

Coordinating Interactions

The Event Coordination Notation

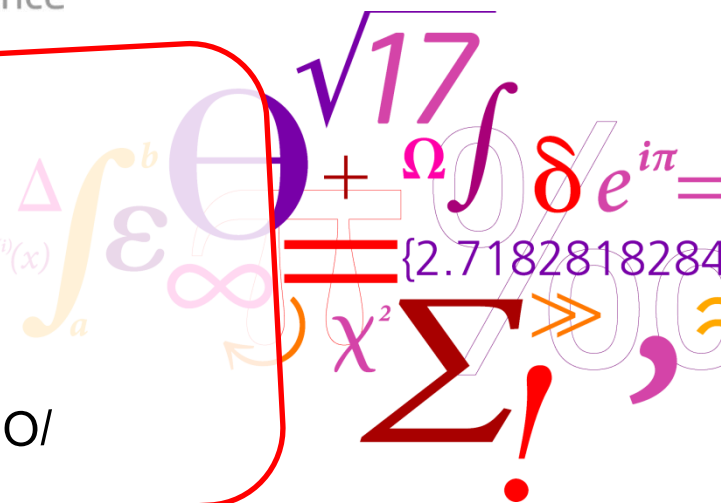
Ekkart Kindler

DTU Compute

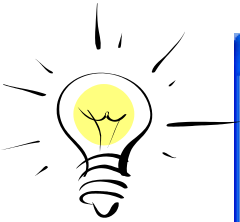
Department of Applied Mathematics and Computer Science

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

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



1. Motivation



Resource - APetriNetEditorIn15MinutesExamples/simple.petrinets_diagram - Eclipse SDK

File Edit Diagram Navigate Search Project Run Window Help

Tahoma 9 B I A 125%

Project Explorer

- APetriNetEditorIn15MinutesExamples
 - simple.petrinets
 - simple.petrinets_diagram
- SE2-ExampleProject [integration/test/tru]
- SE2-Test

simple.petrinets_diagram

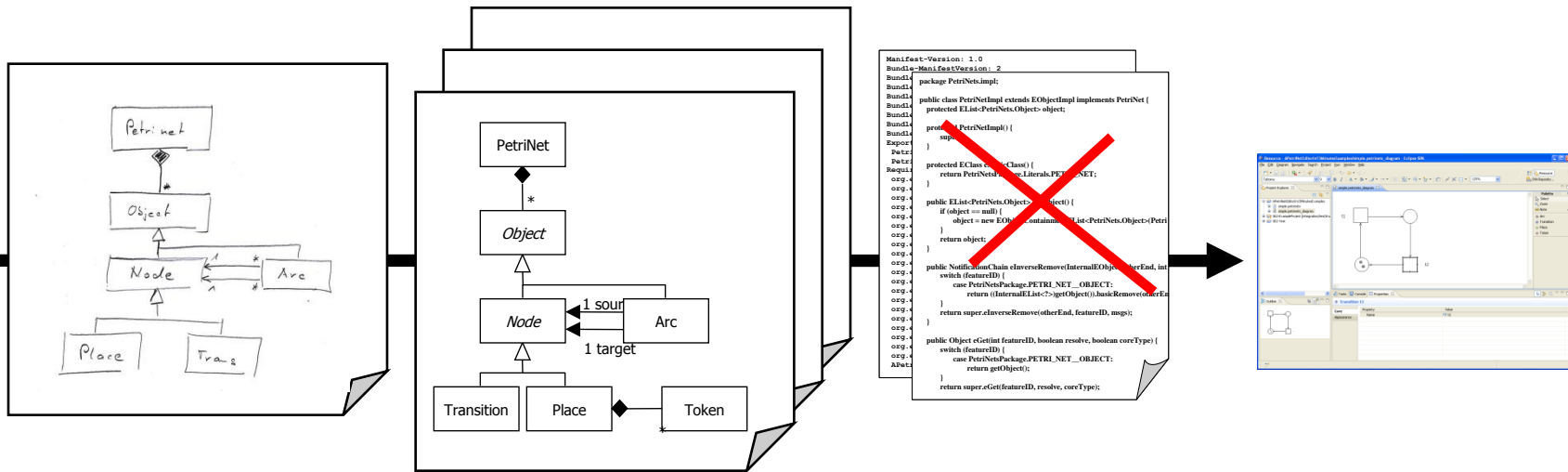
Palette

- Select
- Zoom
- Note
- Arc
- Transition
- Place
- Token

Transition t2

Core	Property	Value
Appearance	Name	t2

How about behaviour ?
(non-standard behaviour)



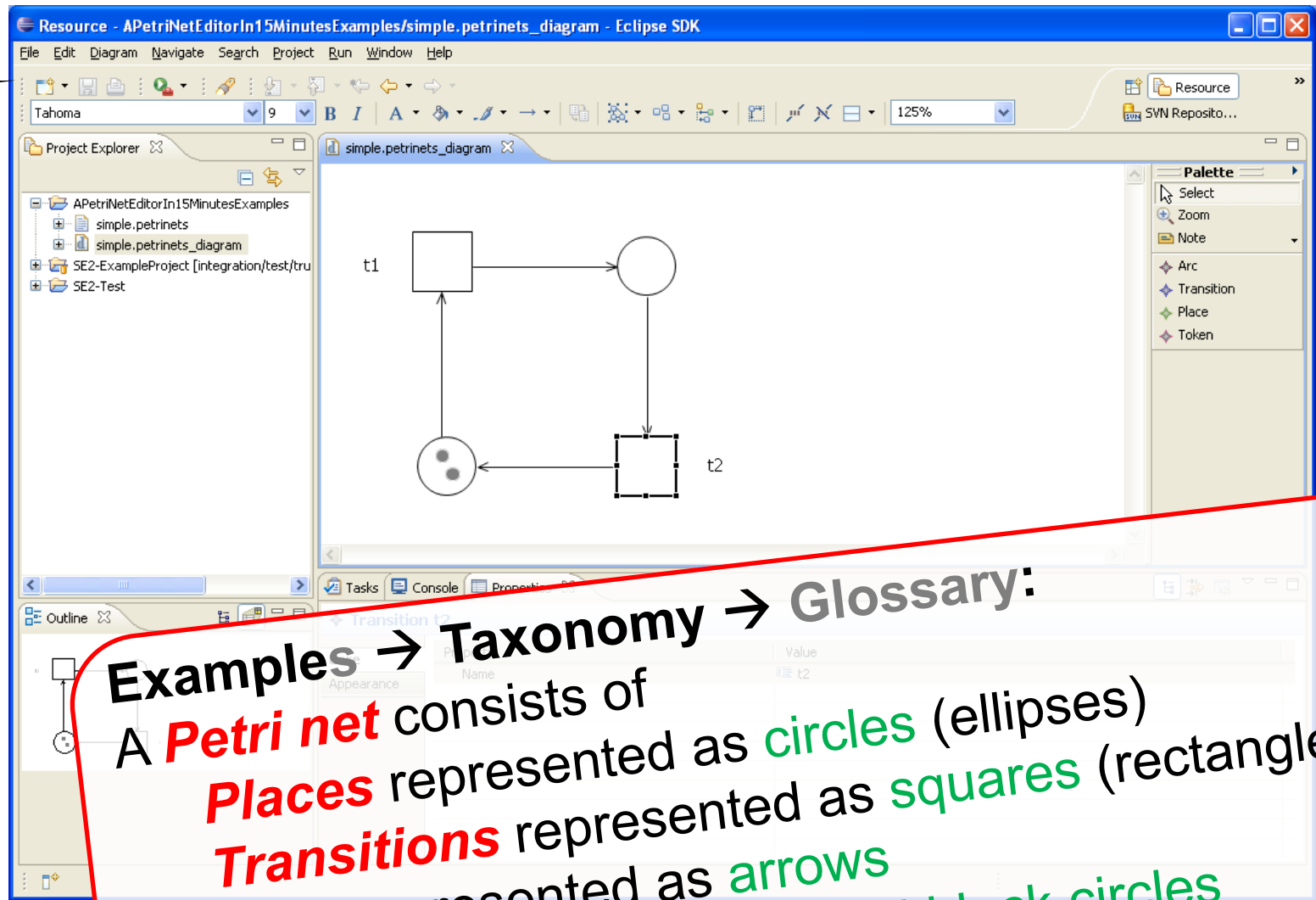
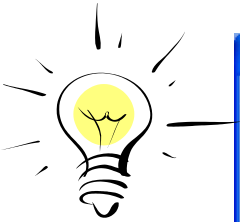
Analysis

Design

Implementation

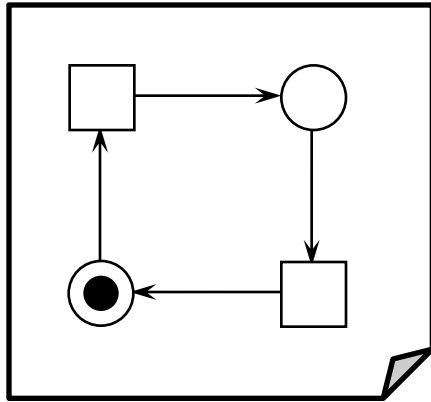
Coding

Some code
is generated



Examples → Taxonomy → Glossary:

- A **Petri net** consists of
- Places** represented as **circles** (ellipses)
- Transitions** represented as **squares** (rectangles)
- Arcs** represented as **arrows**
- Tokens** represented as **filled black circles**



Petri net model

Glossary:

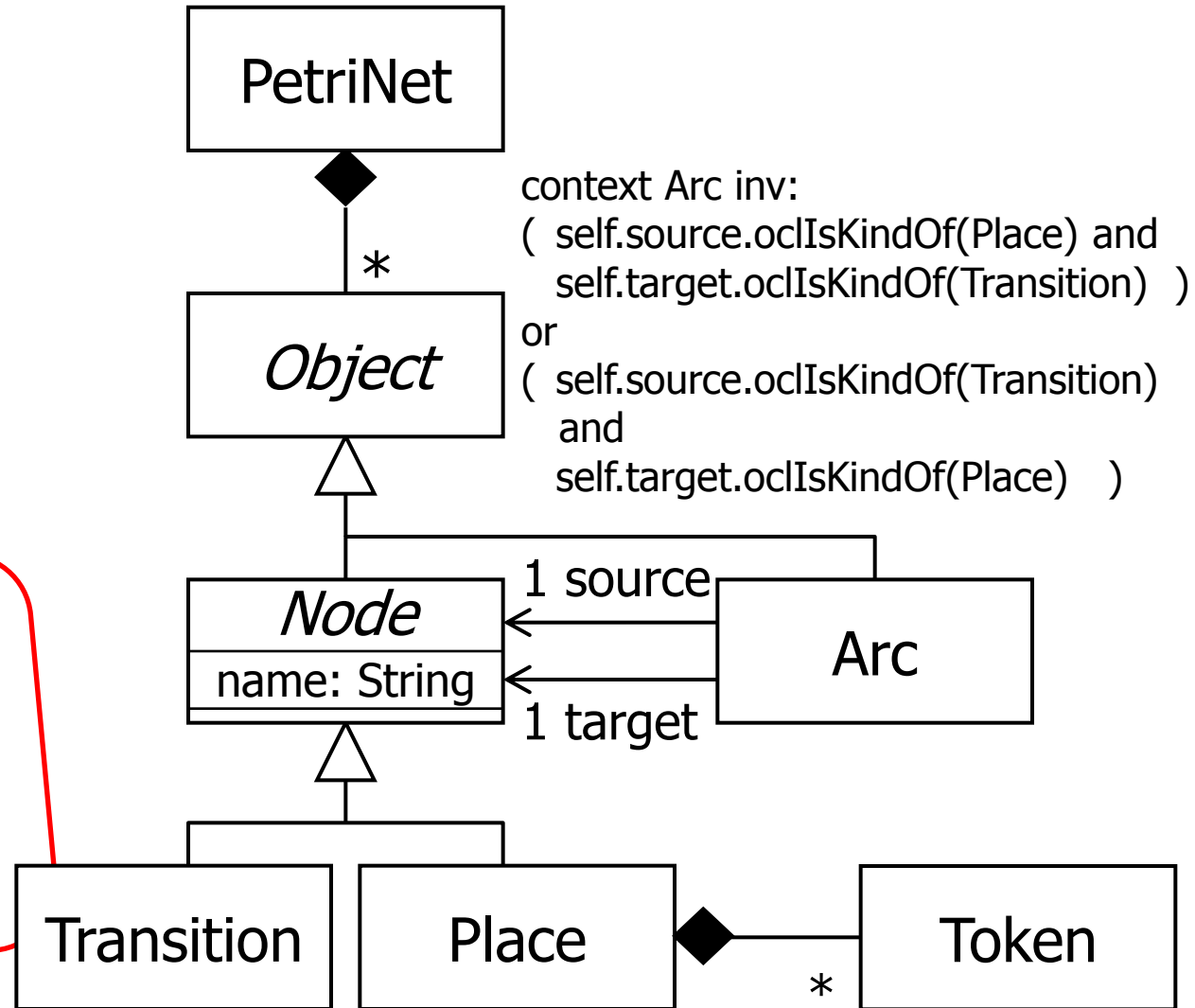
A **Petri net** consists of

Places

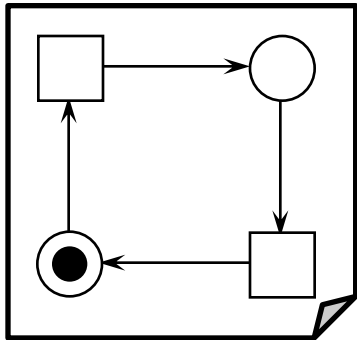
Transitions

Arcs

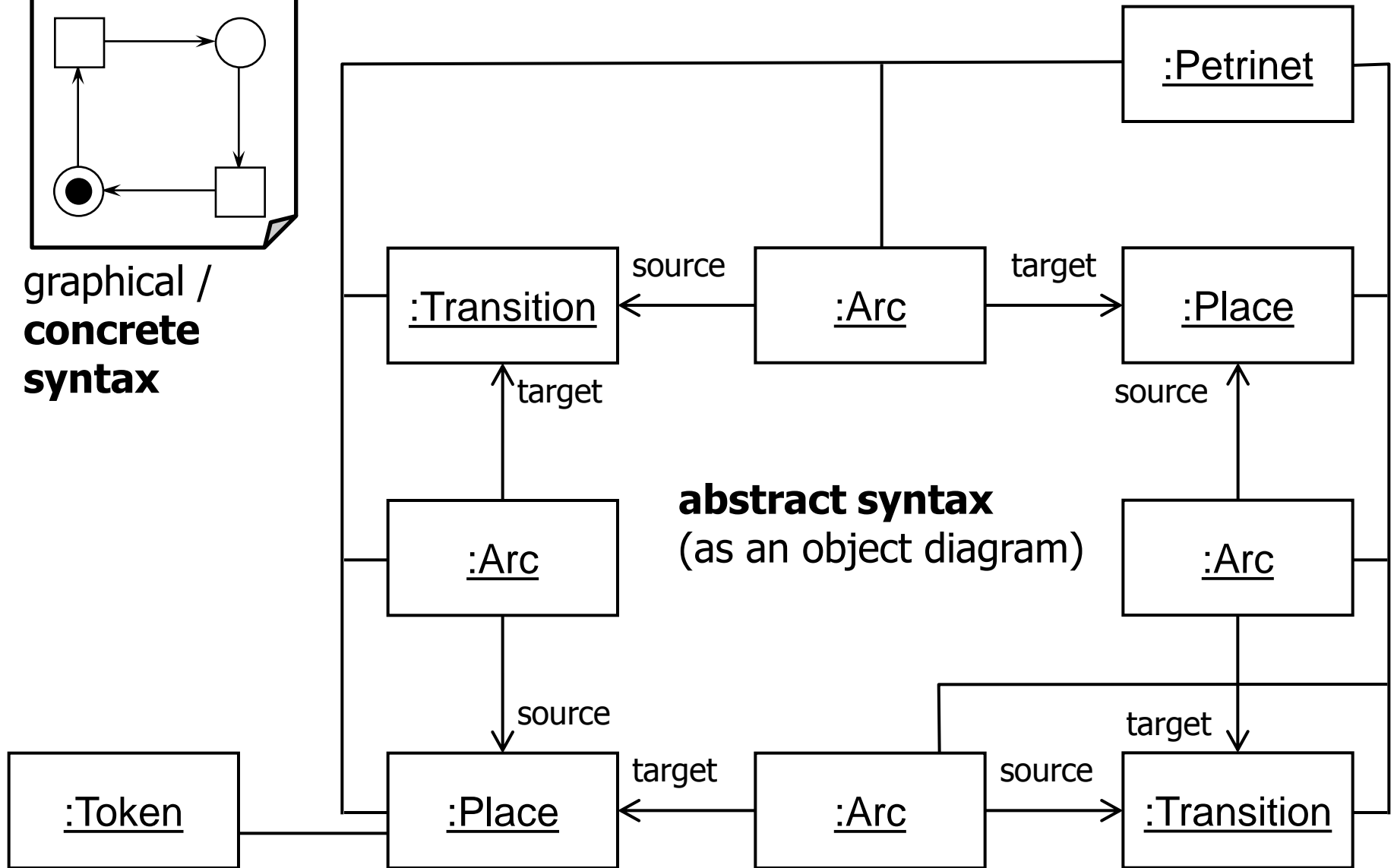
Tokens



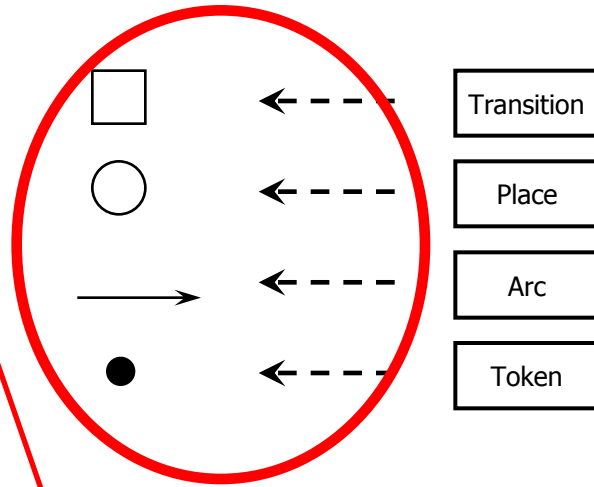
“Meta model” for Petri nets
(as a UML class diagram + OCL)



graphical /
concrete
syntax

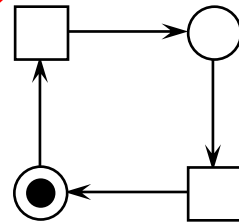


meta model

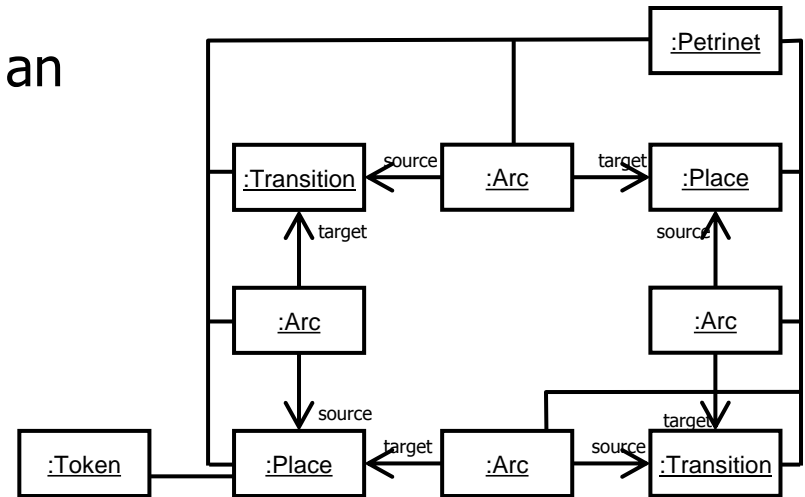
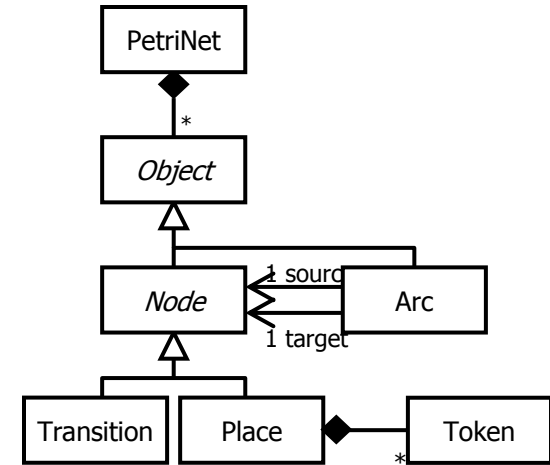


This is about all there is to say about a Petri net editor – conceptually!

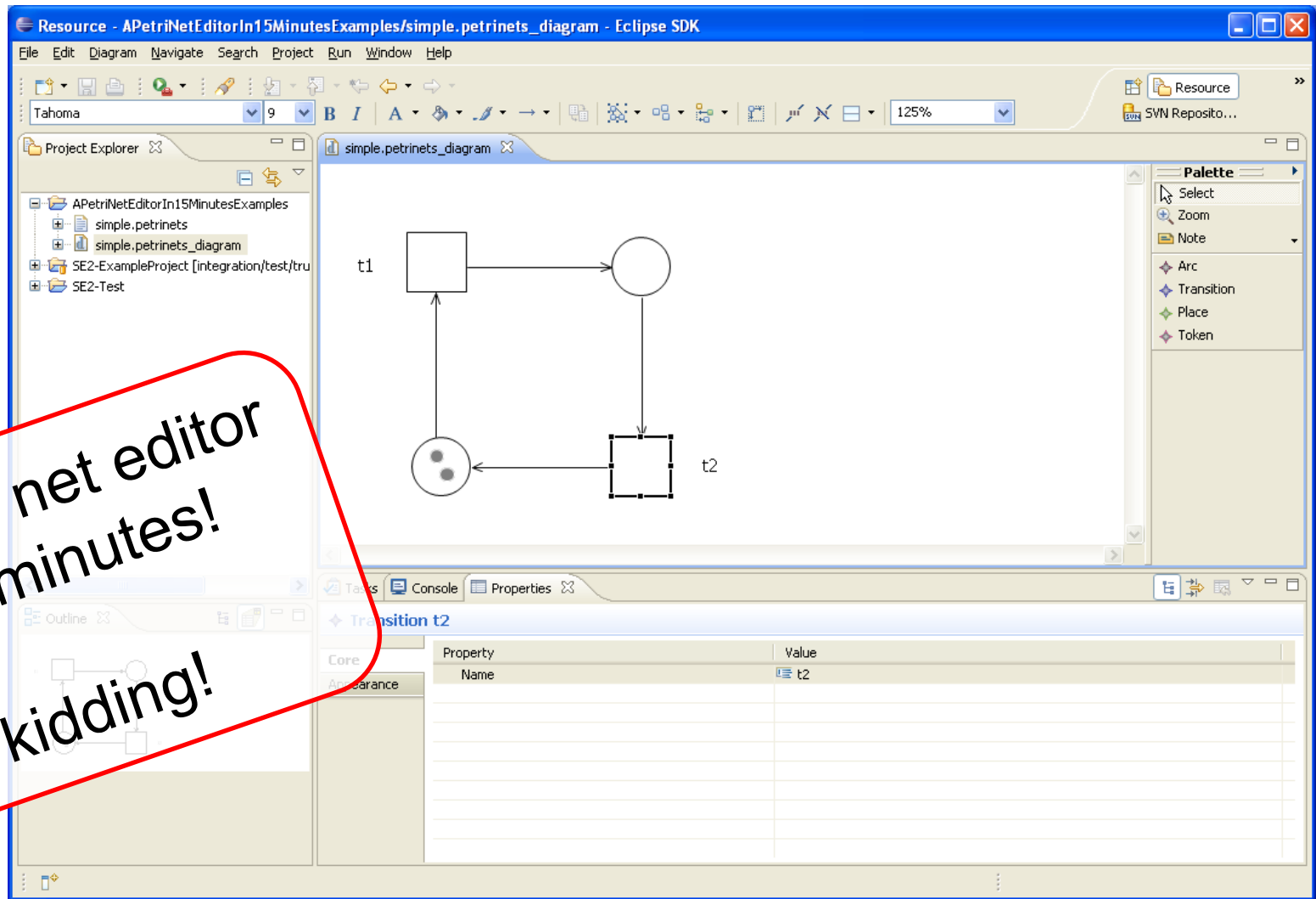
generate an editor



concrete syntax



abstract syntax



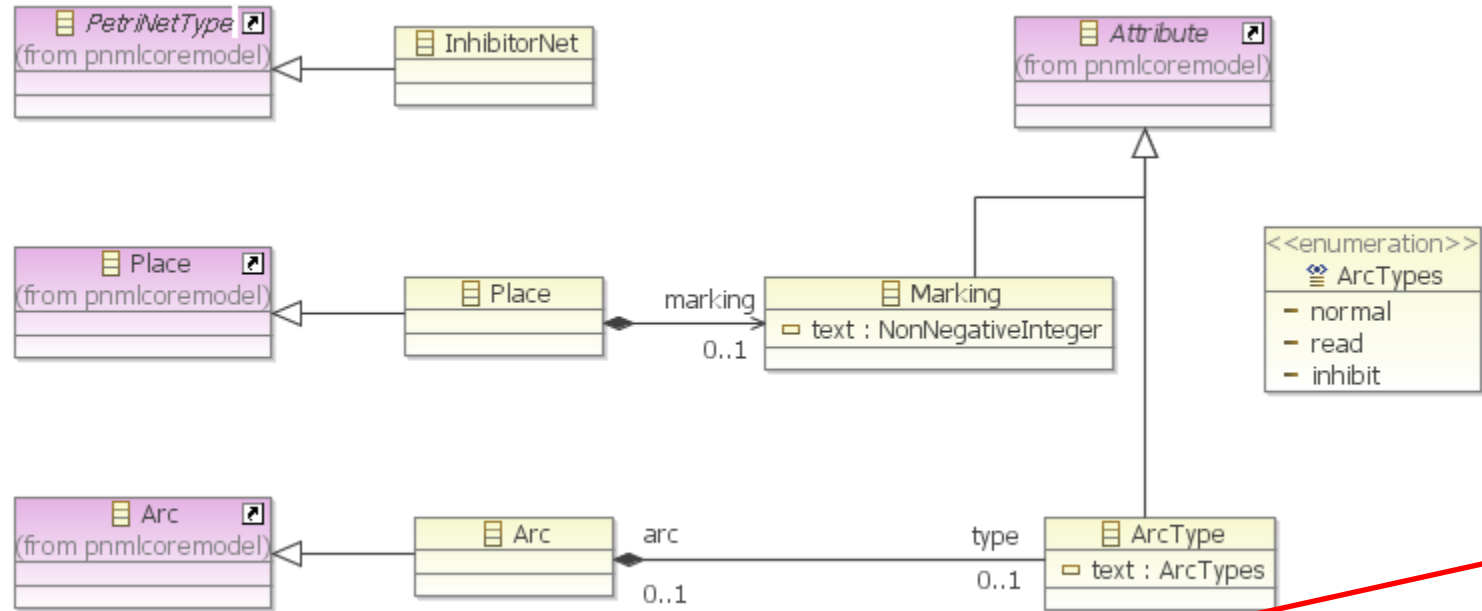
A Petri net editor
in 15 minutes!
Not kidding!

In the large: ePNK

The screenshot shows the Eclipse IDE interface with a Petri net diagram open in the main editor. The diagram consists of several places and transitions. Places are represented by circles, and transitions by squares. The diagram includes a place with one token, a place with two tokens, and a place with one token. Transitions are labeled t1, t2, t3, and end. There is also an 'init' transition. Red circles highlight the place with two tokens, transition t2, transition t1, and the place between t1 and t2. A palette on the right lists Petri net elements: Place, Transition, Arc, Page, RefPlace, RefTransition, Label, Link Label, and Page Label. A properties window at the bottom shows details for transition t1.

Property	Value
Id	t1
In	Arc a1
Out	Arc a2

Define a Petri Net Type



Additional constraints of inhibitor nets (defined in OLC).

```
( self.source.oclIsKindOf (pnmlcoremodel::PlaceNode) and
  self.target.oclIsKindOf (pnmlcoremodel::TransitionNode) )
```

or

```
( self.source.oclIsKindOf (pnmlcoremodel::TransitionNode) and
  self.target.oclIsKindOf (pnmlcoremodel::PlaceNode) and
  not ( self.type.text = ArcTypes::inhibit ) )
```

```
public class InhibitornetsArcFigure extends ArcFigure {
[...]
```

```
    private void setGraphics() {
        RotatableDecoration targetDecorator = null;
        RotatableDecoration sourceDecorator = null;

        if (type.equals(ArcTypes.NORMAL)) {
            targetDecorator = new ReisigsArrowHeadDecoration();

        } else if (type.equals(ArcTypes.INHIBIT)) {
            CircleDecoration circleDecoration =
                new CircleDecoration();
            circleDecoration.setLineWidth(this.getLineWidth());
            targetDecorator = circleDecoration;

        } else if (type.equals(ArcTypes.READ)) {
            targetDecorator = new ReisigsArrowHeadDecoration();
            sourceDecorator = new ReisigsArrowHeadDecoration();
        }
    }
[...]
```

Just a glimpse!

- Better understanding
- Mapping of instances to XML syntax (XMI)
(canonical exchange format for instances)
- Automatic code generation for
 - API for creating, deleting and modifying models
 - Methods for loading and saving models (in XML)
 - Standard mechanisms for keeping track of changes (observers)
 - Transactional changes (TC)
 - Database access/persistence (D)
 - Graphical editor

BUT:
All this is “standard functionality”!
How about “real” functionality or
behaviour?

e.g. a Petri net simulator?

The screenshot displays the ECNO GUI, which is used for simulating Petri nets. The main window shows a Petri net diagram with the following components:

- Places:** req1, idle1, pend1, crit1, sem, crit2, 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. The 'sem' place acts as a semaphore. The 'enter' transitions (enter1, enter2) acquire the semaphore, and the 'exit' transitions (exit1, exit2) release it. The 'pend' places (pend1, pend2) represent the states where a process is waiting to enter its critical section.

The left sidebar shows the ECNO: GUI window with a list of transitions and their corresponding 'fire' buttons:

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

The bottom panel shows the simulation engine registry:

Engine name	Resource name/path
Engine 1	platform:/resource/APetriNetEditorIn15Minutes.runtime/run/semaphor.behaviourstates

~~There are no notations
for modelling behaviour!~~

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

- 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"!

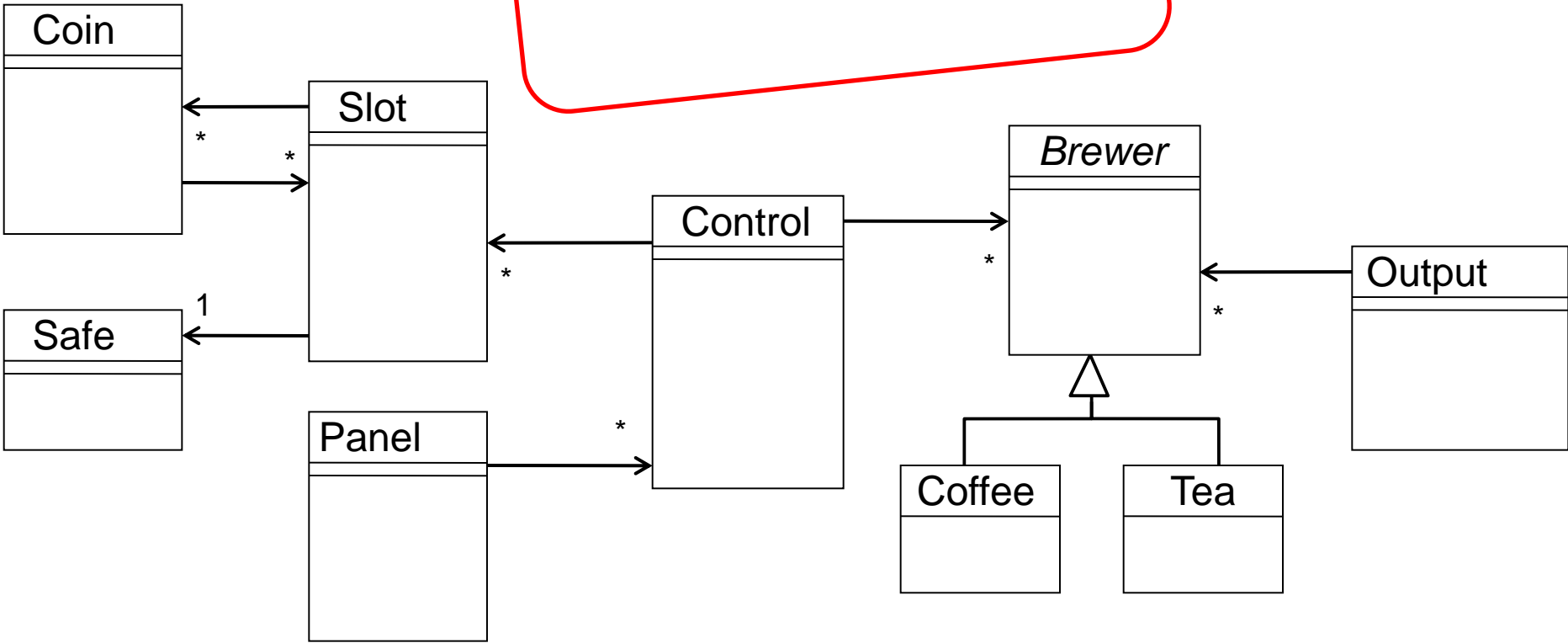
Motivation

- Given some object oriented software with (or without) explicit domain model
 - Model the behaviour on top of it – and make these models executable
 - Model behaviour on a high level of abstraction (domain level): coordination of behaviour
- Integrate behaviour models with structural models
- Integrate different structural models and manually written code (or code generated by different technologies)

Meta-modelling /
domain modelling
including behaviour!

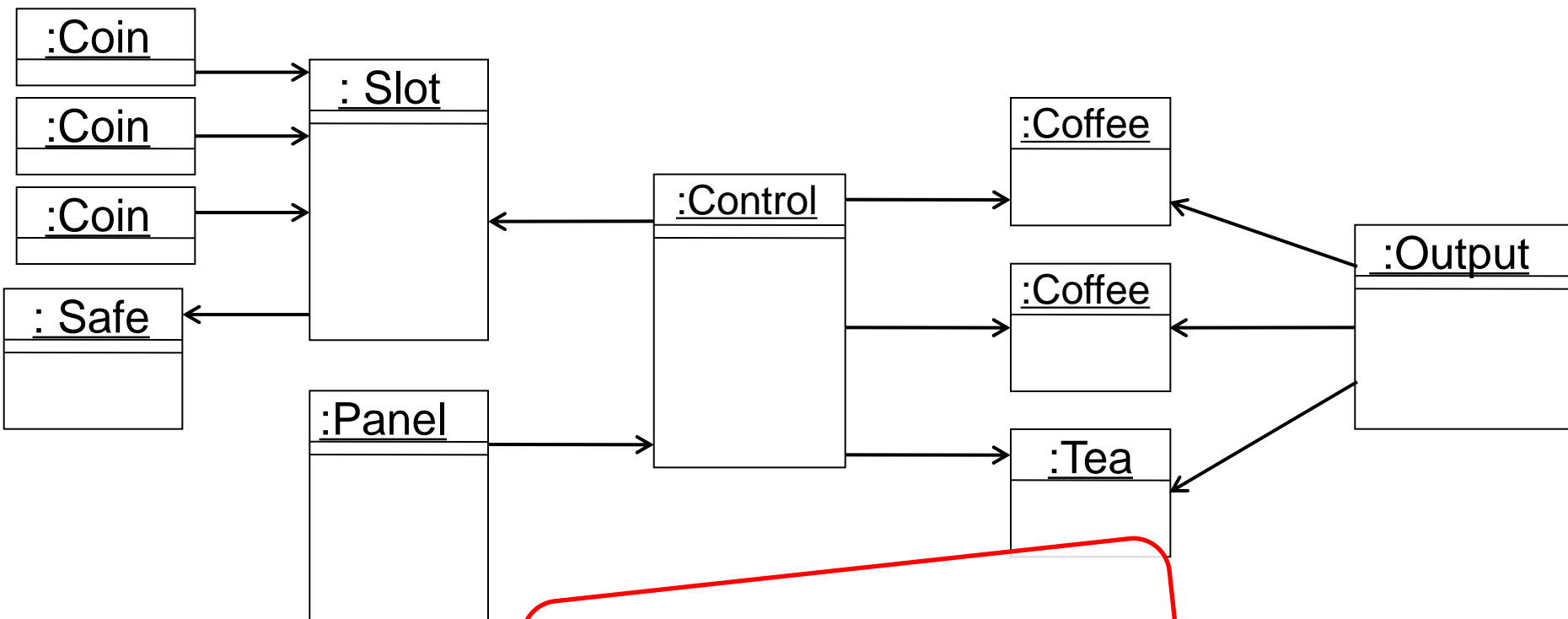
2.1 Example: Vending machine

Class diagram as usual



Instance: Object Diagram

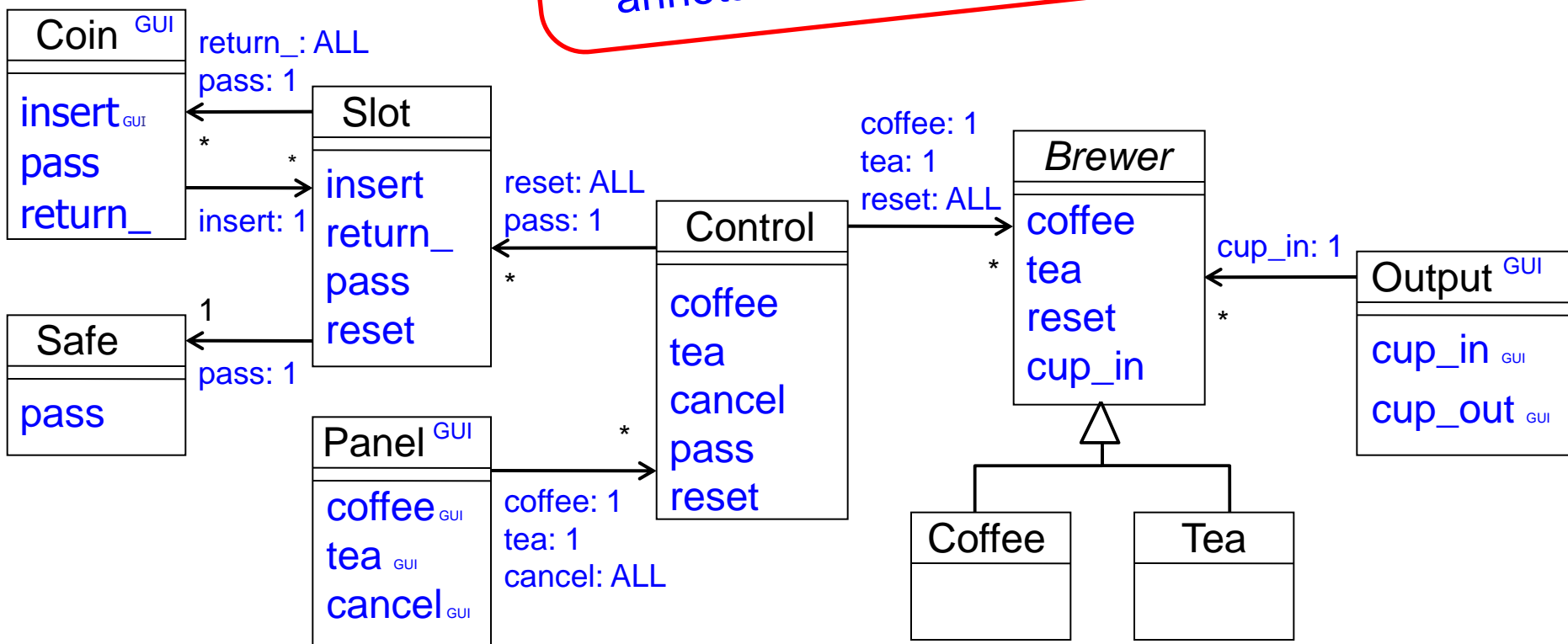
Initial configuration,
current situation



Object diagram as usual

- We call objects elements now!

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



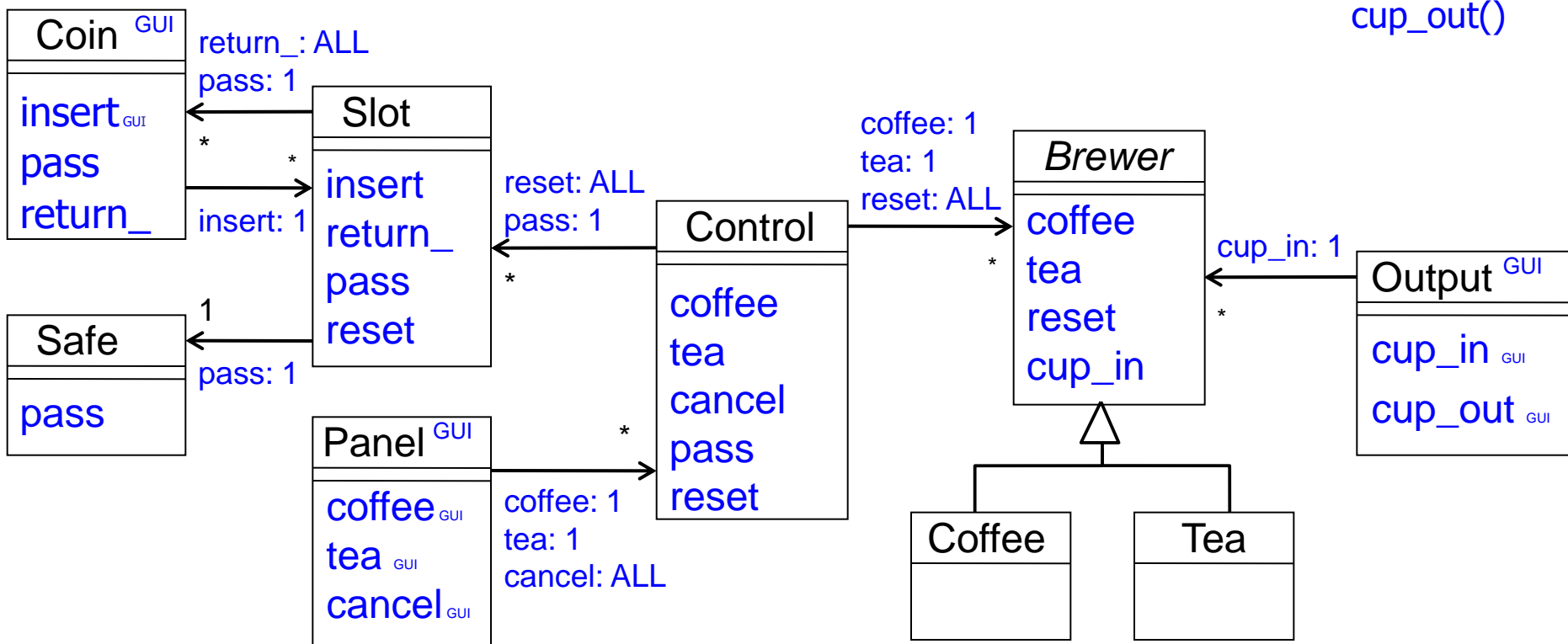
... + Event declaration

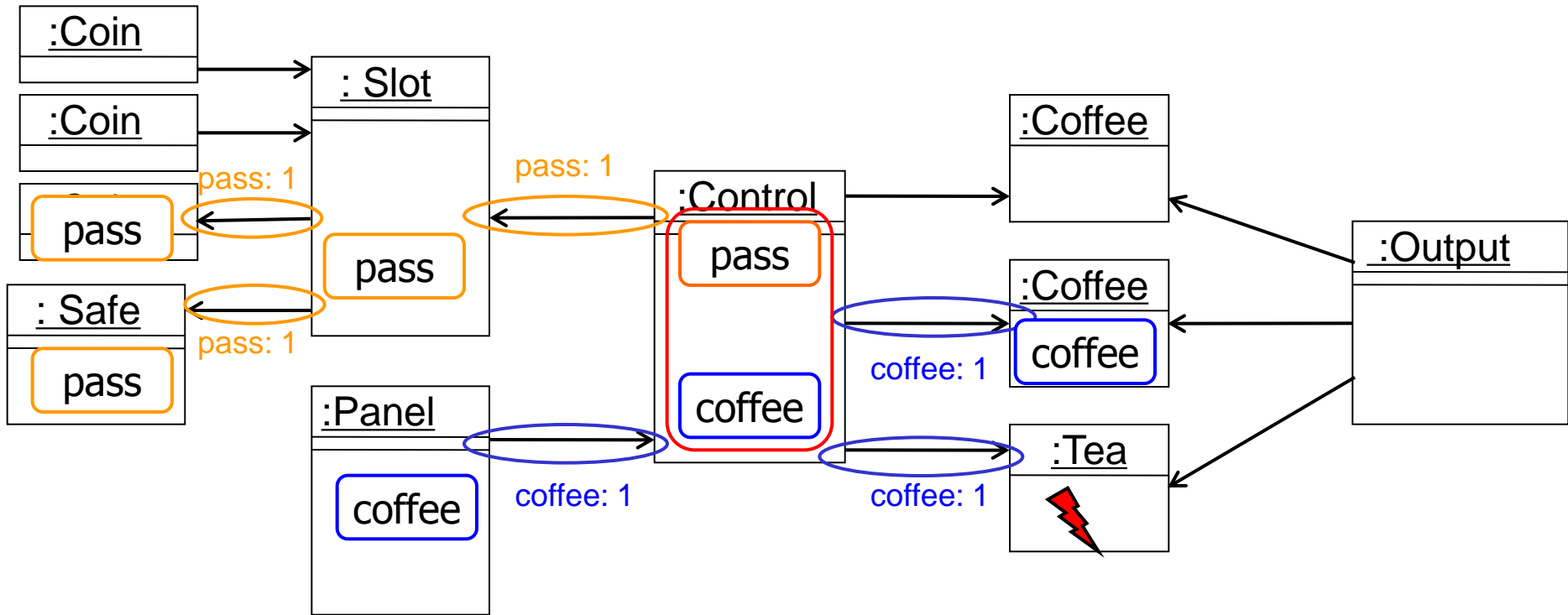
- Event (type) declaration
- Parameters

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

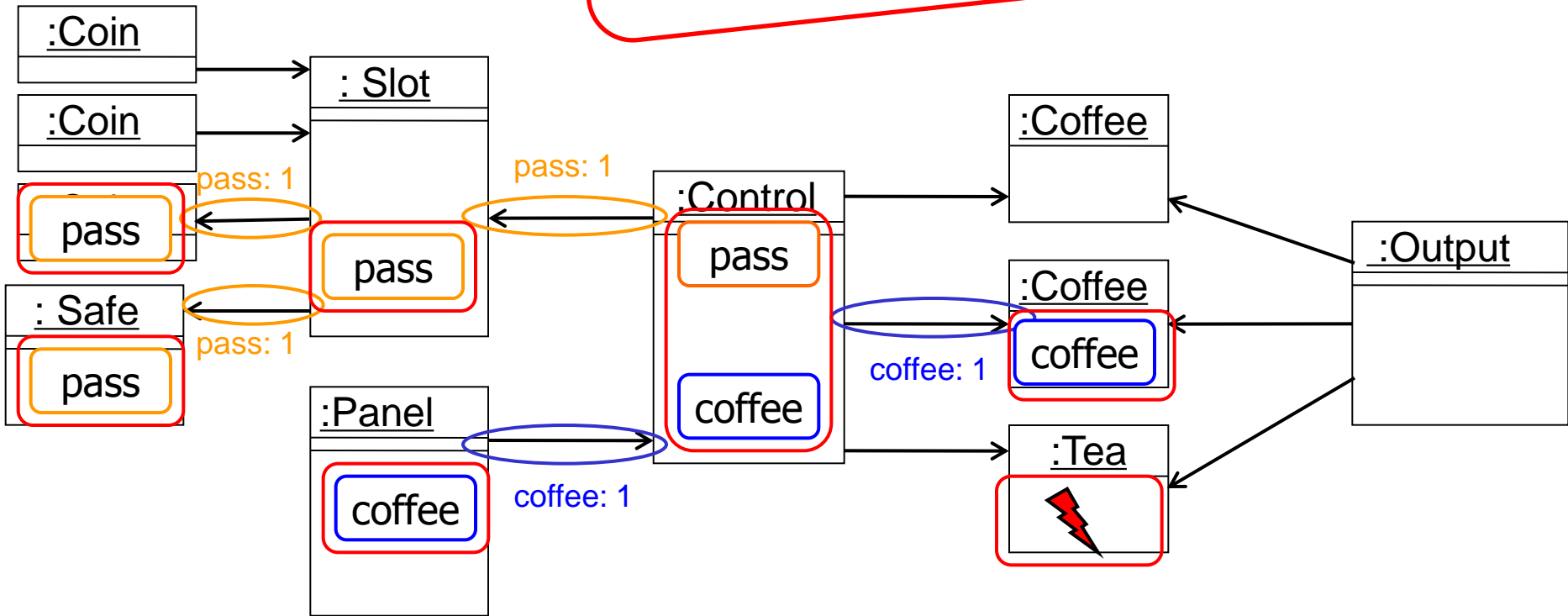
coffee()
tea()
cancel()

cup_in()
cup_out()

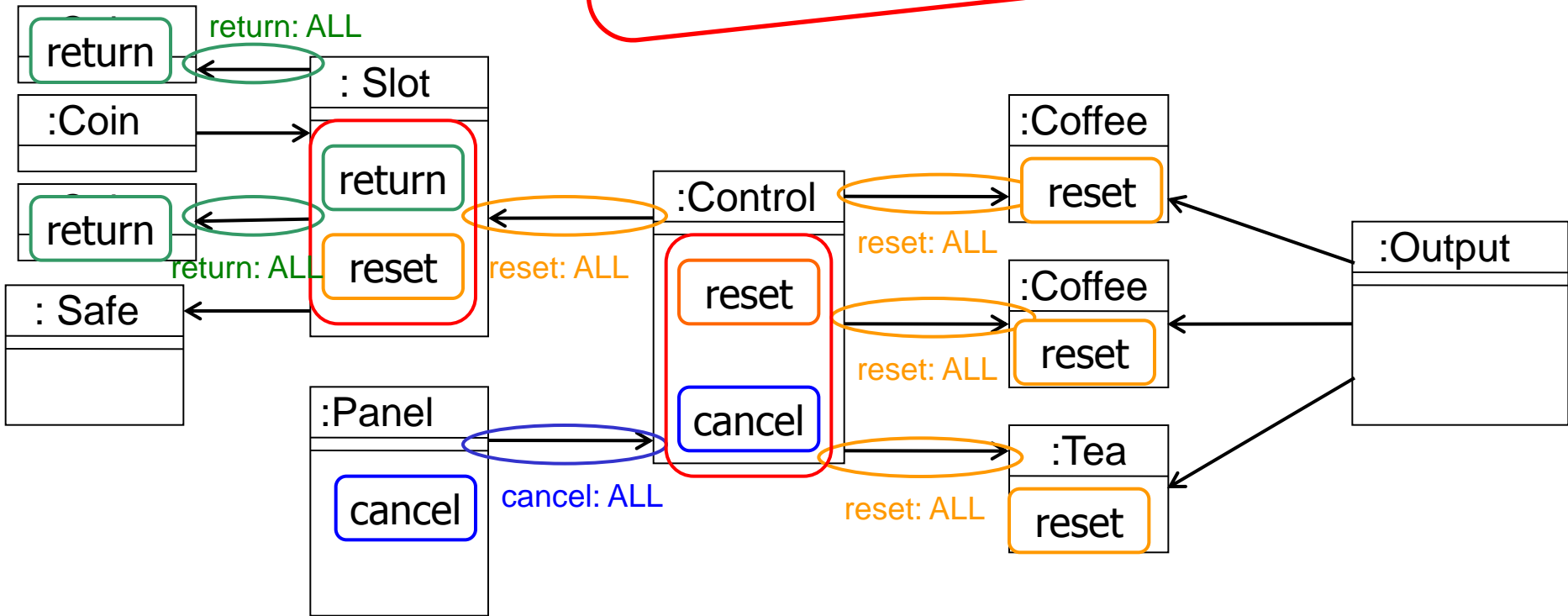




Interaction =
local behavior +
coordination

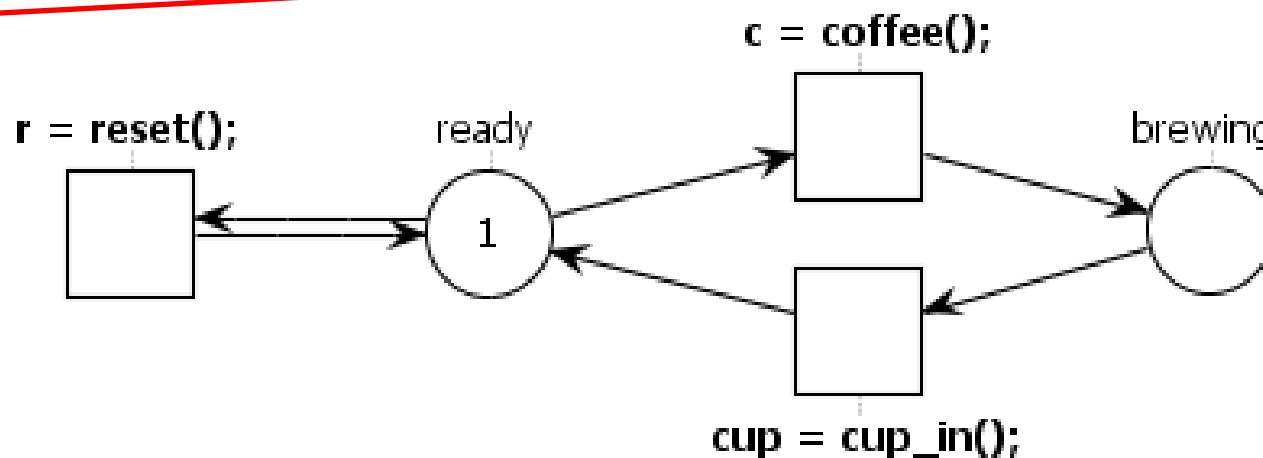


Interaction =
local behavior +
coordination



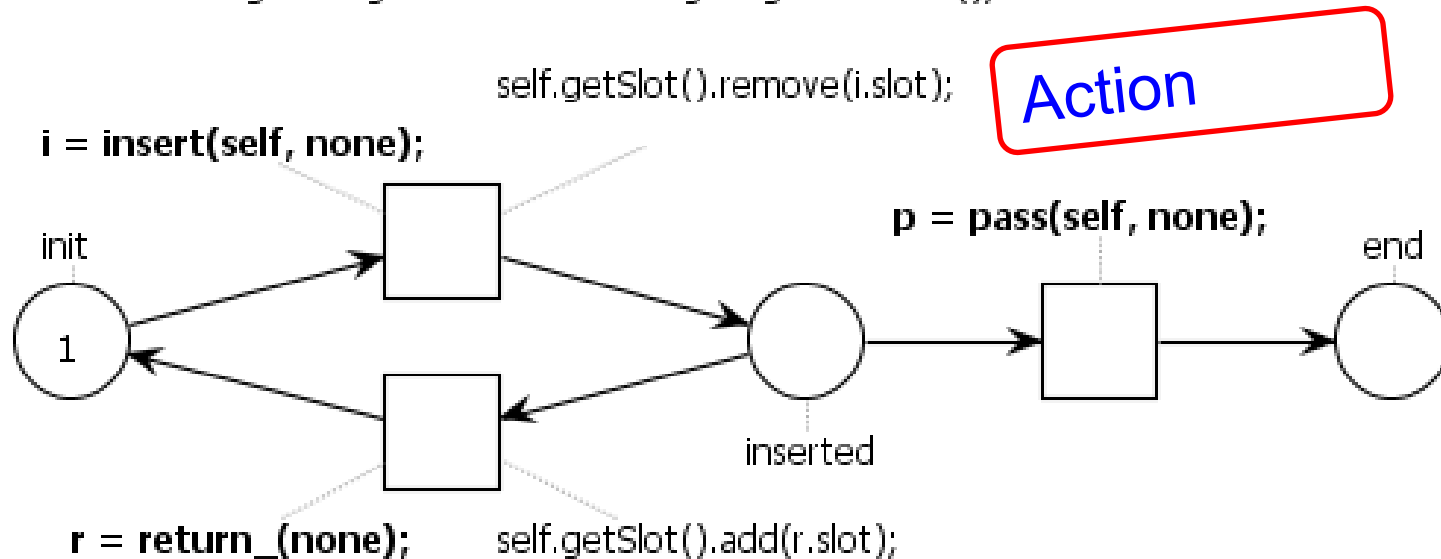
Elements are objects with an explicitly modelled **life-cycle**

Event binding




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

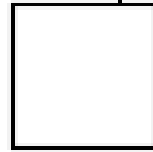
```
final ExecutionEngine engine = ExecutionEngine.getInstance();
```



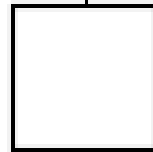
Action

- Event binding
- Parameter assignment

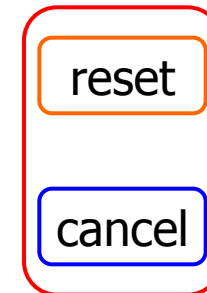
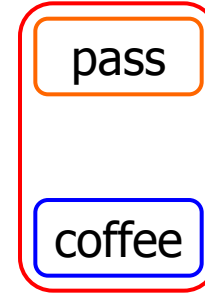
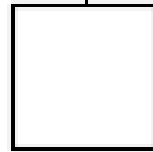
`p = pass(none,none); c = coffee();`



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



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

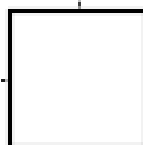


- Event binding with multiple event types!

Condition

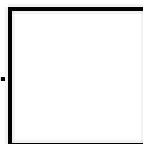
`self.getCoin().size() < 2`

`i = insert(none, self);`



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

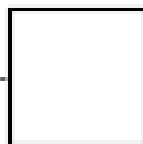
`p = pass(none, self);`



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

`res = reset();`

`r = return_(self);`

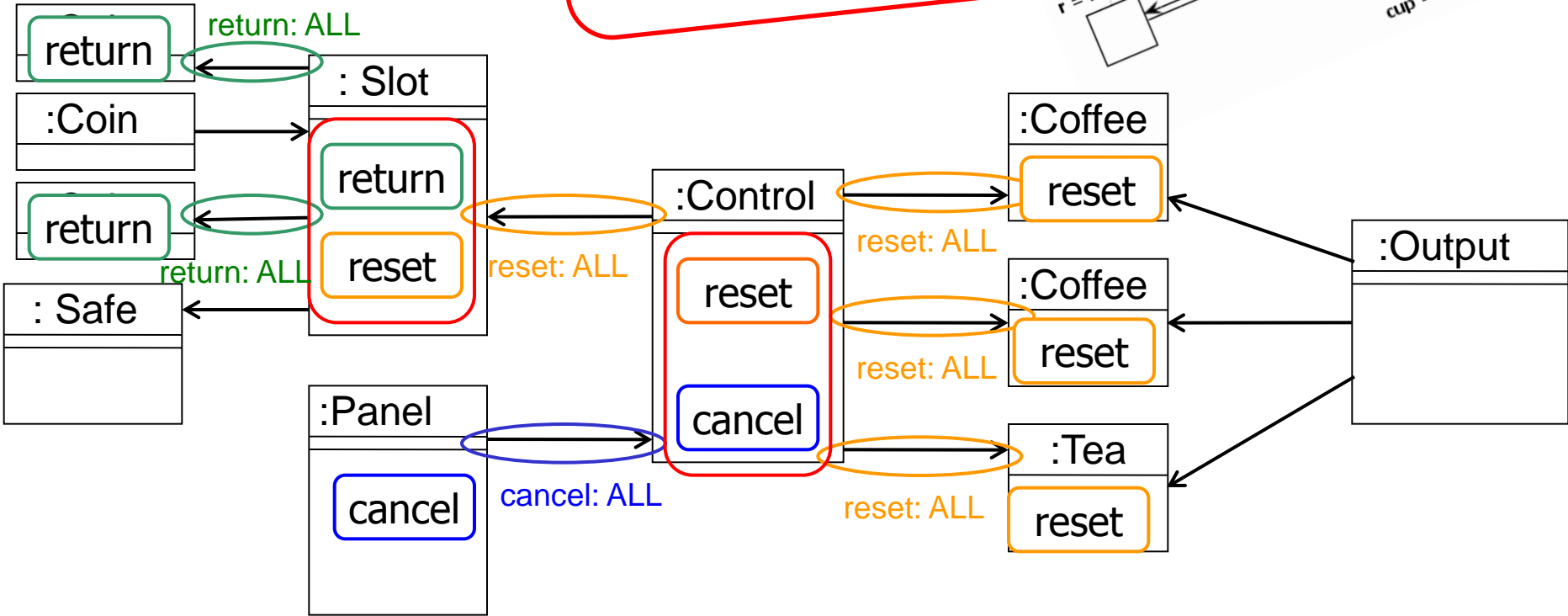
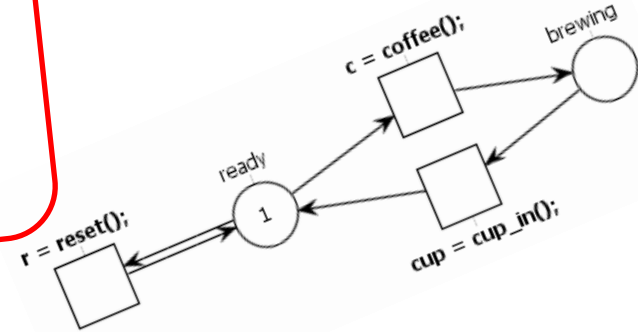


`self.getCoin().clear();`

return

reset

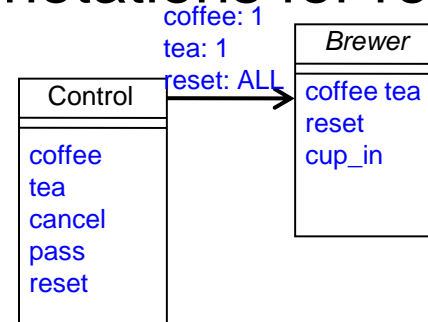
Interaction =
local behavior +
coordination



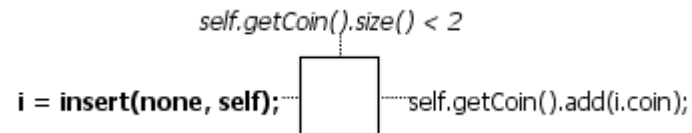
- ElementTypes (Classes)
- EventTypes with
 - parameters

`insert(Coin coin, Slot slot)`

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

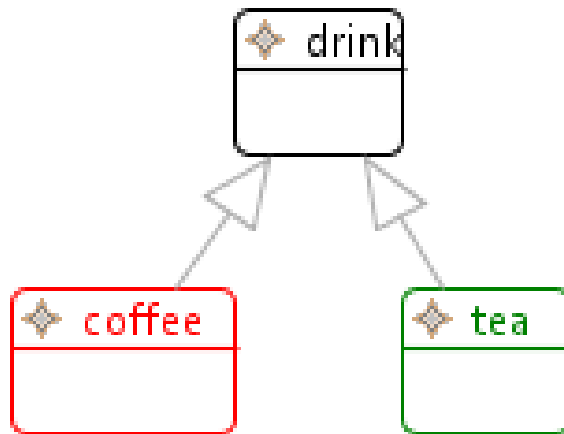


- Local behaviour (life-cycle): ECNO nets (or something else)
 - Event binding (with parameter assignment)
 - Condition
 - Action

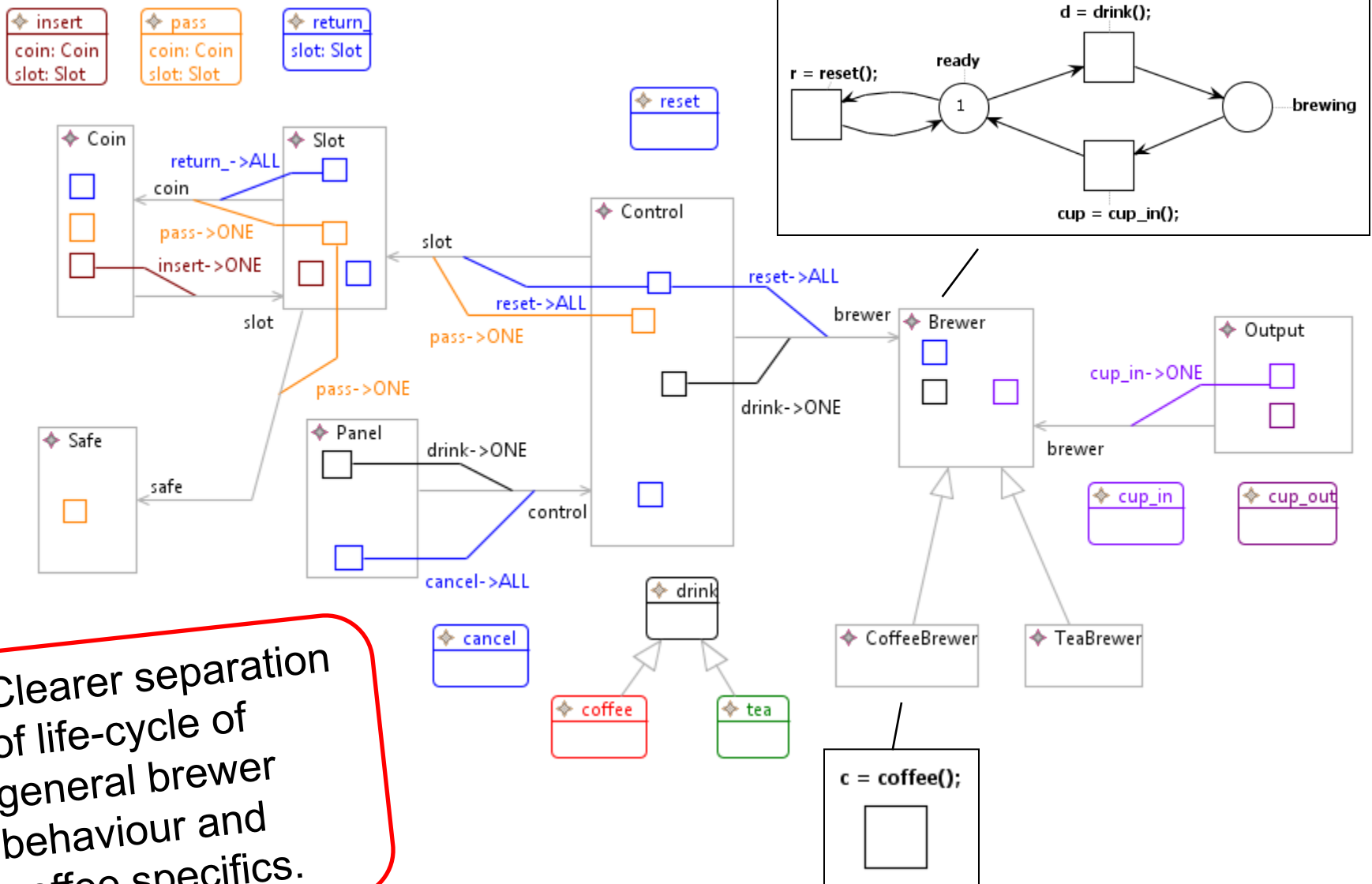


ECNO with its basic concepts has some limitations, which makes modelling things **in an adequate way** a bit painful.

- Sometimes, we want to extend event types later



Behaviour inheritance

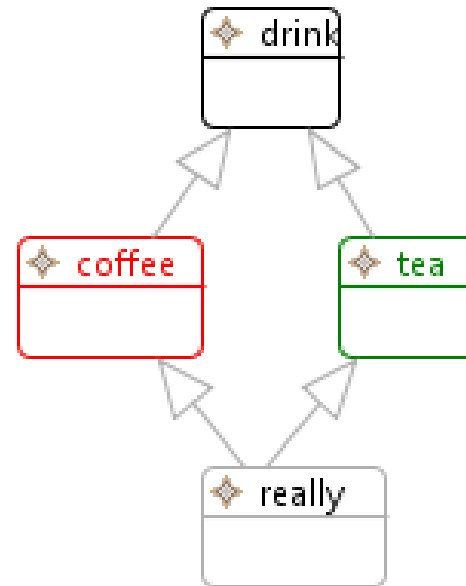


Clearer separation of life-cycle of general brewer behaviour and coffee specifics.

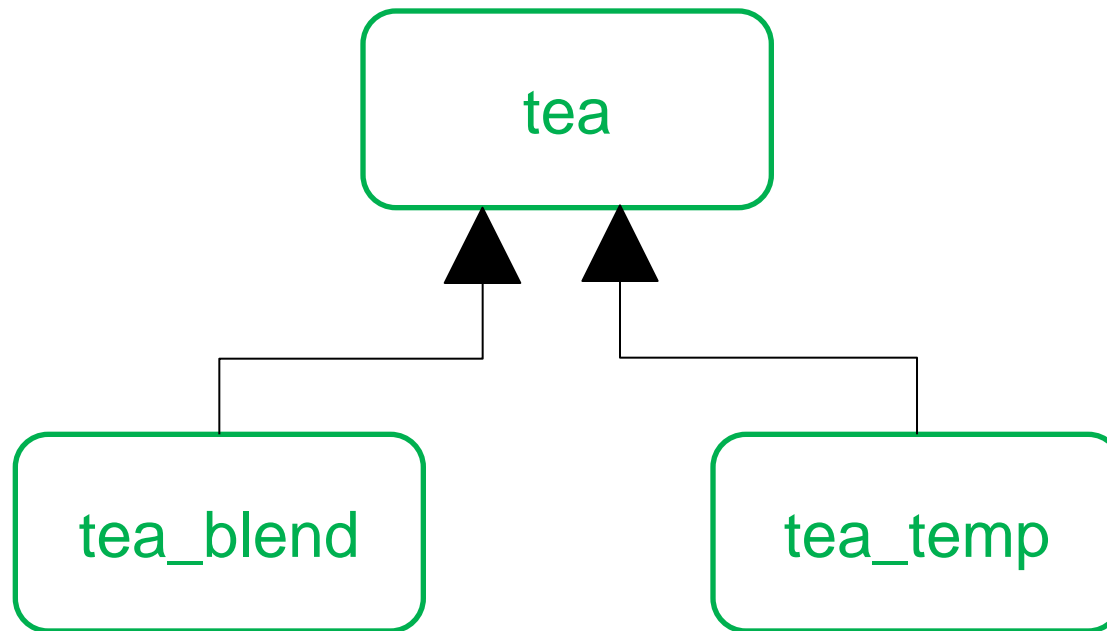
Question: Would we like to have multiple inheritance on event types?

Problems:

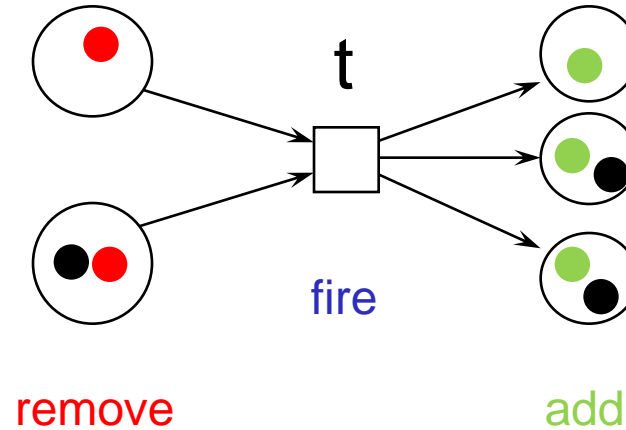
- We could never be sure that two event types that were meant to be different are different!
- We would not know which event type an instance of subtype would represent!



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



=> Two forms of inheritance on event types!



How can we model that behaviour in ECNOs?

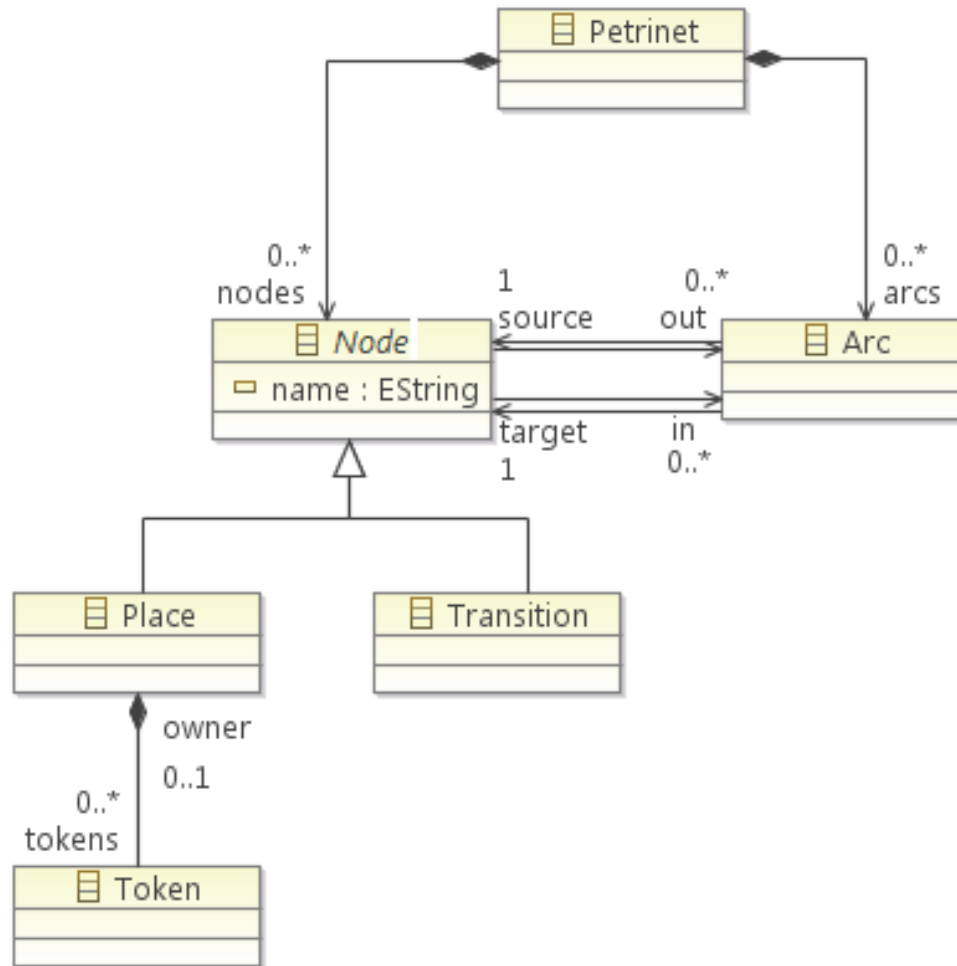
Transition t **enabled**:

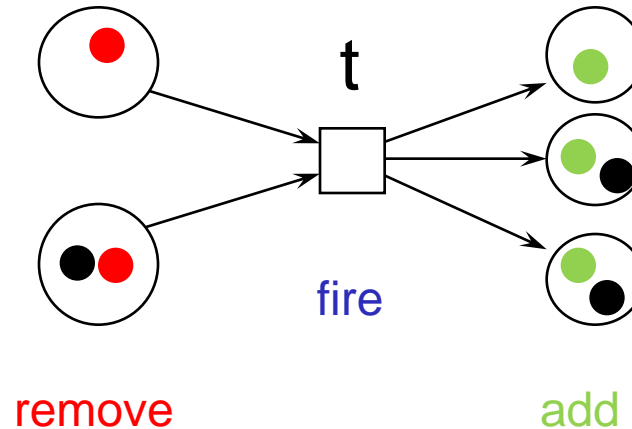
for ALL incoming Arcs a :
for ONE source Place p of Arc a :
find a token

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





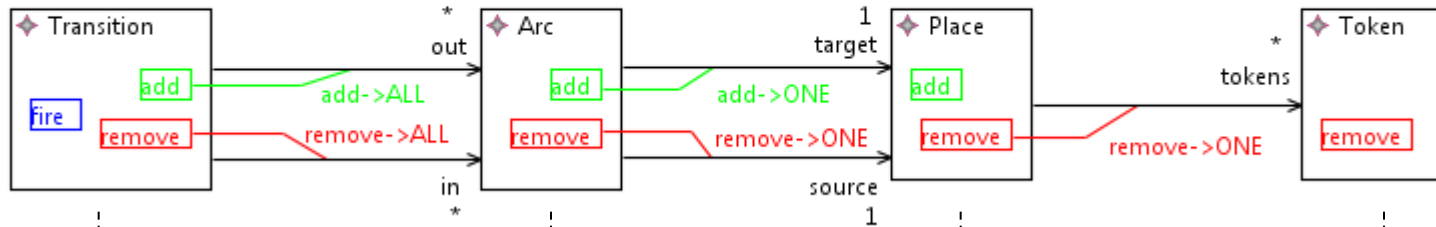
Transition t **enabled**:

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);
```

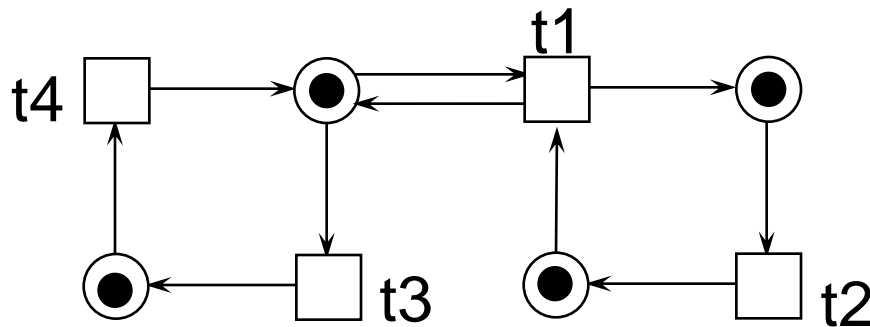
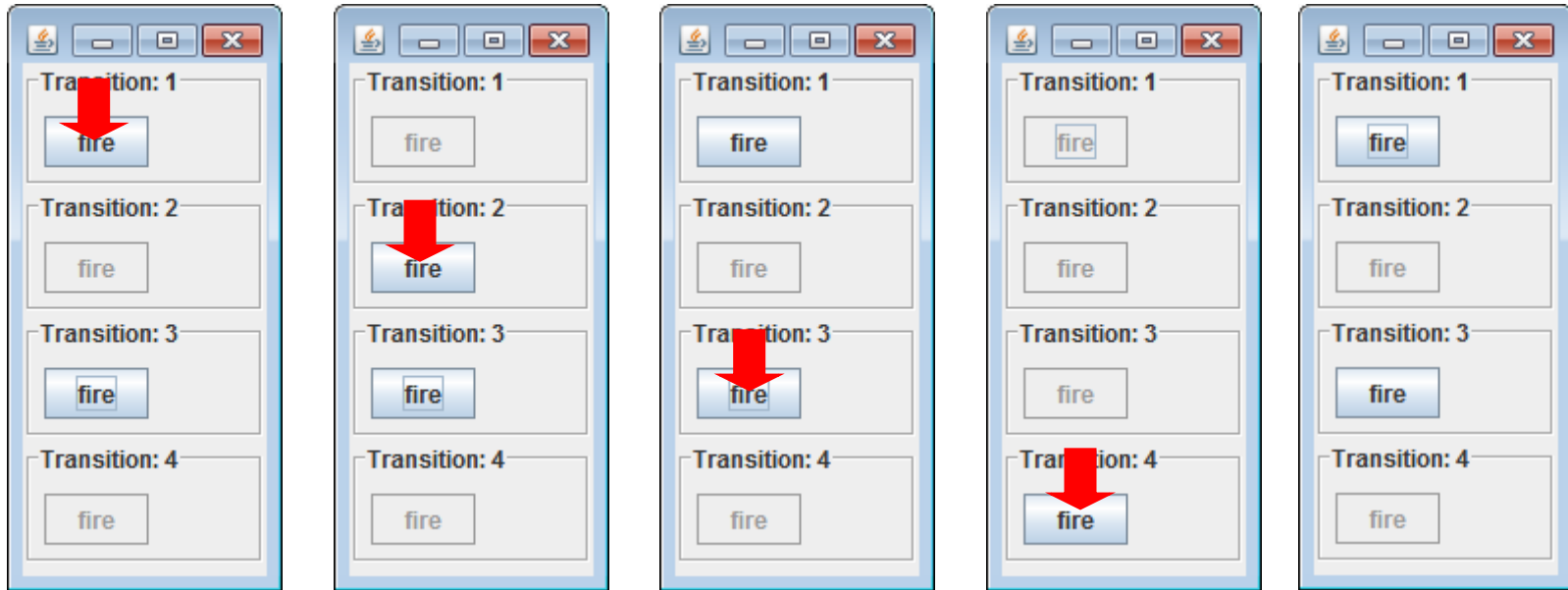
```
import dk.dtu.imm.se.ecno.example.petrinets.PetrinetsFactory;

final PetrinetsFactory factory = PetrinetsFactory.eINSTANCE;

a = add();
```

```
self.getTokens().add(factory.createToken());
```

```
r = remove();
```



The screenshot displays the Petri net simulator interface. The main window shows a Petri net diagram for a semaphore. The diagram includes a central place labeled 'sem' with one token. It is connected to two critical sections, 'crit1' and 'crit2', each with an 'enter' transition and an 'exit' transition. 'crit1' also includes a 'pend1' place and an 'idle1' place, while 'crit2' includes 'pend2' and 'idle2'. Requests 'req1' and 'req2' are shown as places connected to the 'idle' places. The interface also features a palette on the right with elements like Arc, Transition, Place, and Token.

On the left, a window titled 'ECNO: GUI' lists transitions and their 'fire' buttons:

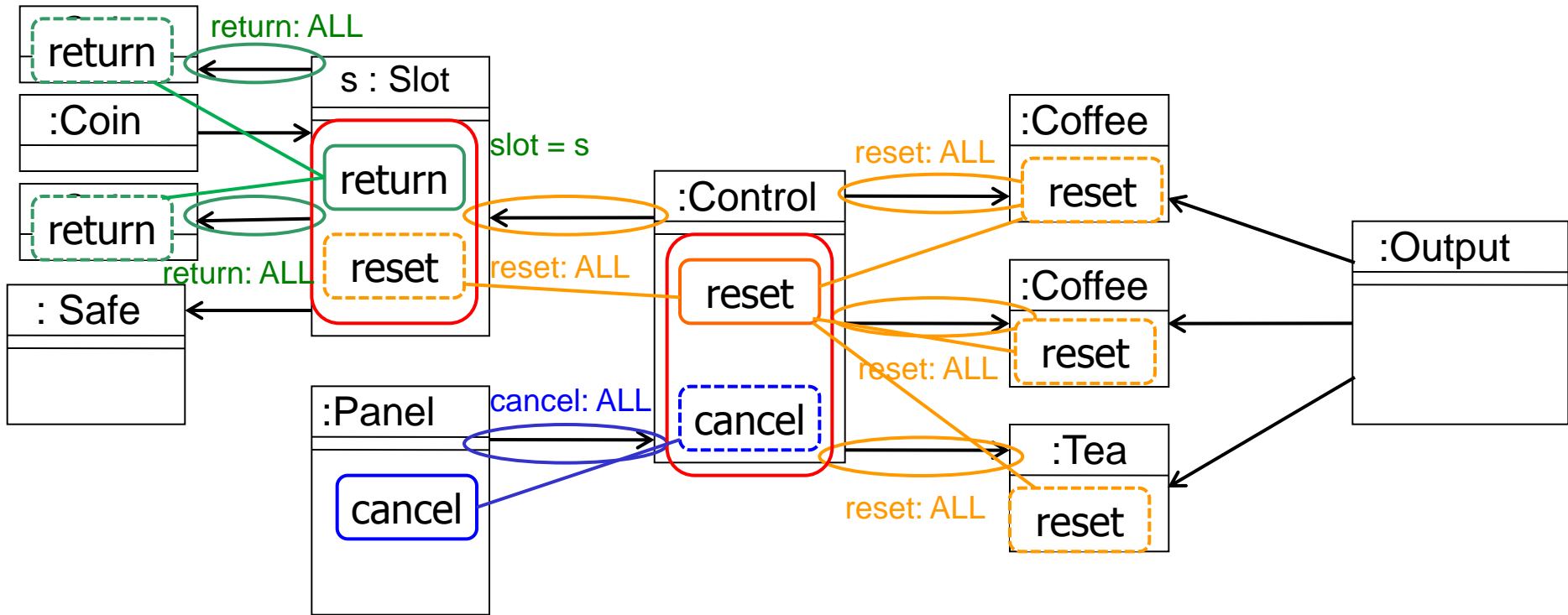
- req1 : Transition [1] - fire
- enter1 : Transition [7] - fire
- exit1 : Transition [11] - fire
- req2 : Transition [16] - fire
- enter2 : Transition [22] - fire
- exit2 : Transition [26] - fire

At the bottom, the 'ECNO: Engine registry' table shows the simulation engine configuration:

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

3. ECNO: Semantics

- Can be formalized in mathematics
(done for core concepts → ECNO Technical Report)



- The semantics of ECNO could, probably, be expressed in ECNO itself!

I have a sketch (but not fully worked out yet).

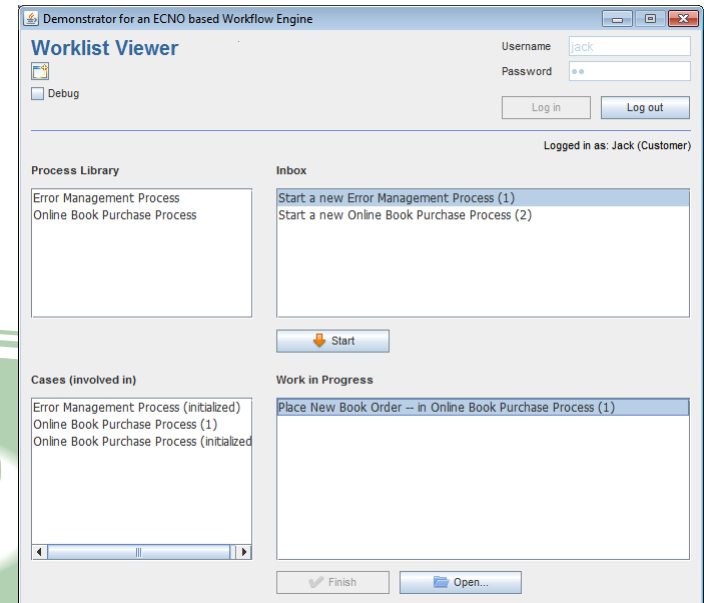
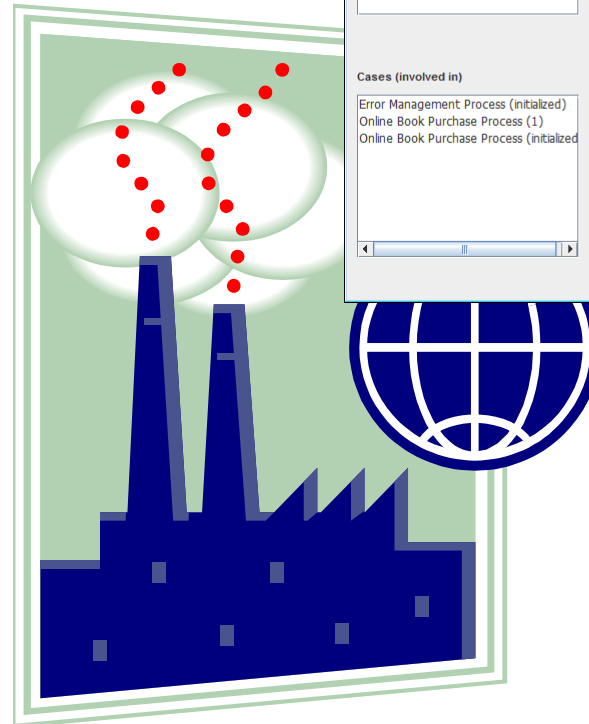
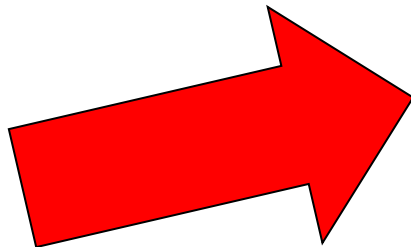
Idea similar to PN semantics; needs most of the advanced concepts (except inheritance).

4. Conclusion

Better ask me for a not yet published release of version 0.3.3!

Find release ECNO Tool (version 0.3.2):
<http://www2.compute.dtu.dk/~ekki/projects/ECNO/>

Next Steps



- Documentation
- Database integration (e.g. Hibernate/Teneo)
- GUI modelling (DSL for GUI for ECNO applications)
- Better IDE support & debugger
- Case studies
- Efficiency / distributed execution
- Standard adapters for other technologies
- **Methodology & education**