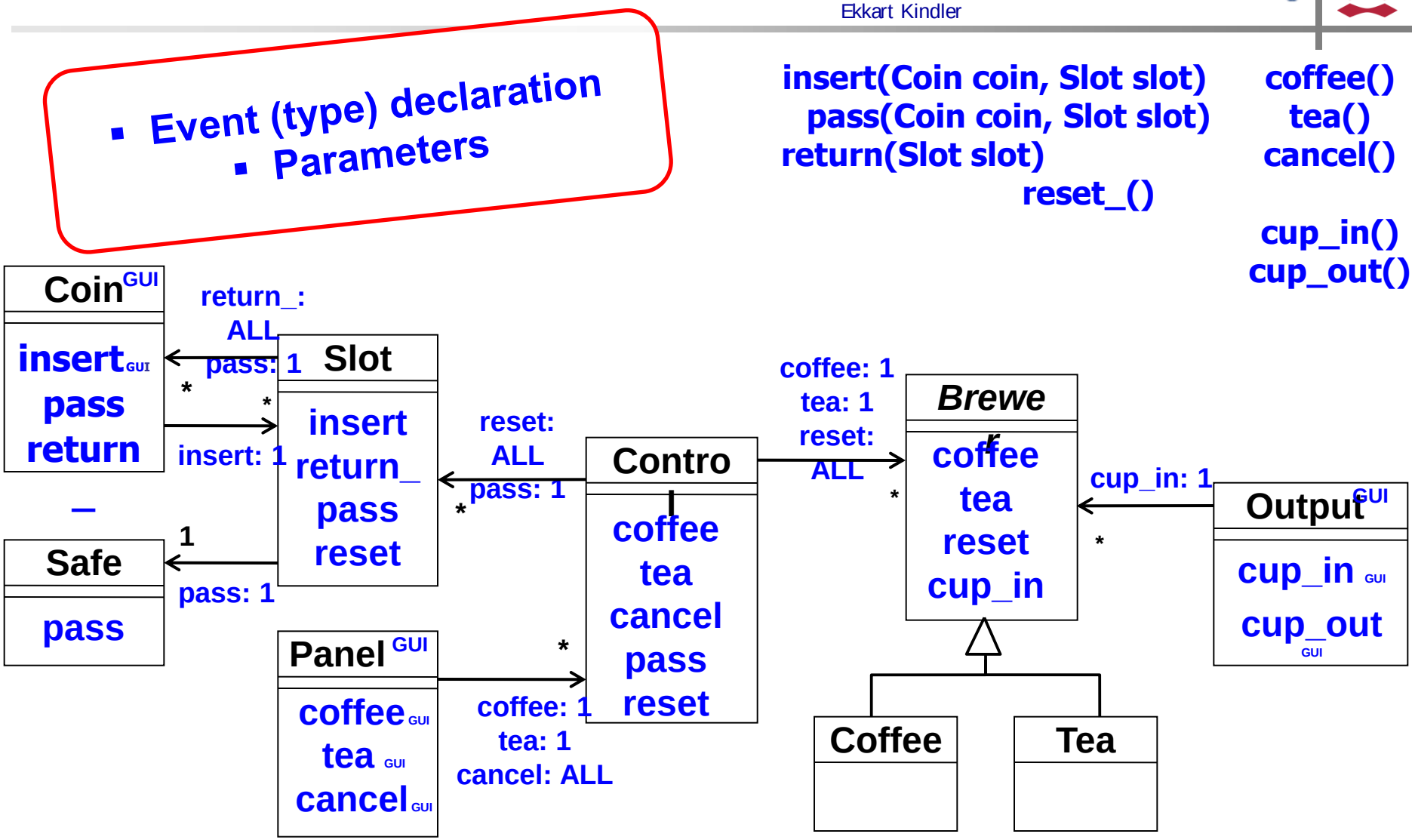
- We call objects elements now!

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

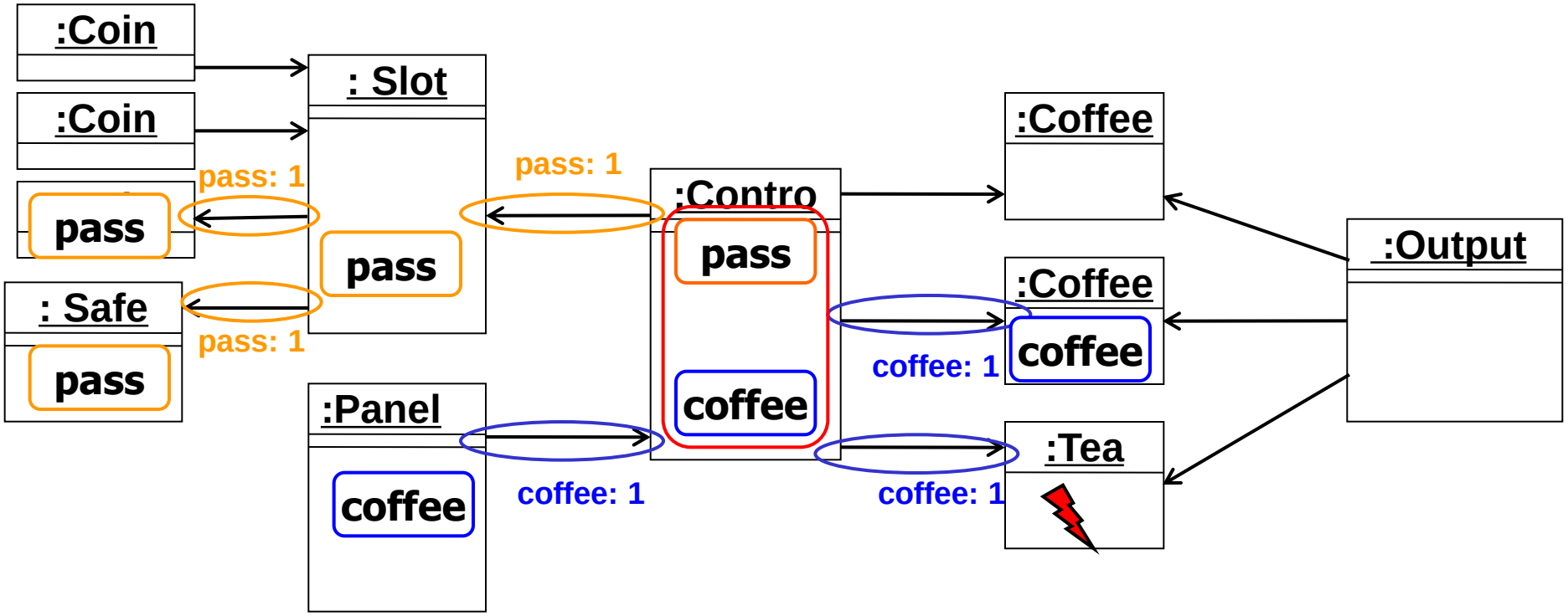


... + Event declaration

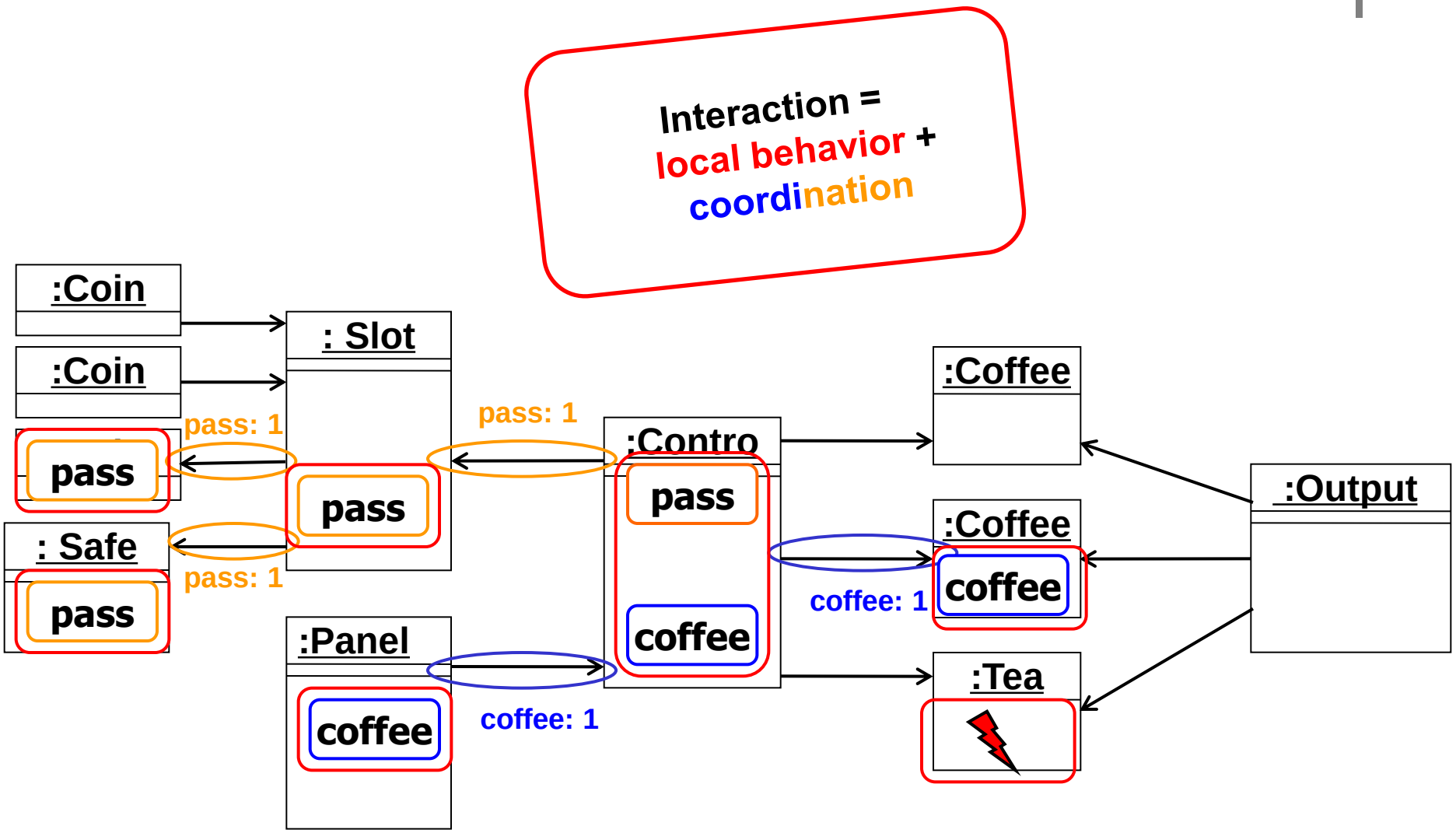
- Event (type) declaration
  - Parameters



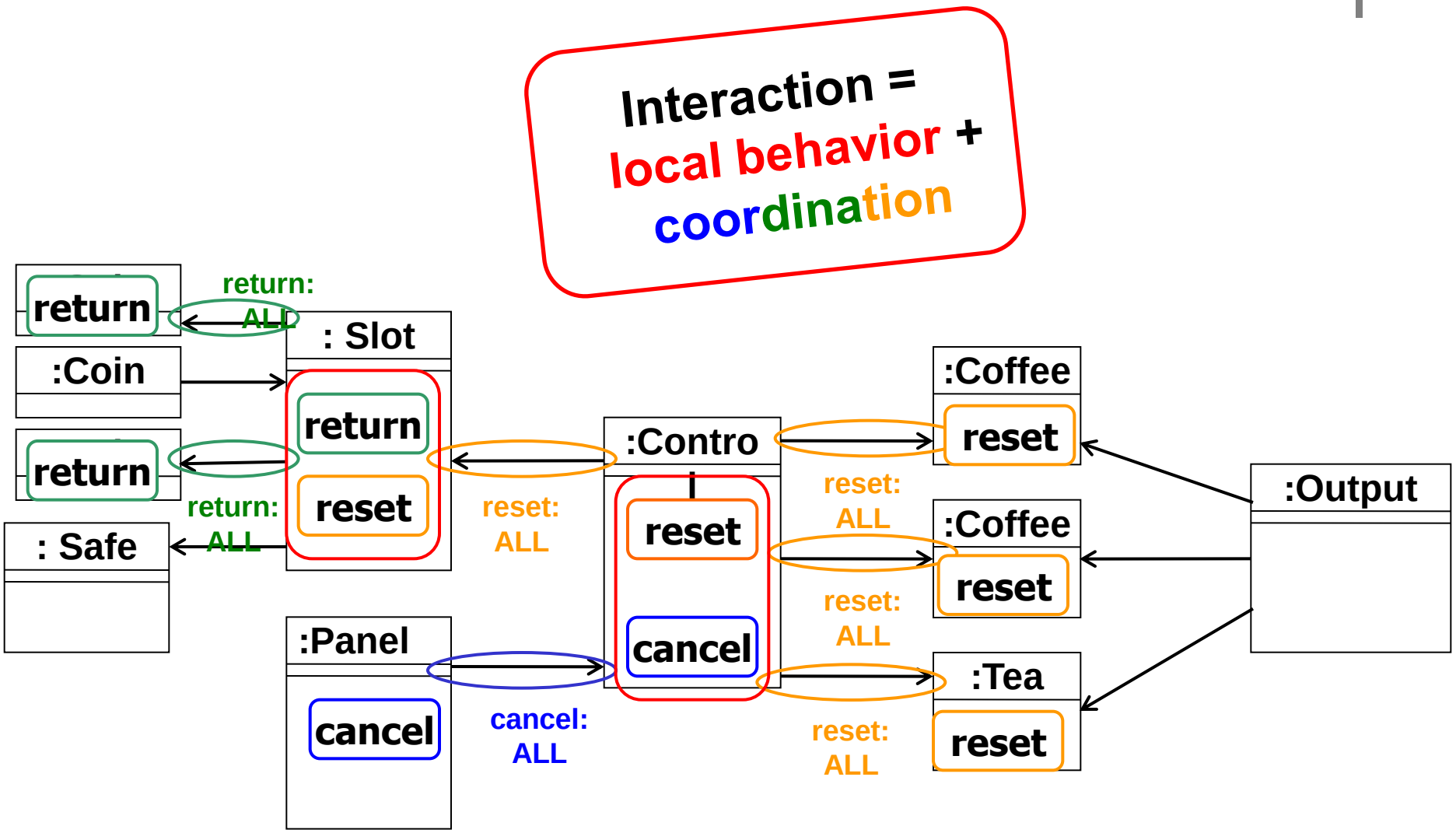
Interactions



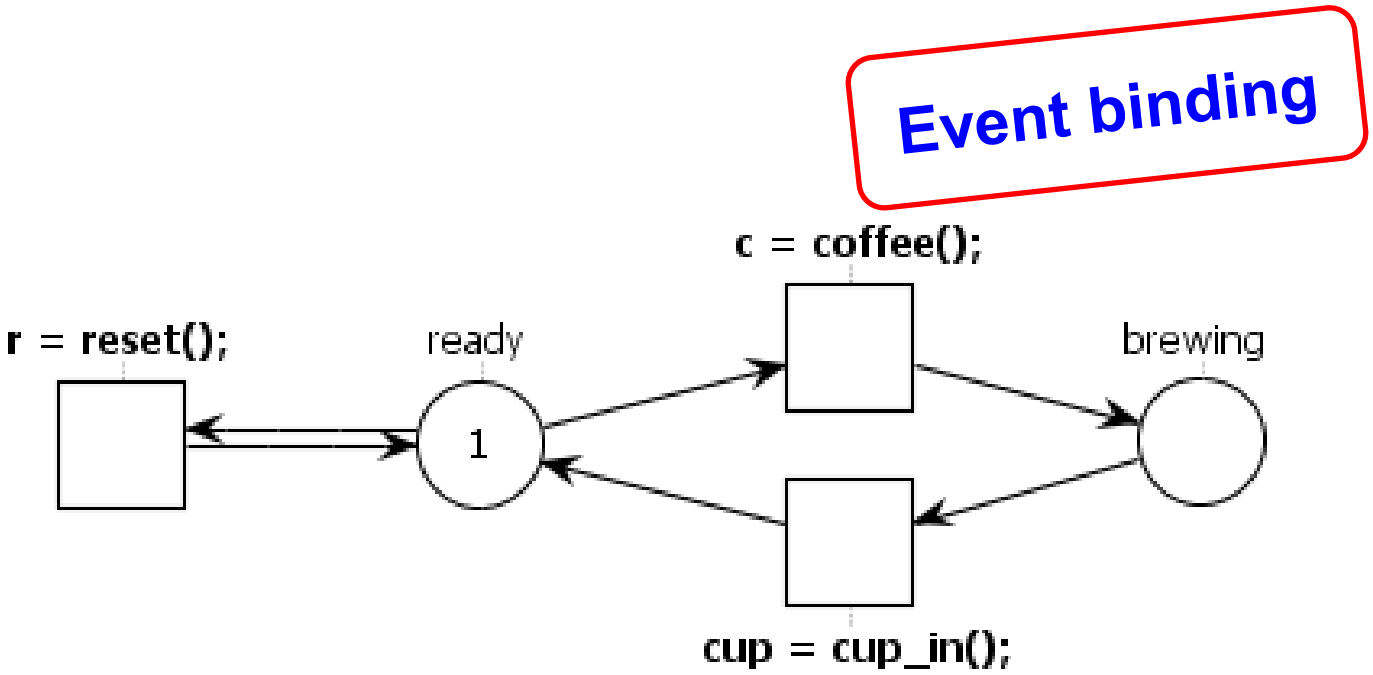
Interactions



Another Interaction

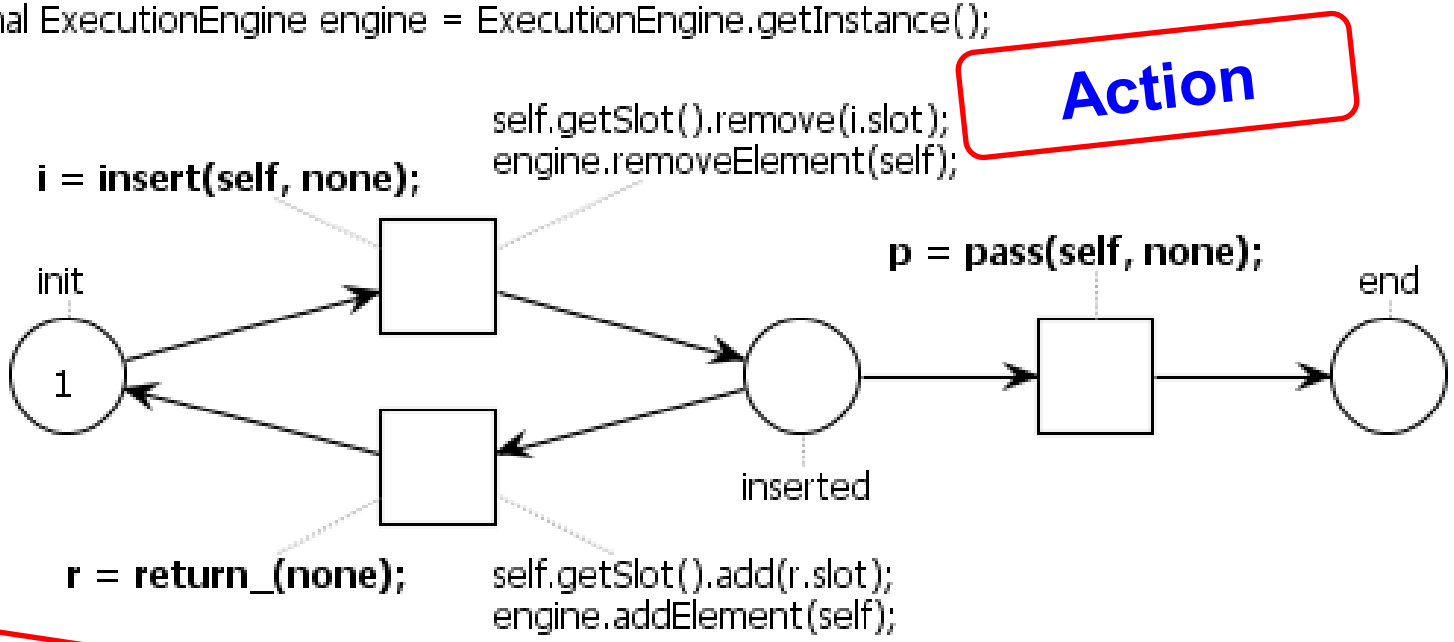


Local behaviour: Coffee



Local behaviour: Coin

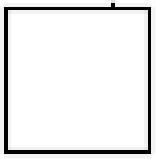

```
import dk.dtu.imm.se.ecno.engine.ExecutionEngine;

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



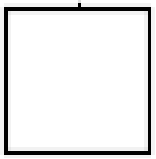
- **Event binding**
- **Parameter assignment**

Local behaviour: Control

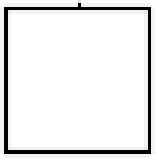
p = pass(none,none); c = coffee();

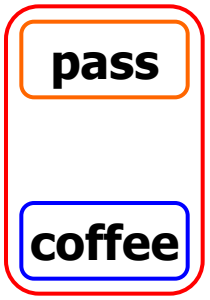


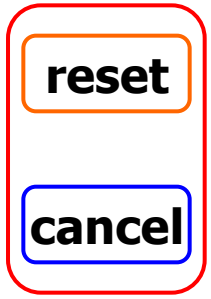
p = pass(none,none); t = tea();



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



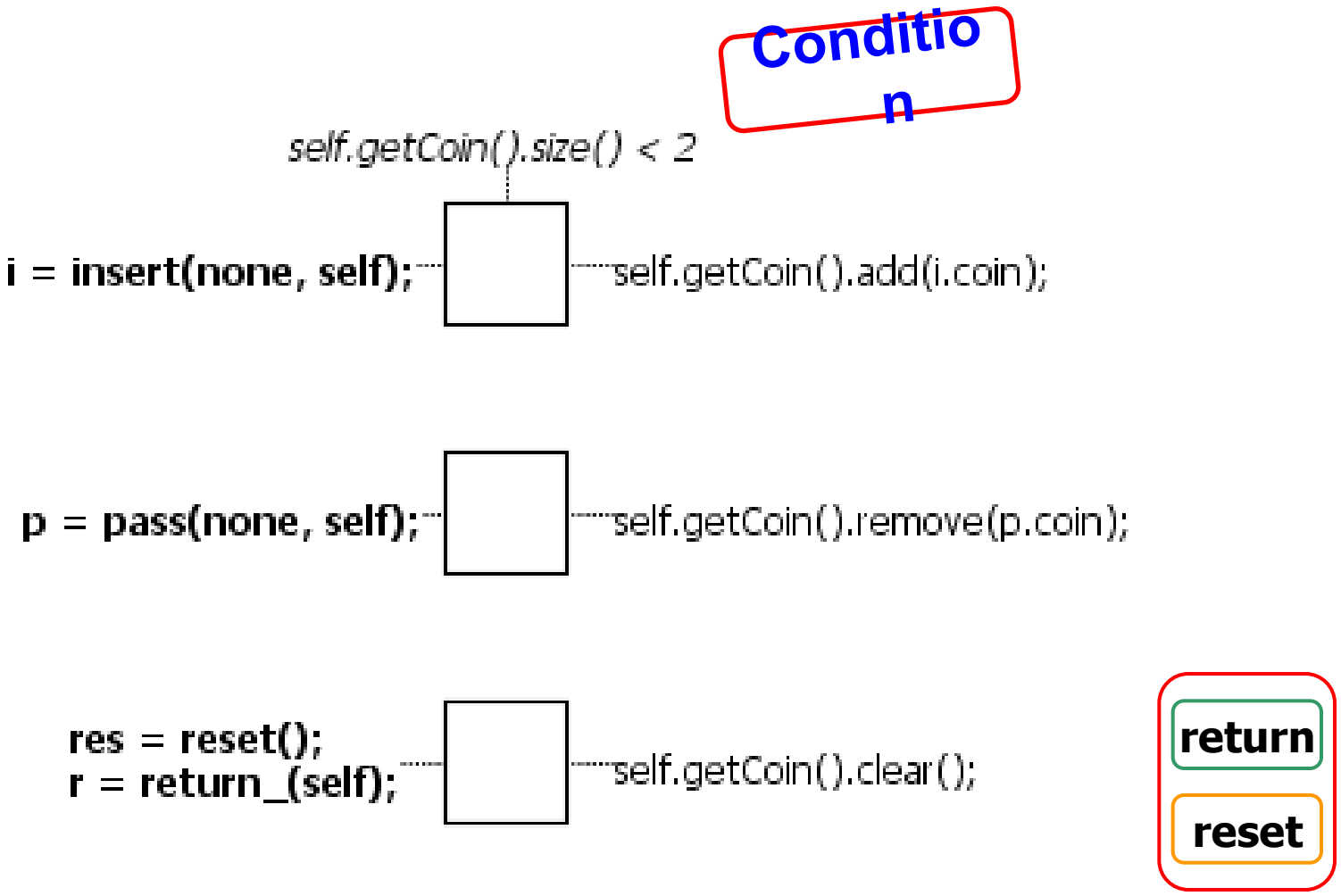




- Event binding with multiple event types!

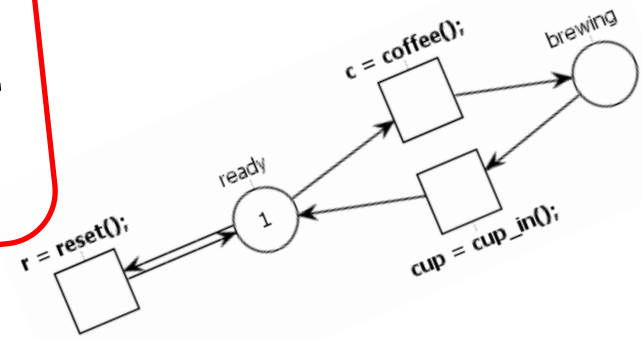
Local behaviour: Slot

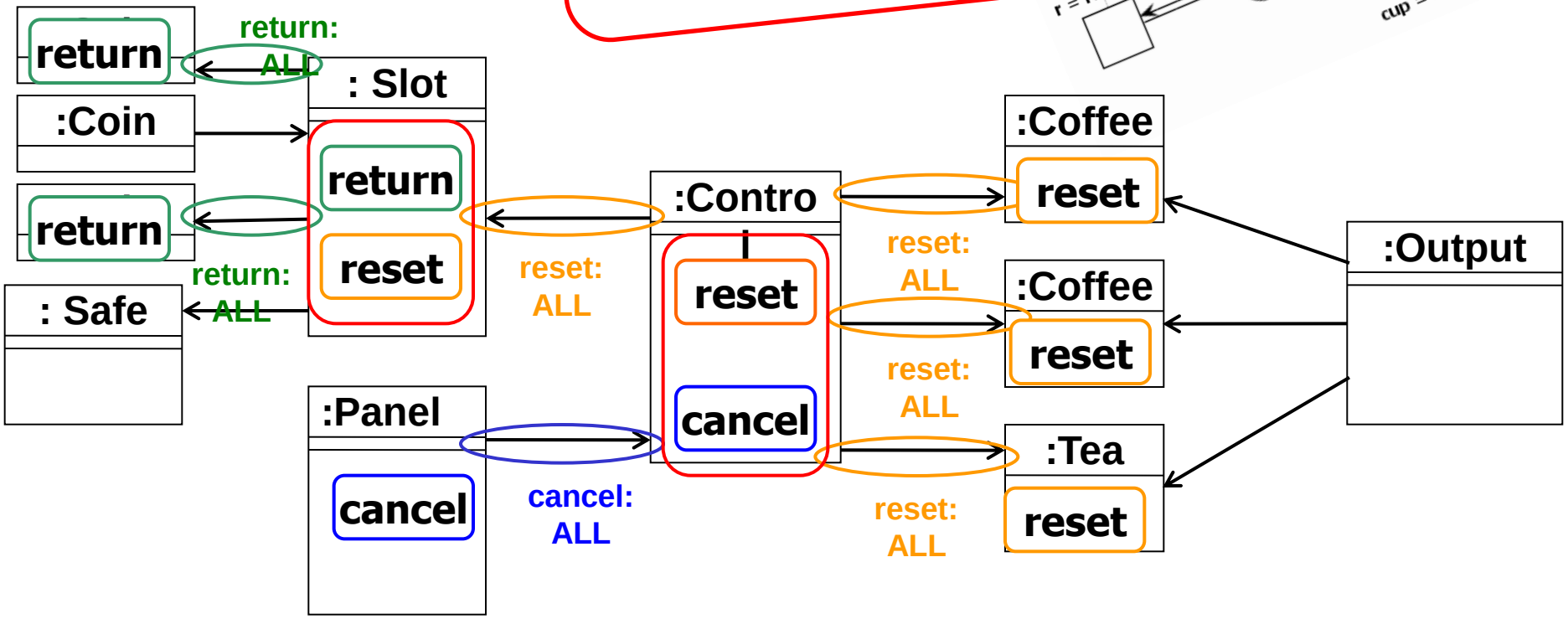
Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler



Interactions

Interaction = local behavior + coordination



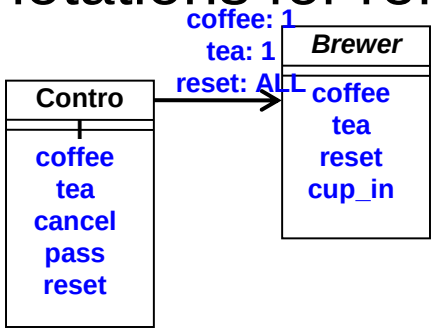


2.2 ECNO: Basic Concepts

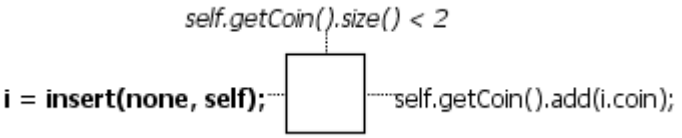
- ElementTypes (Classes)
- EventTypes with
  - parameters

insert(Coin coin, Slot slot)

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



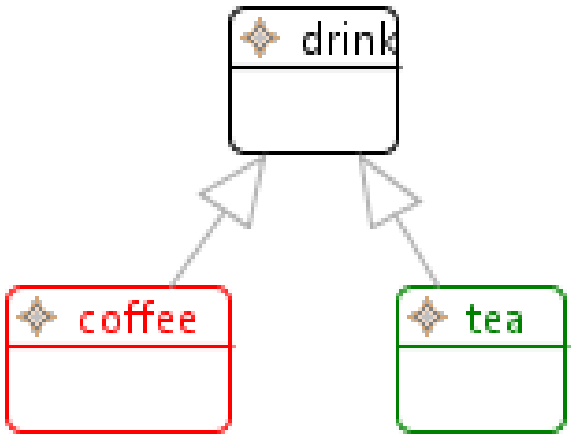
- Local behaviour: ECNO nets (or something else)
  - Event binding (with parameter assignment)
  - Condition
  - Action



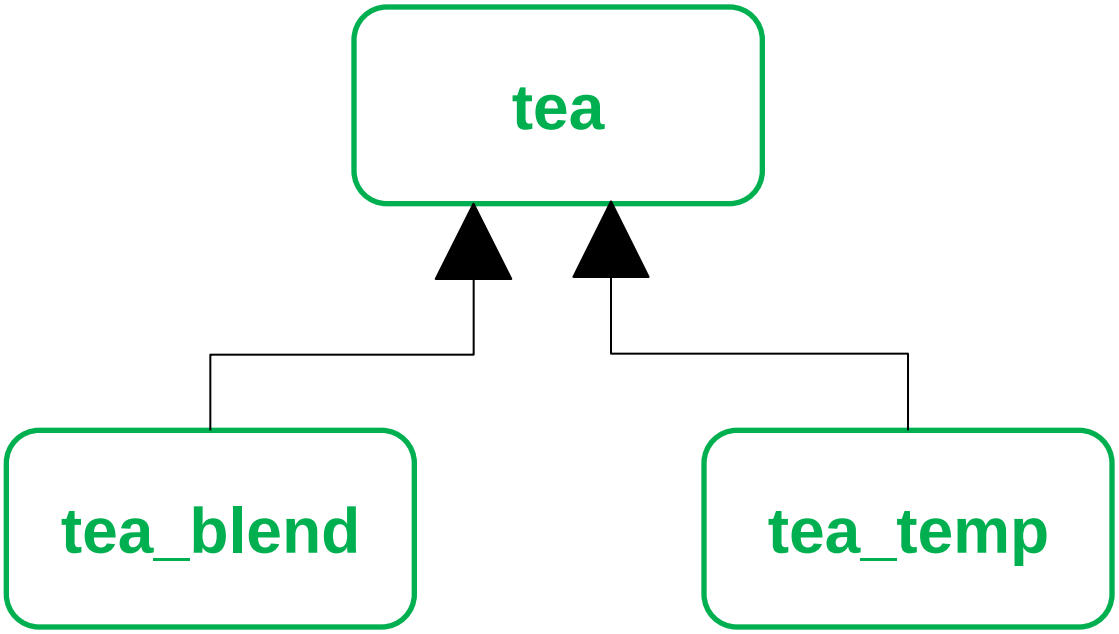
2.3 Extensions

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

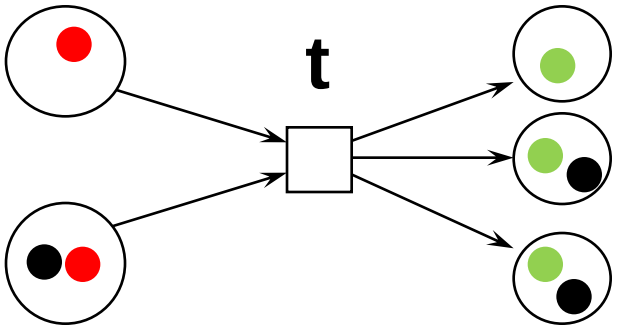


Event Extension



=> Two forms of inheritance on event types!

2.4 Example2: Petri nets



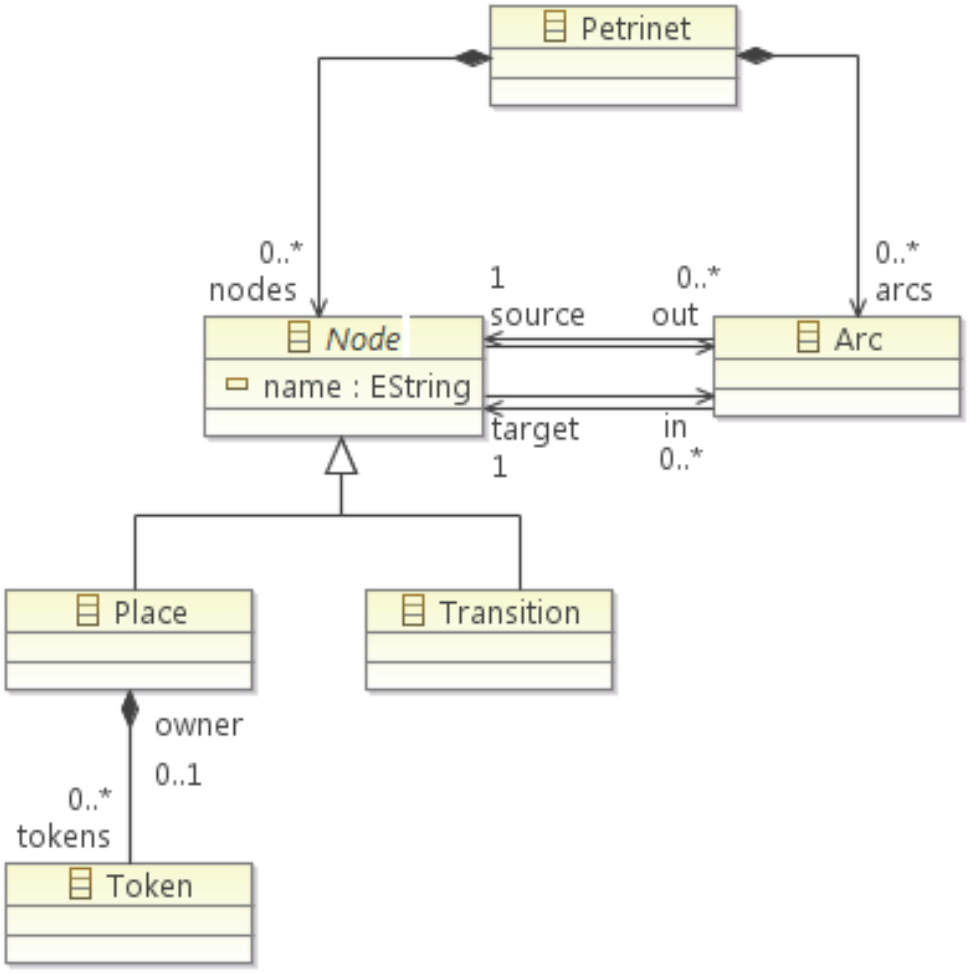
How can we model that behaviour in ECNO nets?

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

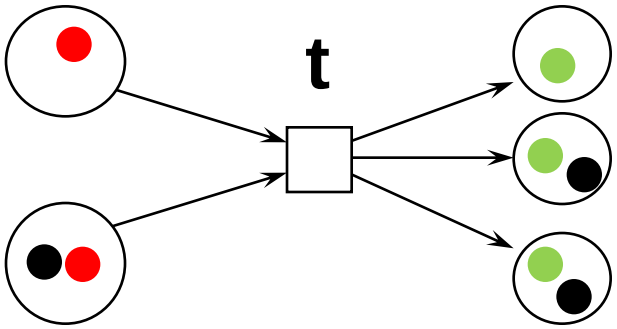
Fire Transition t:
for ALL incoming Arcs a:
for ONE source Place p of Arc a:
find a token and remove it

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

Petri net: Abstract Syntax



Example2: Petri nets



remove **add**

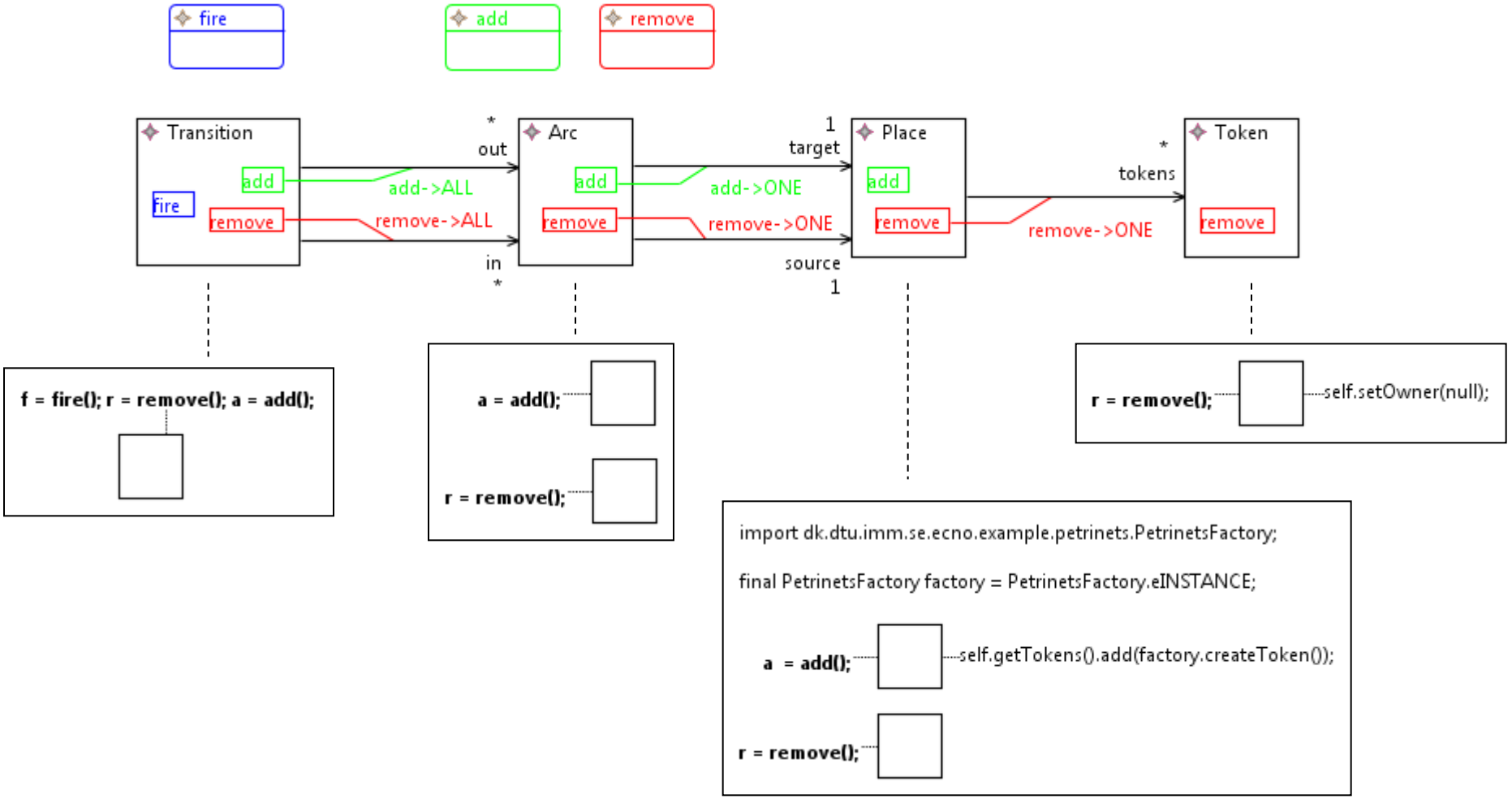
Transition t enabled:
for ALL incoming Arcs a:
for ONE source Place p of Arc a:
find a token

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

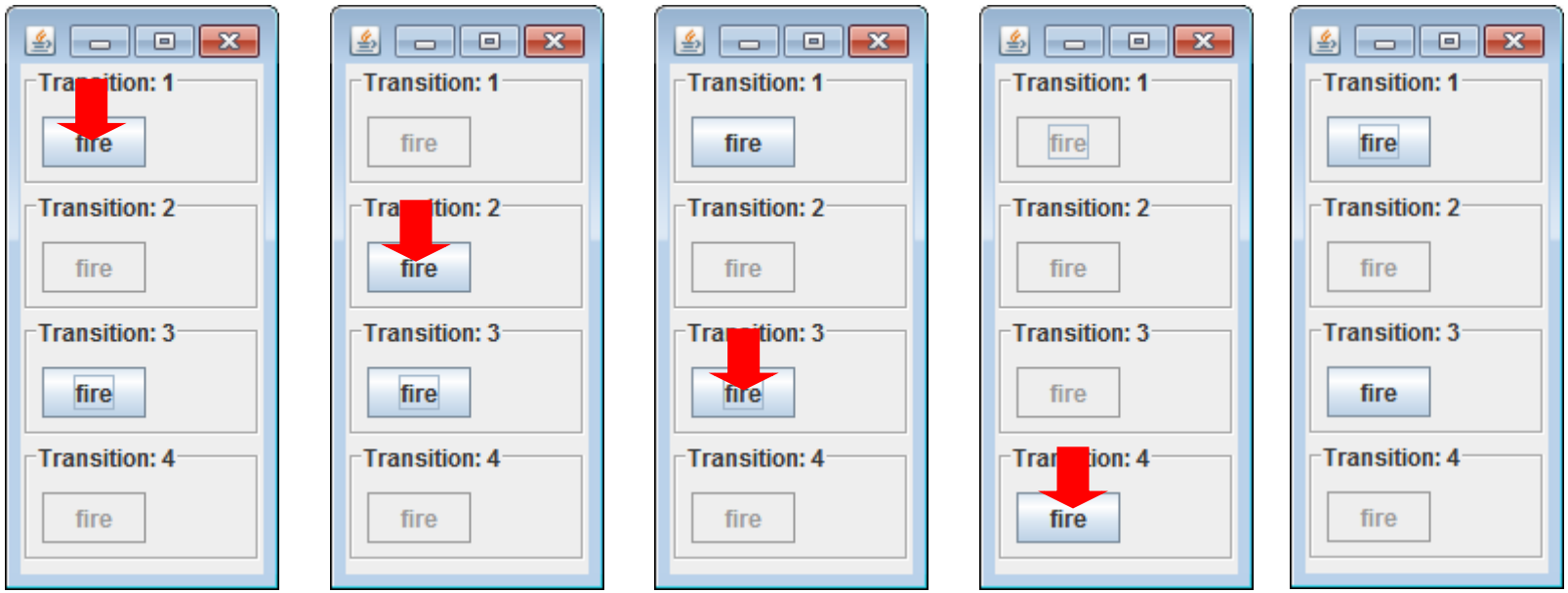
for ALL outgoing arcs a:
for ONE target Place p of Arc a:
add a new Token

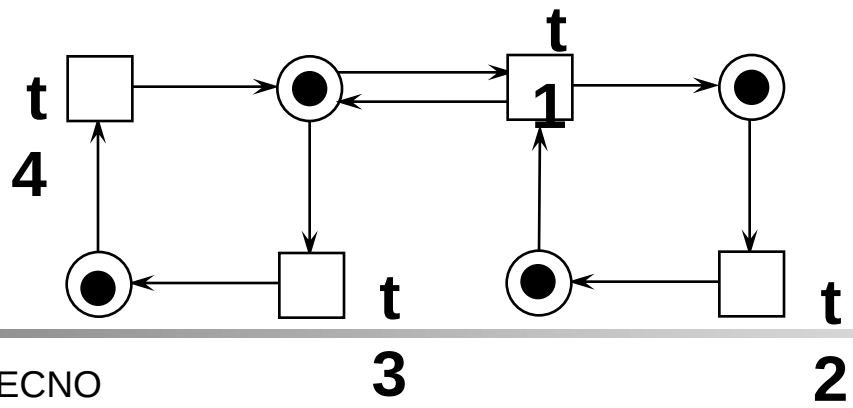
ECNO Semantics of PN

Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler



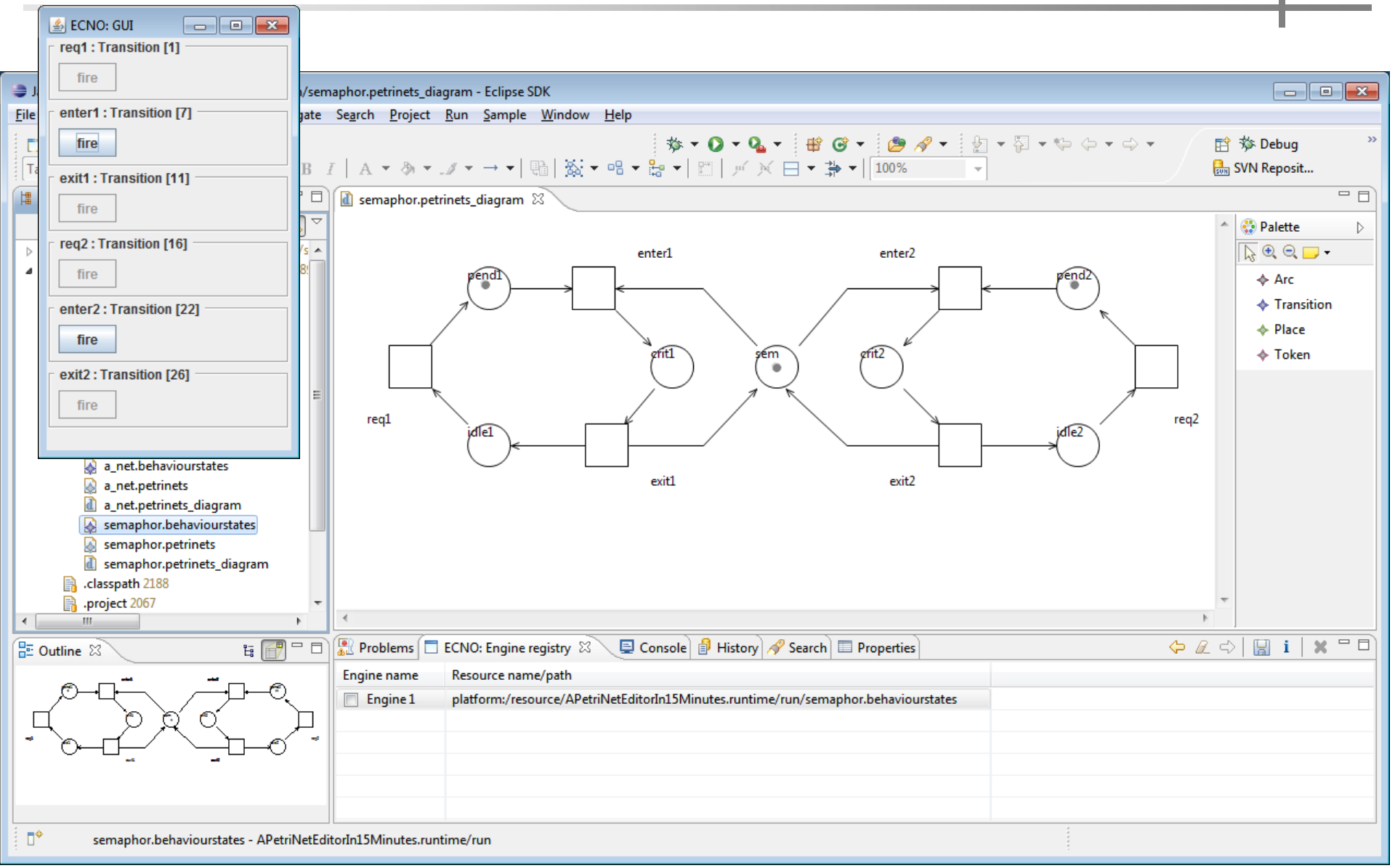
Result


Transition: 1
fire
Transition: 2
fire
Transition: 3
fire
Transition: 4
fire


t
1
t
4
t
3
t
2

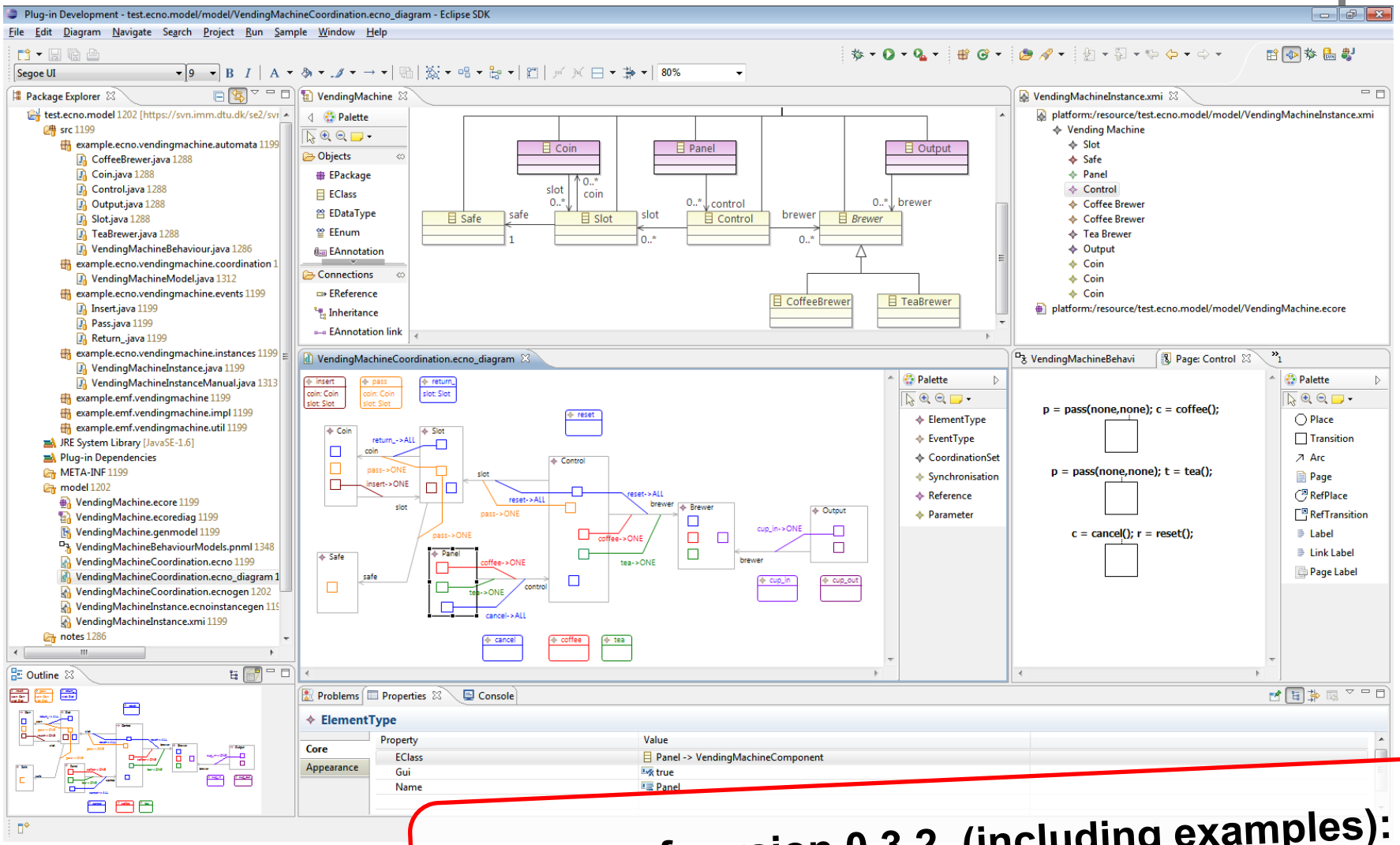
Petri net simulator

Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler



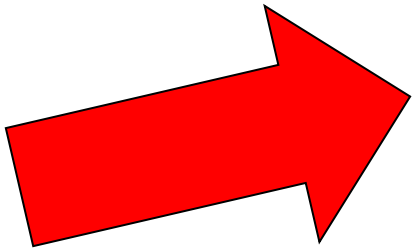
3. Conclusion

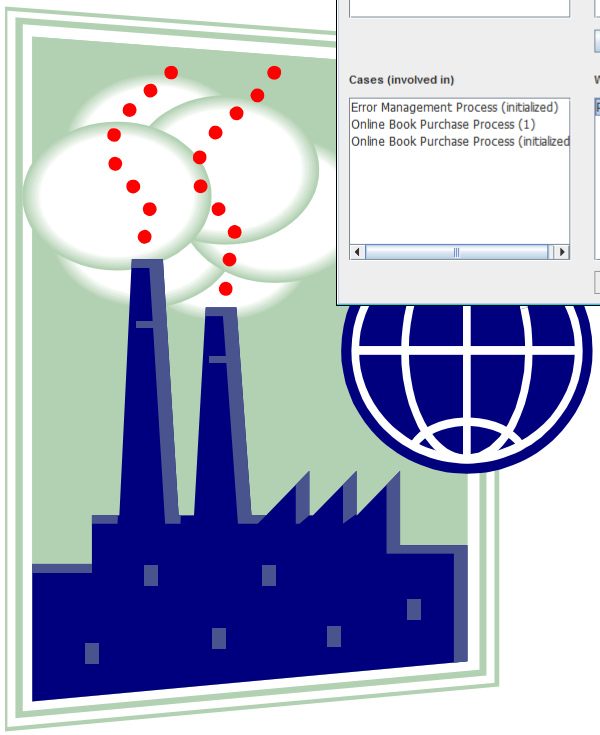
Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler

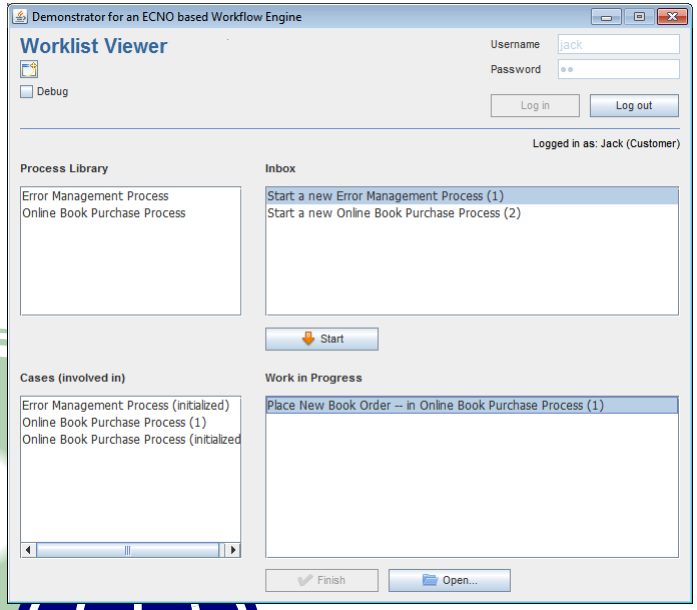


Release of version 0.3.2 (including examples):
http://www2.imm.dtu.dk/~ekki/projects/ECNO/

Next Steps






Demonstrator for an ECNO based Workflow Engine
Worklist Viewer
Debug
Username
jack
Password
Log in
Log out
Logged in as: Jack (Customer)
Process Library
Error Management Process
Online Book Purchase Process
Inbox
Start a new Error Management Process (1)
Start a new Online Book Purchase Process (2)
Start
Cases (involved in)
Error Management Process (initialized)
Online Book Purchase Process (1)
Online Book Purchase Process (initialized
Work in Progress
Place New Book Order -- in Online Book Purchase Process (1)
Finish
Open...