Extensions of Petri Nets by Aspects to Apply the Model Driven Architecture Approach

E.E. Roubtsova

Open University of the Netherlands

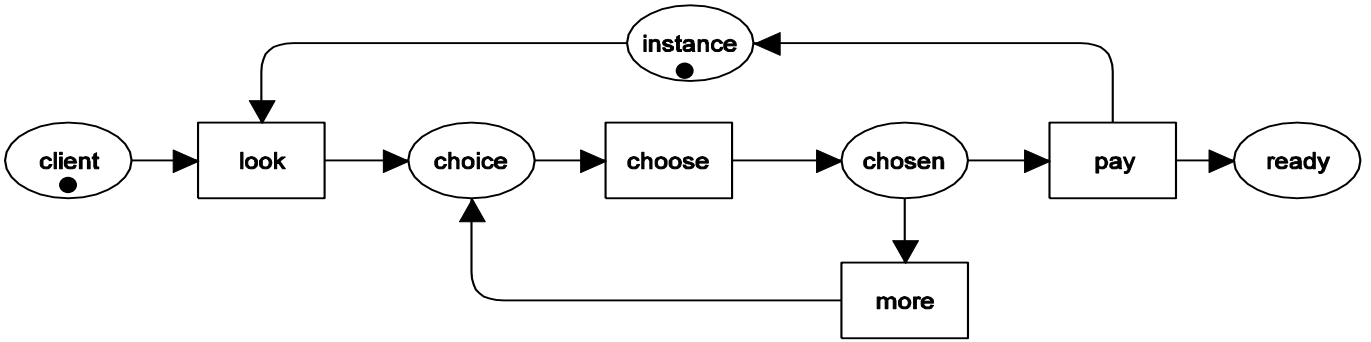
M. Aksit

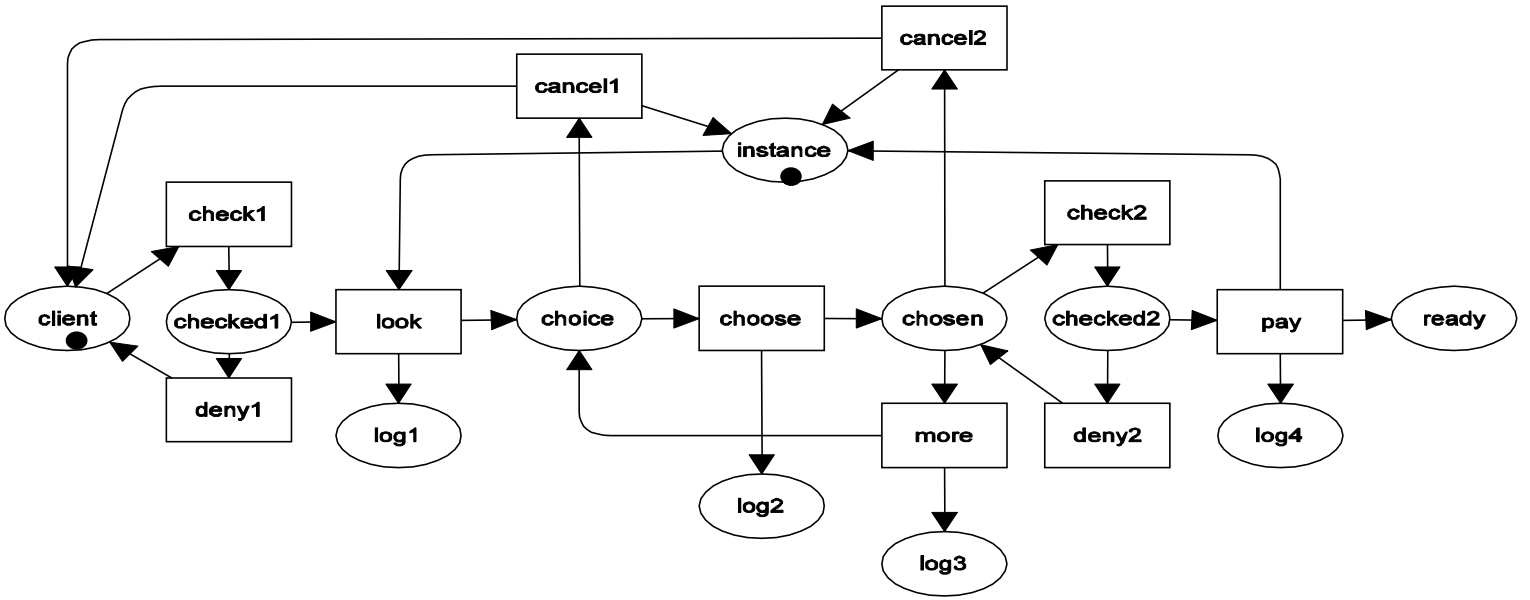
University of Twente, the Netherlands

Outline

- Problems with separation of concerns in Petri Nets
- Aspect Petri Nets
- Transformation of Models in Aspect Petri Nets
- Conclusion and Future work

Problems with separation of concerns in Petri Nets



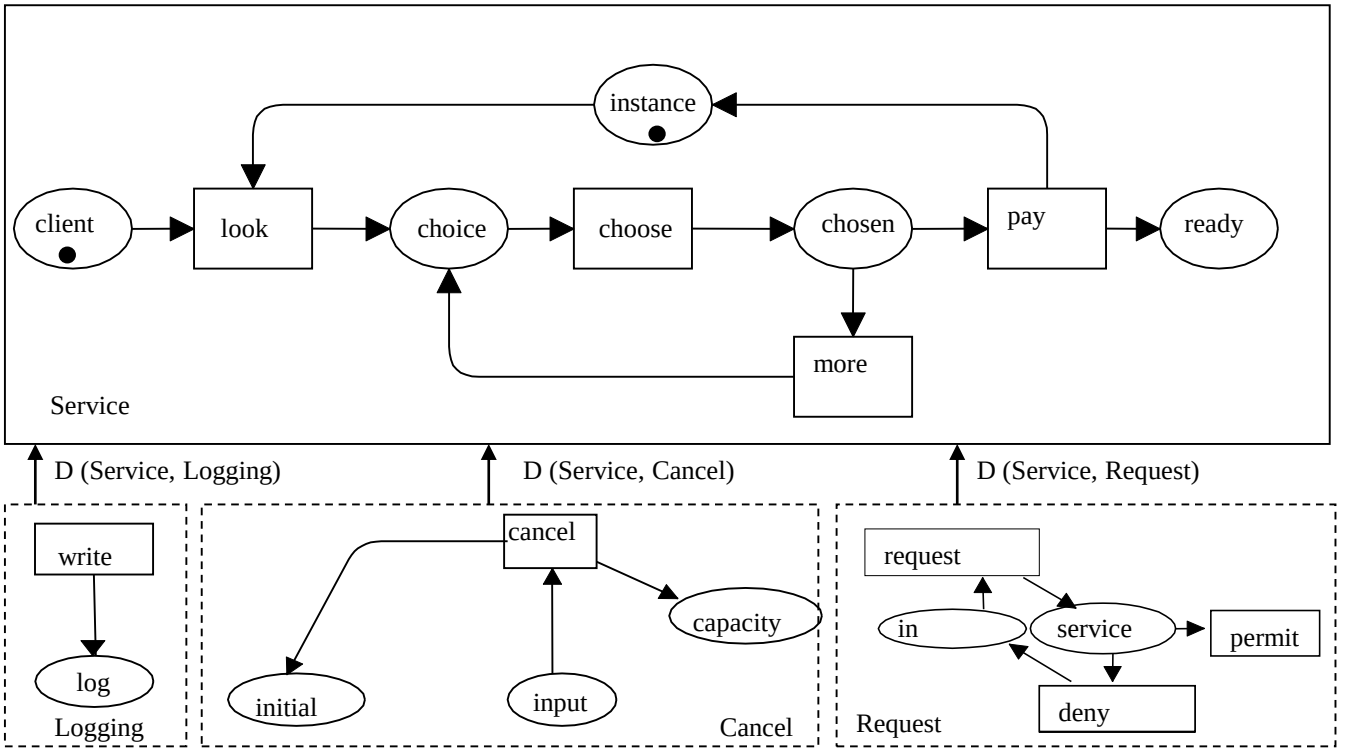


Aspect-Oriented Approach

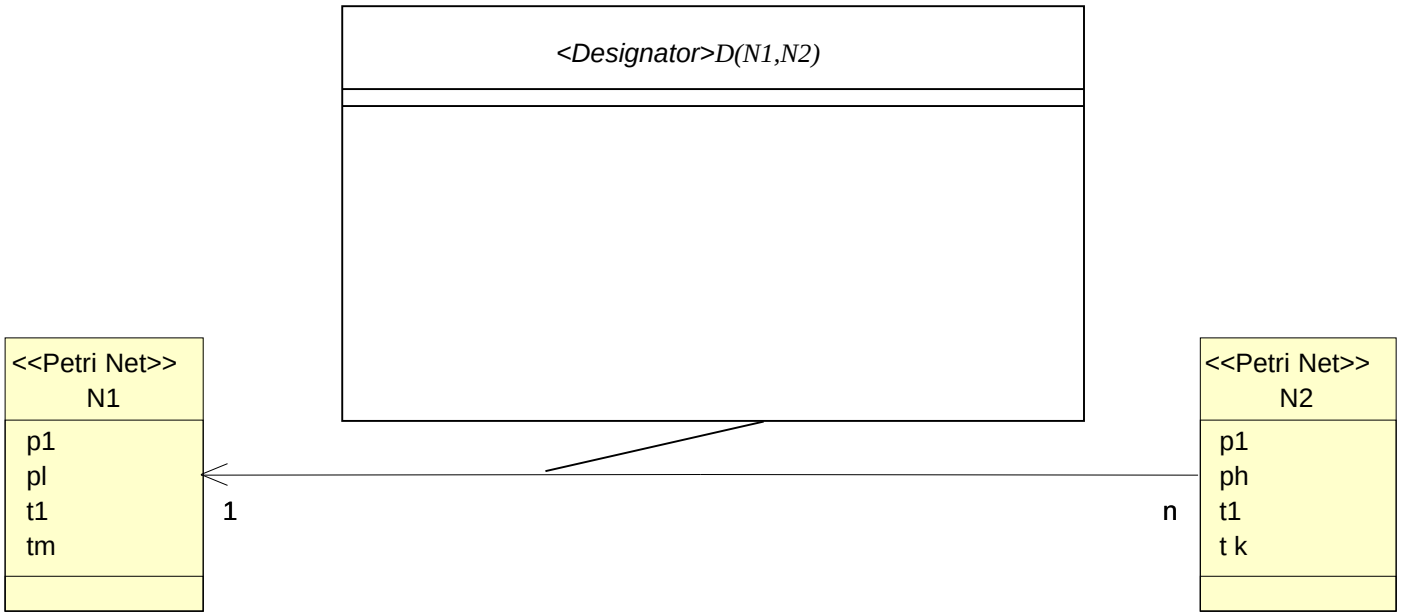
- Notation for modelling a concern;
- Join point model, i.e. elements of notation where concerns can be attached;
- Quantification mechanism presented by weaving expressions (designators) of concerns showing how concerns can be attached;

Achieve the principle of obliviousness of concern specification which means that the concern on which we quantify ”should not know” about other concerns and the mechanisms used for their quantification. Obliviousness allows designers to produce independent specifications of aspects.

Separation of concerns in Petri Nets



Aspect Petri Nets



A=(SN1, SN2,D(SN1,SN2))

Join Point Model

- A Petri Net N=(P,T,F,Mo)

$F \subseteq (P \times T) \cap (T \times P)$

M0: $P \rightarrow \{0,1,2,3,\ldots\}$

- $A=\{\varnothing, \{N\}, true\}$
- P set of places
- T set of transitions

A Language for Weaving of Petri Nets

- Copy

  copy(N1.p,N2); creates *p.N2*

  copy(N1.t, H2); creates *t.N2*

- Join

  joinToPlace(N1.p1, N2.p2)

  joinToTransition (N1.t1,N2.t2)

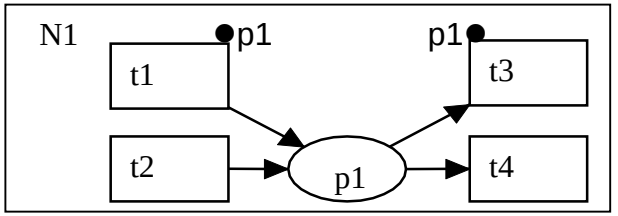
- Insert

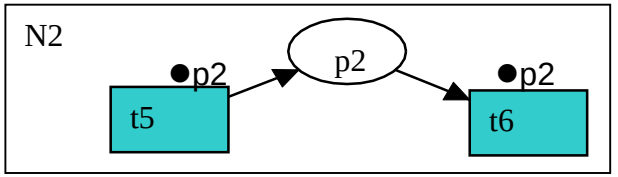
  InsertToPlace(N1.p1, N2.p2a, N2.p2b)

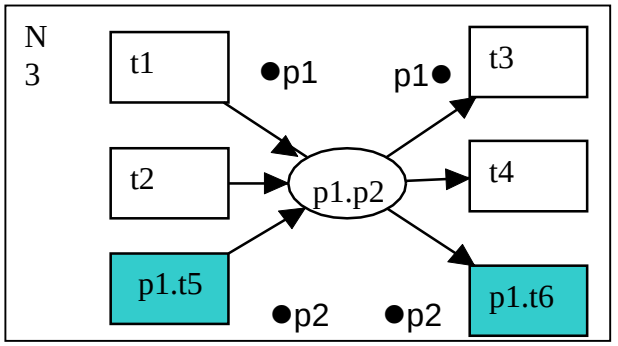
  InsertToTransition(N1.t1, N2)

Join Operations

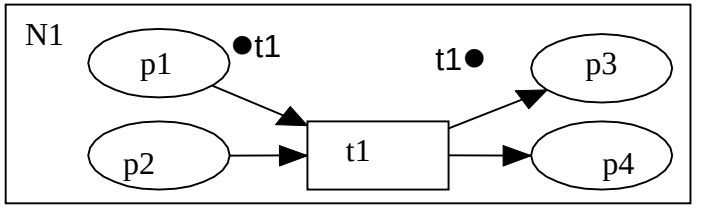
joinToPlace(N1.p1, N2.p2);

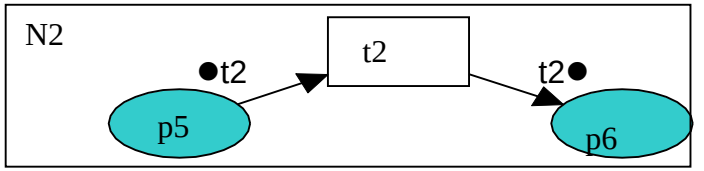


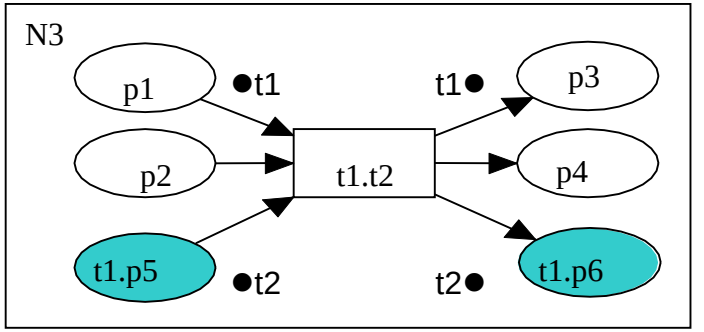




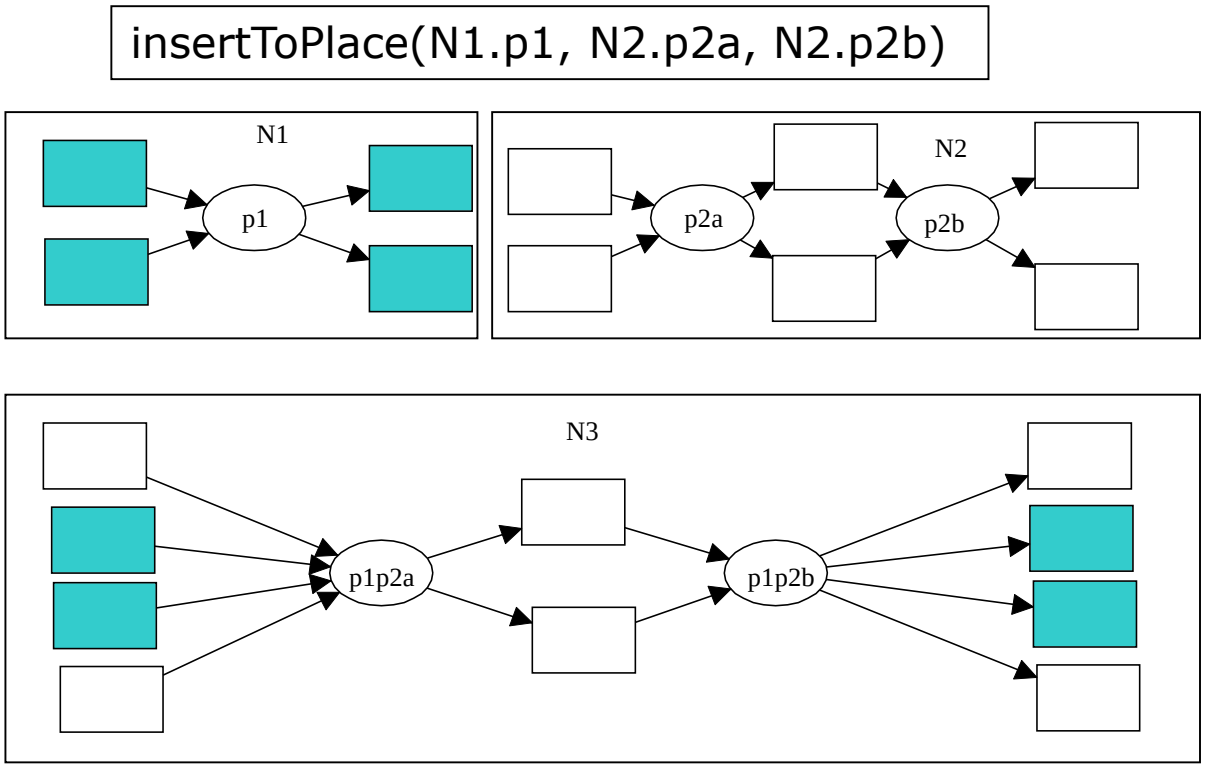
joinToTransition(N1.p1, N2.p2);





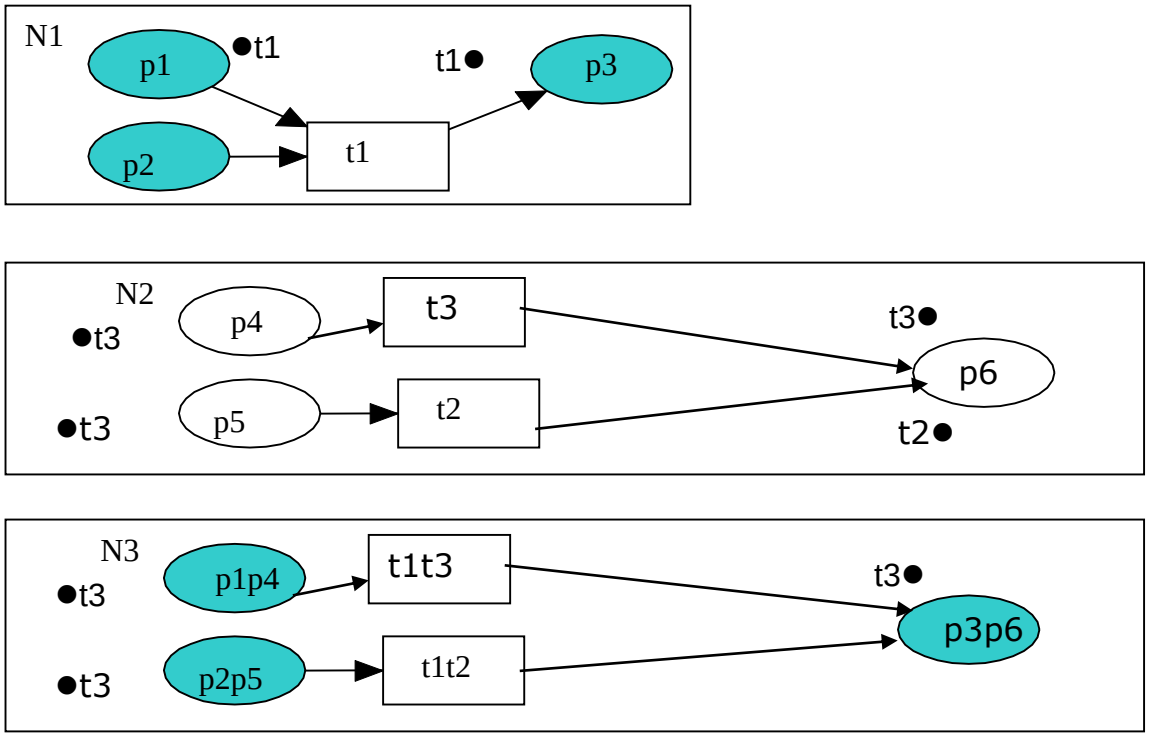


InsertToPlace



InsertToTransition

insertToTransition(N_1.t_1, N_2) is used in hierarchical Petri Net



A Language for Weaving of Petri Nets

WE::=∀N1.e1: B(N1.e1)
[copy(N1.e1,N2) & Expr(N1.e1, N2.e2)] | WE ∨ WE | WE ∧ WE]
B(N1.e1)::= N1.e1=<place name> |
N1.e1=<transition name>|
N1.e1 ∈<set of names>;
Expr(N1.e1, N2.e2)::=true | false |
joinToPlace(N1.p1,N2.p2)
joinToTransition(N1.t1, N2.t2)
InsertToPlace(N1.p1, N2.p2a, N2.p2b)
InsertToTransition(N1.t1, N2)
Expr(N1.e1, N2.e2) ∨ Expr(N1.e1, N2.e2)
Expr(N1.e1, N2.e2) ∧ Expr(N1.e1, N2.e2)

Modelling using Aspect Petri Nets

Modelling cancelling functionality

D(Service,Cancel):

$\forall$Service.p:(Service.p={ choice, chosen}

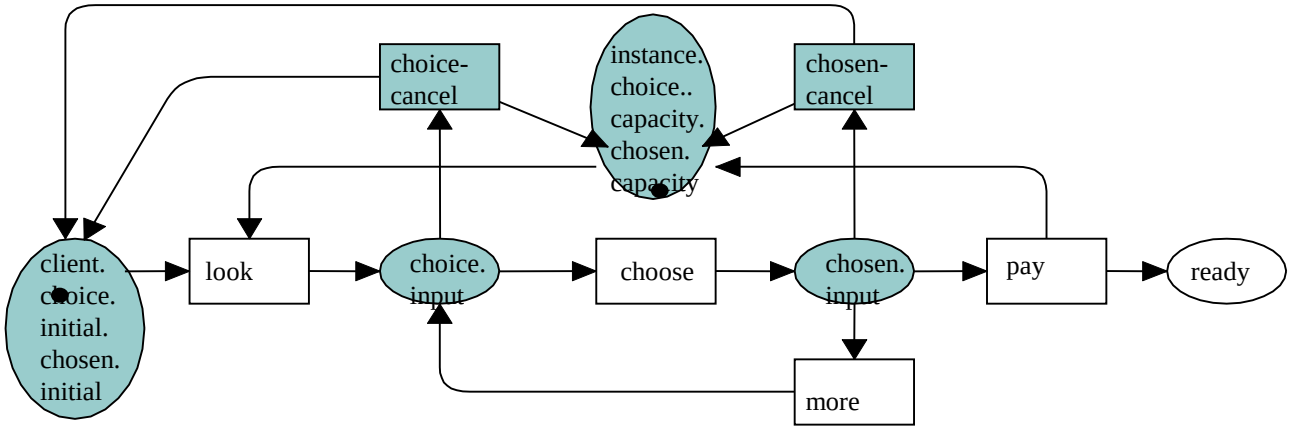
[copy(Service.p,Cancel) $\wedge$

joinToPlace(Service.p, Cancel.input) $\wedge$

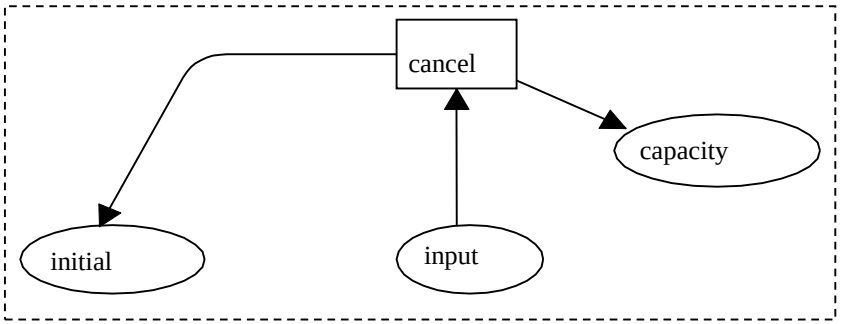
joinToPlace(Service.client, Cancel.initial) $\wedge$

joinToPlace(Service.instance,Cancel.capacity)];

Modelling cancelling functionality

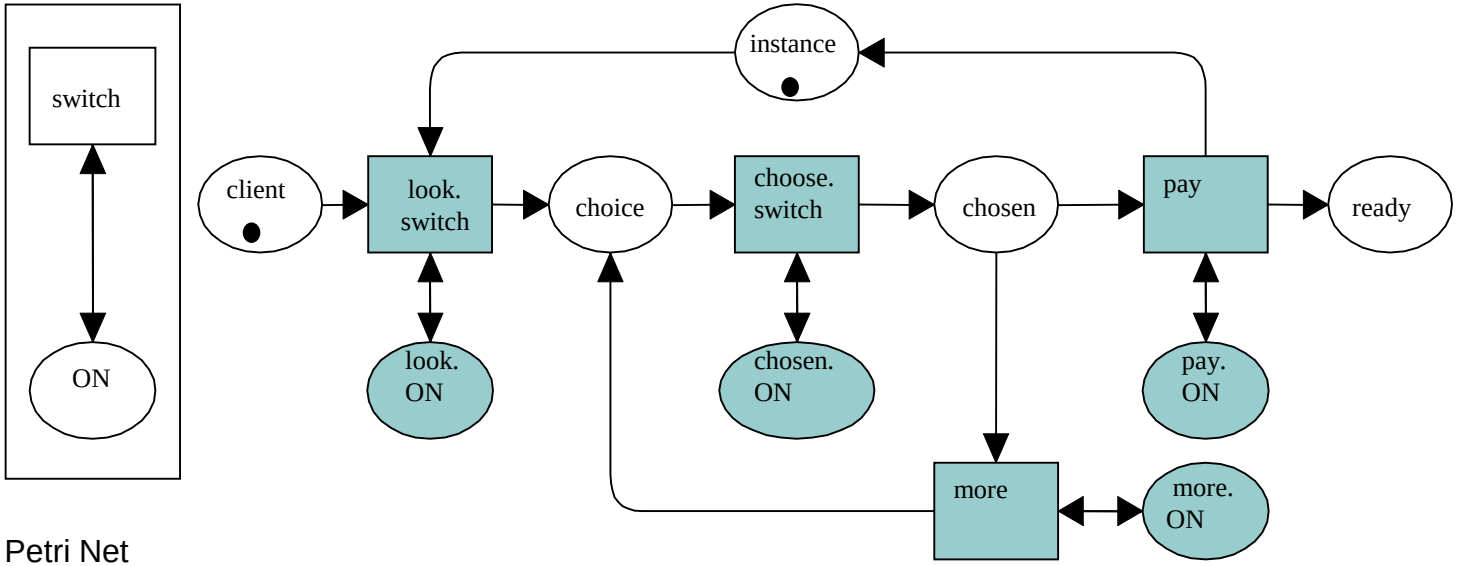


The result of weaving Petri Nets *Service* and *Cancel*



Petri net *Cancel*

Modelling logging functionality



Petri Net
Power

D(Service,Logging): ∀ Service.t:

[copy(Service.t, Logging)∧joinToTransition(Service.t, Logging.write)]

Useful quantifiers

insertAfterTransition(N1.t, N2.t,N2.p2a, N2.p2b)::=

$\forall$p:(p$\in$N1.t1●

[copy(N1.p,N2) $\wedge$

insertToPlace(N1.p, N2..p2a,N2l.p2b)]

Conclusion and Future work

Extending Petri Nets with aspect-oriented modelling concepts allows application the model driven architecture approach when modelling in Petri Nets.

Model based and aspect based approaches are complementary in this application domain.

Application of both approaches allows reduce complexity of Petri Nets models and support evolution of models without growing complexity.