


Separation of Control Flow and Data Flow in High-Level Petri Nets

Towards Readability and Verification of Complex Systems

Berndt Farwer

Fachbereich Informatik

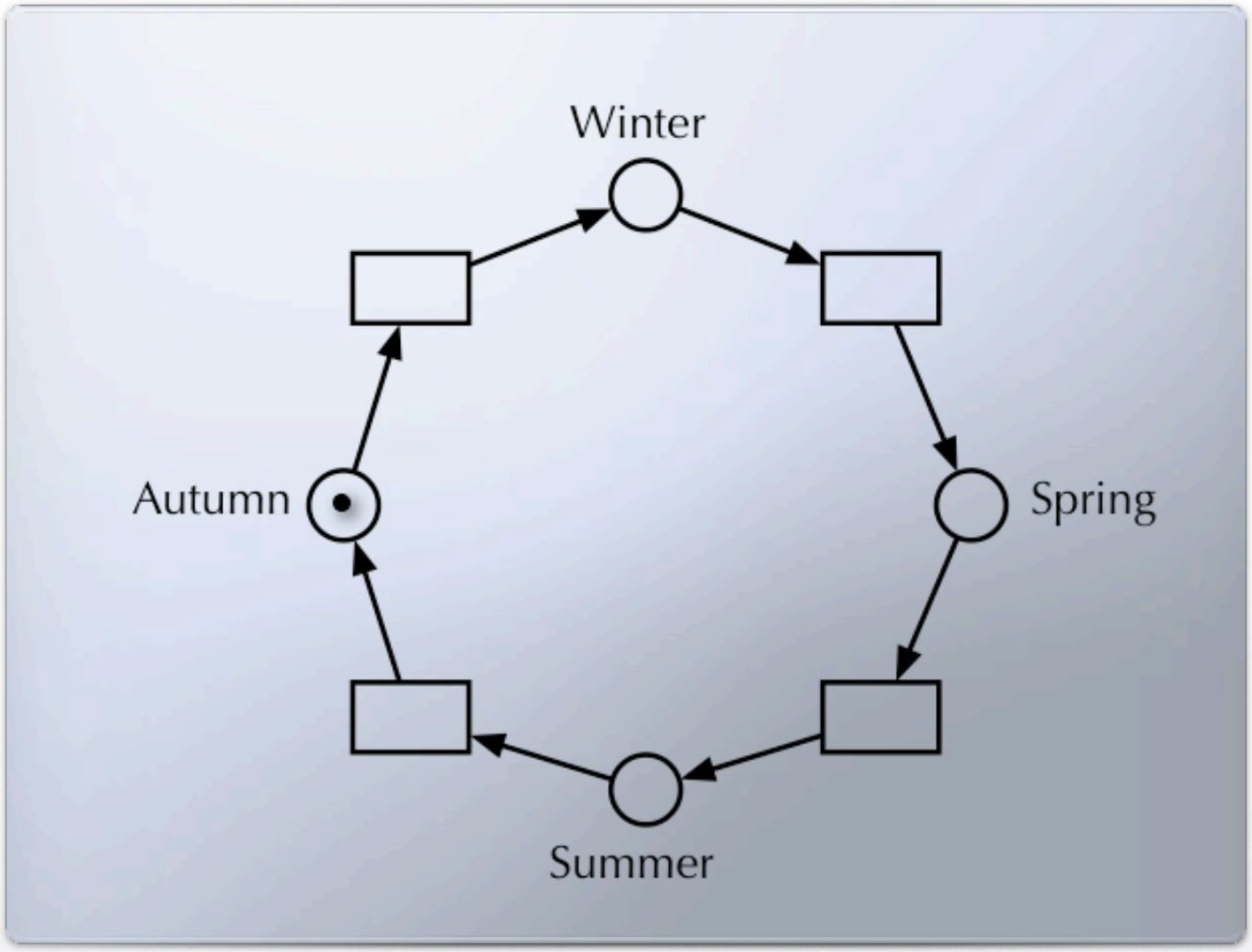
Universität Hamburg

farwer@informatik.uni-hamburg.de

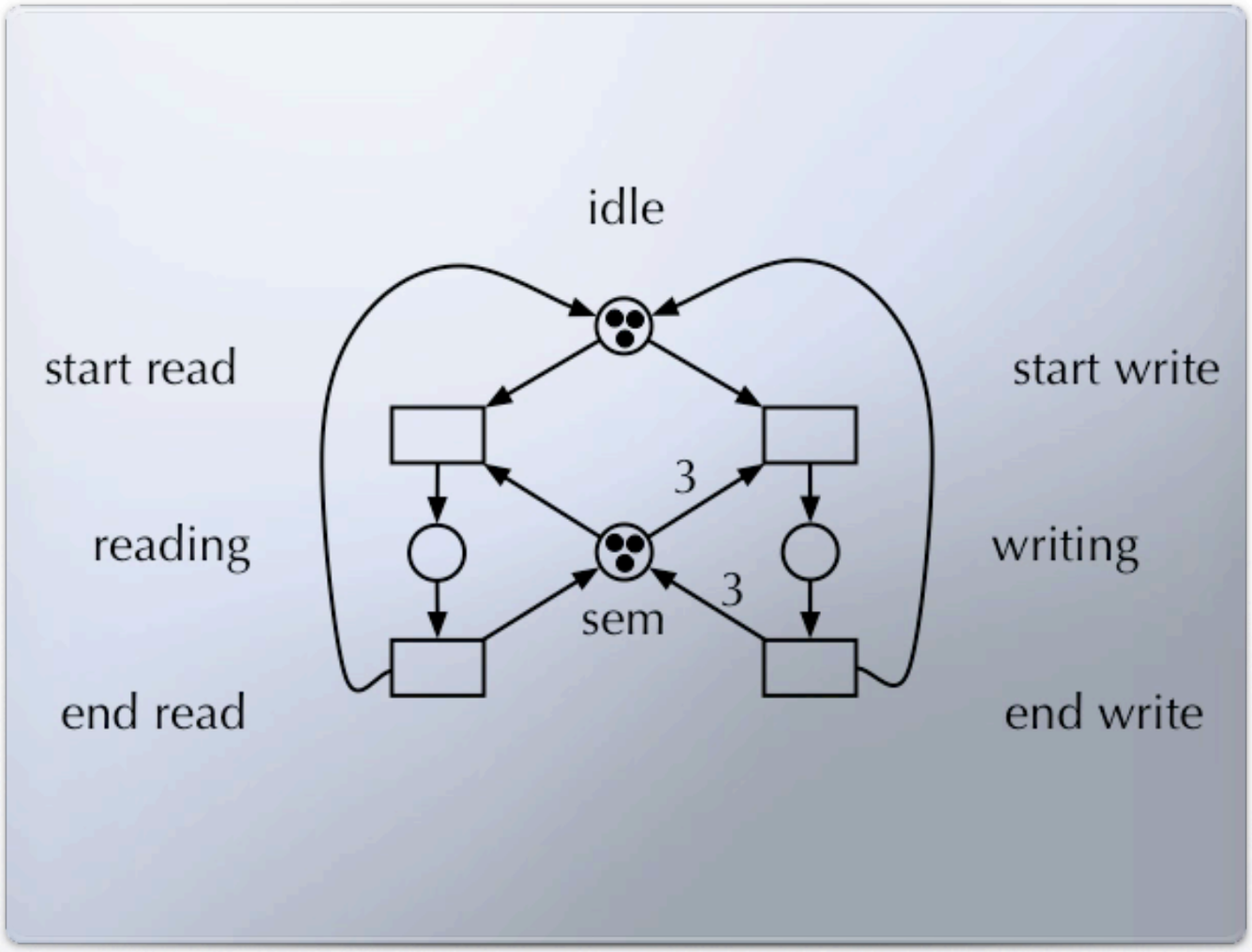

Séminaire MeFoSyLoMa, 18 March 2005

Overview

1. Object Petri Nets
2. Tools & Modelling
3. Verification
4. Data Flow ↔ Control Flow



Petri Nets


idle
start read
start write
3
reading
writing
3
sem
end read
end write

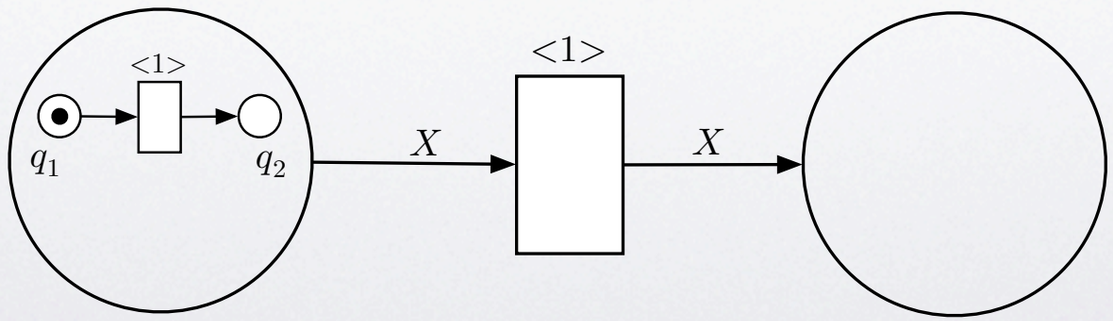
Petri Nets


Some Observations

- formal semantics
- **but:**
  - can become hard to read
- **solutions:**
  - abstraction/refinement
  - coloured & object Petri nets
  - Petri nets patterns/components

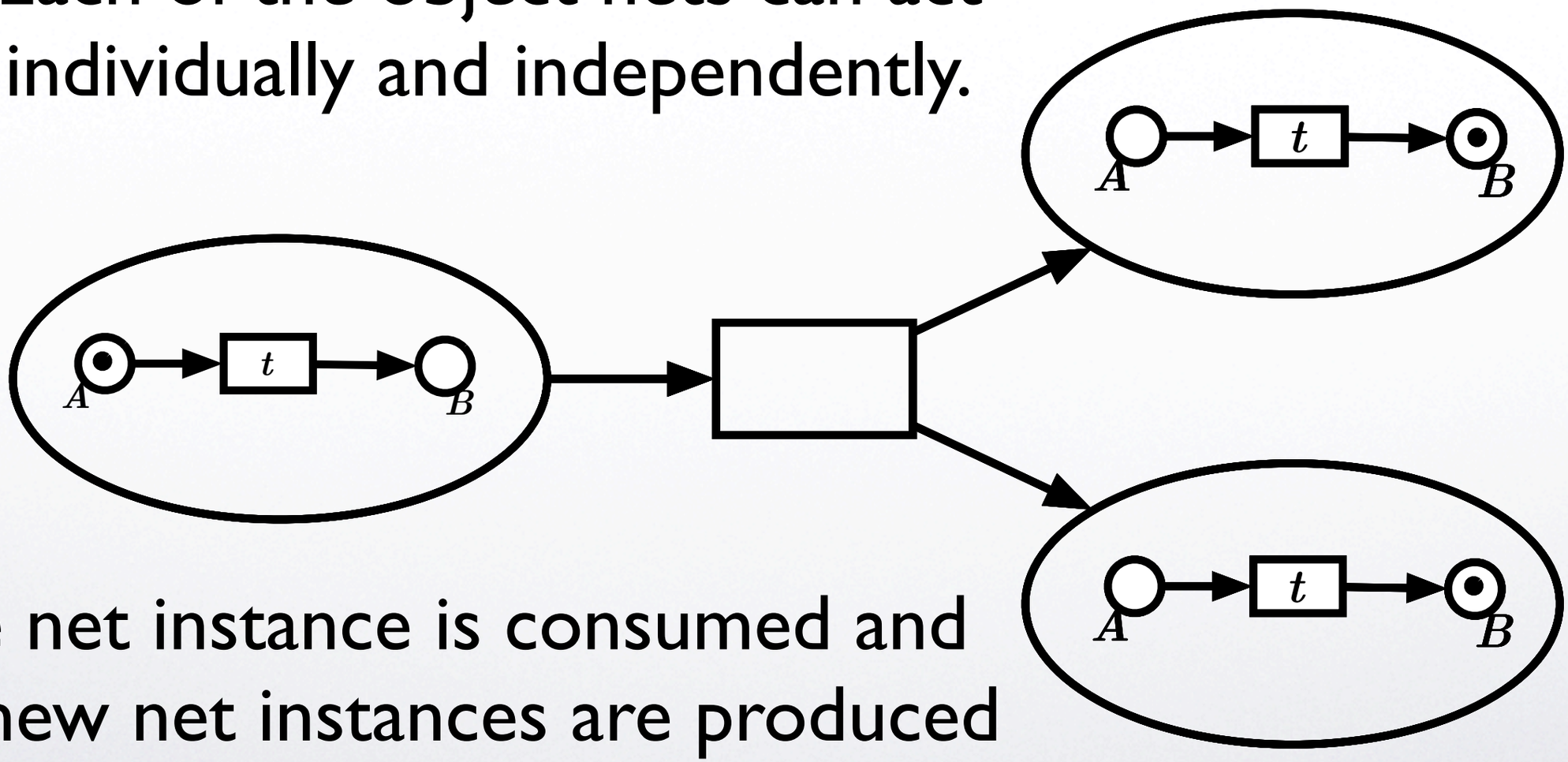

What are Object Petri Nets?

- "Nets-within-nets" paradigm
  - Petri nets as tokens (token nets) of a Petri net (system net)
- reference and value semantics



Value Semantics

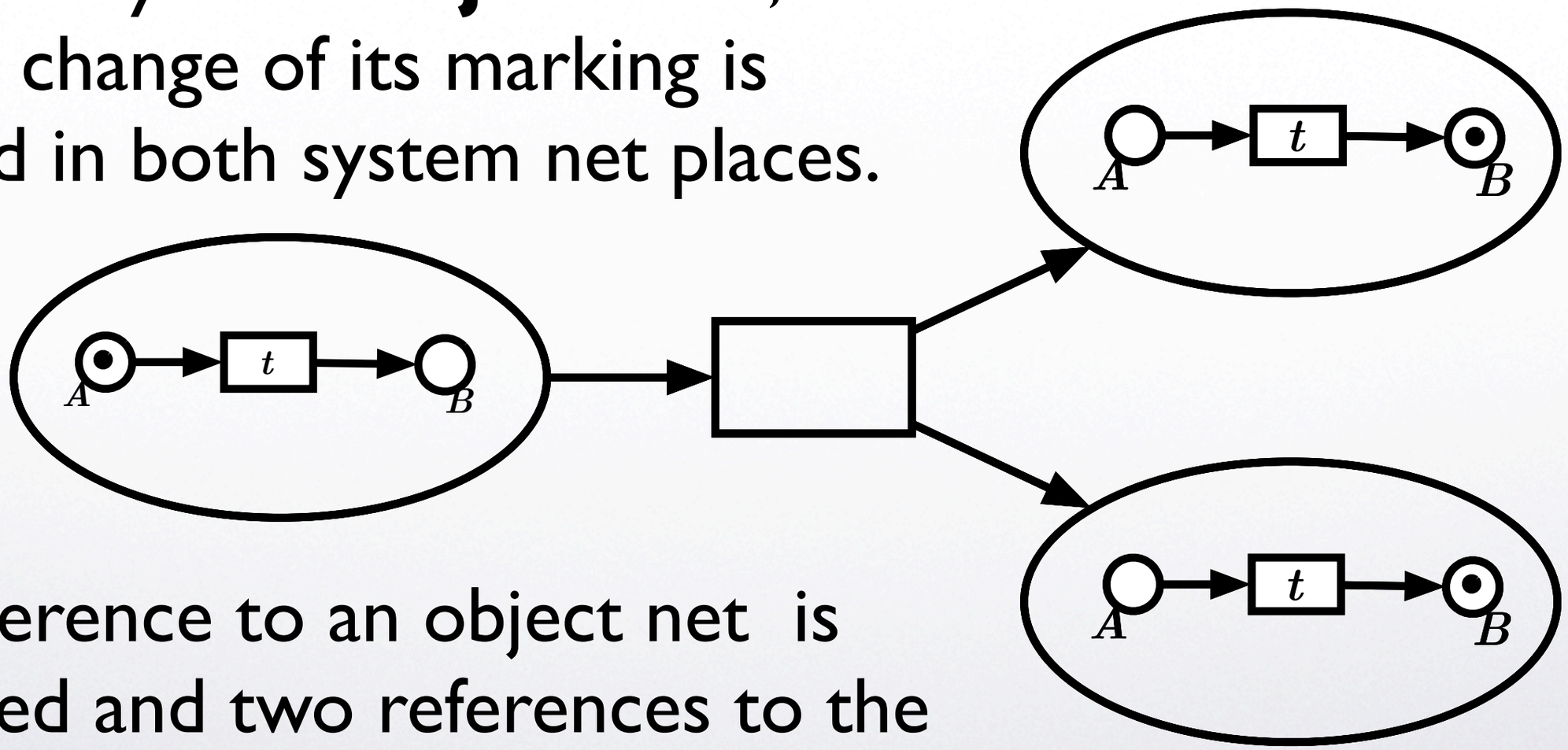
Each of the object nets can act individually and independently.



One net instance is consumed and two new net instances are produced by the system net transition.

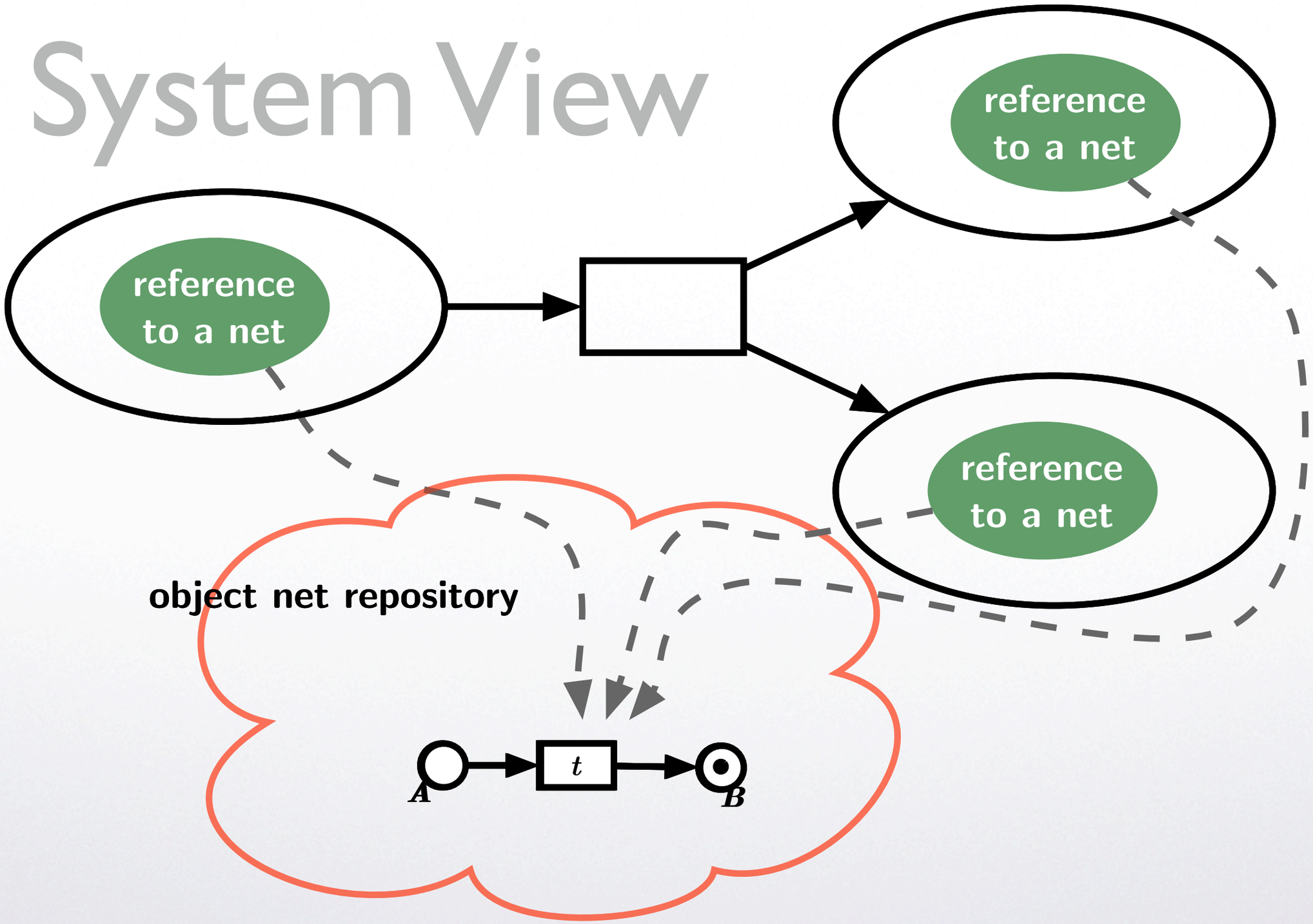
Reference Semantics

There is only **one object net**, such that any change of its marking is reflected in both system net places.



One reference to an object net is consumed and two references to the same net are produced by the system net transition.

System View


reference
to a net
reference
to a net
reference
to a net
object net repository
t
A
B

Tools

- Renew (Reference Net Workshop)
  - graphical editor
  - simulator
  - analysis frontend
  - free download: *www.renew.de*
- XTL
  - model checking (CTL)

Projects

- Settlers of Catan (software development)
- Workflow (business process modelling)
- Flexible Manufacturing
- Security Protocols
- Philosophers (academic)

Settlers of Catan

tool bar

net in execution

game board

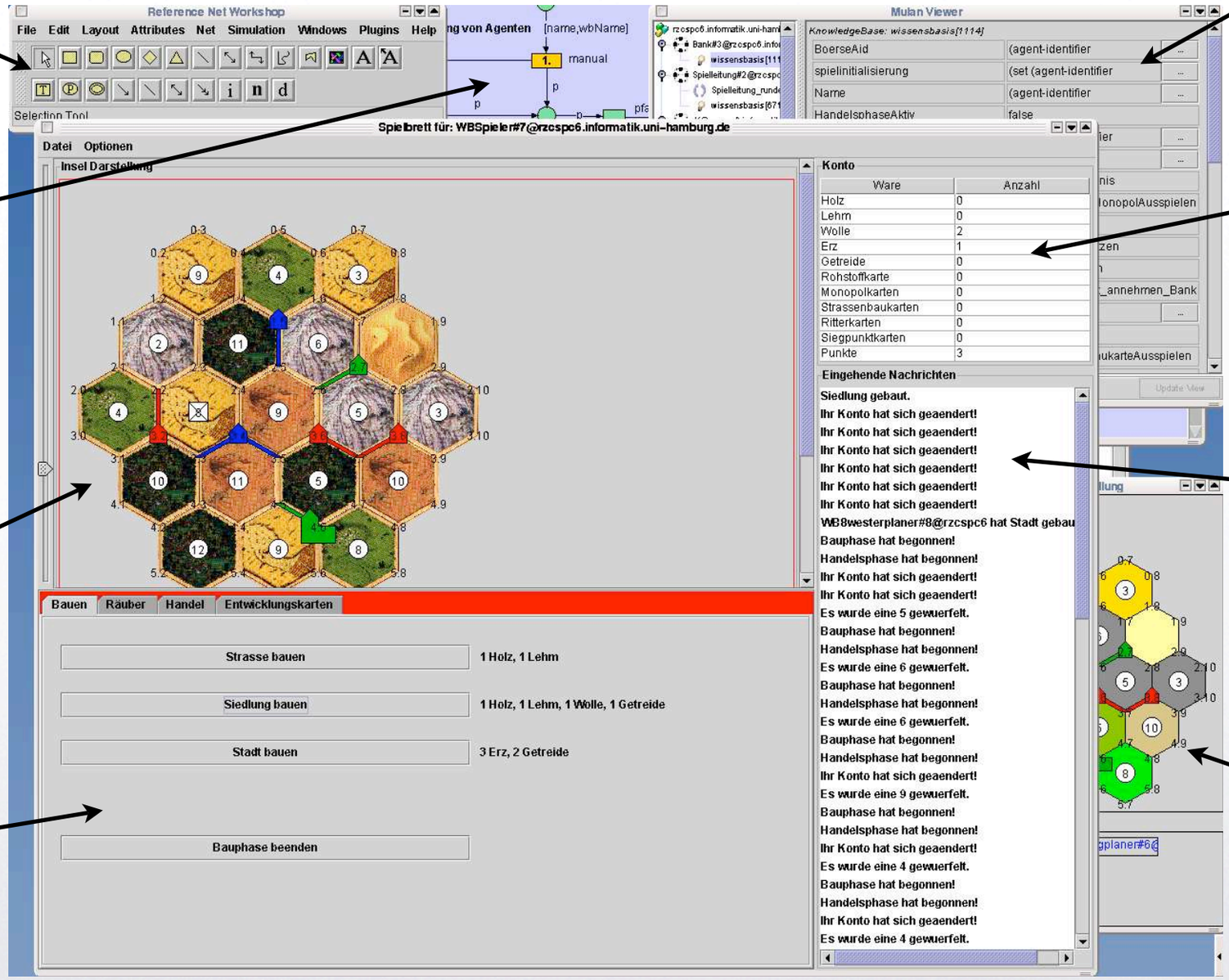
user interaction

Mulan viewer

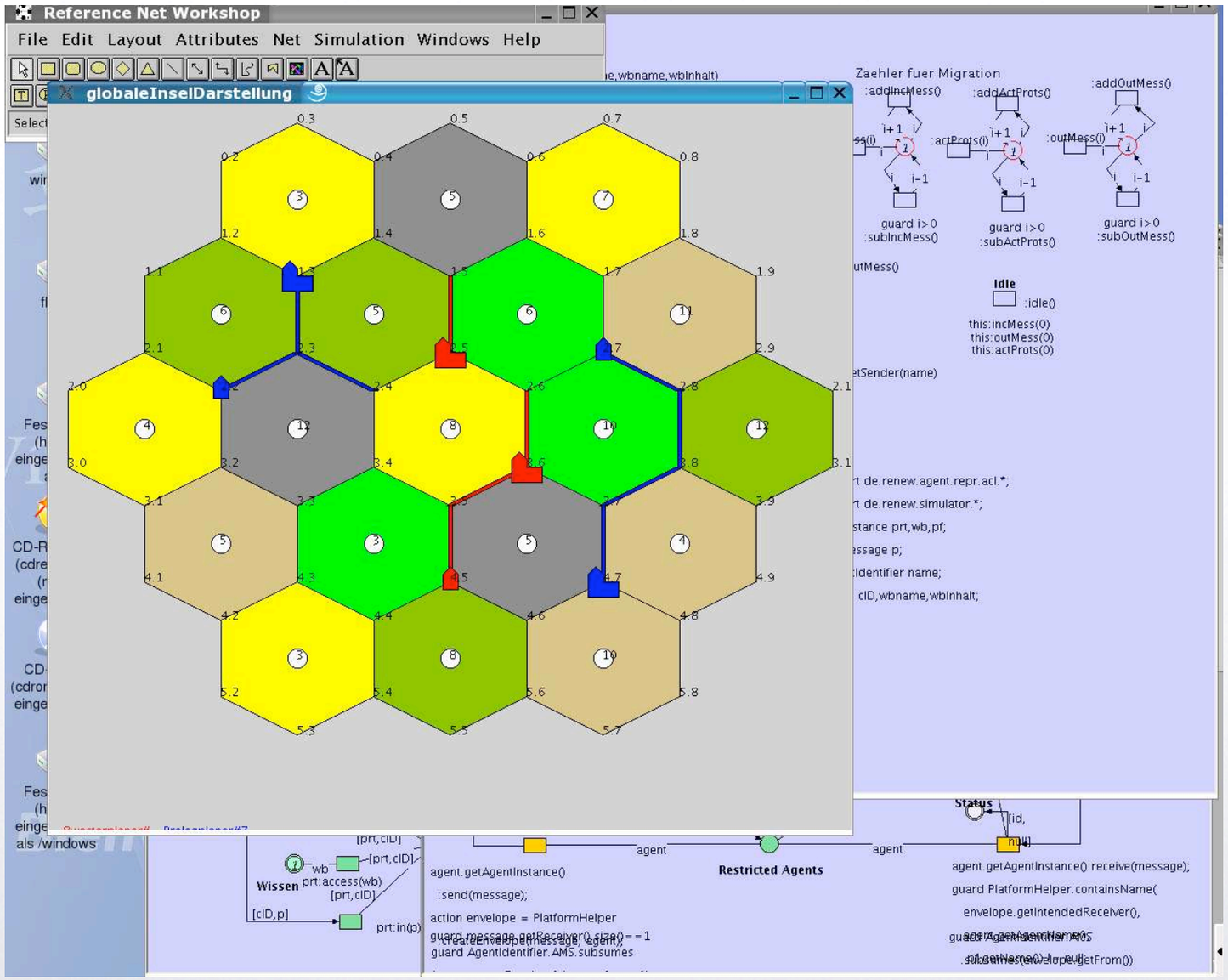
resources

incoming messages

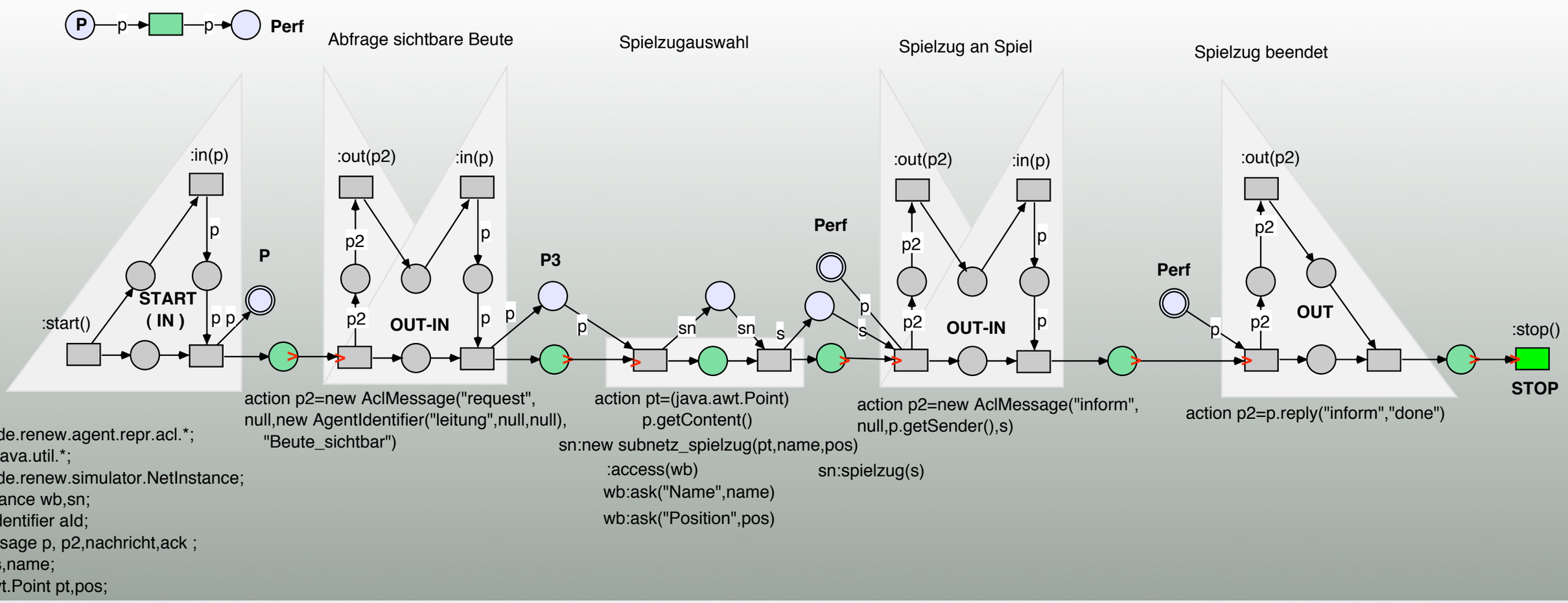
admin view



The User Interface



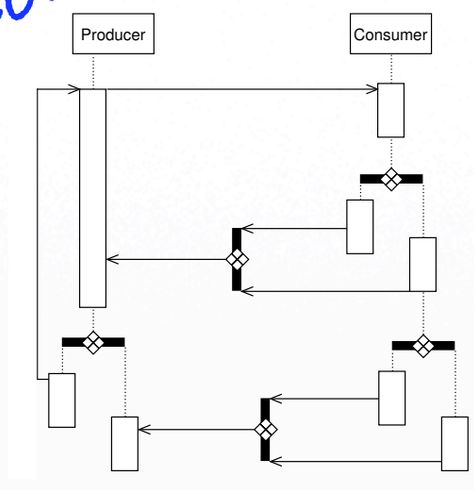
Modular Composition

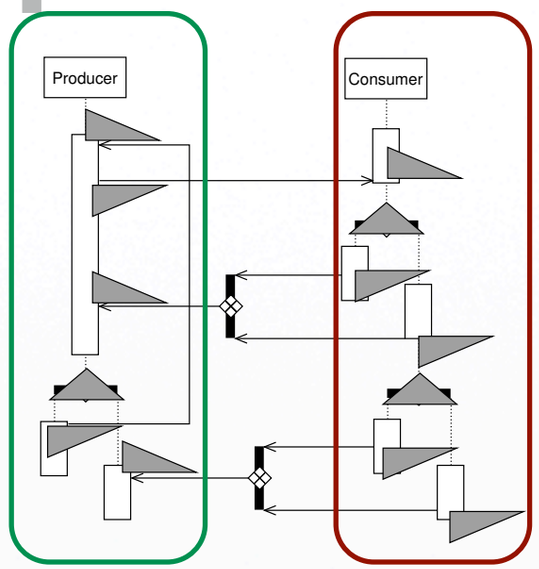


player's sub-process

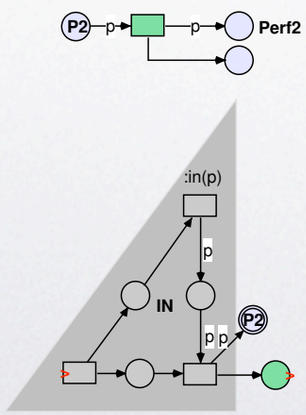
Net Components

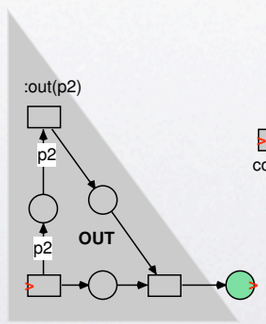
AUML Agent Interaction Protocols

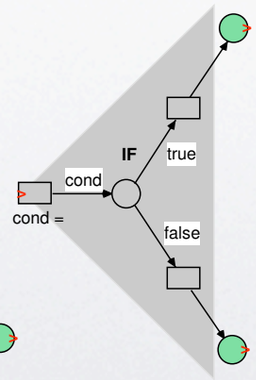


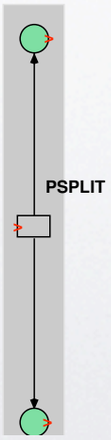


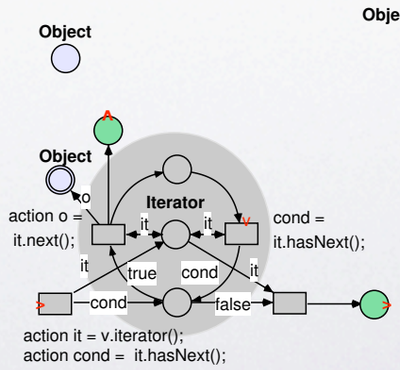
NC in NC out NC cond NC psplit NC iterator NC forall

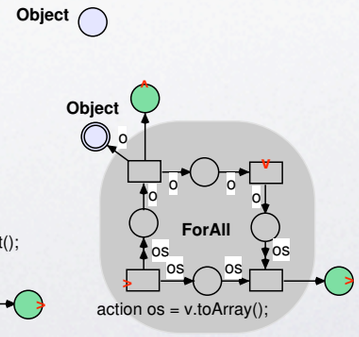




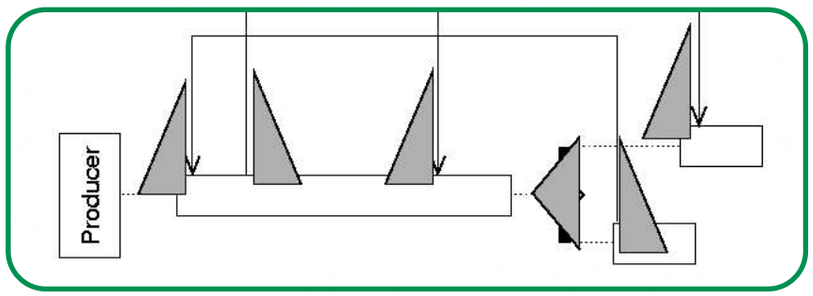




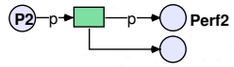


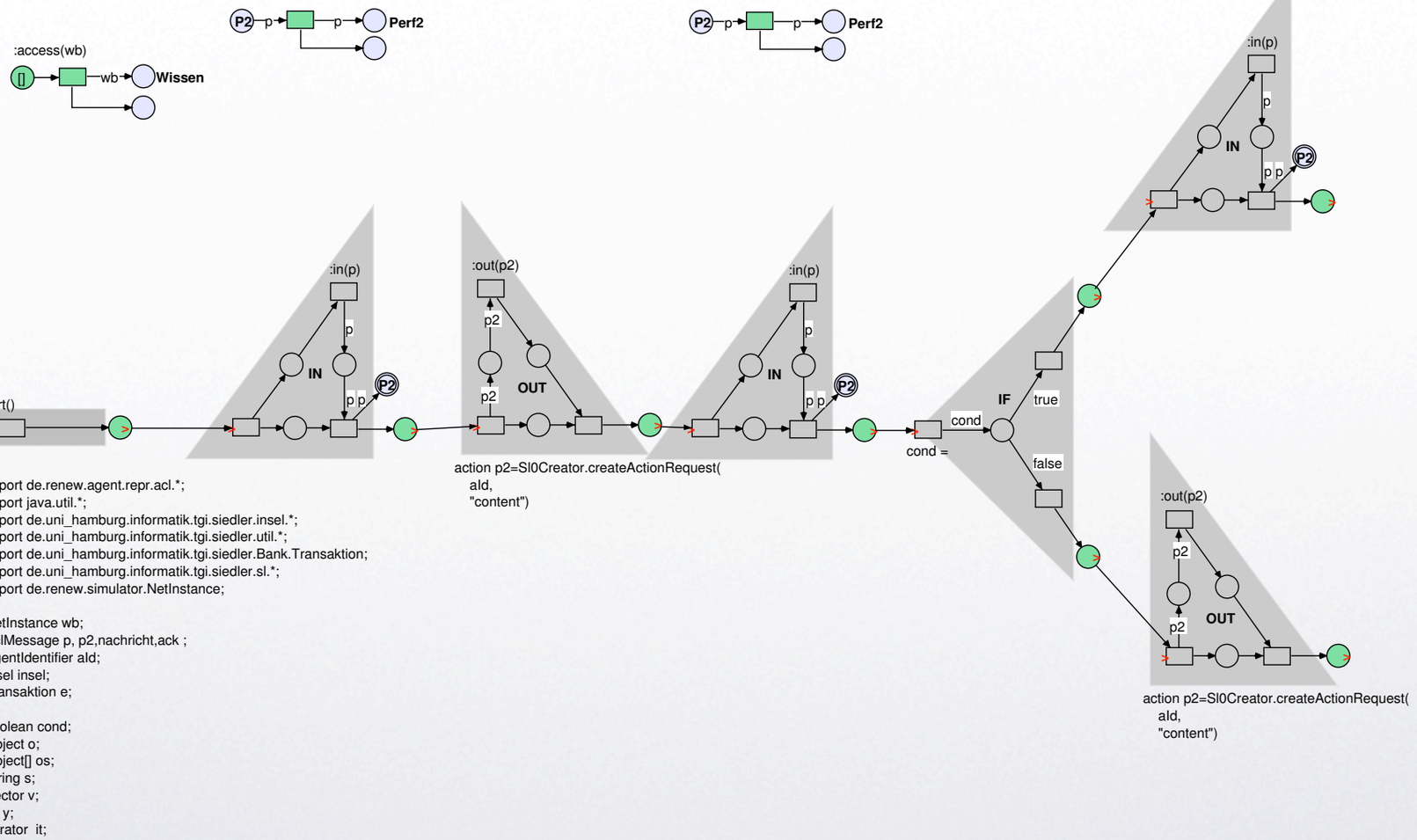


action p2=Sl0Creator.createActionRequest(



Producer

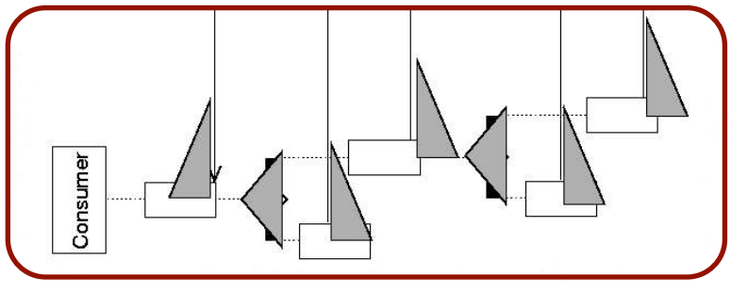




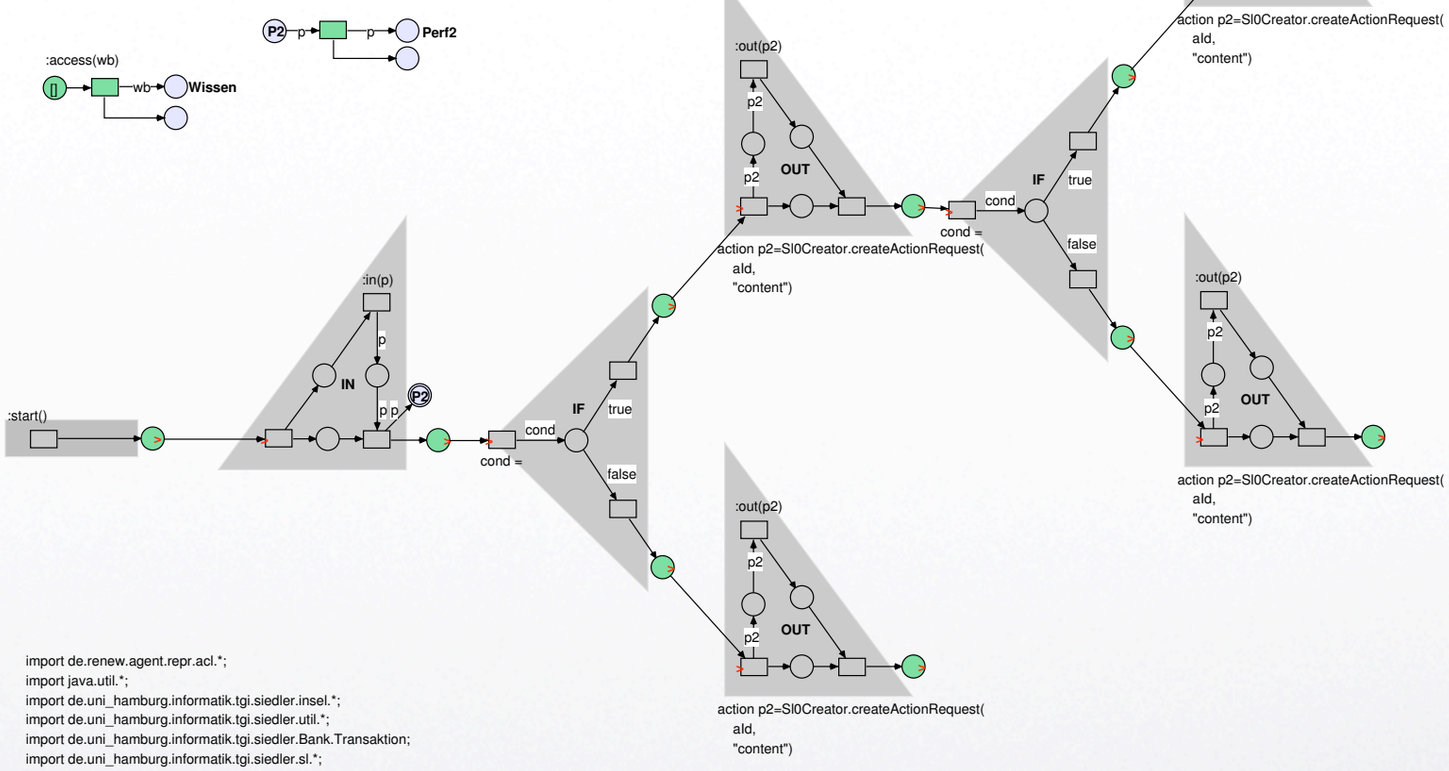

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.uni_hamburg.informatik.tgi.siedler.insel.*;
import de.uni_hamburg.informatik.tgi.siedler.util.*;
import de.uni_hamburg.informatik.tgi.siedler.Bank.Transaktion;
import de.uni_hamburg.informatik.tgi.siedler.sl.*;
import de.renew.simulator.NetInstance;

NetInstance wb;
AclMessage p, p2,nachricht,ack ;
AgentIdentifier aId;
Insel insel;
Transaktion e;

boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y;
Iterator  it;
```



Consumer



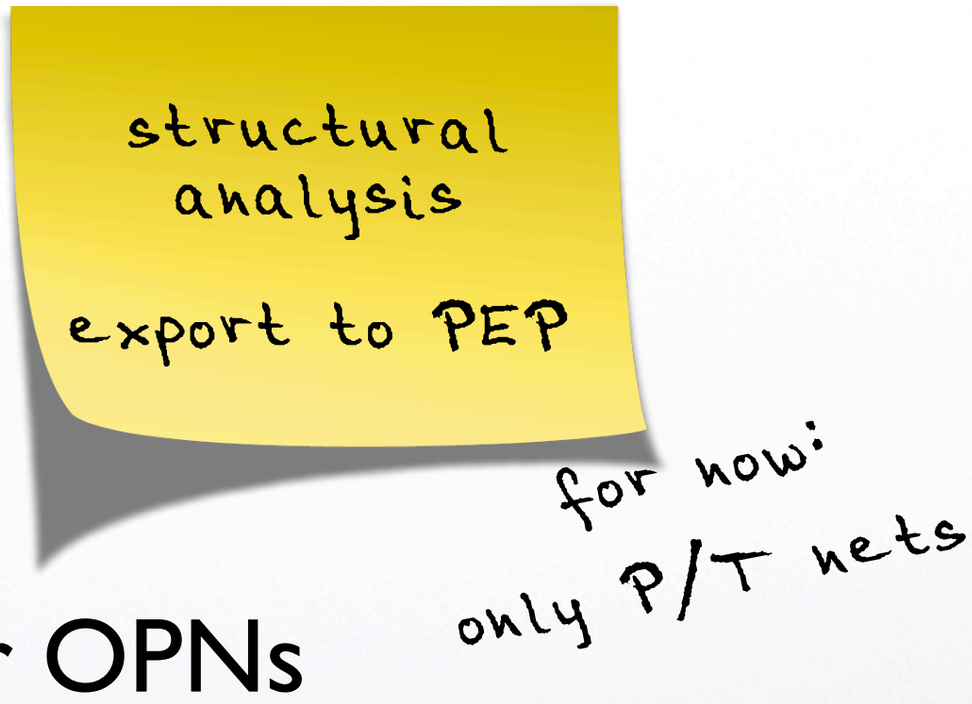

```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.uni_hamburg.informatik.tgi.siedler.insel.*;
import de.uni_hamburg.informatik.tgi.siedler.util.*;
import de.uni_hamburg.informatik.tgi.siedler.Bank.Transaktion;
import de.uni_hamburg.informatik.tgi.siedler.sl.*;
import de.renew.simulator.NetInstance;

NetInstance wb;
AclMessage p, p2,nachricht,ack ;
AgentIdentifier aId;
Insel insel;
Transaktion e;

boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y;
Iterator  it;
```

Why study translations?

- Renew supports
  - modelling
  - execution/animation
- Lack of analysis tools for OPNs
  - verification/validation
  - model checking




Encoding Transitions

```
obj_trans(task(_),print,[]) -->
    p2 =-=> X,
    p3 <=+= X,
    p5 <=+= X.
```

- remove a token from place p2
- add a token to p3
- add a token to p5
- no synchronisation is required


Synchronisation

```
obj_trans(sn,t,Ref) -->
  p1 =-=> TokenNet,
          {token_trans( t',
                        TokenNet,
                        TokenNetAfterFire,
                        Ref )},
        q1 <=+= TokenNetAfterFire.
```

- `token_trans`: synchronisation with TokenNet

The Initial Marking

```
start(
  [obj(sn, [bind(p1,[ref(low),ref(low),b,b,c]),
            bind(p2,[b]),
            bind(p3,[c])]),
   obj(low,[bind(q1,[1,1]),
            bind(q2,[])])
]).
```

- `1` = black token
- `b, c` = coloured tokens
- `ref(xxx)` = reference to net `xxx`

Some Results

- Faithfulness of the translation
  - Every firing sequence in the OPN is executable in its translation
  - and vice versa
- Extensibility to value semantics

Why choose Prolog?

- Value semantics usually relies on some notion of **unification** for join transitions
- Tools exist that allow efficient model checking
  - ProXTL / ProB
- can handle: *clear arcs*, *flexible arcs*, etc.
- integrate reference and value semantics

ProXTL / ProB

ProXTL 1.1.0: [flow_tribunal.P] : (c) Michael Leuschel

```
/* ['Examples/ColourObjectNets/flow_tribunal']. */

:- ensure_loaded('object_interpreter.P'). /* for Sicstus */
:- ['Examples/ColourObjectNets/object_interpreter.P']. /* for XSB */

top_level_net(flow).

start([ NET ]) :- new_net(flow,NET).

place(task(_),PN,[]) :-
   PN = p1 ; PN = p2 ; PN = p3 ; PN = p4 ; PN = p5 ; PN = p6 ; PN = p7 ;
   PN = p8 ; PN = p9 ; PN = p10 ; PN = p11 ; PN = p12.

place(flow,f1,[1]).
place(flow,PN,[]) :-
   PN = f2 ; PN = f3 ; PN = f4 ; PN = f5 ; PN = f6 ; PN = f7 ;
   PN = f8 ; PN = f9.

obj_trans(task(_T),new,[]) -->
   p1 <=+= 1,
   {print(new(task(_T))),nl}.
obj_trans(task(_),record,[]) -->
   p1 ==> X, p2 <=+= X.
obj_trans(task(_),print,[]) -->
   p2 ==> X, p3 <=+= X, p5 <=+= X.
```

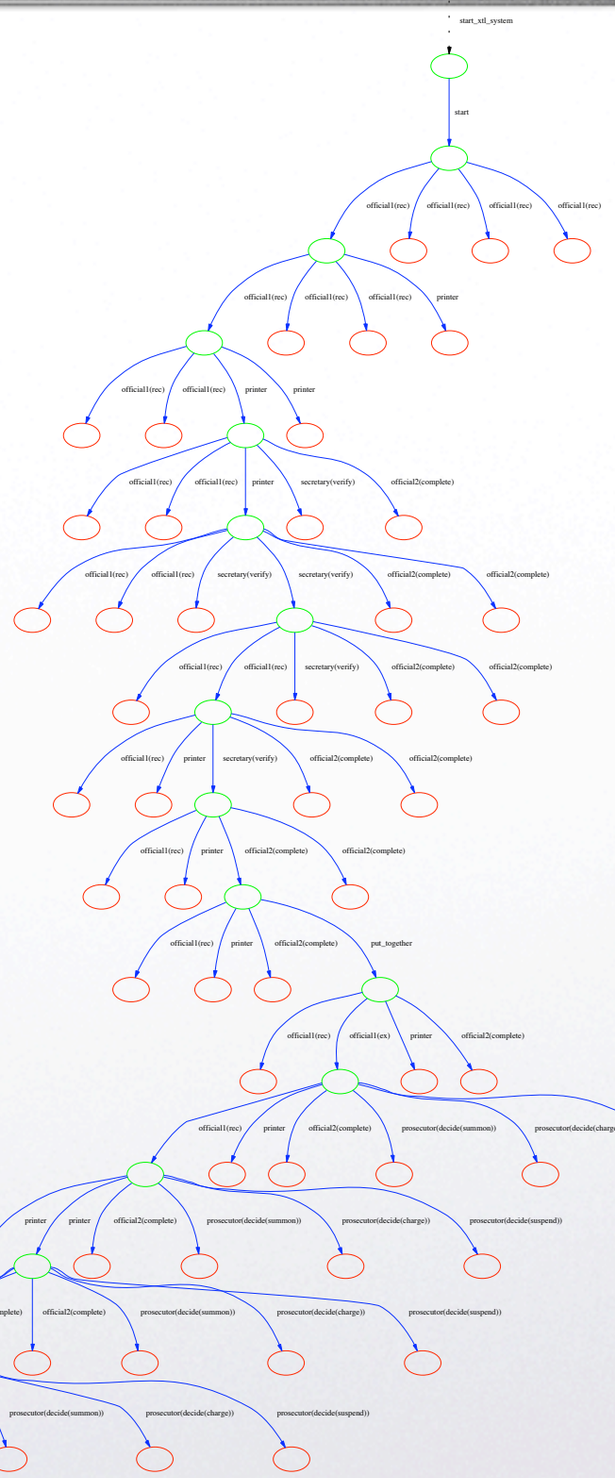
code window: encoding of OPN

StateProperties	EnabledOperations	History
card(flow,f4,1)	printer	official2(complete)
card(flow,f5,1)	secretary(verify)	printer
card(flow,f6,1)	official2(complete)	official1(rec)
card(flow,f7,1)	put_together	official1(ex)
card(flow,f8,0)	prosecutor(decide(summon))	put_together
card(flow,f9,0)	prosecutor(decide(charge))	official2(complete)
f1({})	prosecutor(decide(suspend))	secretary(verify)
f2({})	BACKTRACK	official1(rec)
f3({ref(task(t)),ref(task(t2))})		secretary(verify)
f4({ref(task(t3))})		printer
f5({ref(task(t3))})		printer
		official1(rec)
		official1(rec)
		start
		start_xtl_syste

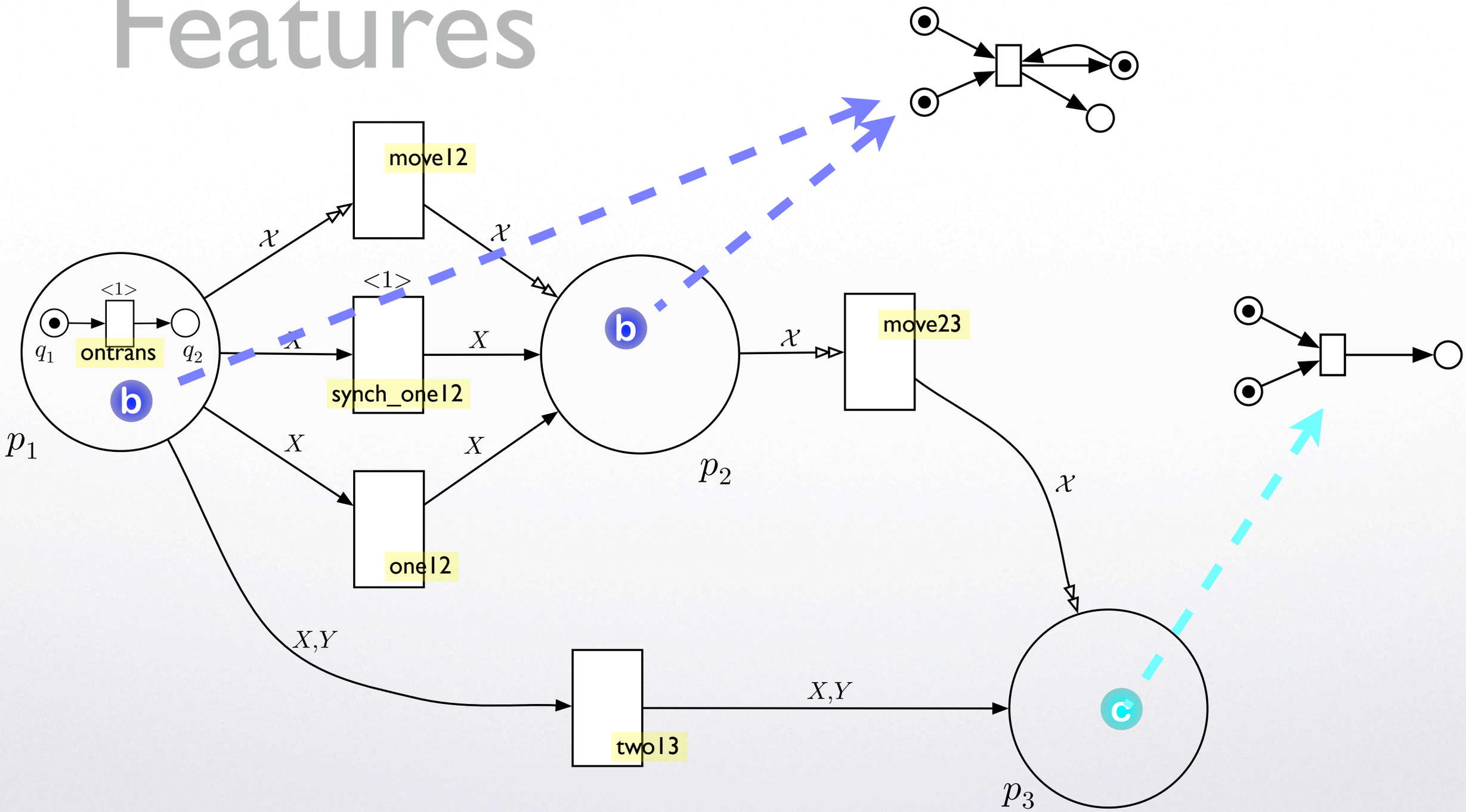
marking & state properties

enabled transitions

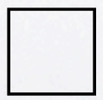
firing history



Features


move12
<1>
<1>
ontrans
q_1
q_2
b
synch_one12
move23
one12
two13
p_1
p_2
p_3
X
X,Y
b
c

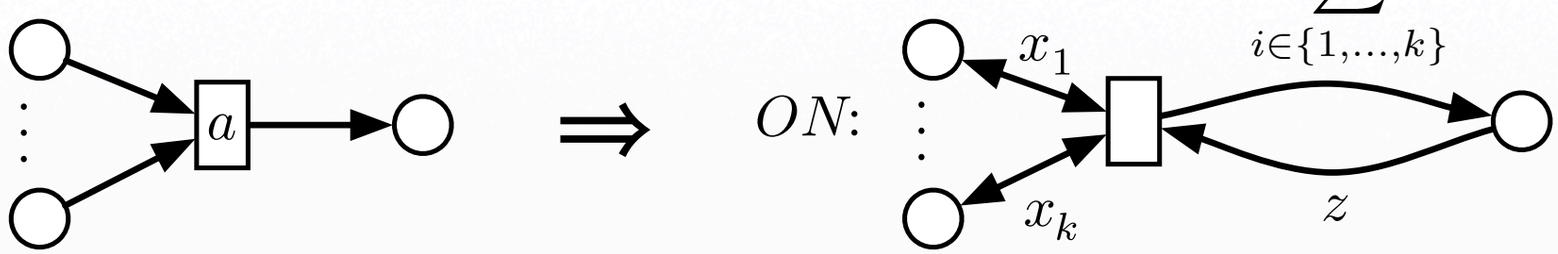
Dual Flow Nets

- Dual Flow Nets [Varea, 2002]
  - separation of data flow from control flow
  - two kinds of transitions:
    - transitions (control) 
    - hulls (data) 
  - tokens: (*control part*;*data part*)

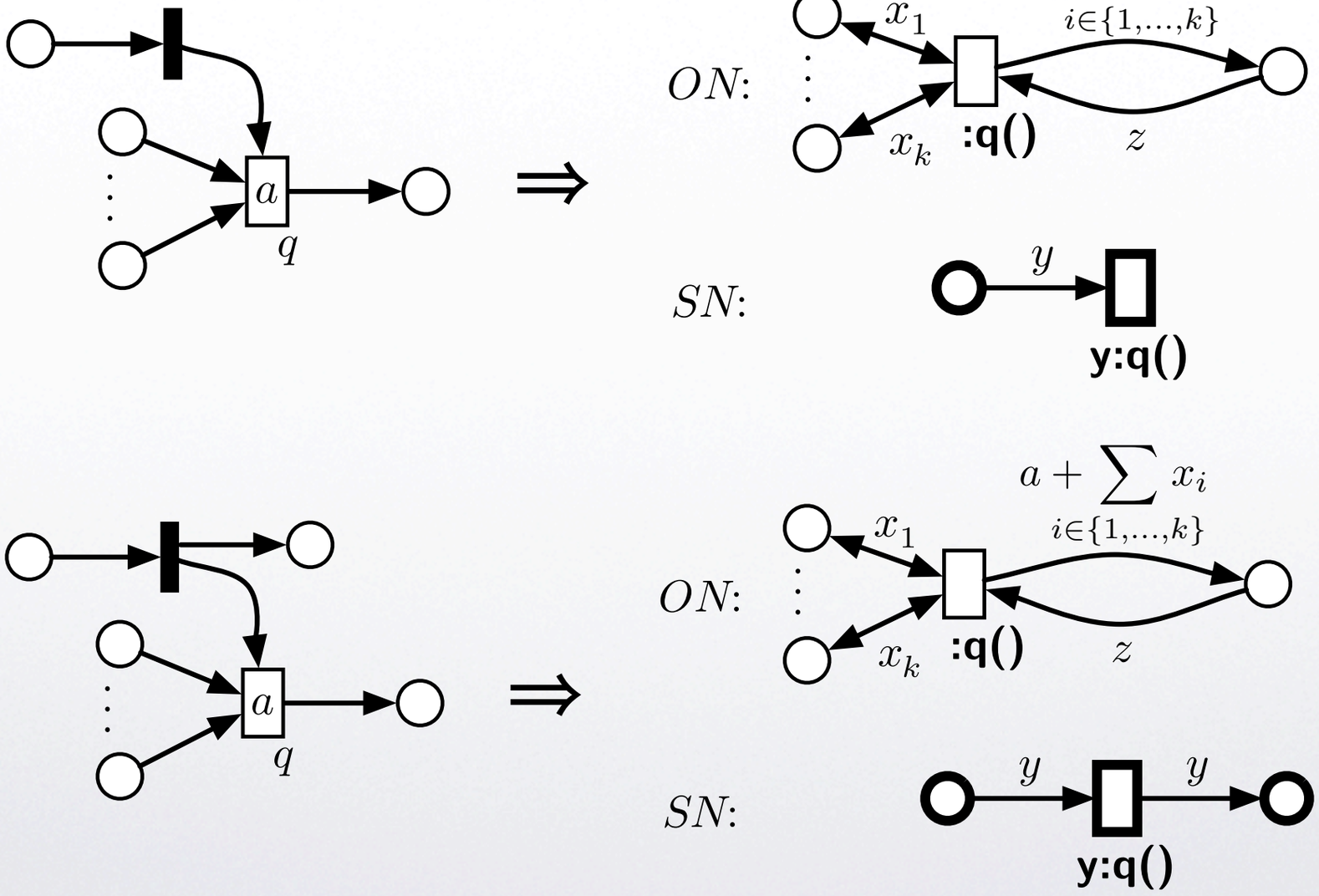
From DFNs to OPNs

- Translation:
  - control part --› system net
  - data part --› object net
  - triggering of hulls --› synchronisation

Dealing with Data …

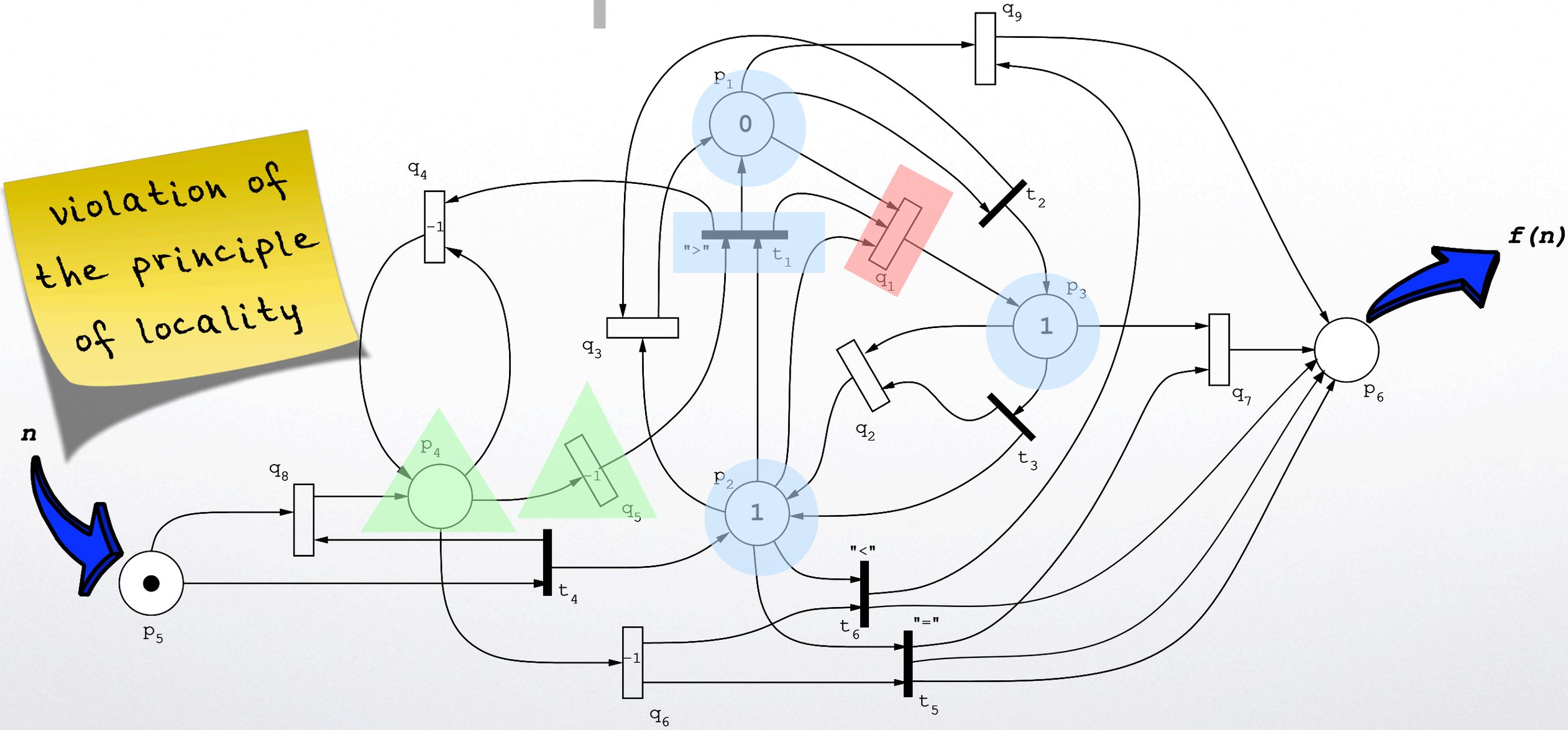


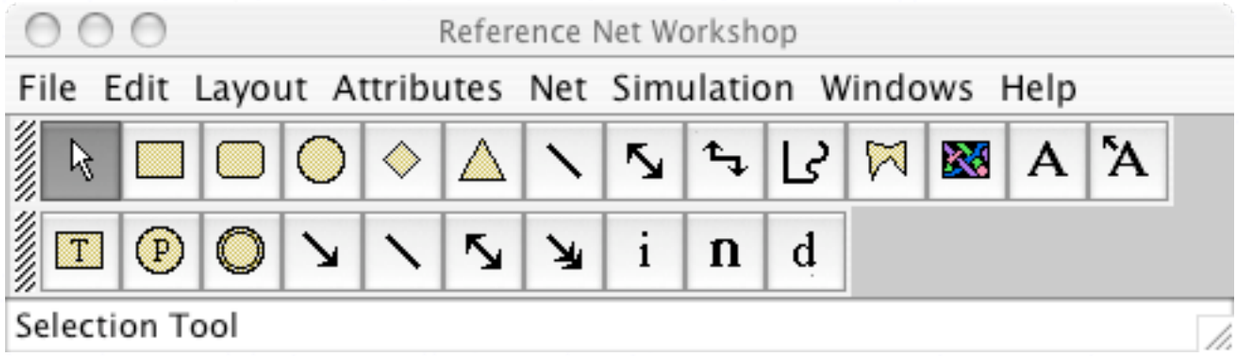
Transforming Triggers

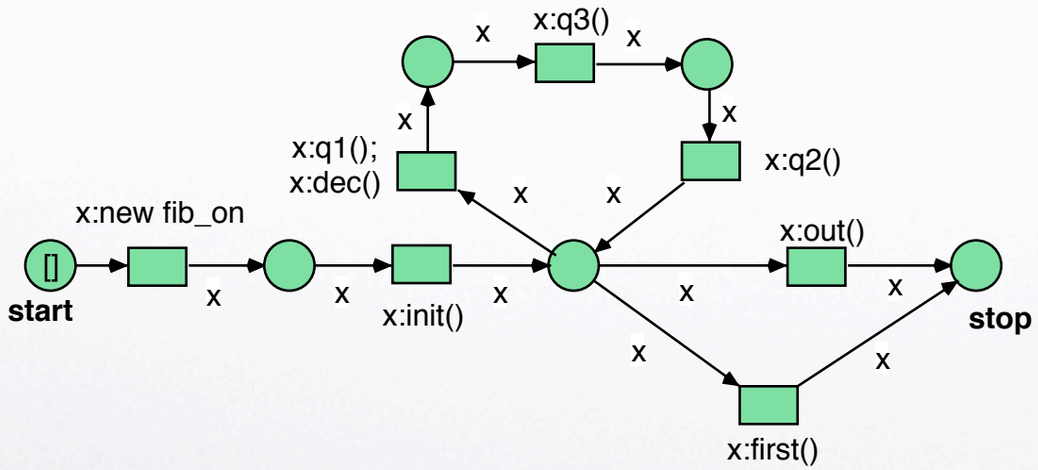


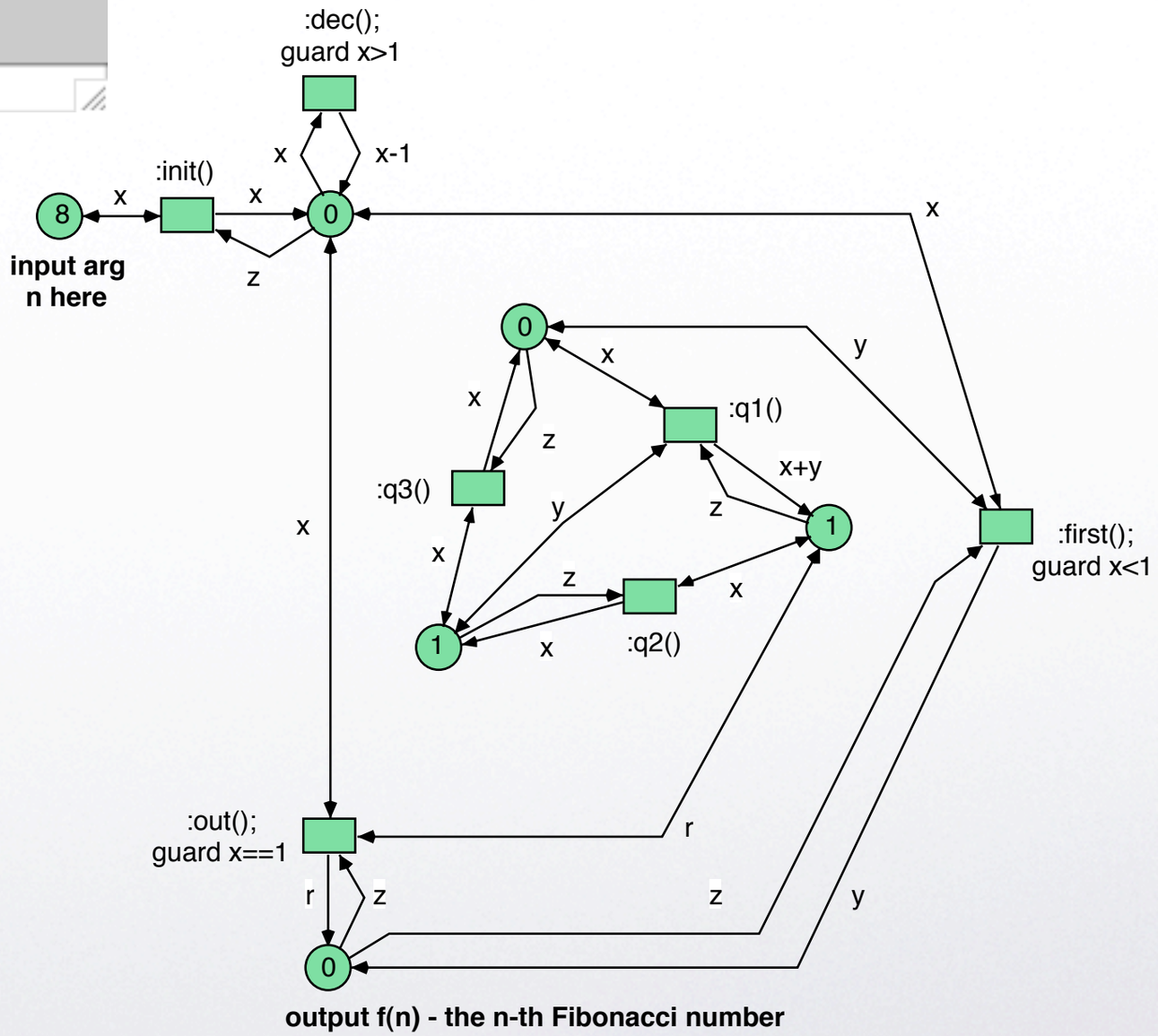
Example: Fibonacci

violation of the principle of locality




Reference Net Workshop
File Edit Layout Attributes Net Simulation Windows Help
Selection Tool


x:q3()
x:q1();
x:dec()
x:q2()
x:new fib_on
x:out()
start
x:init()
stop
x:first()


:dec();
guard x>1
:init()
input arg
n here
:q1()
x+y
:q3()
:q2()
:first();
guard x<1
:out();
guard x==1
output f(n) - the n-th Fibonacci number

Conclusion

- Software development with OPNs
  - suitable for large groups
    (tested with 25–40 developers)
  - rapid prototyping
  - analysis (sub-class HORNETS)
  - visualisation helps communicate problems

Multi-Level Nesting

You can hide all concurrency some of the time,
and you can hide some concurrency all of the time,
but you can't hide all concurrency all the time.

Luca Cardelli, 2003

... thank you for your attention!