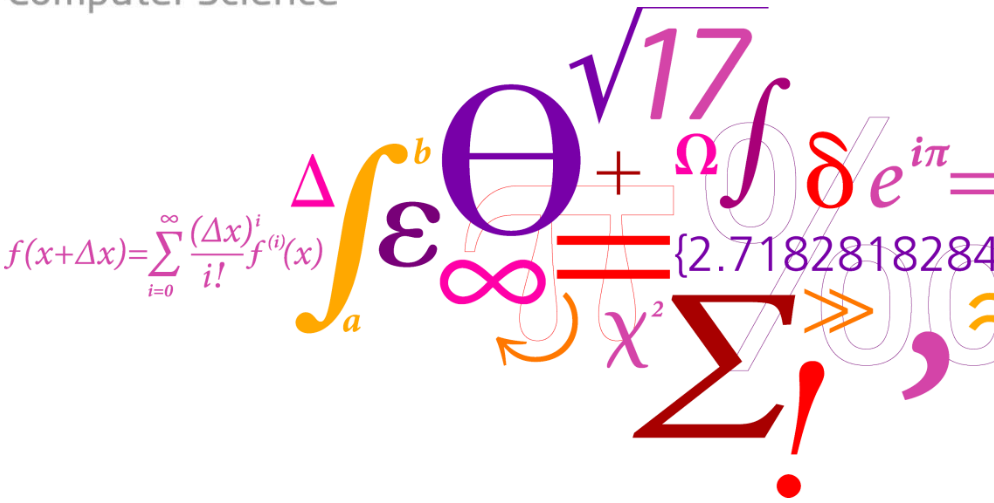
Advanced Topics in Software Engineering (02265)

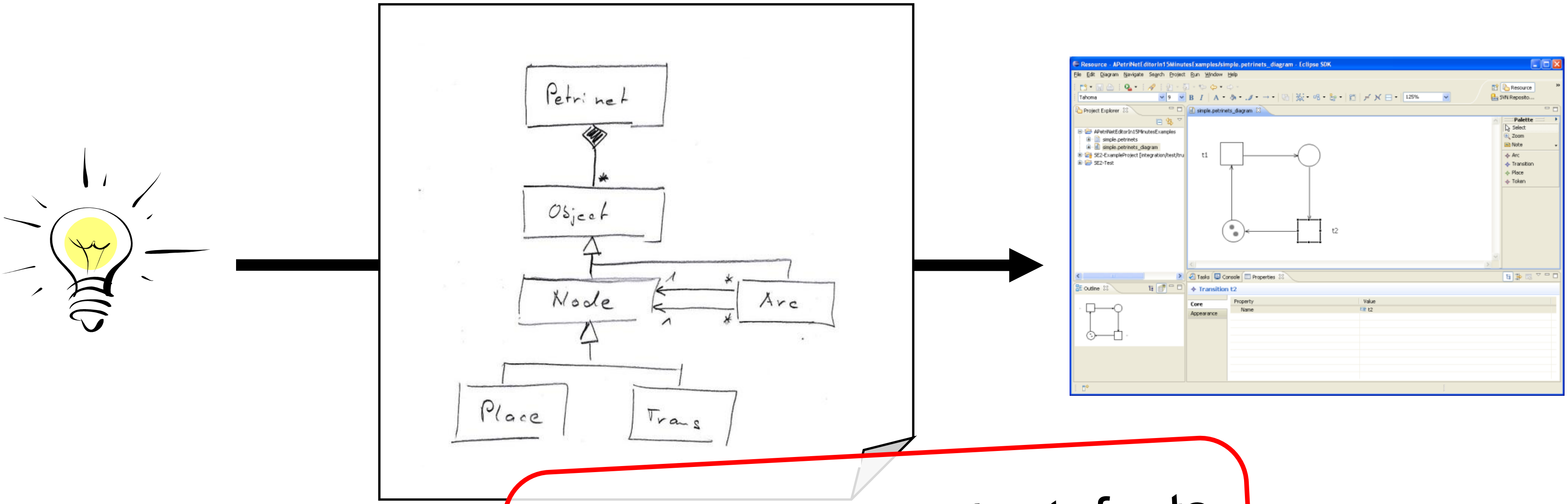
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science


$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
$\sqrt{17}$
$e^{i\pi}$
{2.7182818284

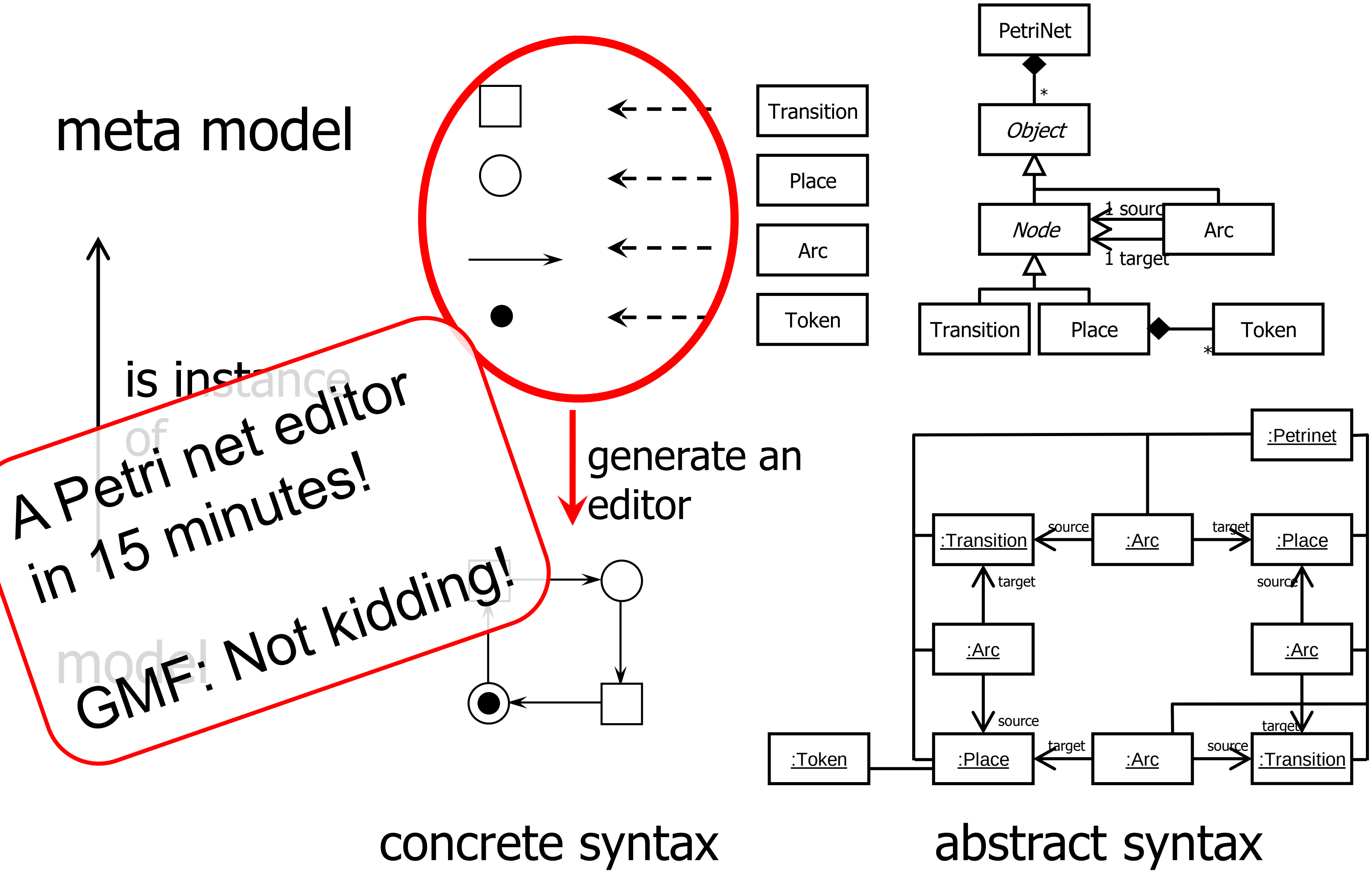
VII. Modelling Behaviour (cntd.)

Vision (reminder from L01)



Exploit conceptual artefacts (and generate software from them).

Today (reminder from L01)



meta model

is instance of model

A Petri net editor in 15 minutes!

GMF: Not kidding!

generate an editor

concrete syntax

abstract syntax

Benefits of Modelling (today)

- 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

How about “real” functionality / behaviour?

Motivation

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

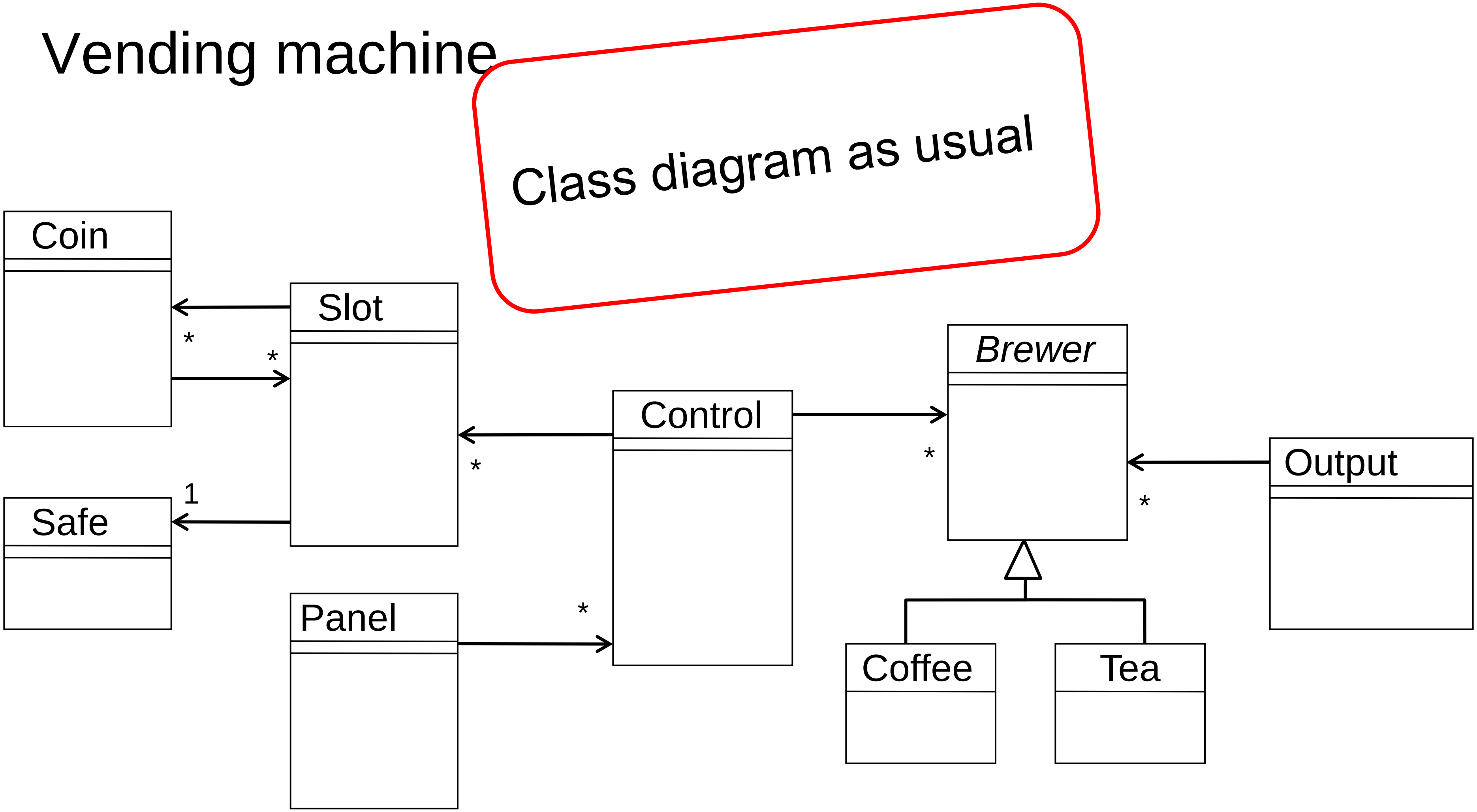
- Model behaviour on top of it – and make these models executable
- Model behaviour on a high level of abstraction (domain level)

→ Integrate behaviour models with structural models

→ Integrate different structural models (even from different technologies and without underlying models)

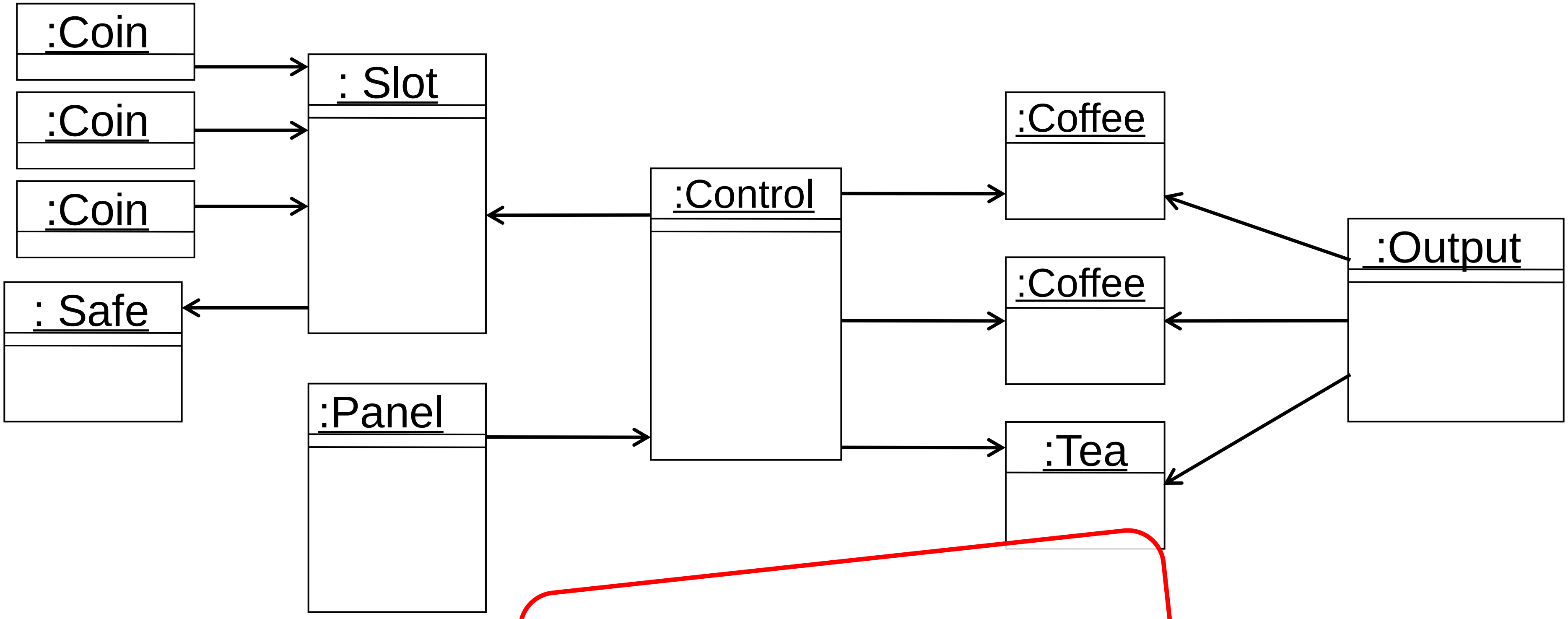
5.1 Example

Vending machine



Class diagram as usual

Instance: object diagram



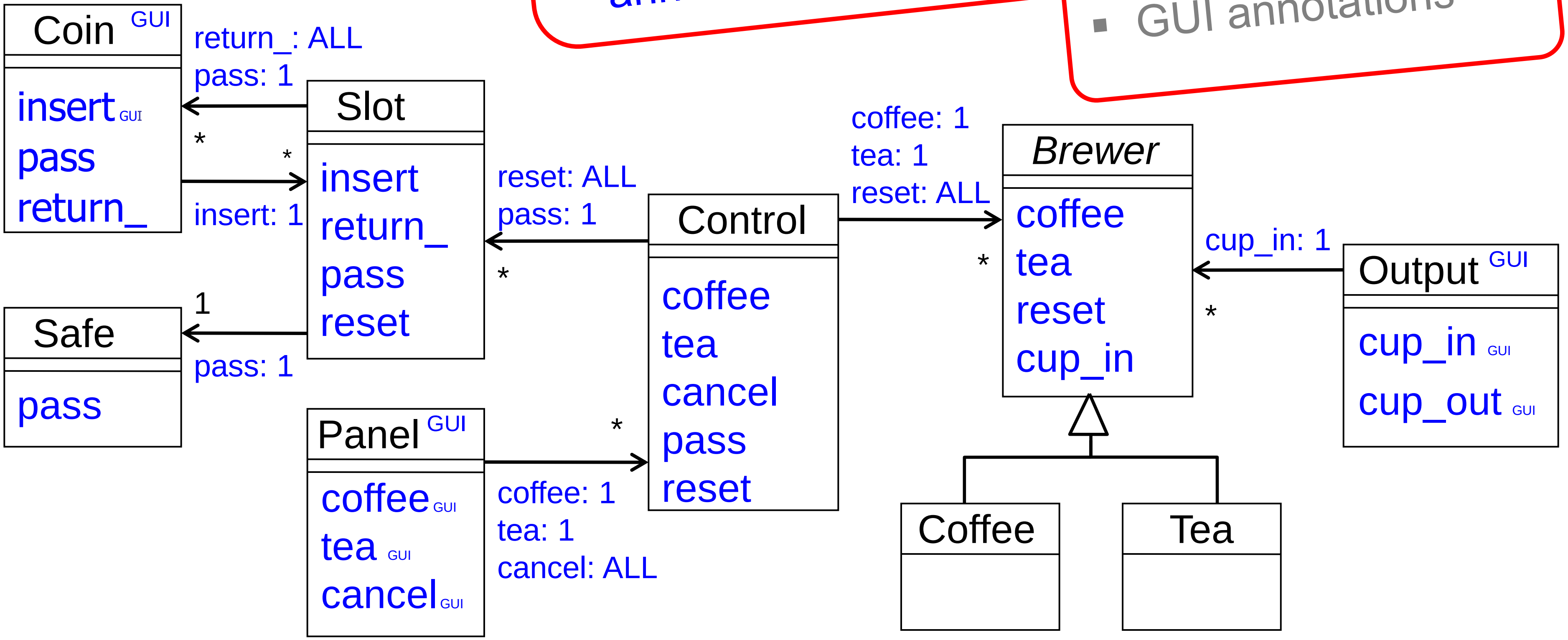
Object diagram as usual

Coordination Diagram

- We call objects elements now!

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

- GUI annotations



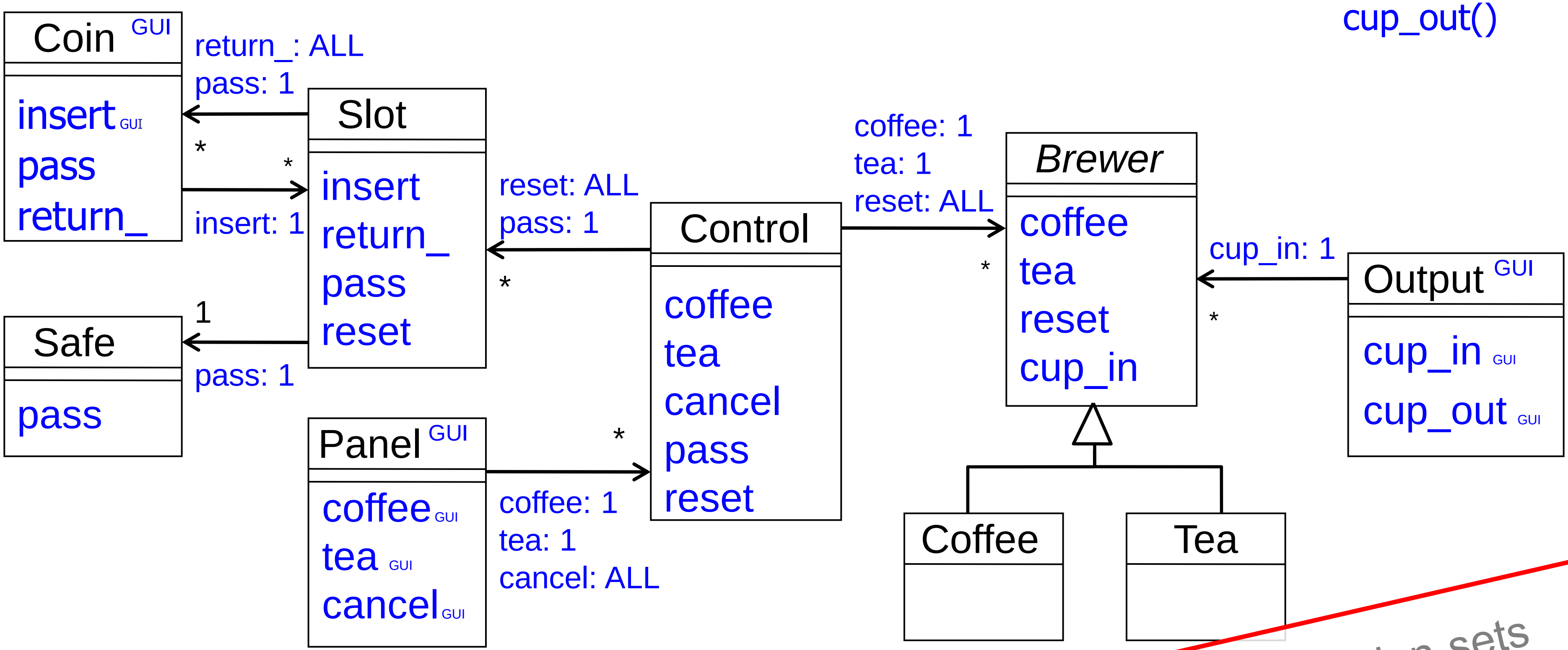
... + Event declaration

- Event (type) declaration
- Parameters

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

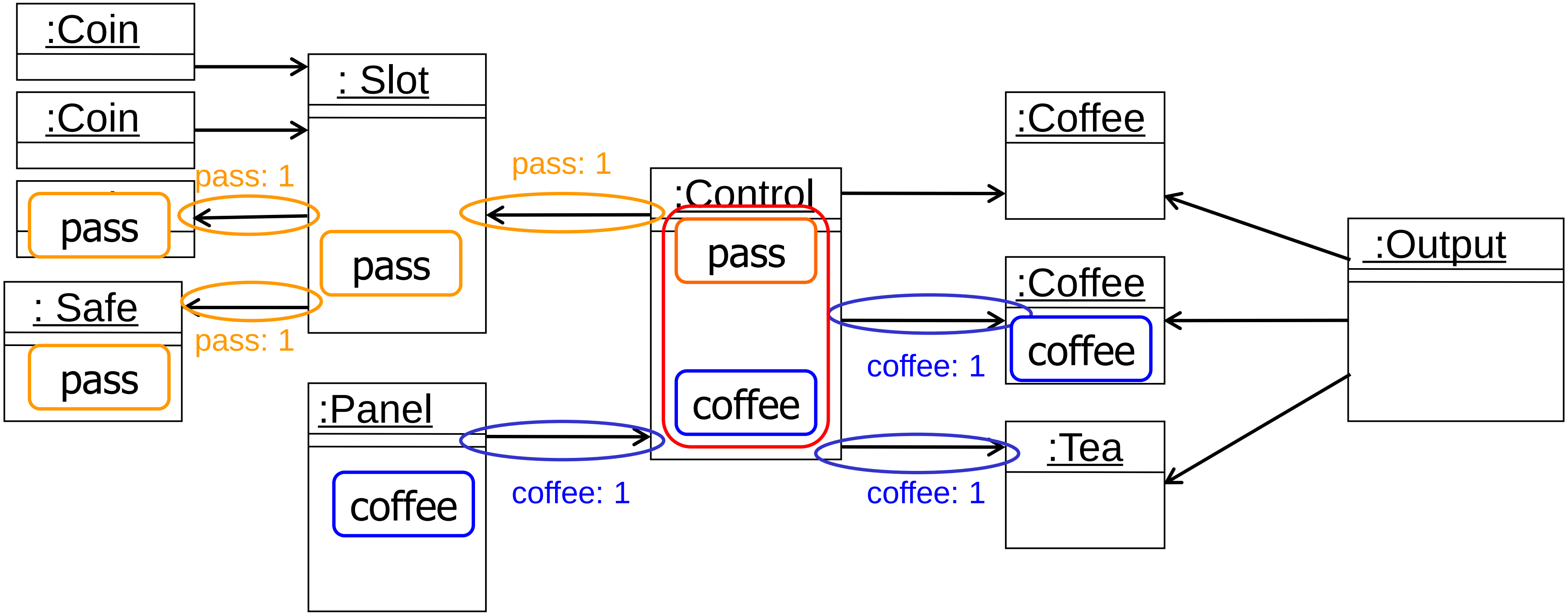
coffee()
tea()
cancel()

cup_in()
cup_out()



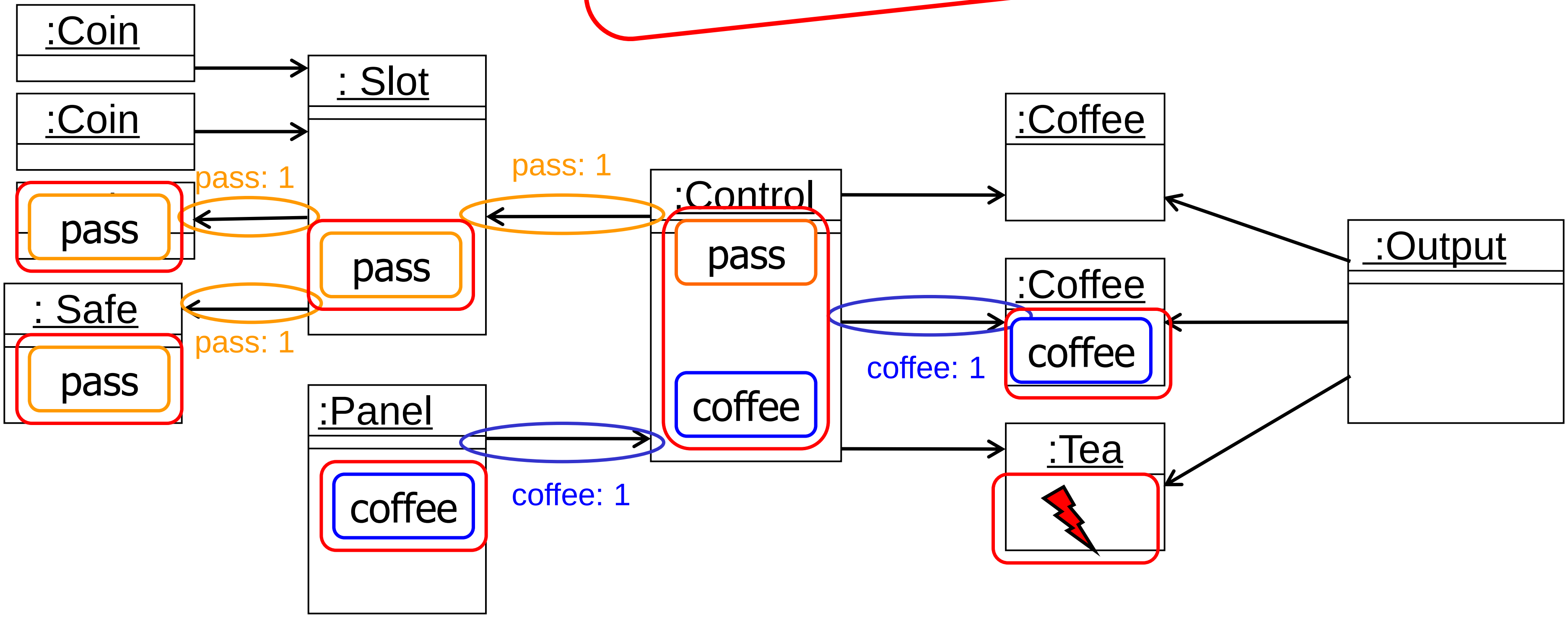
- Coordination sets (not discussed here)

Interactions



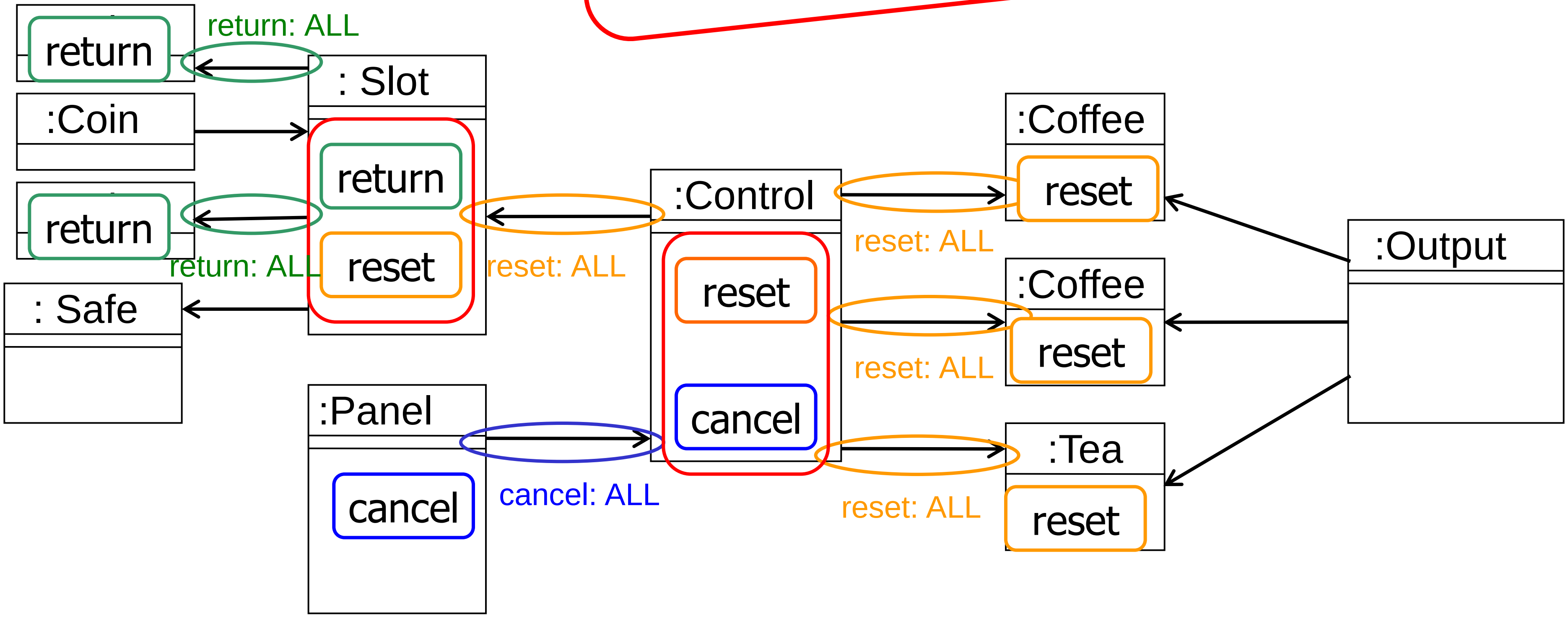
Interactions

Interaction =
local behavior +
coordination



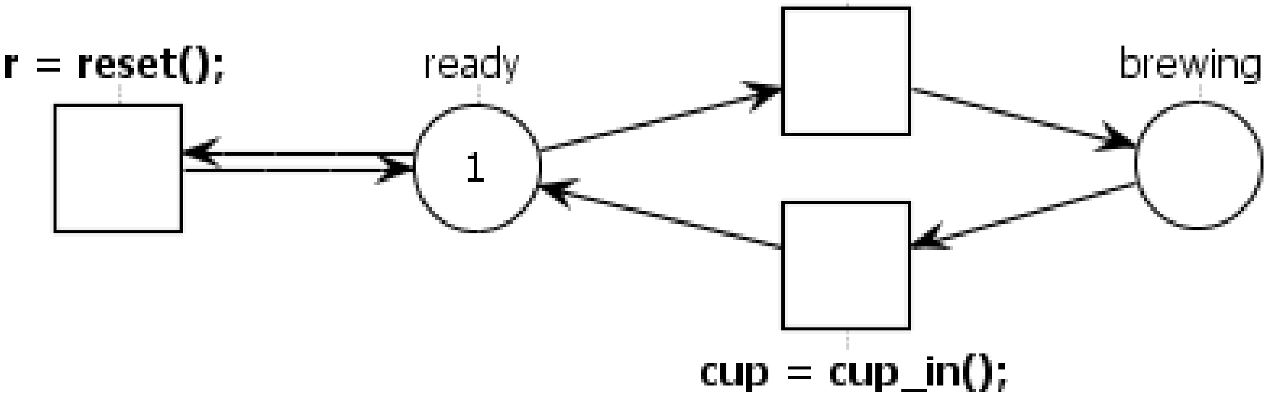
Another Interaction

Interaction =
local behavior +
coordination



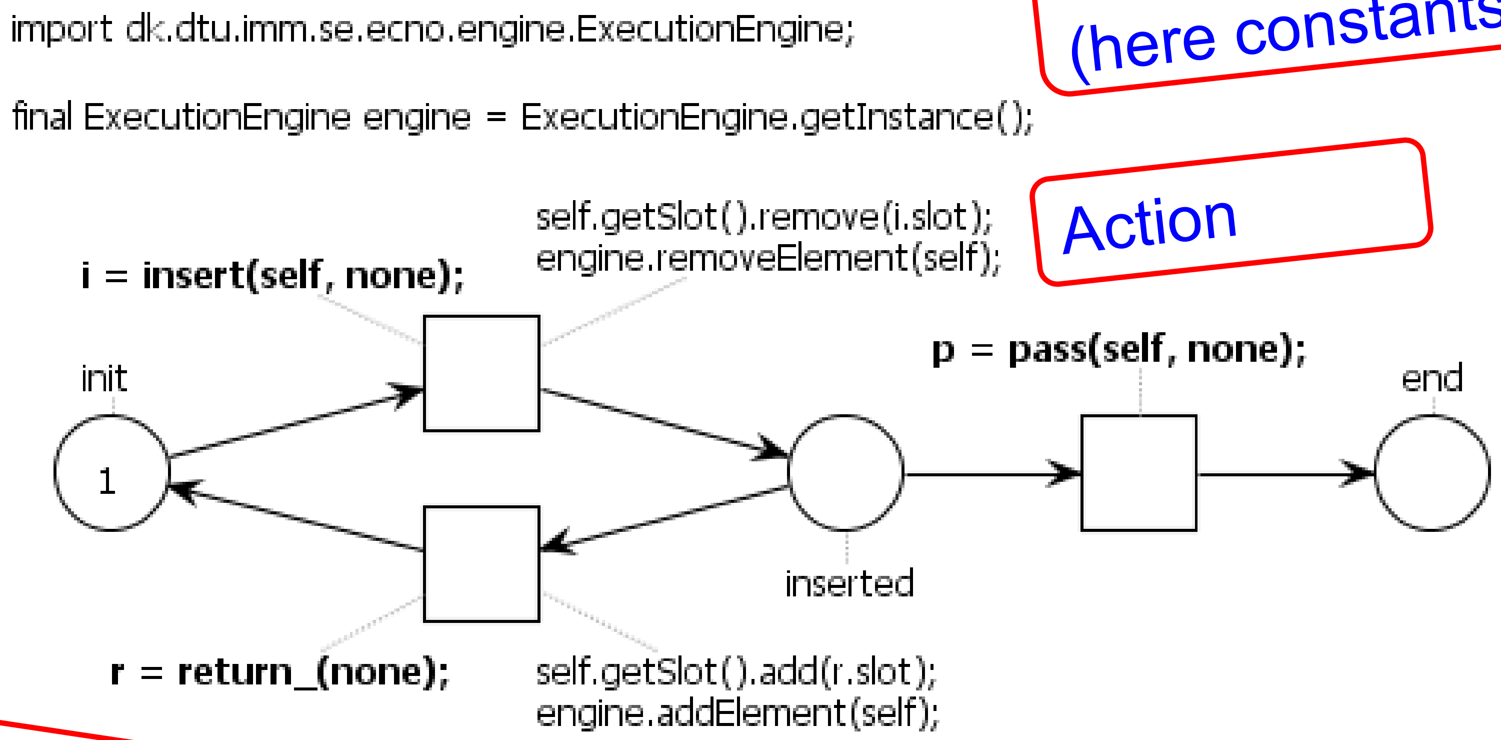
Local behaviour: Coffee

Event binding



Local behaviour: Coin

Attribute declaration (here constants)



```
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);
```

```
p = pass(self, none);
```

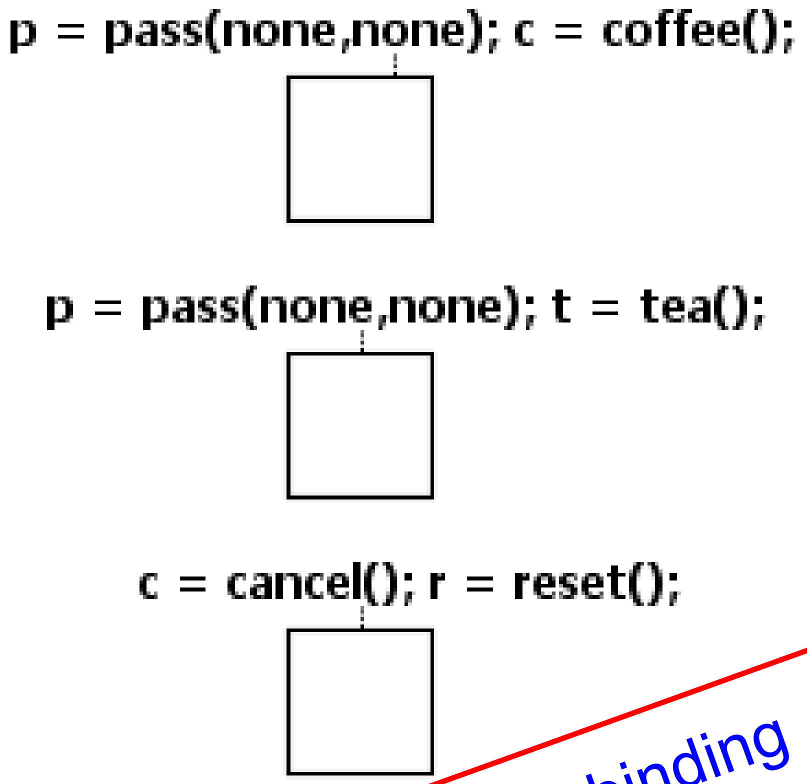
```
r = return_(none);
```

```
self.getSlot().add(r.slot);
engine.addElement(self);
```

Action

- Event binding
- Parameter assignment

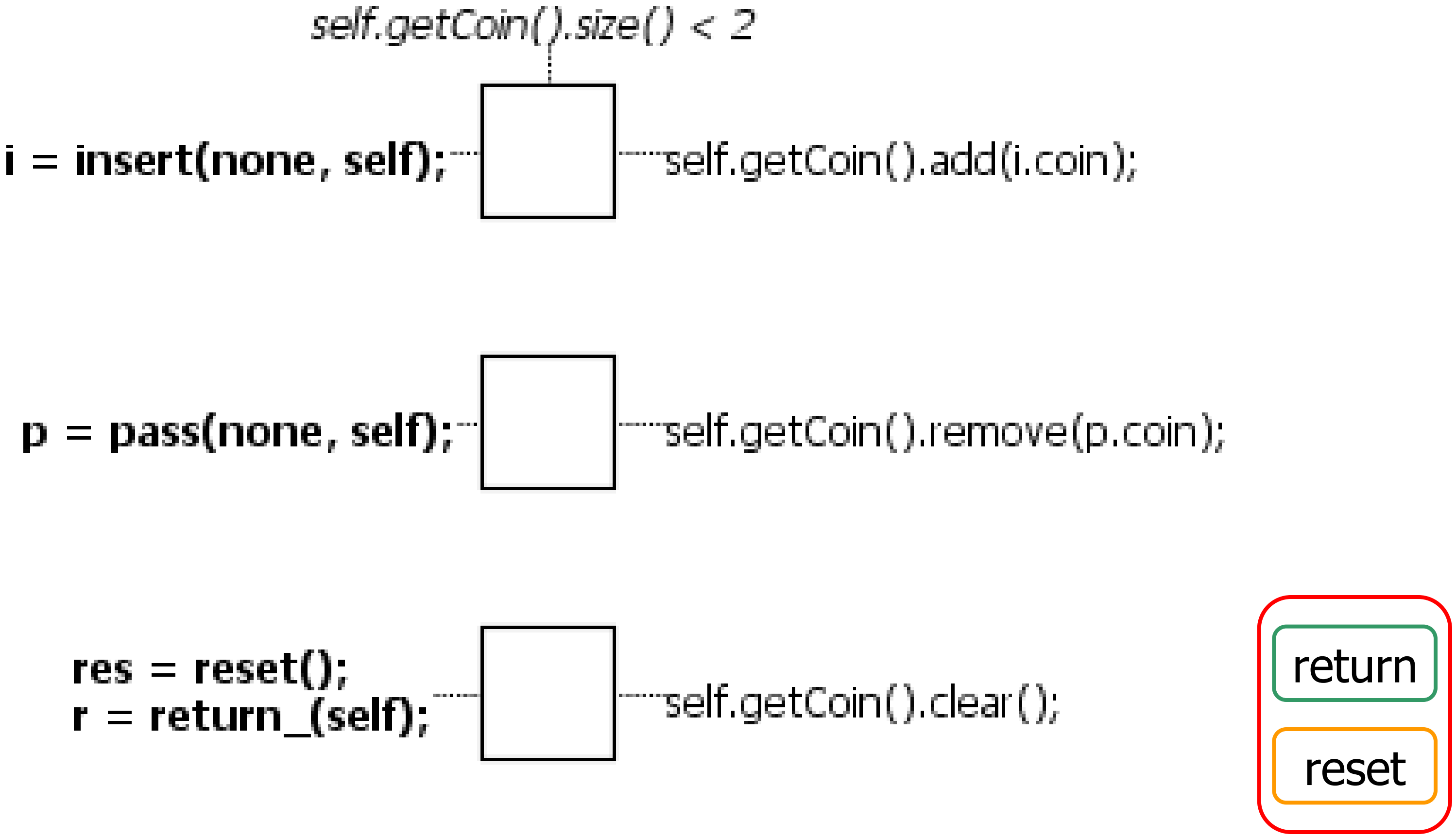
Local behaviour: Control


p = pass(none,none); c = coffee();
p = pass(none,none); t = tea();
c = cancel(); r = reset();

pass
coffee
reset
cancel

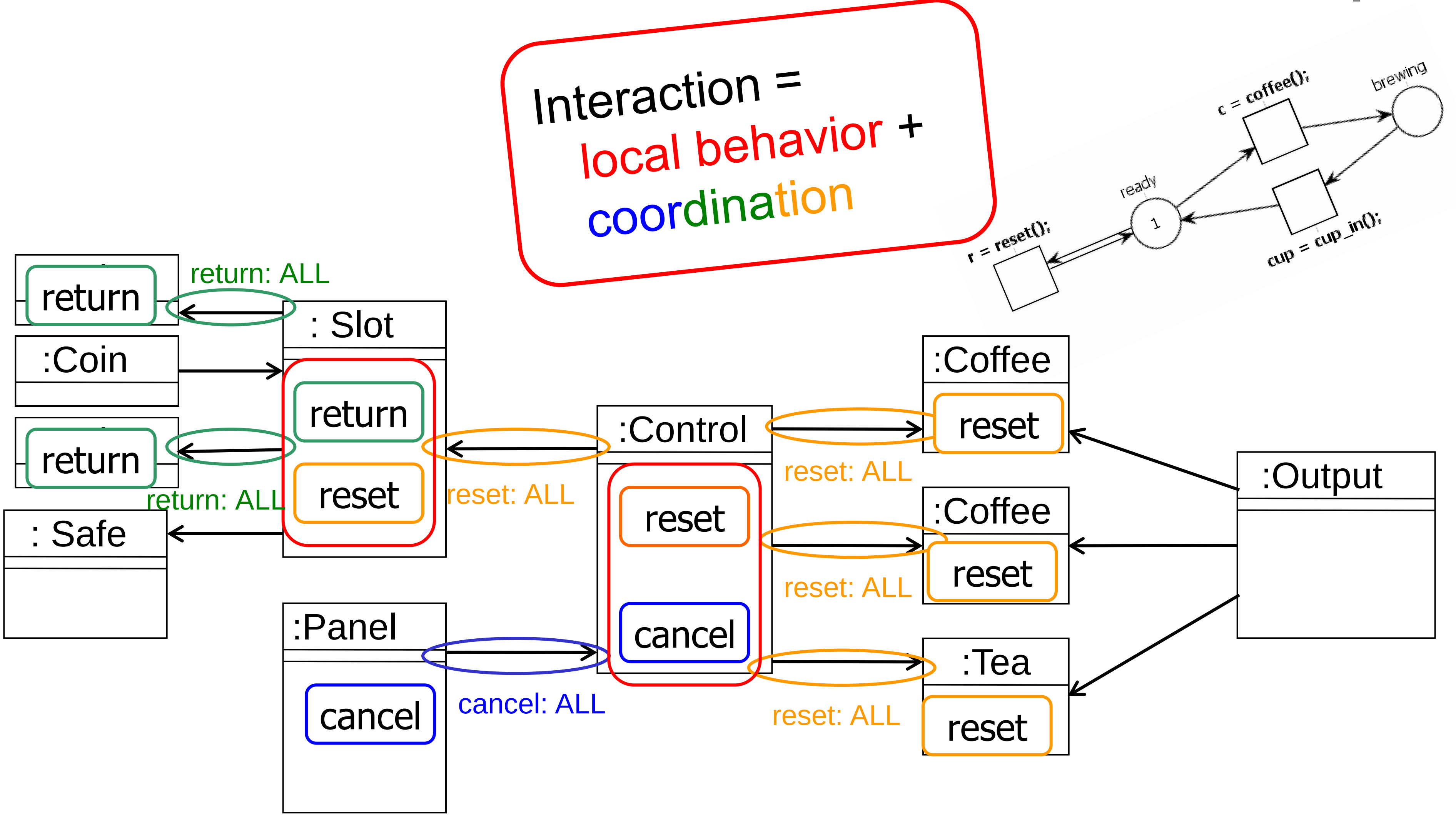
- Event binding with multiple event types!

Local behaviour: Slot



```
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();
```

Interactions



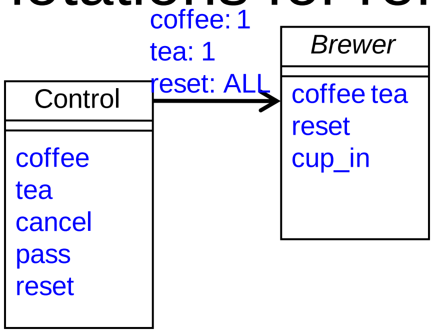
Interaction = local behavior + coordination

5.2 ECNO: Basic Concepts

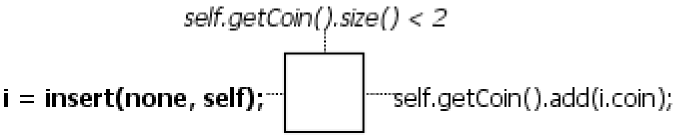
- ElementTypes (Classes)
- EventTypes with
  - parameters

insert(Coin coin, Slot slot)

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



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



```
self.getCoin().size() < 2
i = insert(none, self);
self.getCoin().add(i.coin);
```

ECNO: Basic Concepts

- ElementTypes (Classes)
- EventTypes with
  - parameters

Parameter passing is different from classical method invocations!

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

Interactions can span many elements; which depends on the current situation. Circles are possible.

- Local behaviour: Or something else
  - Event binding (with parameter assignment)
  - Condition
  - Action

ECNO nets are but one way of modelling local behaviour.

5.3 Extensions

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

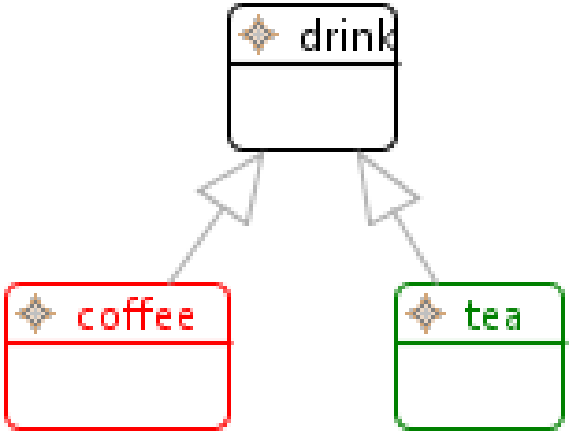
- Right now, for one event type we need to consider all coordination annotations for that event type starting from the element.

  Sometimes, we just want to follow either one or another or a subset together.

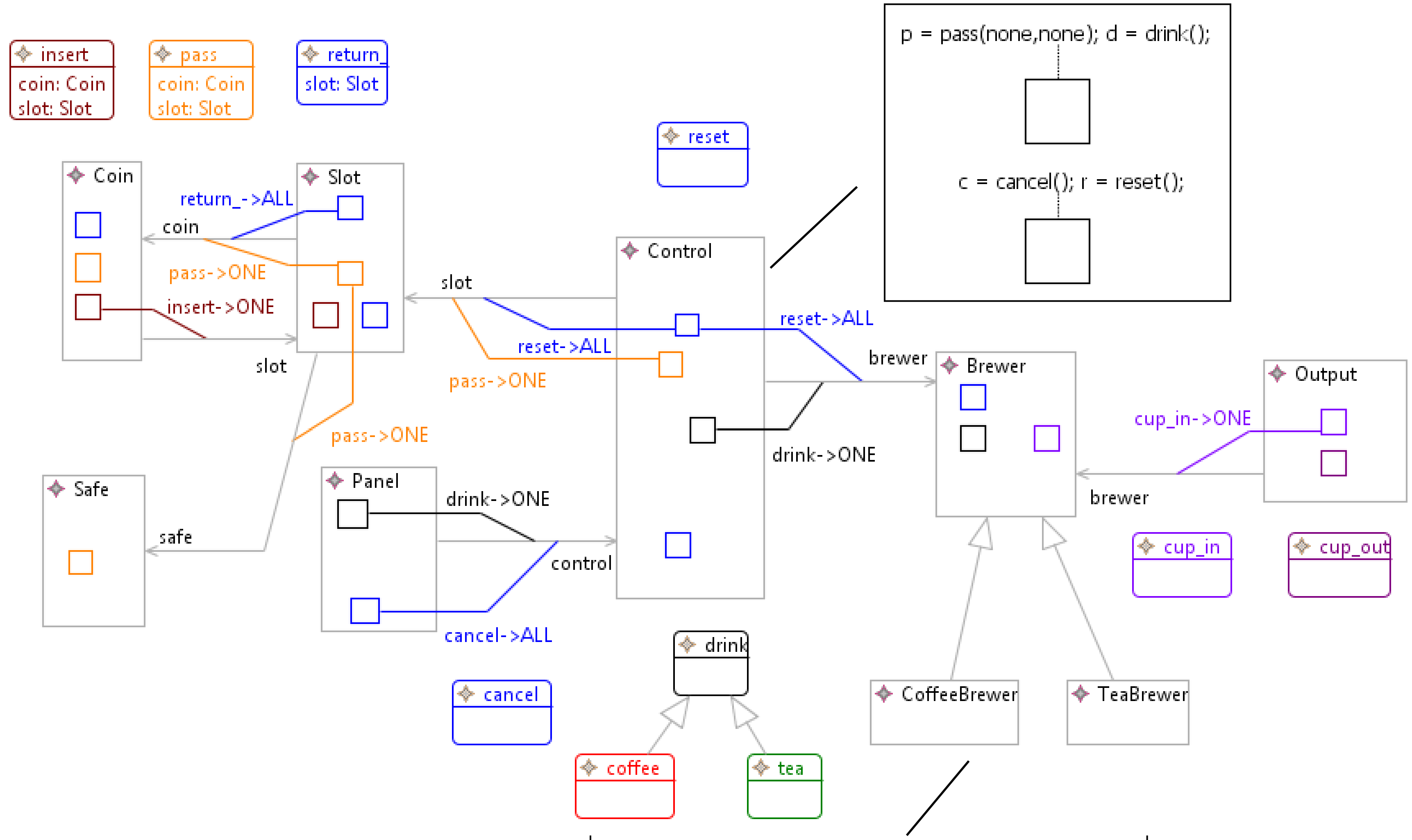
Limitations (cntd.)

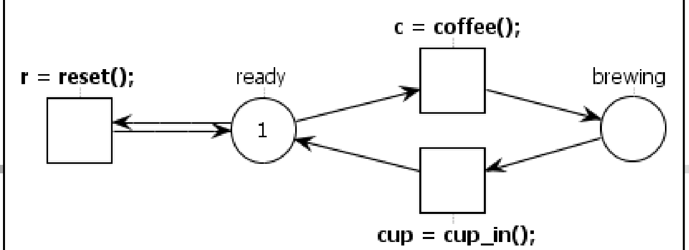
ECNO with its basic concepts has some limitations, which makes modelling things **in an adequate way** a bit painful.

- Sometimes, we want to extend event types later



”Nicer Vendingmachine”



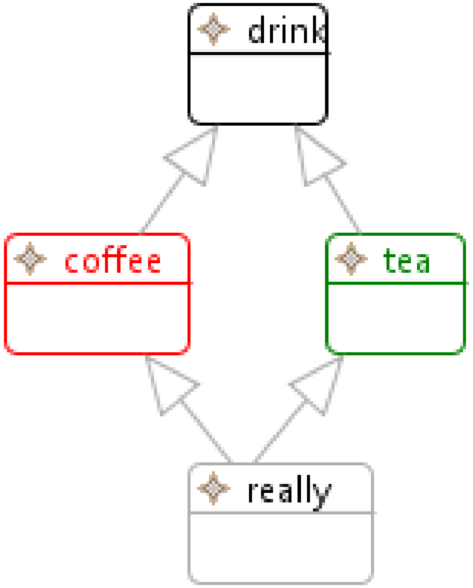


Event Inheritance

Question: Would we like to have multiple inheritance on event types?

Problems:

- We could never be sure that two event types that were meant to be different are different!
- We would not know which event type an instance of subtype would represent!

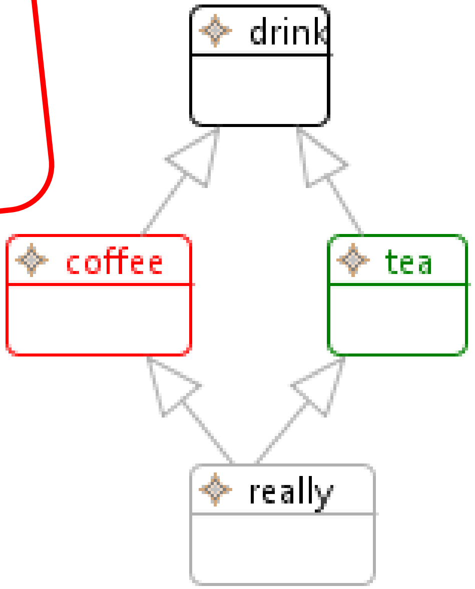


Avoid confusion! Without a really compelling argument, do not introduce multiple inheritance on event types.

Event Inheritance

Question: Would we like to have multiple inheritance on event types?

Avoid confusion! Without a really compelling argument, do not introduce multiple inheritance on event types.

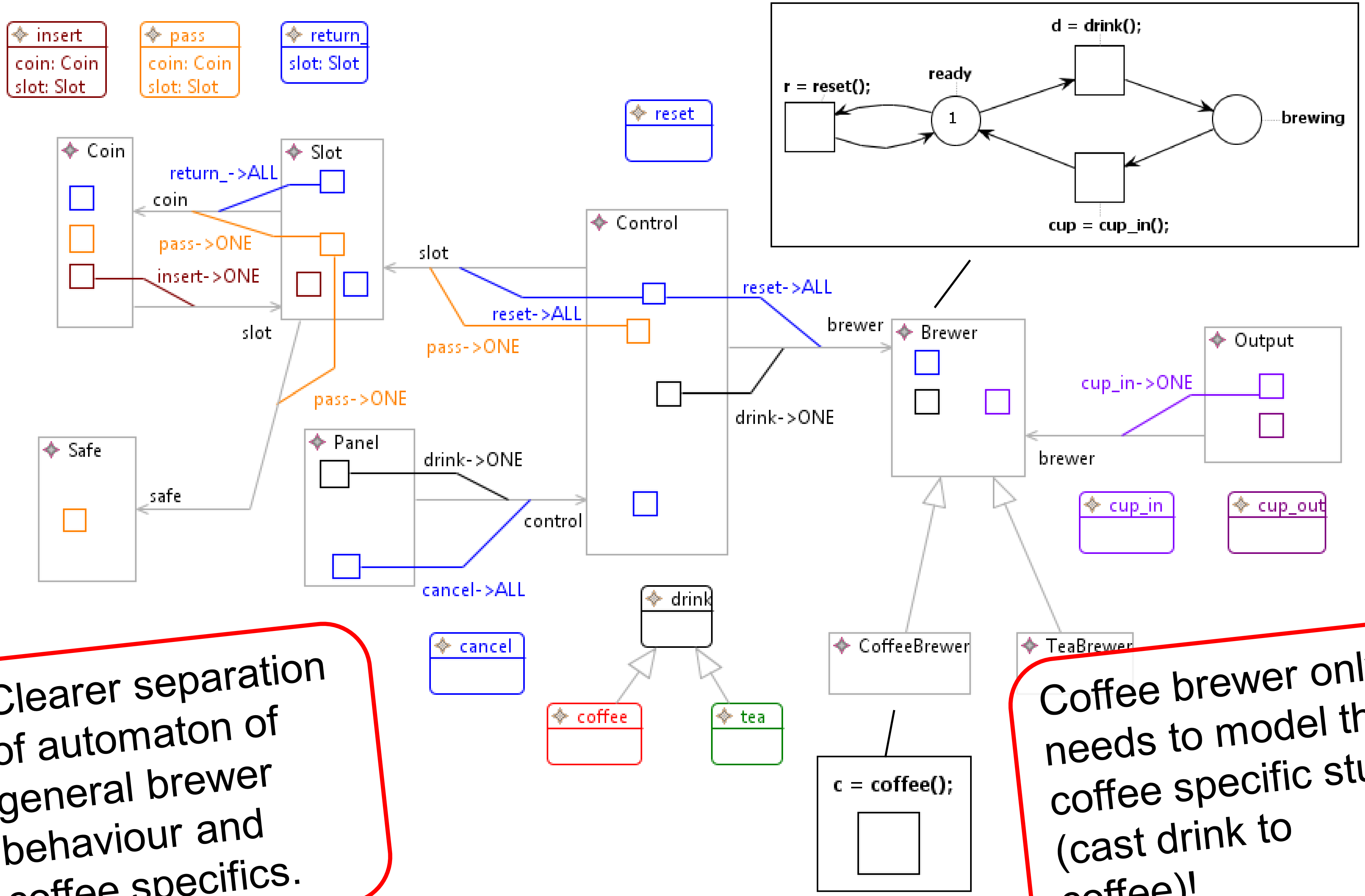


Problems:

- We could never be sure that two event types that were meant to be different are different!
- We would not know which event type an instance of subtype would represent!

We also do not introduce a top-level event type (like Object in Java) from which all inherit. Why?

Behaviour inheritance



Clearer separation of automaton of general brewer behaviour and coffee specifics.

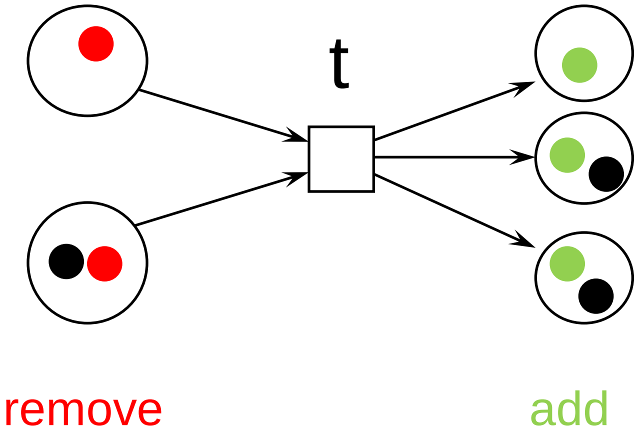
Coffee brewer only needs to model the coffee specific stuff (cast drink to coffee)!

Behaviour inheritance

- Every level in the element type hierarchy of an element can have a behaviour. These behaviours will be synchronized, and jointly executed.

- Only if the behaviour on all levels can participate (has a choice) for an event, the element can participate in this event.

5.4 Another example: Petri nets



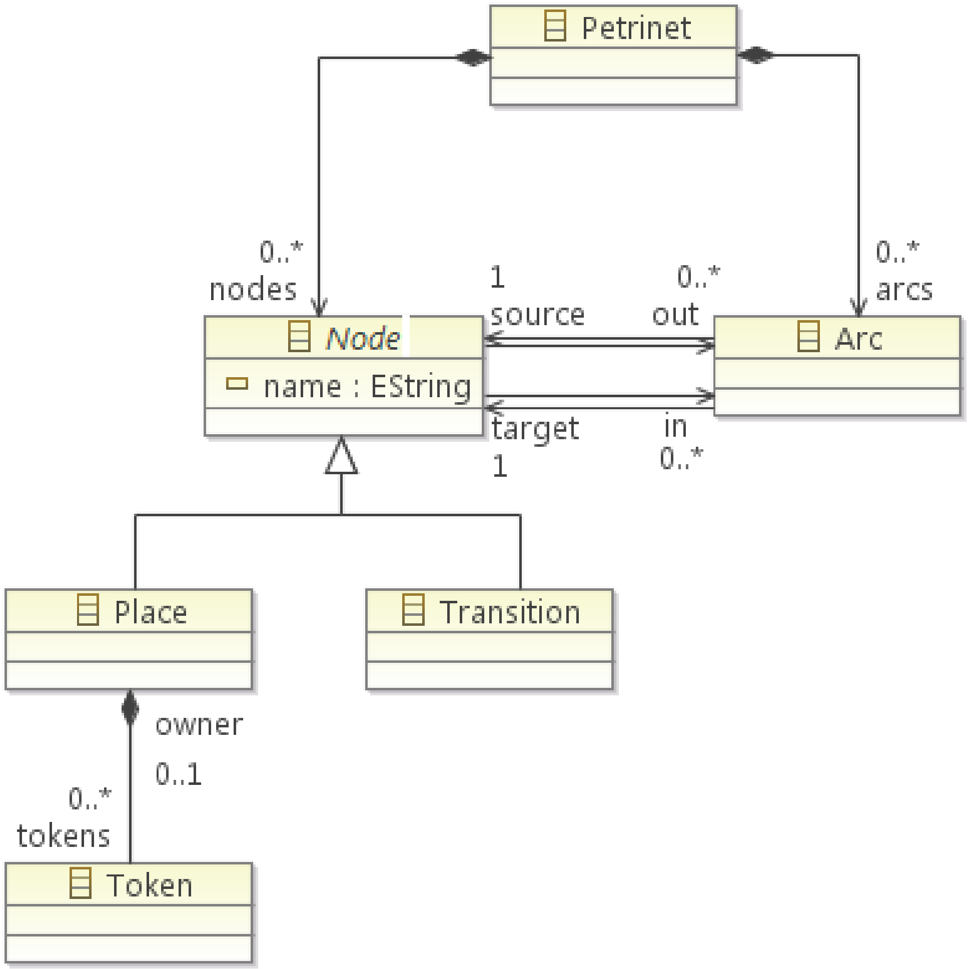
How can we model that behaviour in ECNO nets?

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

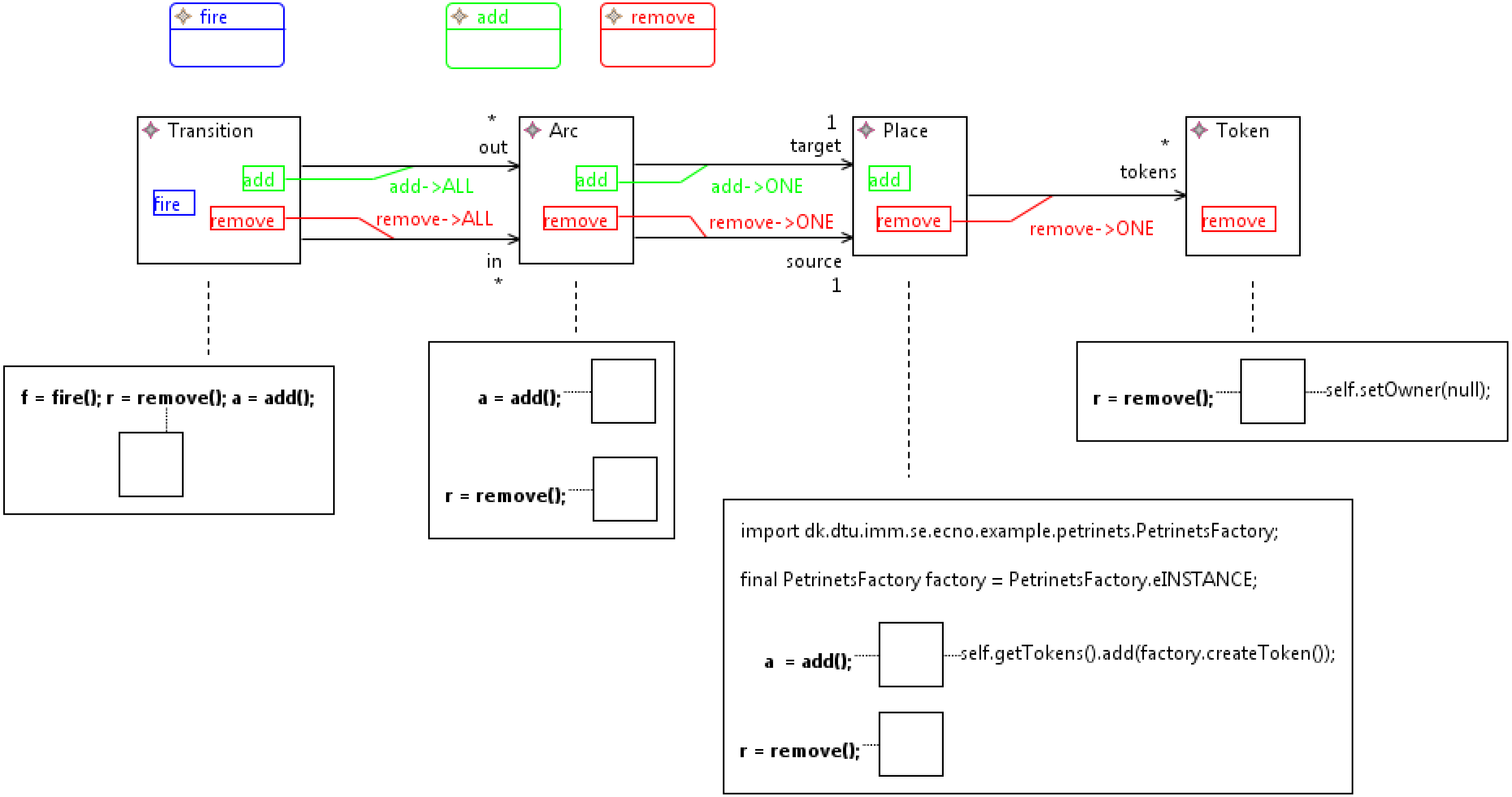
```
**fire** Transition t:
  for ALL incoming Arcs a:
    for ONE source Place p of Arc a:
       find a token and remove it

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

Petri net: Class Diagram



ECNO Semantics of PN



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

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

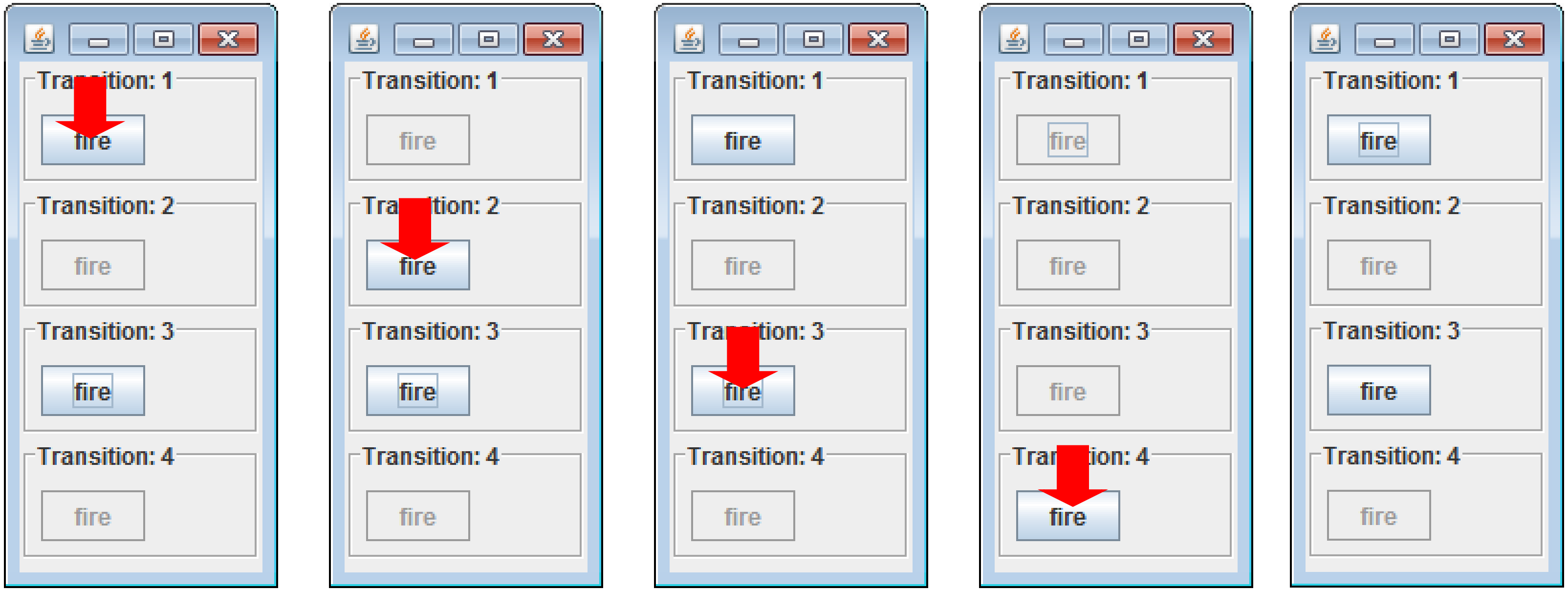
```
r = remove(); self.setOwner(null);
```

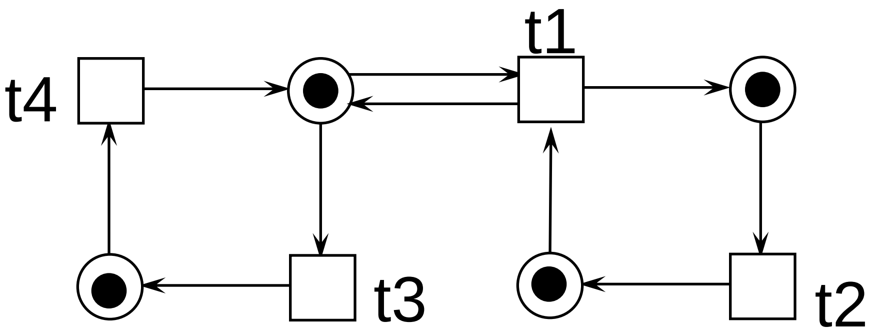
```
import dk.dtu.imm.se.ecno.example.petrinets.PetrinetsFactory;

final PetrinetsFactory factory = PetrinetsFactory.eINSTANCE;

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

r = remove();
```





With the ECNO concepts as presented up to now, t1 would never be enabled! Why?

ECNO: Student projects

- ACI**D**: Run ECNO on top of a data base (hibernate for persisting the current state)

- **PDE support**:
  - Debugging ECNO models
    - visualize enabled and not-enabled interactions
    - formulate conditions / create break points
  - Better integration with Java

- **DSL for GUI** of ECNO application:
  (cf. project 3 → larger project (e.f WFMS))

ECNO: Student projects

- Theory:
  - **formalisation** of ECNO semantics and verification
  - formalisation of ECNO semantics in ECNO itself (which concepts are needed for that)
- Methodology:
  - Larger examples
  - Best practices: make good use of the features
  - ECNO for which kinds of systems
  - ...
- ...

VIII. Summary and Outlook

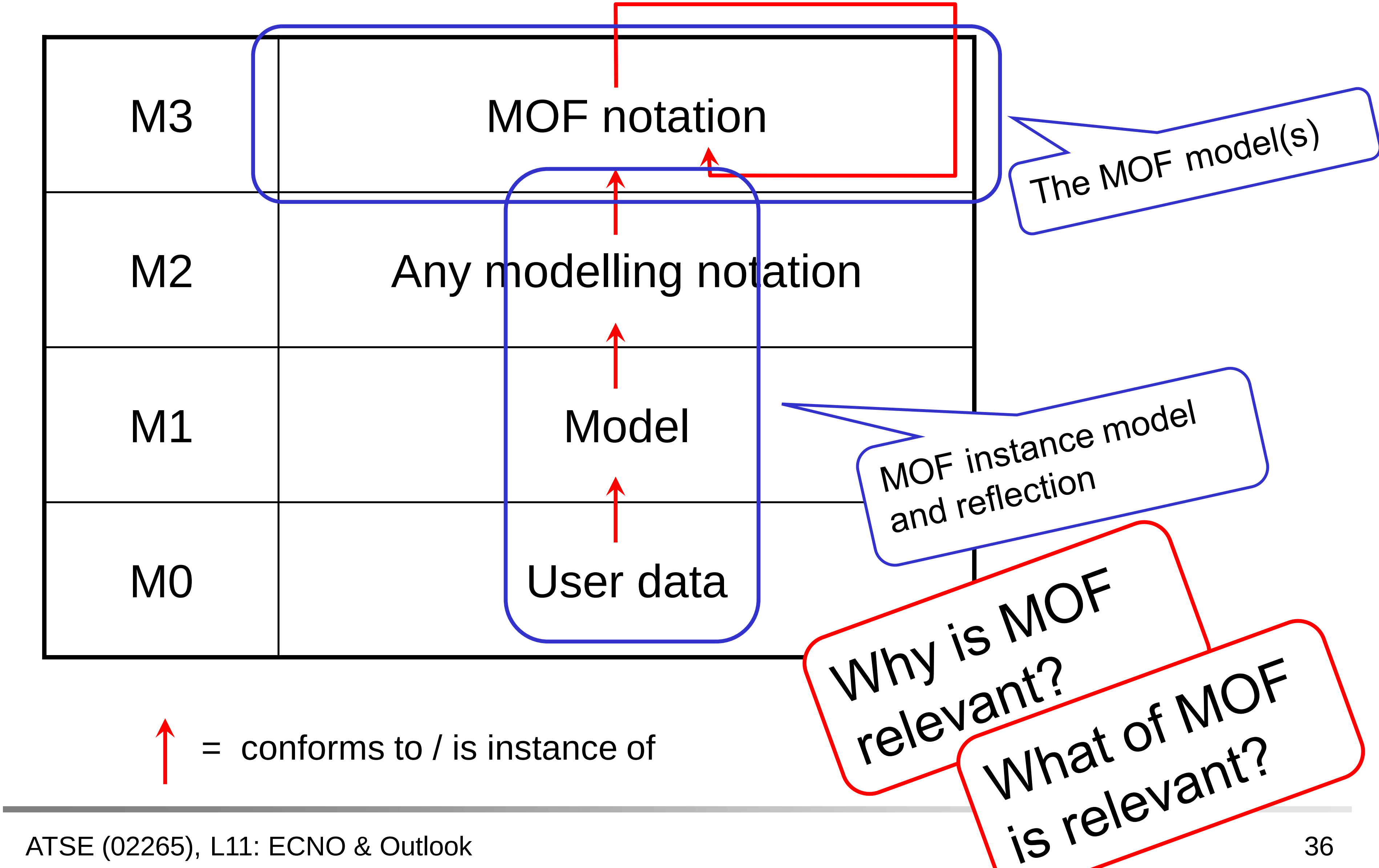
Overview

- Modelling
  - OCL
  - Automata
  - Petri nets
  - BPM (concepts only)
- Meta-modelling (MOF)
  - self-referential
  - self-describing (M3)

Works nicely for structure; but not so nicely for behaviour yet!

- Transformation & synchronisation of models
  - M2T (JET, Codegeneration)
  - M2M (TGG & QVT)
- DSLs (Domain Specific Languages)

The Meta Object Facility (MOF)



M3	MOF notation
M2	Any modelling notation
M1	Model
M0	User data

↑ = conforms to / is instance of

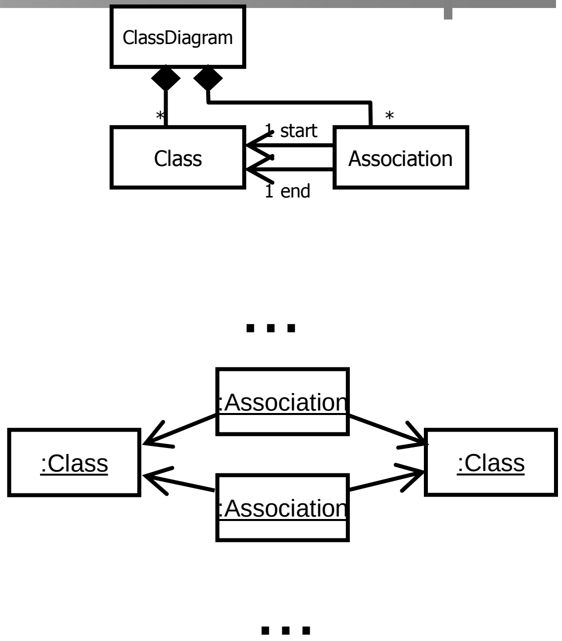
The MOF model(s)

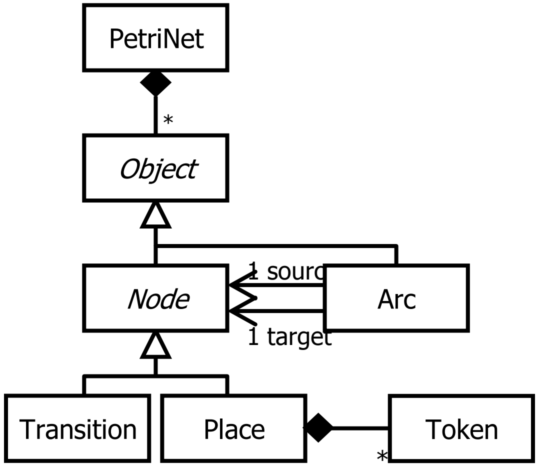
MOF instance model
and reflection

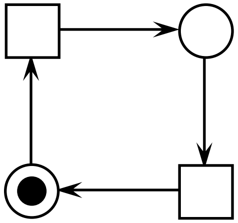
Why is MOF
relevant?

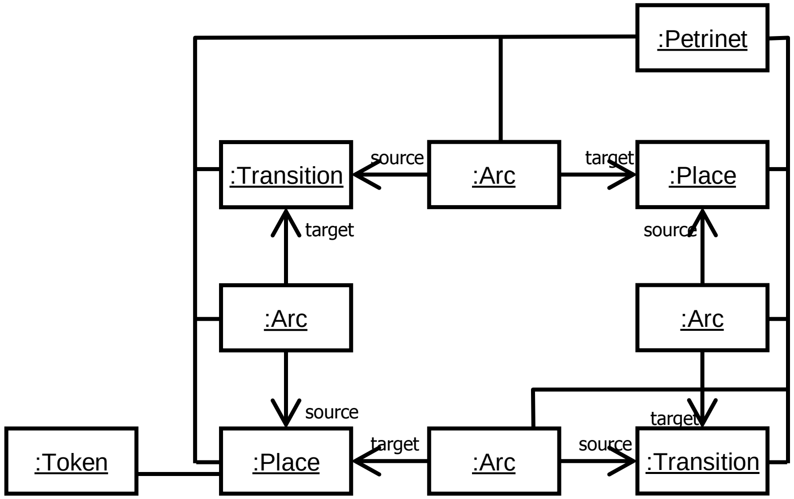
What of MOF
is relevant?

Meta-levels


ClassDiagram
*
*
Class
1 start
1 end
Association
...
:Association
:Class
:Class
:Association
...


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




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

Meta modelling

- understanding and clarifying concepts and making them explicy (independently of concrete syntax)
- building tools that support Model-based Software Engineering (MBSE)

On the model as well as on the meta modelling level!

- bootstrapping: developing tools in their own technology (ultimate litmus test)

Model-driven Architecture (MDA)

Use models on different levels of detail and granularity for modelling software and generating software out of them

- CIM: Computation Independent Model (conceptual)
- PIM: Platform Independet Model (technical but not platform specific)
- PSM: Platform Specific Model

what

↓

how

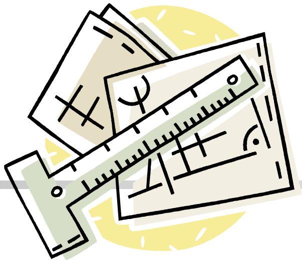
SE2: Specifying Software

Reminder

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

what

how



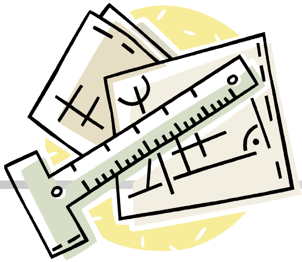
SE2: Specifying Software

Reminder

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

rough

detailed



Model-driven Architecture (MDA)

- CIM: Computation Independent Model
  (conceptual)
- PIM: Platform Independet Model
  (technical but not platform specific)
- PSM: Platform Specific Model

what

↓

how

Strictly speaking, only the first one is MDA (that's why I often use the term MBSE)

Two approaches:

- Transformation CIM → PIM → PSM → Code
- Incremental CIM + PIM + PSM → Code

Discuss:
Pro and Cons

Domain model

”Domain model” vs. ”Software model”

- A ”software model”, in a sense, models the ”how” of the software
- A ”domain model” models the ”what”

BTW:
What is a DSL?

How far can we come with domain models for making software?