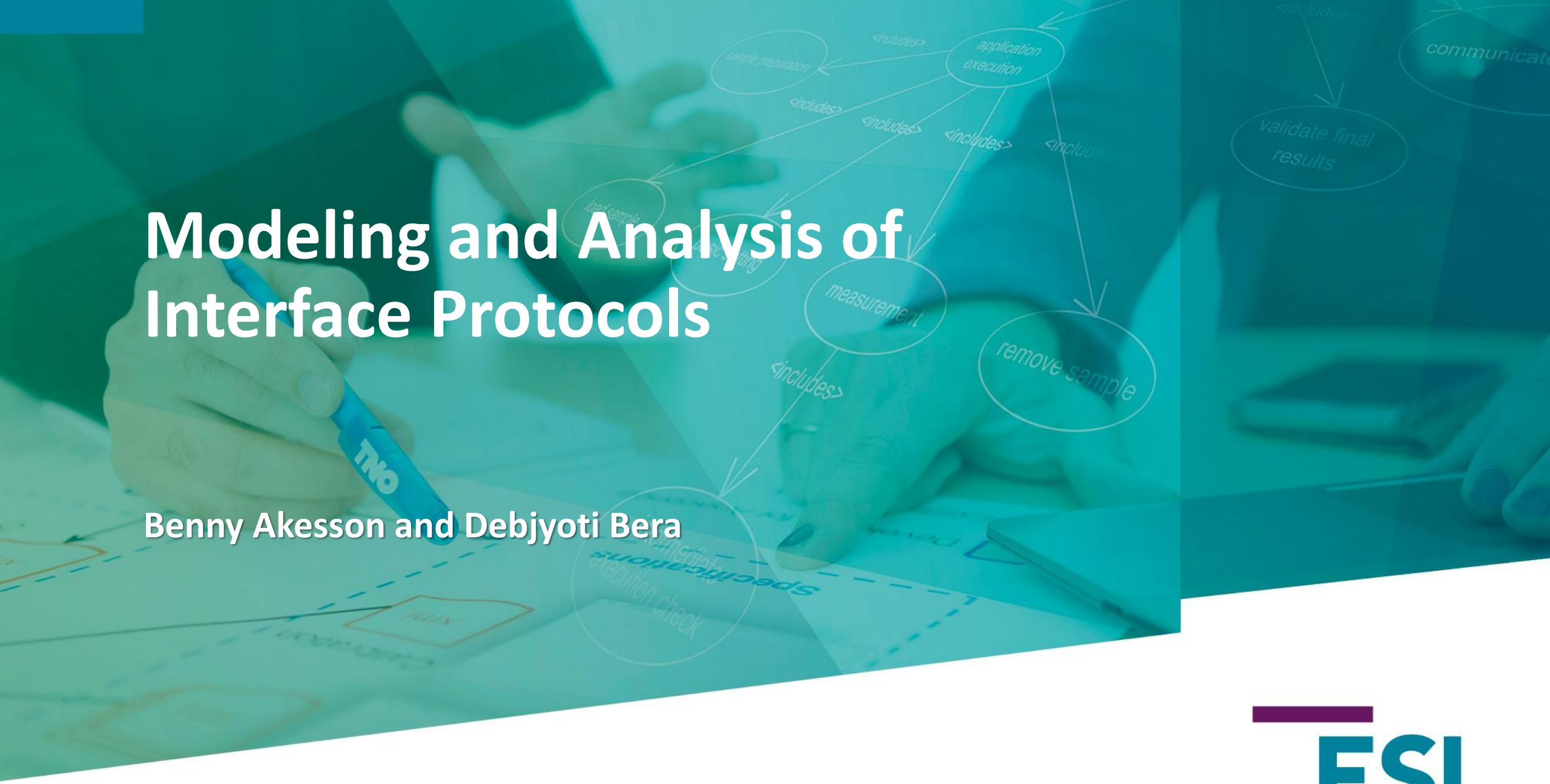
Modeling and Analysis of Interface Protocols

Benny Akesson and Debjyoti Bera



An initiative of industry, academia and TNO

ESI

Learning Outcomes

You will be able to…

1. **explain** how concurrent, non-deterministic, and cyclic behavior is captured using Petri Nets
2. **describe** the concept of termination and its relation to deadlock, livelock, and buffer overflow
3. **model** a given interface description as a Petri Net and **analyze** whether it terminates
4. **state** how concurrency, non-determinism, and cyclic execution behavior are represented in ComMA and how Petri Nets and associated analysis is used to validate/enforce correct behavior
5. **explain** how module contents connect to own daily work

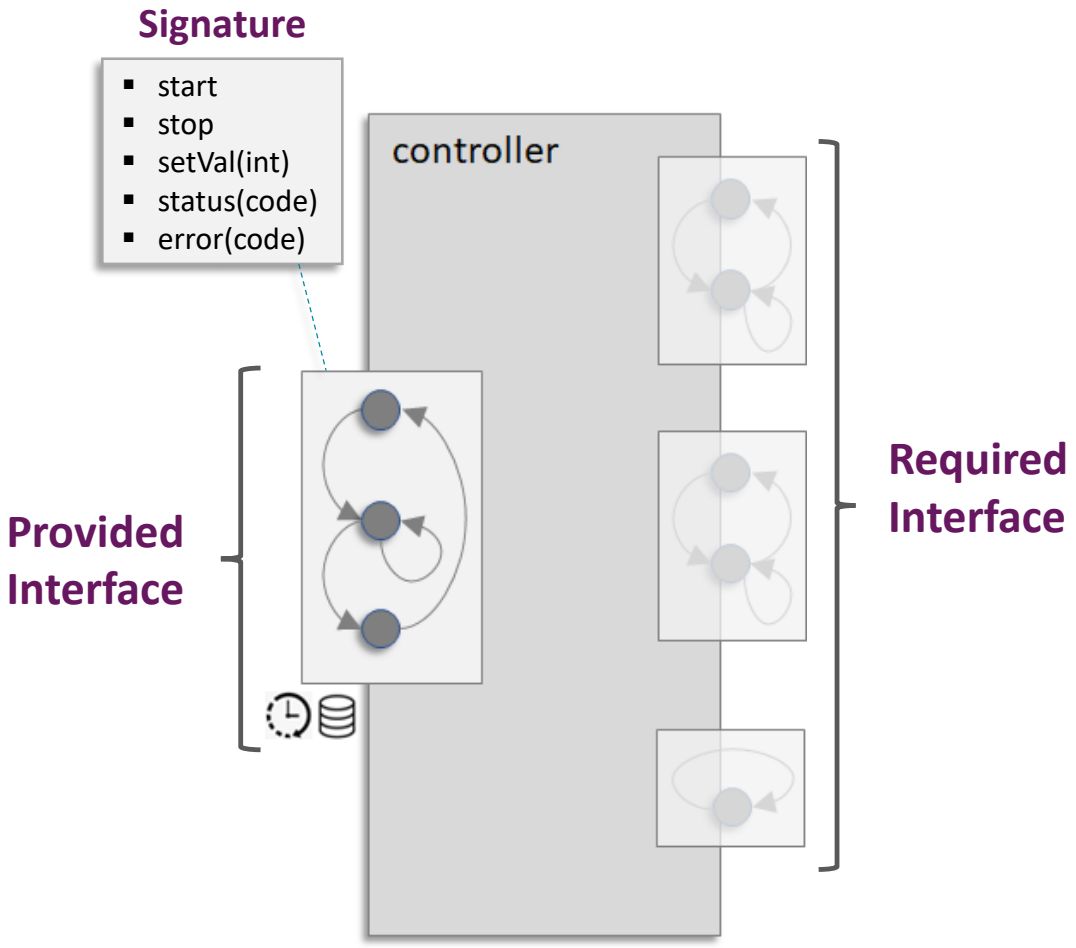
Interface Specifications

An interface must describe more than just a set of incoming and outgoing events

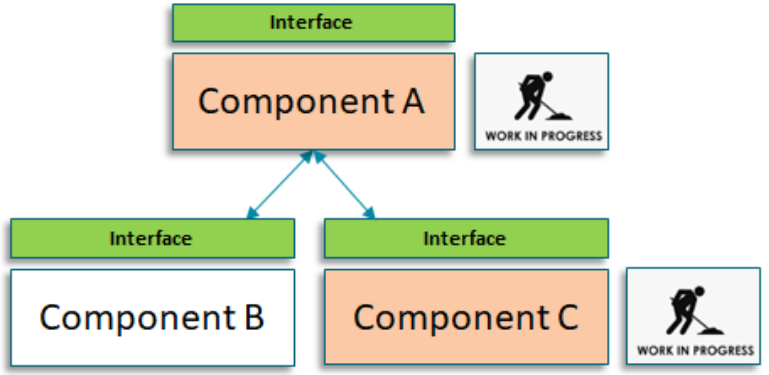
- The implementation of an interface induces an ordering between its set of events, a **protocol**
- Not making protocol behavior explicit often leads to **costly mistakes** later due to wrong assumptions

Interface descriptions must specify

- Signature (Events and Data Types)
- Order of events: protocol state machine



Benefits of Modeling Interfaces

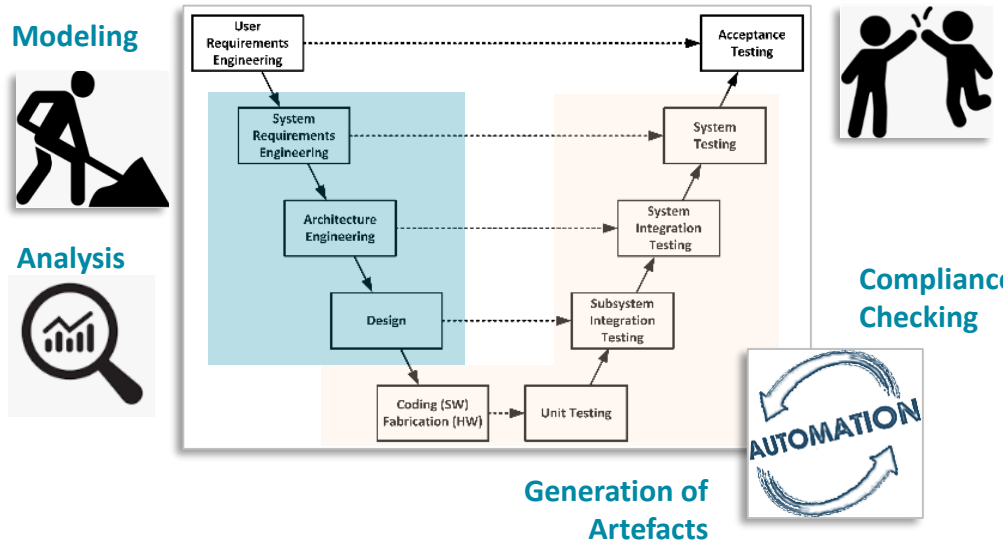


Avoid integration nightmares due to hidden assumptions

Reduce dependencies during software development

Improve quality and efficiency of software development

- Analysis and validation of behavior
- *Generation of code, documentation, etc.*
- *Monitoring, test automation, etc.*



Choice of Formalism

Carl Adam Petri

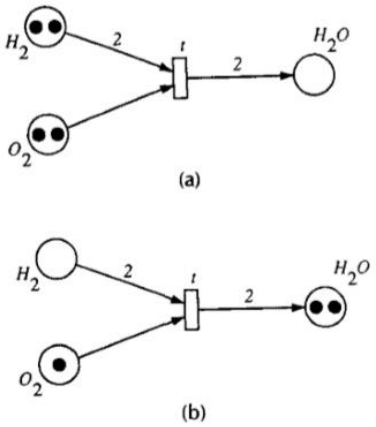
Many formalisms for modeling behavior, e.g.

- (Communicating) Finite State Machines
- Process algebra
- Petri Nets

Petri nets are a generalization of an Automata – PhD thesis, Carl Adam Petri in 1962

- **Good expressive power**: Turing complete in some variants
- Ideal for describing **concurrency** and **asynchronous communication**
- High modeling comfort due to its **graphical syntax**
- There are **many classes and extensions** of Petri nets for modeling different aspects
- Many existing **theoretical results** and **analysis techniques**

Applications of Petri Nets



Many application domains

- distributed systems, communication protocols, business processes,
- manufacturing and logistics, decision support systems,
- multiprocessor systems, programmable logic and VLSl arrays,
- modeling processes in biology and chemistry
- anything with behavior

Petri Nets are a Biologist's Best Friend

Nicola Bonzanni[1,2], Anton Feenstra[1], Wan Fokkink[1], and Jaap Heringa[1]

[1] VU University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands,
{n.bonzanni,k.a.feenstra,w.j.fokkink,j.heringa}@vu.nl
[2] The Netherlands Cancer Institute, Plesmanlaan 121, 1066 CX Amsterdam, The Netherlands

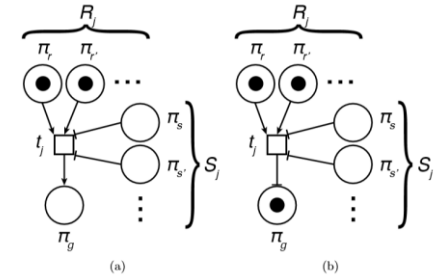


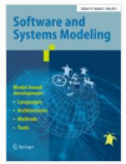
Figure 2. Graphical representation of the positive (a) and negative (b) gene interactions depicted in Fig. 1 using the folded network definition $B = \langle \Pi, T, F, A, I \rangle$. Negative arcs belonging to I have a flat arrowhead. Both transitions are enabled by the depicted markings. In practice, R and S are small, further simplifying the representation.

A Petri Nets Model for Blockchain Analysis

Andrea Pinna ✉, Roberto Tonelli, Matteo Orrú, Michele Marchesi

The Computer Journal, Volume 61, Issue 9, September 2018, Pages 1374–1388, https://doi.org/10.1093/comjnl/bxy001

Published: 25 January 2018 **Article history** ▾



Software & Systems Modeling

May 2015, Volume 14, Issue 2, pp 703–710 | Cite as

Petri nets in systems biology

Authors Authors and affiliations

Ina Koch ✉

Stochastic Petri Nets, Chemical Reaction Networks and Feynman Diagrams

John Baez, Jacob Biamonte, Brendan Fong

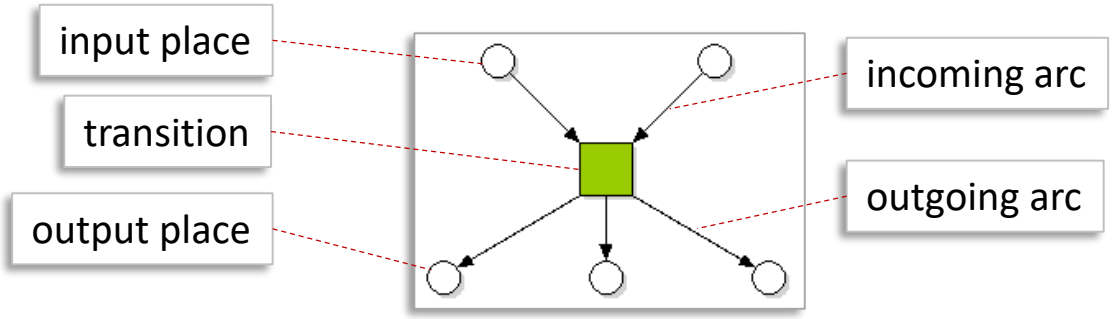
Supervised and Unsupervised Learning by Using Petri Nets

Publisher: IEEE

3 Author(s) Victor R. L. Shen ; Yue-Shan Chang ; Tony Tong-Ying Juang **View All Authors**

Introduction to Petri Nets

Syntax of Petri Nets



Input Places	Transition	Output Places
Preconditions	Event	Postconditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Task or job	Resources released
Conditions	Clause in logic	Conclusion(s)
Buffers	Processor	Buffers

Source: Murata, Tadao. "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE* 77.4 (1989): 541-580.

Petri Nets are a bi-partite graph where arcs connect places and transitions

- Places never connect to places and transitions never to transitions

The direction of arcs are defined with respect to transitions

- Bi-directional arcs are syntactic sugar that indicate both incoming and outgoing arc between a place and a transition

Semantics of Petri Nets

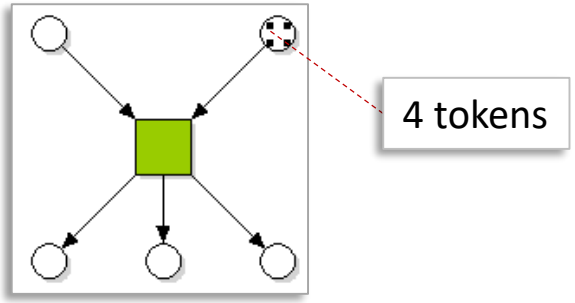
The dynamic behavior of Petri Nets are provided by tokens

- The distribution of tokens in the net is called a **marking**, representing its **state**
- The net starts from a pre-defined **initial marking**

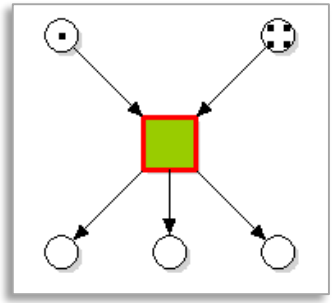
Transitions are enabled when there is at least one token in all of its input places

A firing transition consumes one token from each input and produces one in each output place

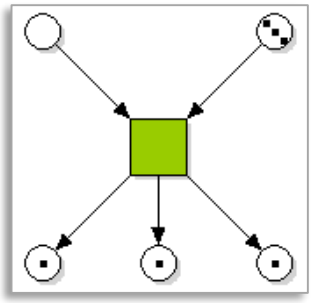
- Enabled transitions are fired in sequence in **non-deterministic order**



Not Enabled!



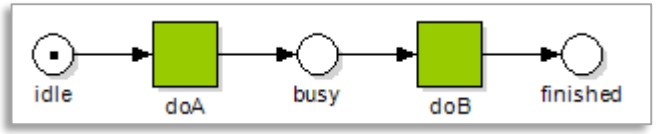
Enabled, Can Fire!



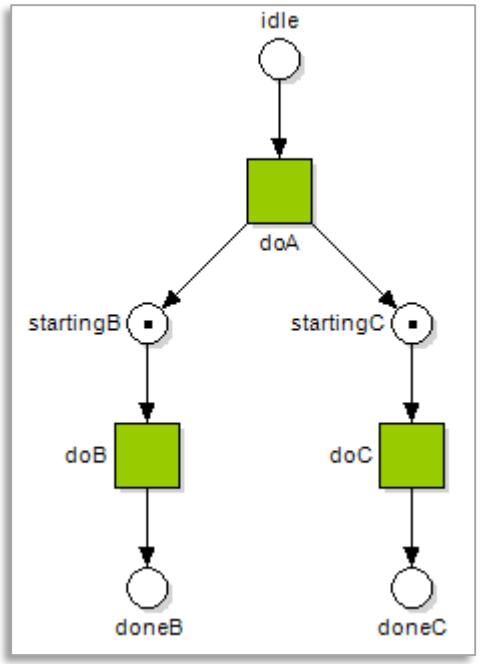
Fired!

Modeling with Petri Nets

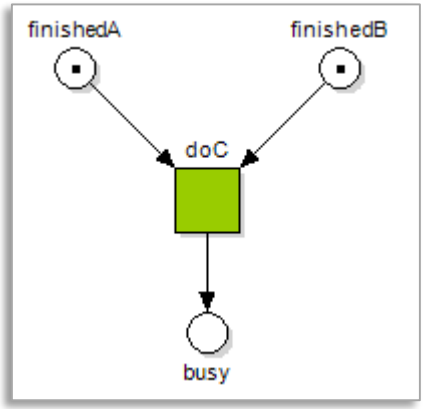
Sequential



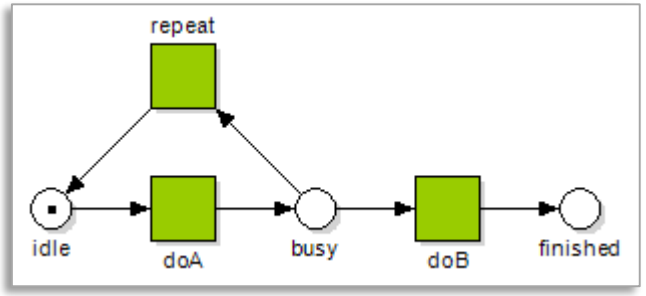
Concurrency



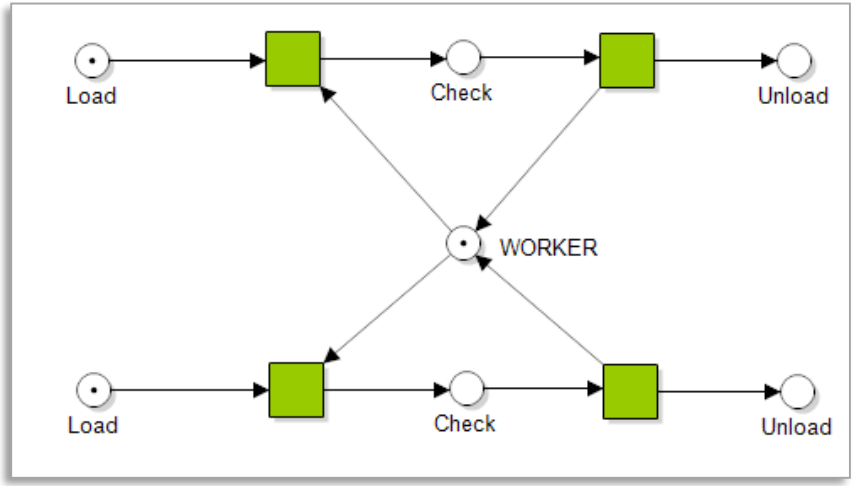
Synchronization



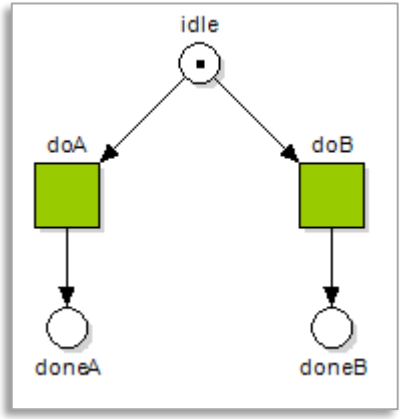
Loops



Mutual Exclusion



Choice / Non-Determinism

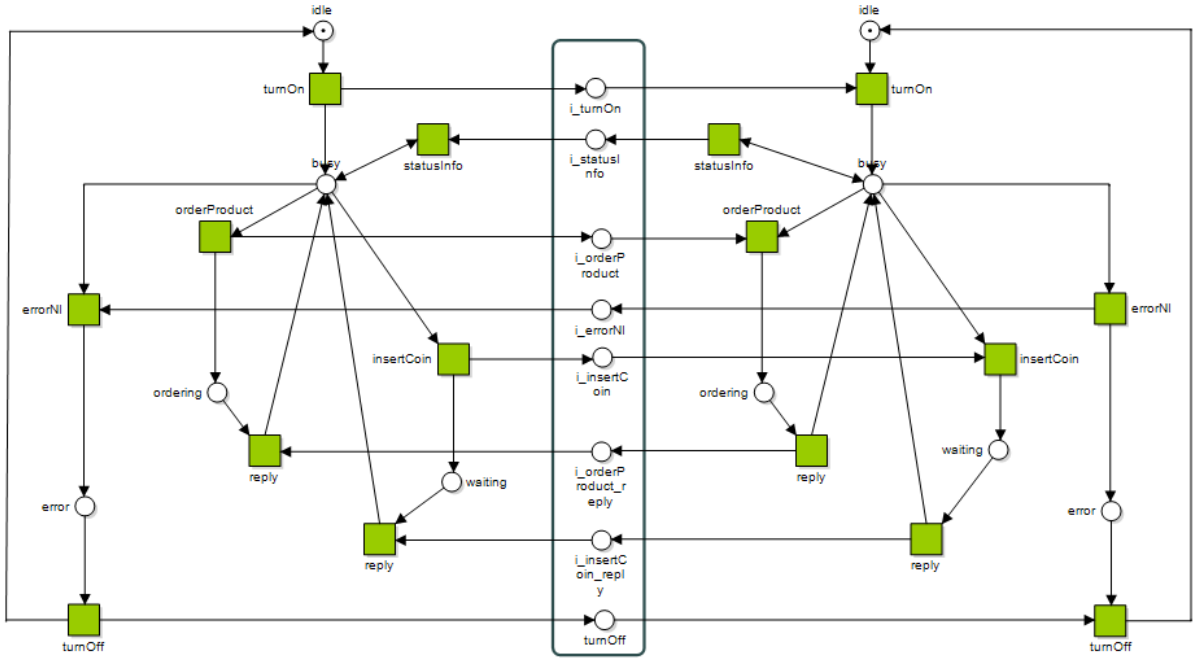


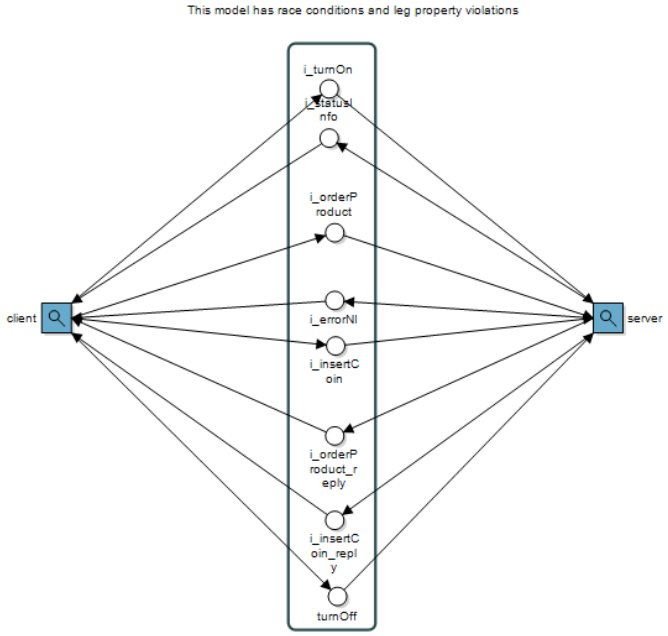
Modelling Assignments

Yasper

Yasper is a popular tool for modeling and simulating Petri nets

- Supports interface and component modeling



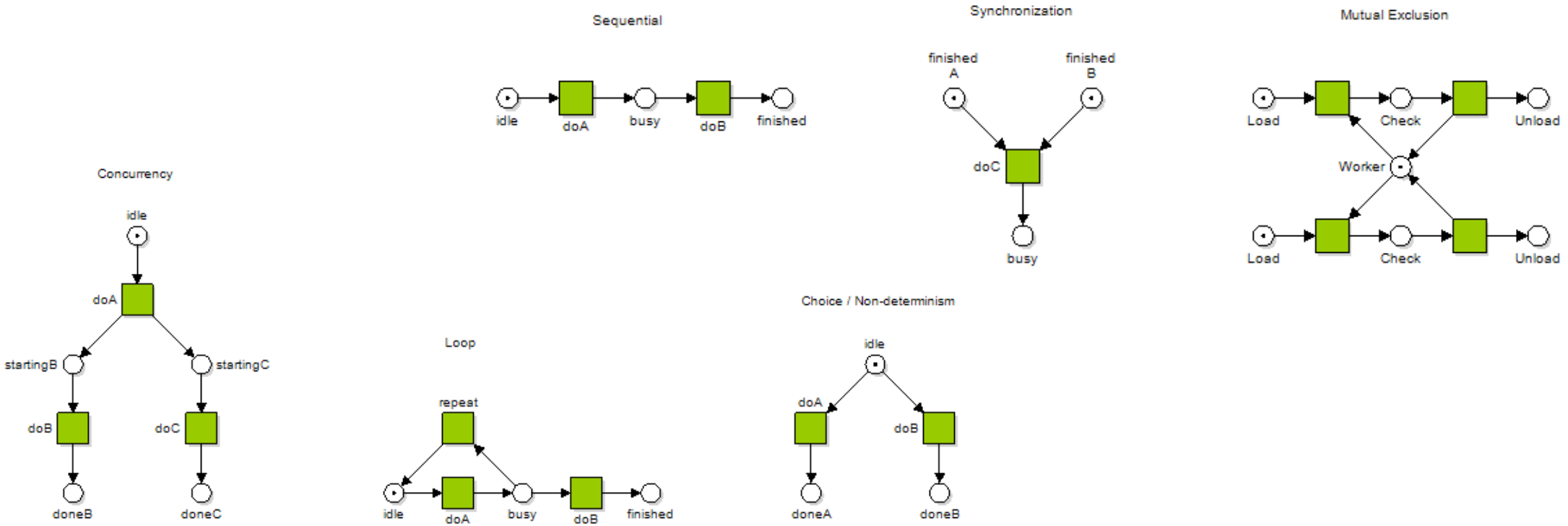


Yasper Demo

Overview

Modeling Exercise

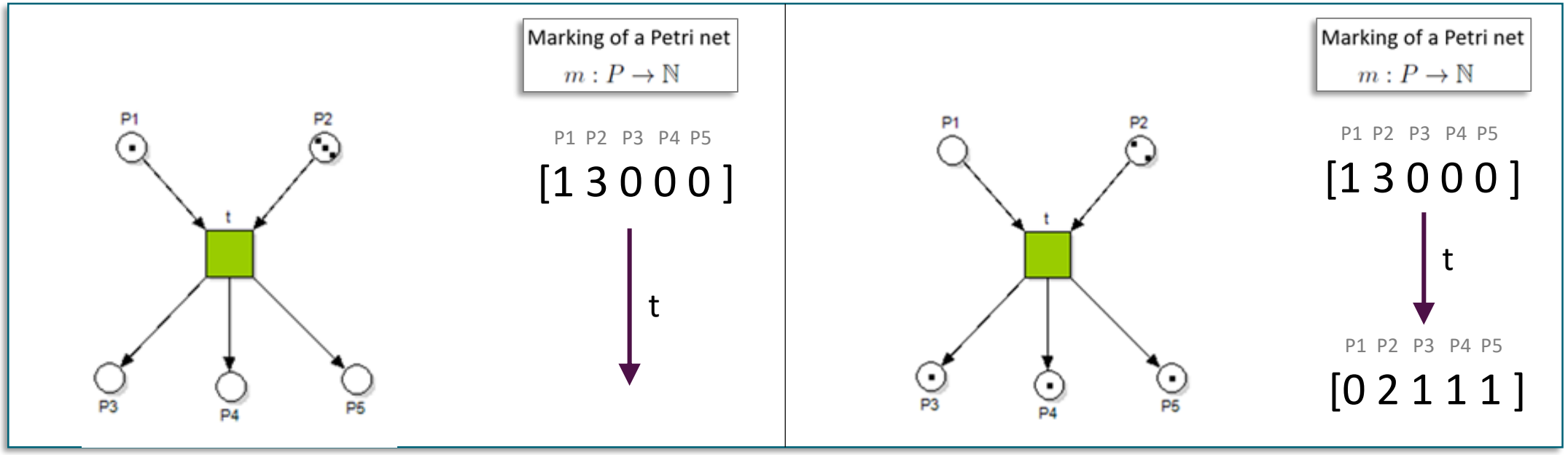
Get some experience with Petri net modeling and simulation in Yasper



Semantics of Petri Nets

Representation of Markings

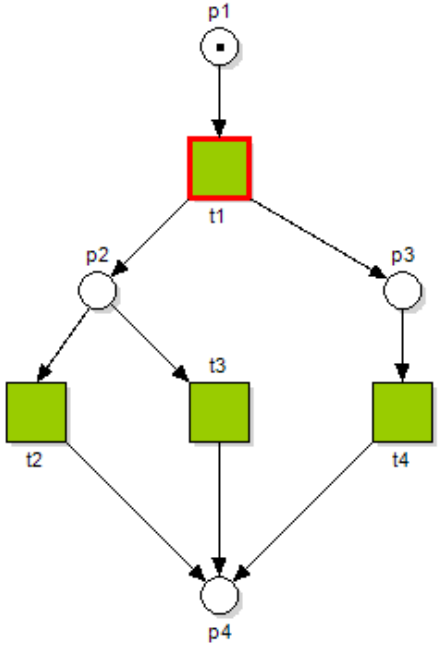
The marking of a Petri Net can be represented as a vector with the number of tokens for each place

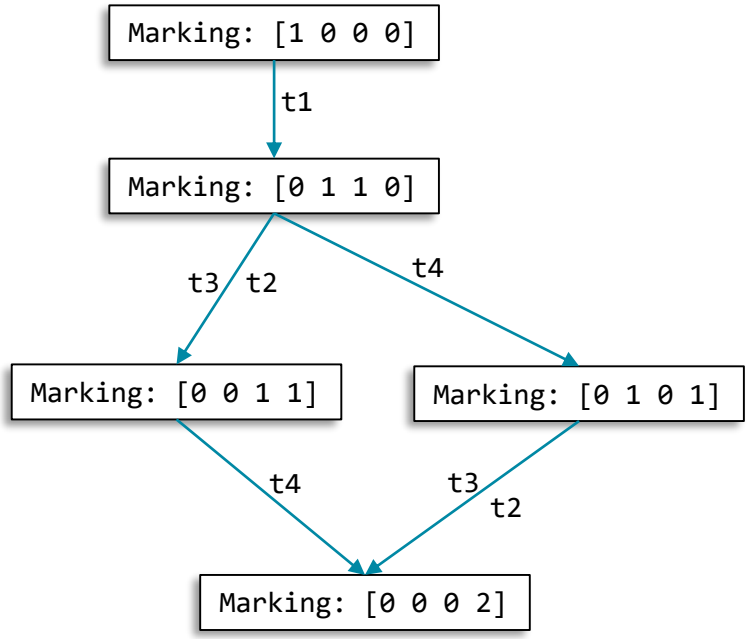


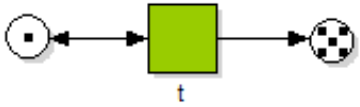
Reachability Graph (State Space)

The state space of a Petri net is captured in terms of its reachability graph

- Reachability graphs are useful for analysis of safety and liveness properties, e.g. deadlock detection
- A finite Petri net structure can have an infinite reachability graph, for e.g. unbounded nets

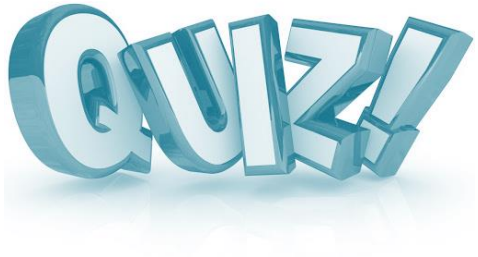






Finite structure, infinite reachability graph

```
Markings:
[1 0]
[1 1]
[1 2]
…
[1 inf]
```



Go to www.menti.com

Mentimeter

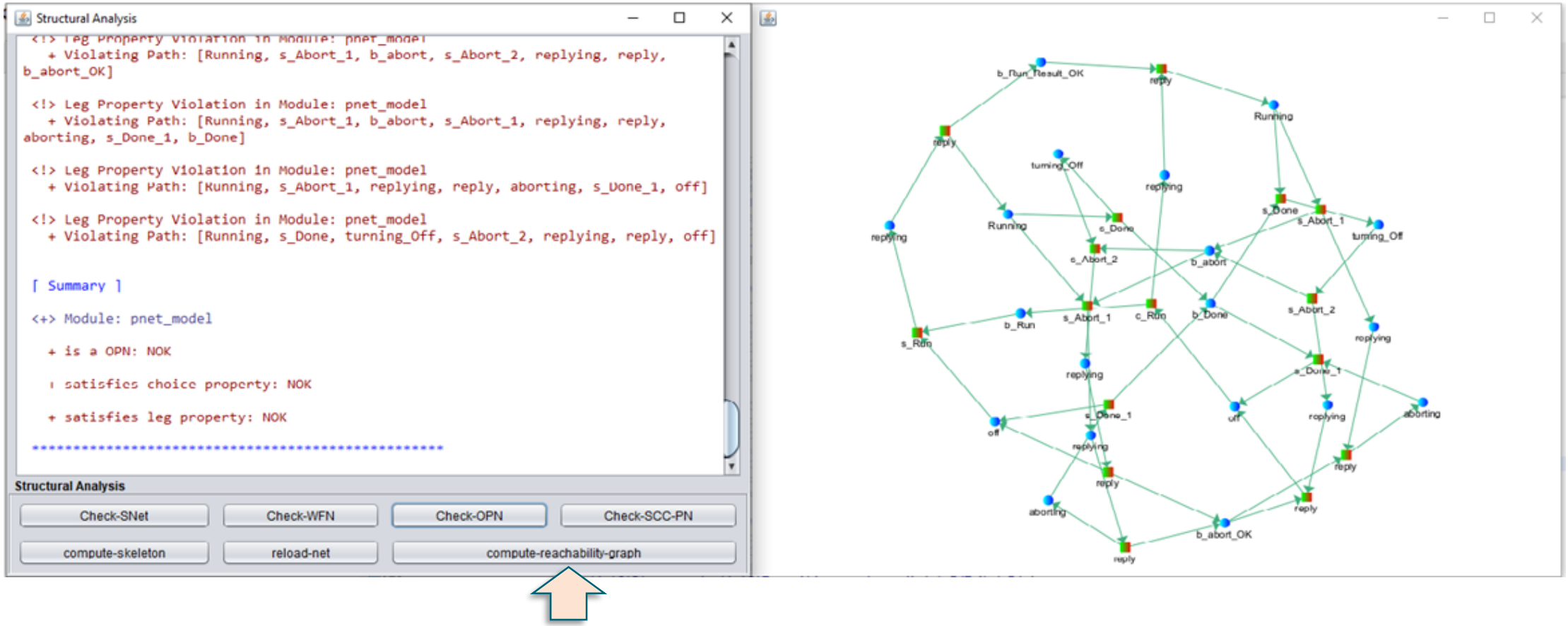
Modelling Assignments

Petri Net Analysis Toolbox (PnAT)

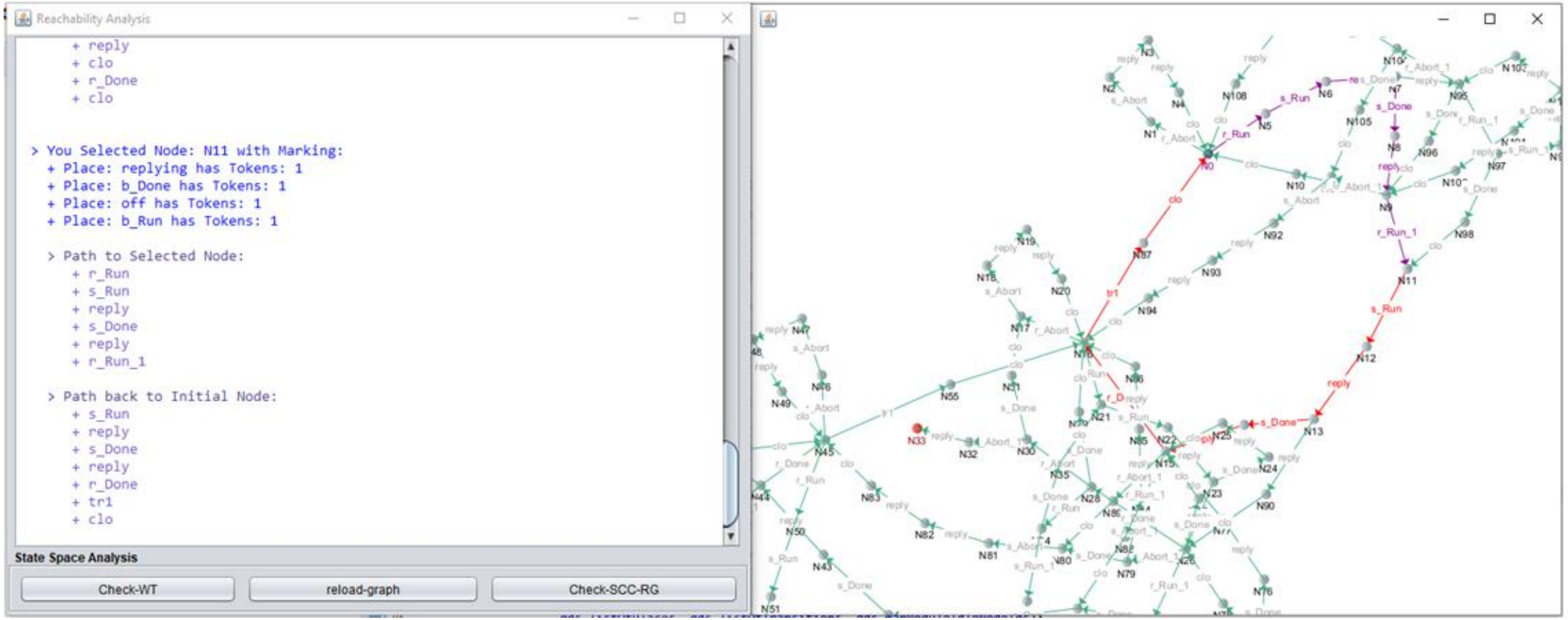
PnAT is a standalone analysis toolbox that accepts Yasper files (*.pnml)

- Checks on net Structure
  - State Machine Net, Workflow Net, Open Petri Net and Skeleton Computation
  - Portnet Properties: Leg and Choice Properties, Strongly Connected Components
- Checks on net Behavior
  - Weak Termination, Boundedness, Strongly Connected Components
  - On-demand Path Finder to Selected Node and Back
- Visualization and Dynamic Layout

Structural Analysis



Behavioral Analysis



Overview

Models for Hands-On during Lectures

BasicPetriNet.pnml

Given a Petri net model in Yasper, generate and explore the reachability graph with PnAT

Classes of Petri Nets

Classes of Petri Nets

There are many classes of Petri Nets with different restrictions on their syntax

- Restrictions on types of elements in the net (e.g. reset and/or inhibitor arcs)
- Restrictions on structure (e.g. no choice and/or no concurrency)
- Restrictions on number of initial tokens
- Restrictions on number of initial / final places

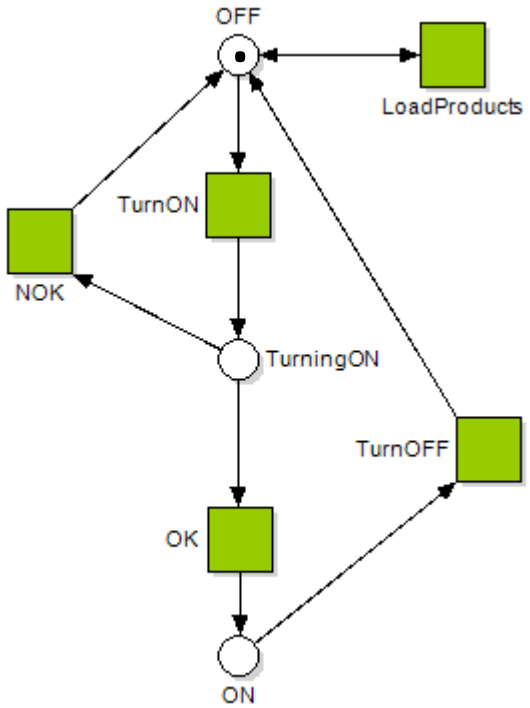
Different restrictions result in different levels of analyzability

- Classic **trade-off** between **expressiveness** and **analyzability**
- Pick the **least expressive** class that suits your purposes to maximize analyzability

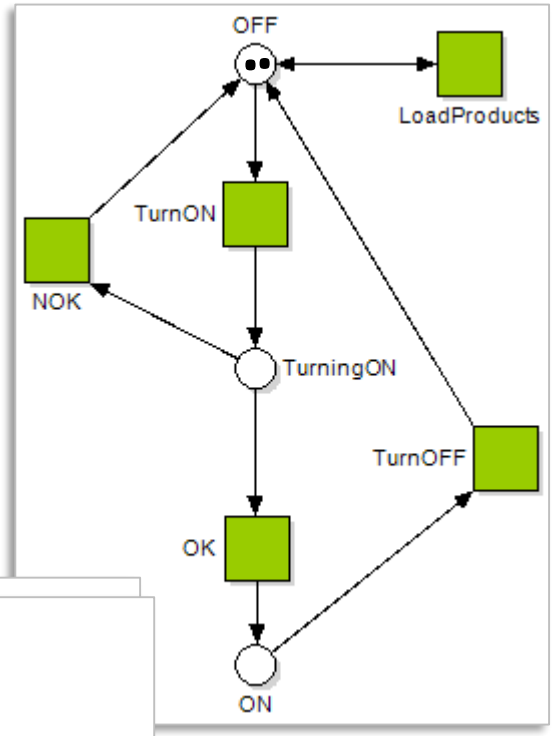
State Machine Net (S-Net)

An S-Net is a Petri net whose transitions cannot be connected to more than one input place and more than one output place

- **Concurrency** cannot be expressed
- **Non-determinism** can be expressed
  (as choice of firing transitions sharing a place)

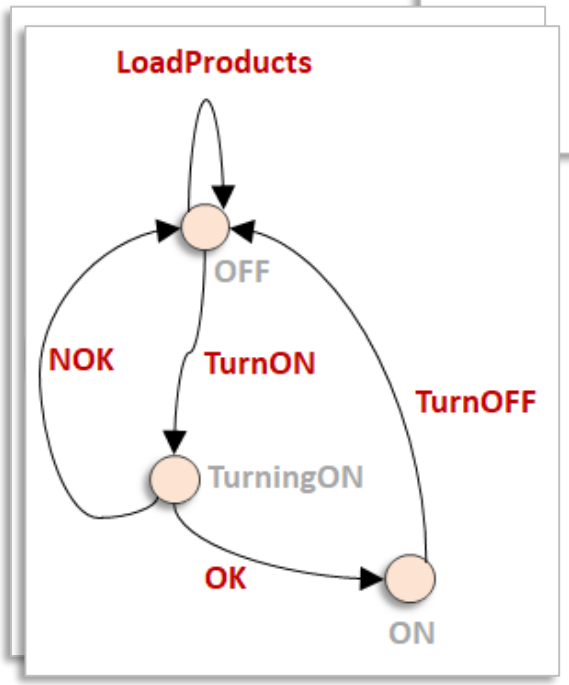


State Machine Nets and Finite State Machines



An S-Net with one token in its initial marking corresponds a classical Finite State Machine (FSM)

- By considering each place as a state and transitions as labeled edges
- Multiple tokens in the initial marking are **equivalent** to multiple independent instances of the S-Net with a single token each in their initial marking



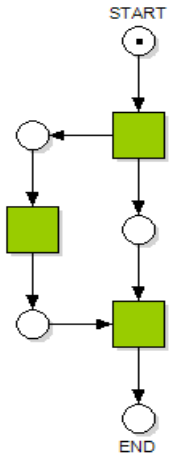
(Multi-)Workflow Nets

Workflow Nets (WFN) model activities that have a clear start and an end

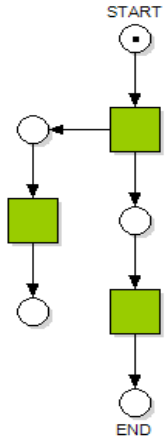
- E.g. workflows, business processes or component behavior

A Workflow Net has a single initial and final place and all nodes are on a path between these

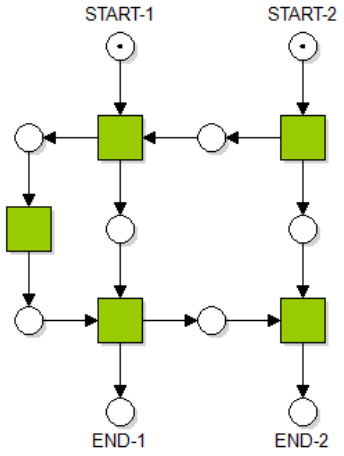
- **Multi-workflow nets (MWFN)** are a generalization that have pairs of initial and final places



Valid WFN



Invalid WFN



Valid MWFN

Open Petri Nets

An Open Petri Net (OPN) is a Petri Net where

- there are **input places** and **output places** (collectively referred to as **interface places**)

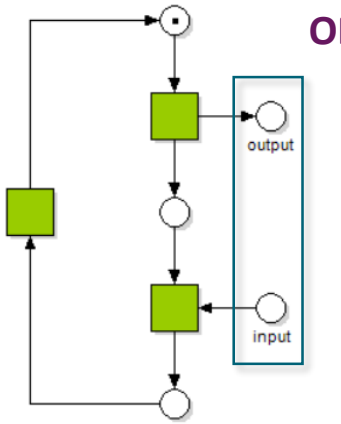
OPNs can be composed, potentially forming a closed net

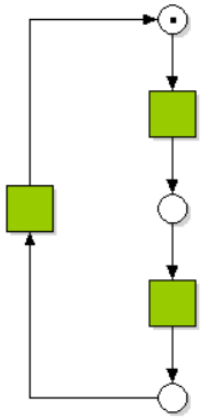
- Input and output places with identical labels are **fused**
- Behavior of OPN **dependent** on composed net(s)

The skeleton of an OPN what remains if interface places and their arcs are removed

- Allows internal behavior to be analyzed **independently**

Valid OPN





Skeleton of net

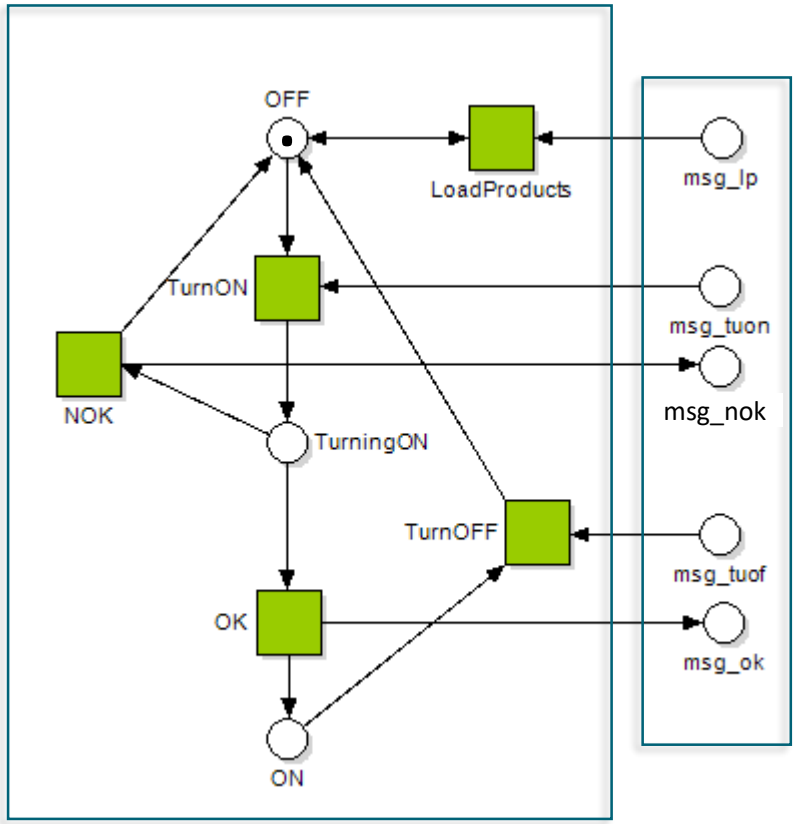
Example Interface as an Open Petri Net

Open Petri Nets are suitable for modelling interactions between components

Example interface OPN with two events:

- **TurnOn** with replies **OK** or **NOK**
- **TurnOff**
- **LoadProducts**

Server side OPN



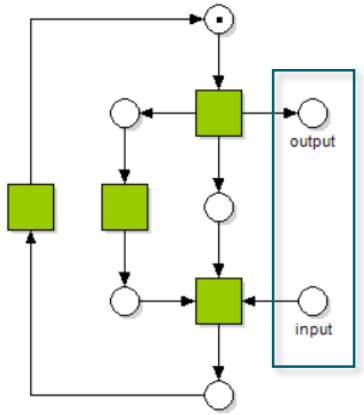
Combined Classes

Classes can be combined to form new classes

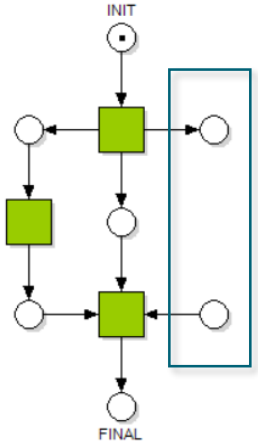
- Essentially a matter of super-imposing their restrictions

Example

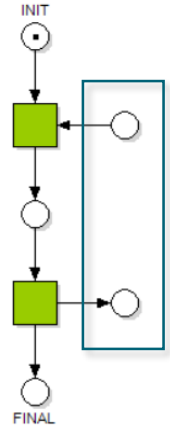
- An OPN is an **Open Workflow Net** (OWFN) if its skeleton is a Workflow Net
- If the skeleton is an S-net, it is even a **State Machine Open Workflow Net** (S-OWFN)



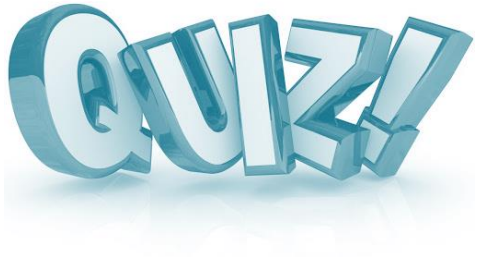
Valid OPN, but not OWFN or S-net



Valid OWFN



Valid S-OWFN



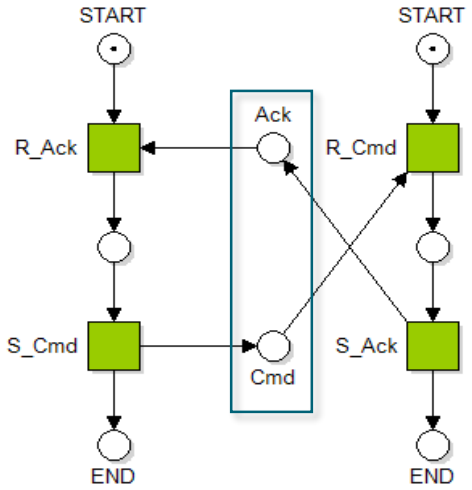
Go to www.menti.com

Mentimeter

Properties of Petri Nets

Deadlocks

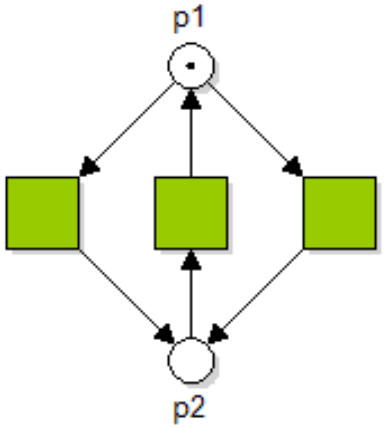
A marking is called a deadlock if there are no enabled transitions. A Petri net is said to be deadlock free if none of its reachable markings are a deadlock marking.



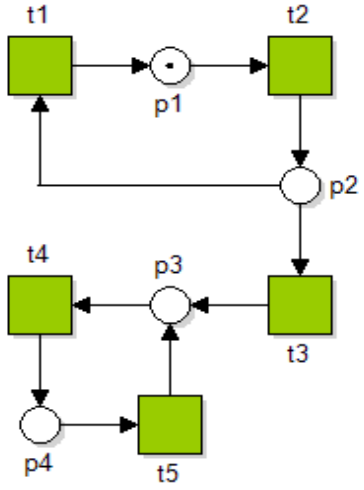
A deadlocked protocol

Liveness and Livelocks

A transition is said to be live if it can be enabled from every reachable marking of a Petri net. A Petri net is said to be livelock free if all its transitions are live.



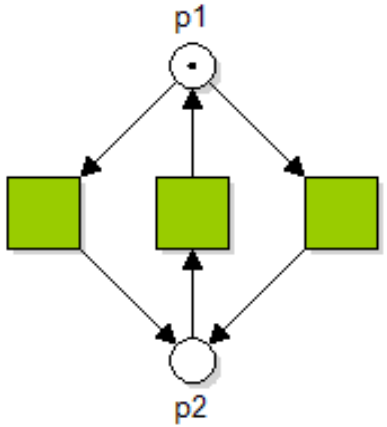
A live net



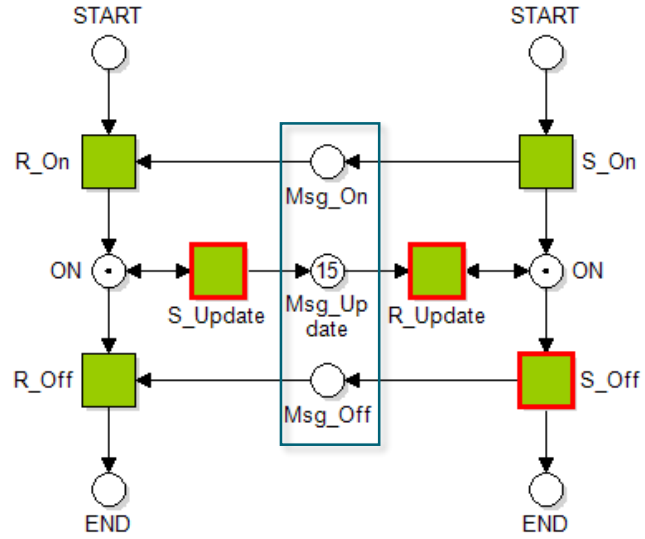
A net with livelock

Boundedness

A Petri net is said to be k-bounded if number of tokens in each place does not exceed k for any reachable marking. If the value of k is one, then the net is said to be safe. A net for which no bounded value for k exists is called an unbounded net



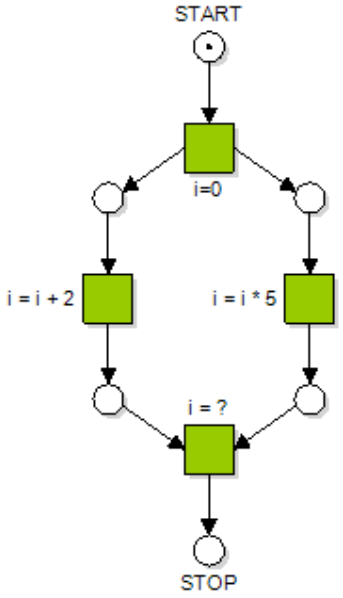
A safe net



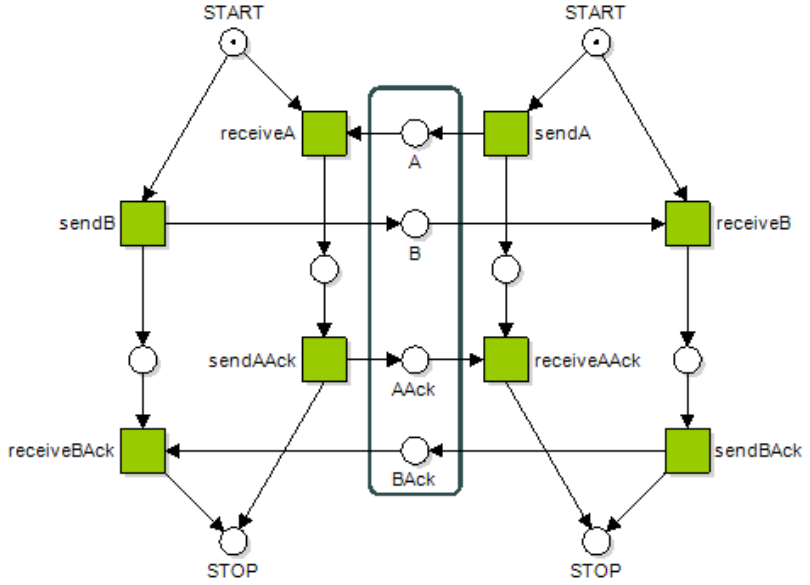
An unbounded protocol

Race Condition

A race condition is a situation where the system's substantive behavior is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when one or more of the possible behaviors is undesirable.



Different value of i for different transition firing sequences



Deadlock for some possible transition firing sequences

Termination

A net is weakly terminating if and only if from every reachable marking, it is always possible to reach the final marking

- Initial and final marking may be equal in a cyclic net

Termination guarantees freedom from deadlocks and livelocks, and boundedness

- Does **not guarantee freedom from race conditions**, only that they **do not prevent termination**

Two types of termination

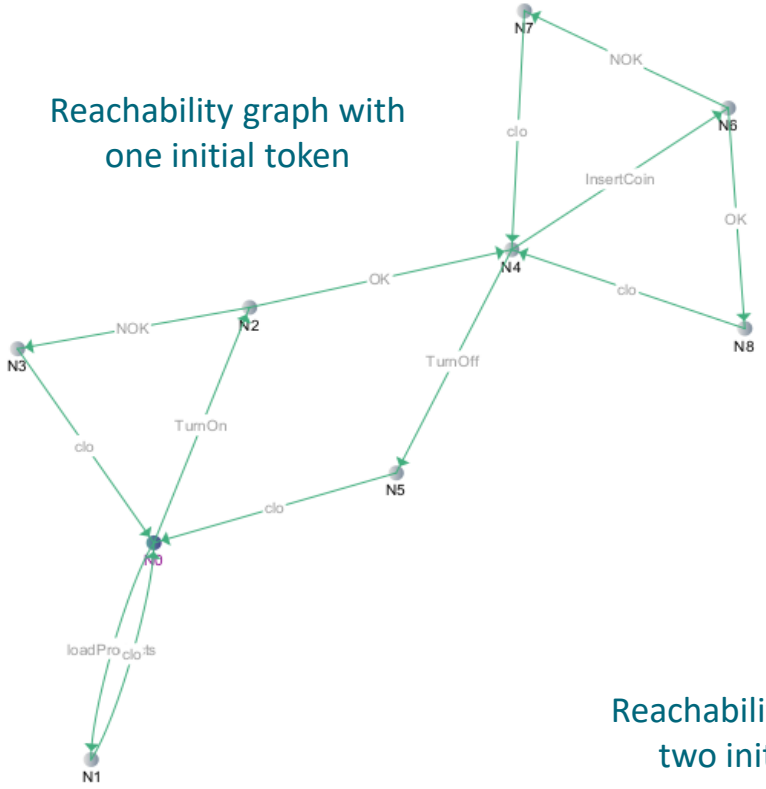
1. **Strong termination** requires that the net terminates with a **finite number** of firing steps
2. **Weak termination** allow **infinite firing sequence(s)** but is always **able** to terminate

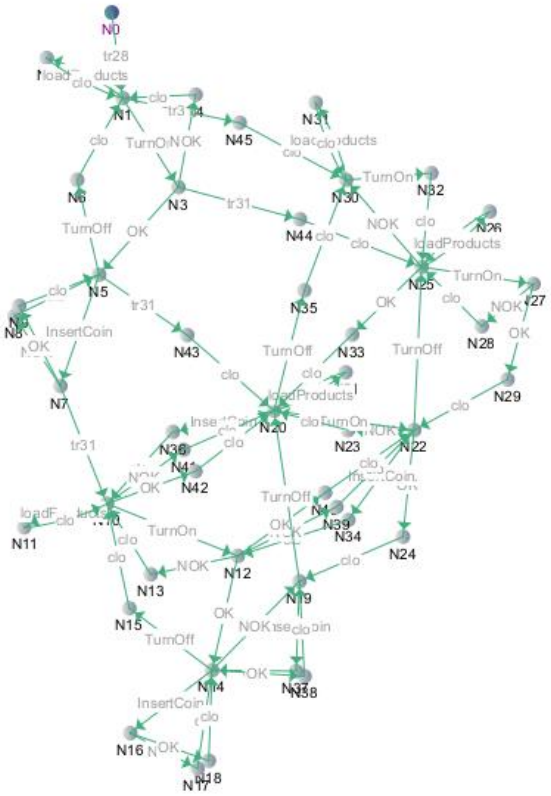
This presentation focuses on weak termination

Example of a Weakly Terminating Net

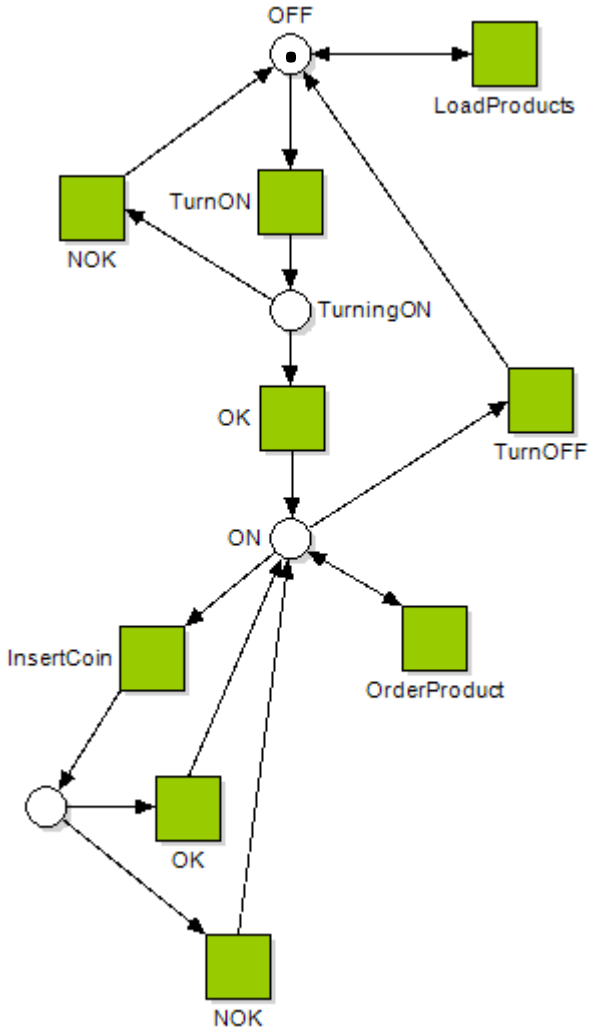
```
Initial Marking = Final Marking = [OFF -> 1]
```

Reachability graph with one initial token





Reachability graph with two initial tokens



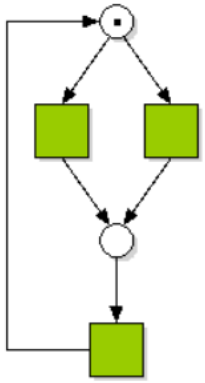
Weak Termination in State Machine Nets

A strongly connected S-net is guaranteed to weakly terminate

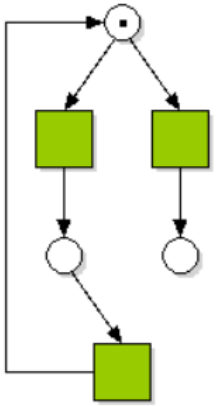
- Strongly connected means every state is reachable from every other state (no dead states)
- No concurrency means no dependencies between tokens

All S-WFN are guaranteed to weakly terminate

- All nodes on path from initial to final state
- **Closure** of S-WFN (connecting final state back to initial) is a strongly connected S-net

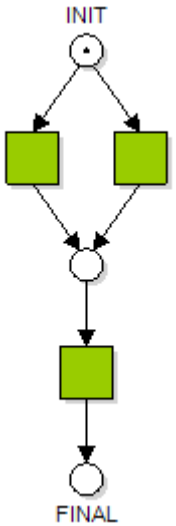


Strongly connected and weakly terminating S-net



Not strongly connected nor weakly terminating S-net

Weakly terminating S-WFN



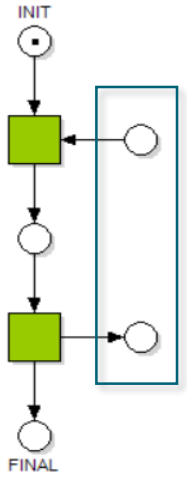
Weak Termination in Open Petri Nets

Weak termination of an OPN is typically considered for its skeleton

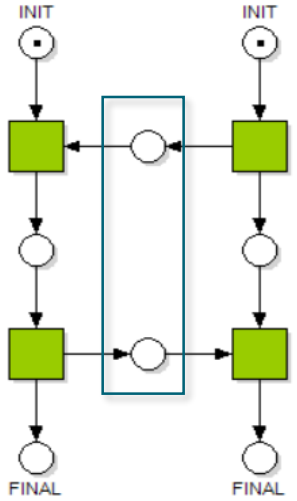
- OPN cannot terminate without input tokens from its environment

Weak termination can be determined for compositions of open nets that result in a closed net

- Methods exist to determine if an open net has a partner net that allows it to weakly terminate

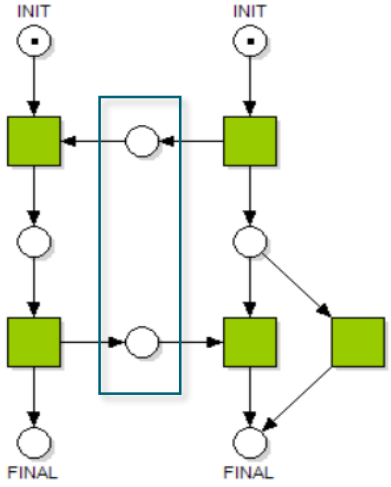


Valid S-OWFN



Weakly terminating composition

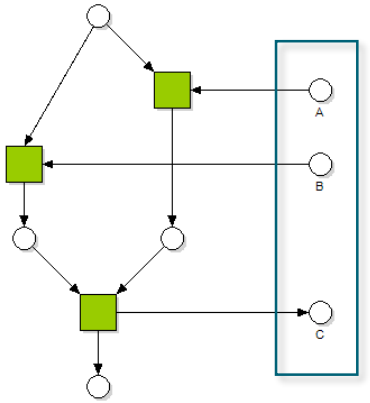
Not weakly terminating composition



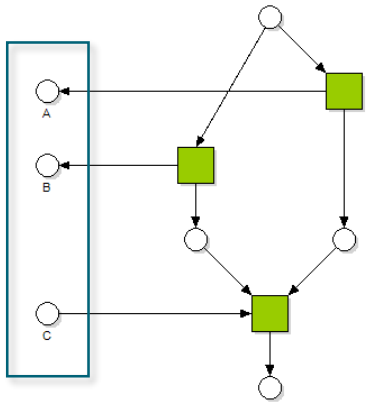
The Mirrored Port Clients

We will often consider mirrored OPN clients in this course

- The mirrored client uses the same interface protocol **symmetrically**
- Same skeleton and interface places as server, but direction of communication is reversed
- **Maximal client** that exercises the entire protocol

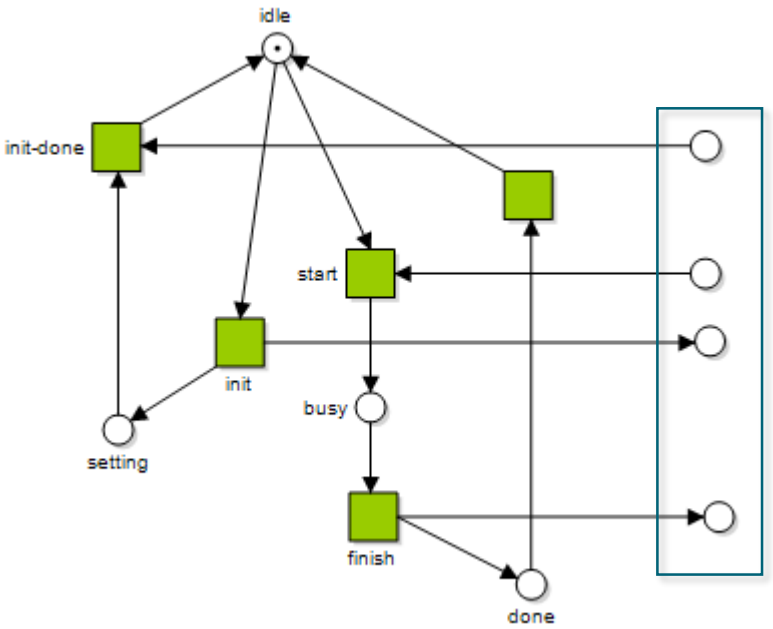


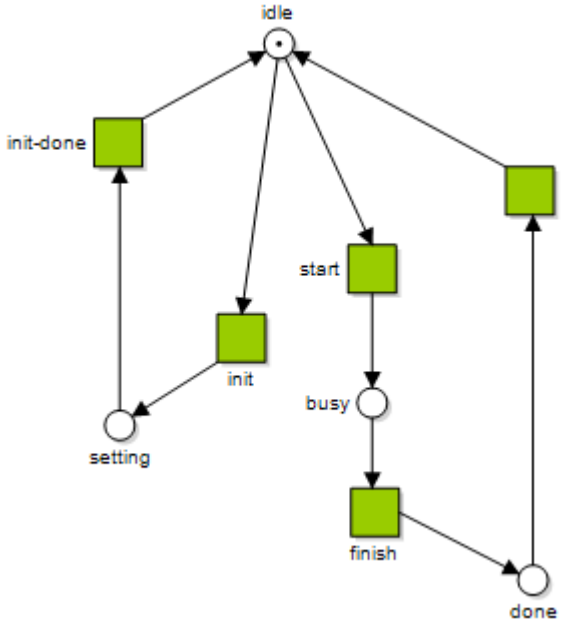
Server

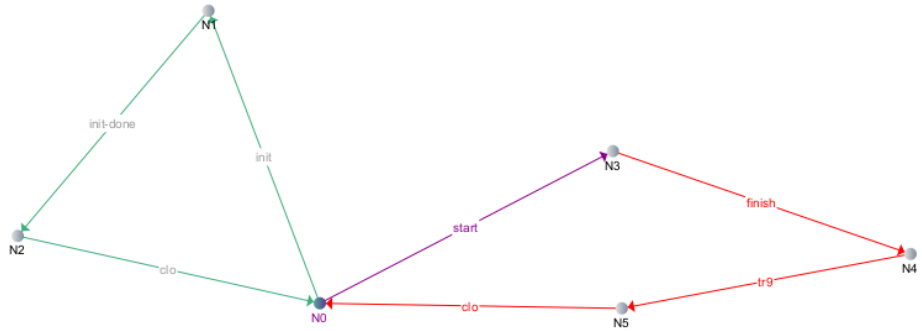


Mirrored Client

Weak Termination of OPN and its Skeleton





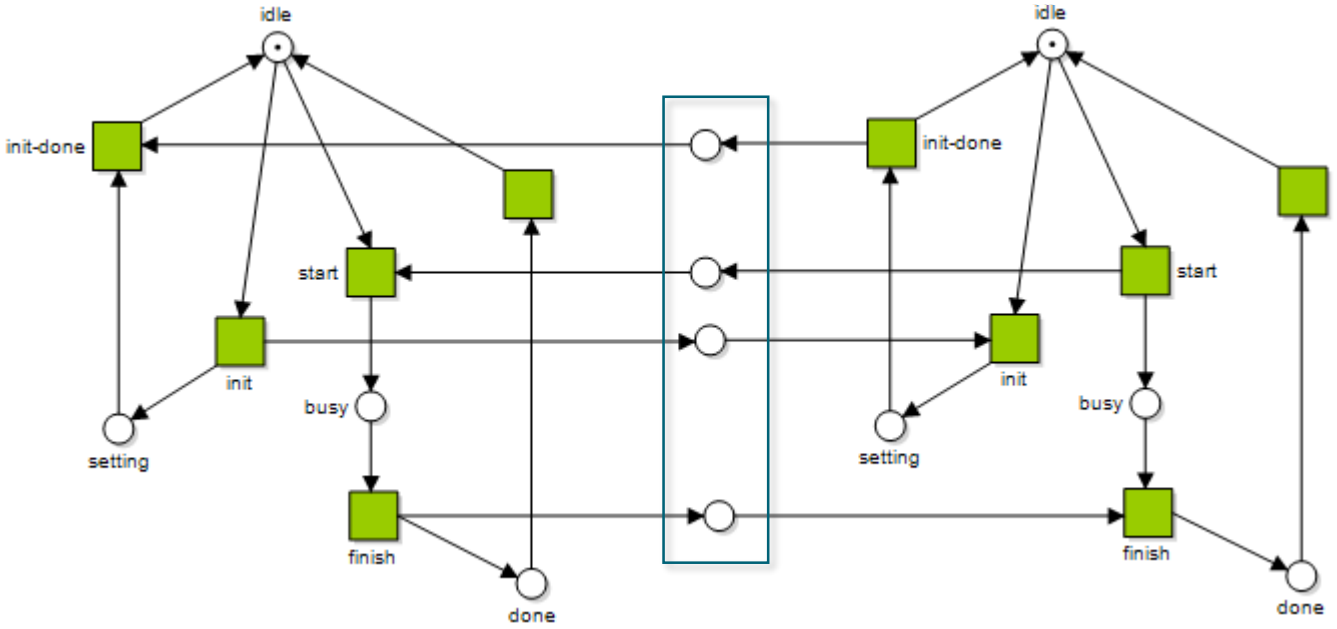


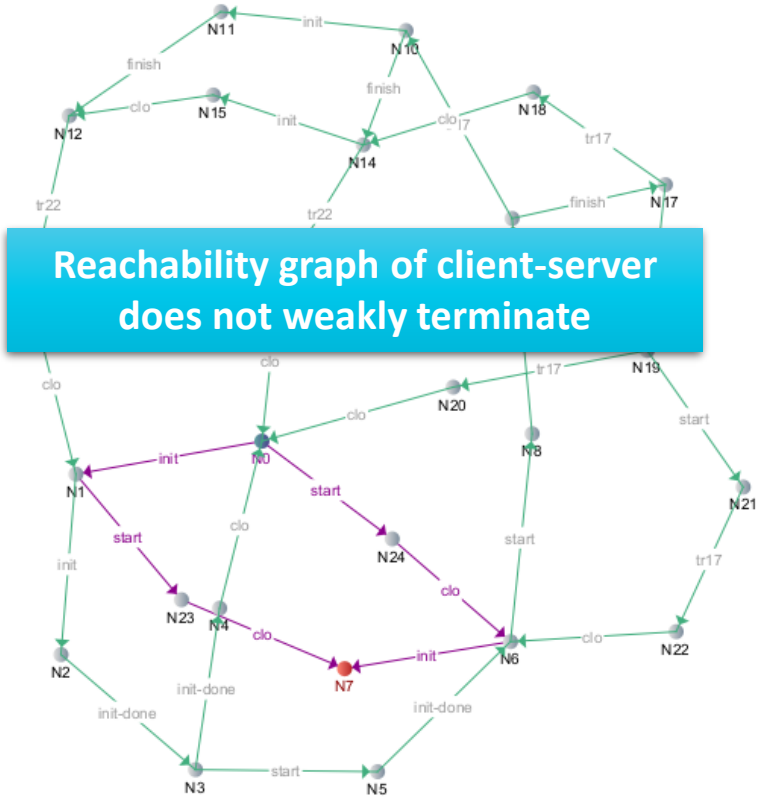
An OPN and its weakly terminating skeleton, as indicated by the reachability graph

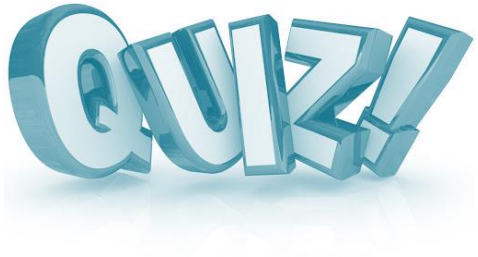
Weak Termination of OPN and its Skeleton

A weakly terminating skeleton does not guarantee that composition of OPN is weakly terminating

- Example shows a composition with a mirrored client







Go to www.menti.com

Mentimeter

Reflection

Please take a minute to individually reflect on what you have learned

Write down any insights that you would like to remember after the course on a PostIt note

Modelling interfaces could reduce design errors in SW department

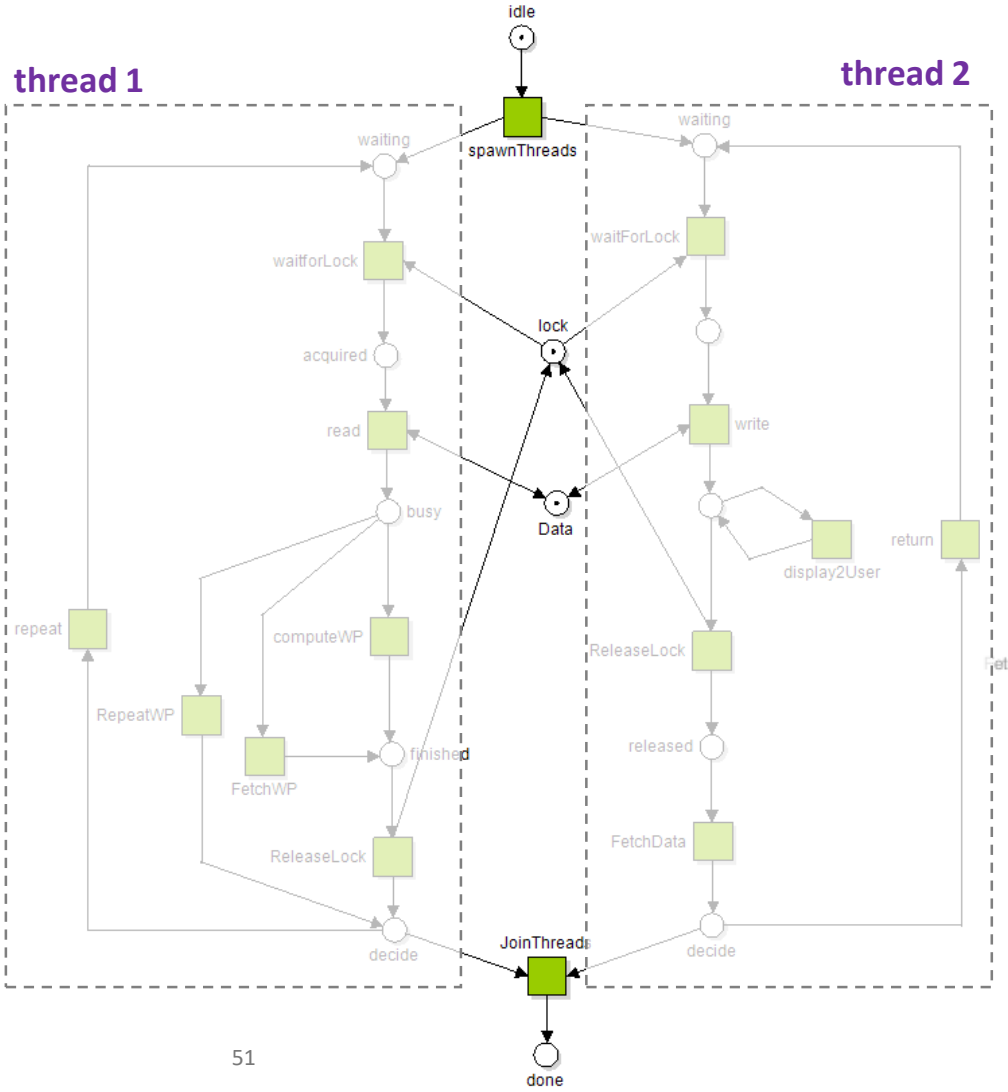
Modelling Assignments

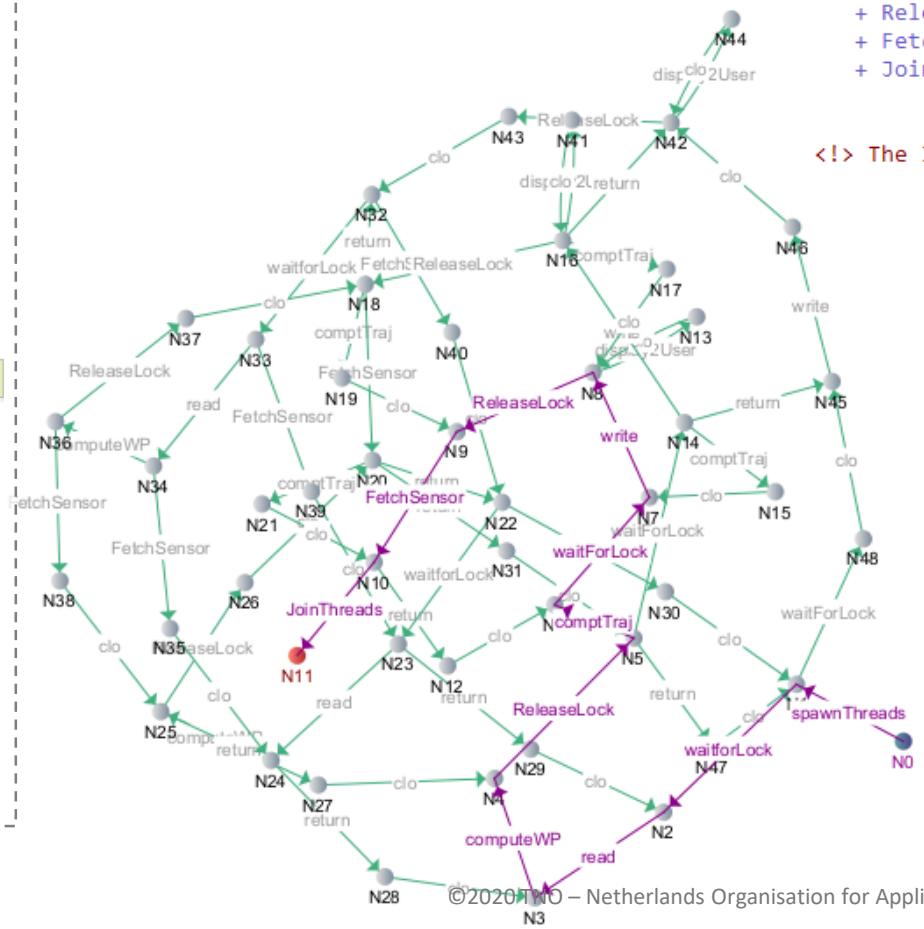
Overview

ThreadingAndDeadlock.pnml

- Check Workflow net and other properties
- First experience with resolving problematic models by reachability analysis

ThreadingAndDeadlock.pnml






```
> You Selected Node: N11 with Marking:
  + Place: Data has Tokens: 1
  + Place: lock has Tokens: 1
  + Place: done has Tokens: 1

  > Path to Selected Node:
    + spawnThreads
    + waitforLock
    + read
    + computeWP
    + ReleaseLock
    + comptTraj
    + waitForLock
    + write
    + ReleaseLock
    + FetchSensor
    + JoinThreads

<!> The Initial Node is unreachable from Selected Node!
```

Generate reachability graph using PnAT with value 30. Inspect it.

Overview

Module Exercise

ServerProtocolStateMachine.pnml:

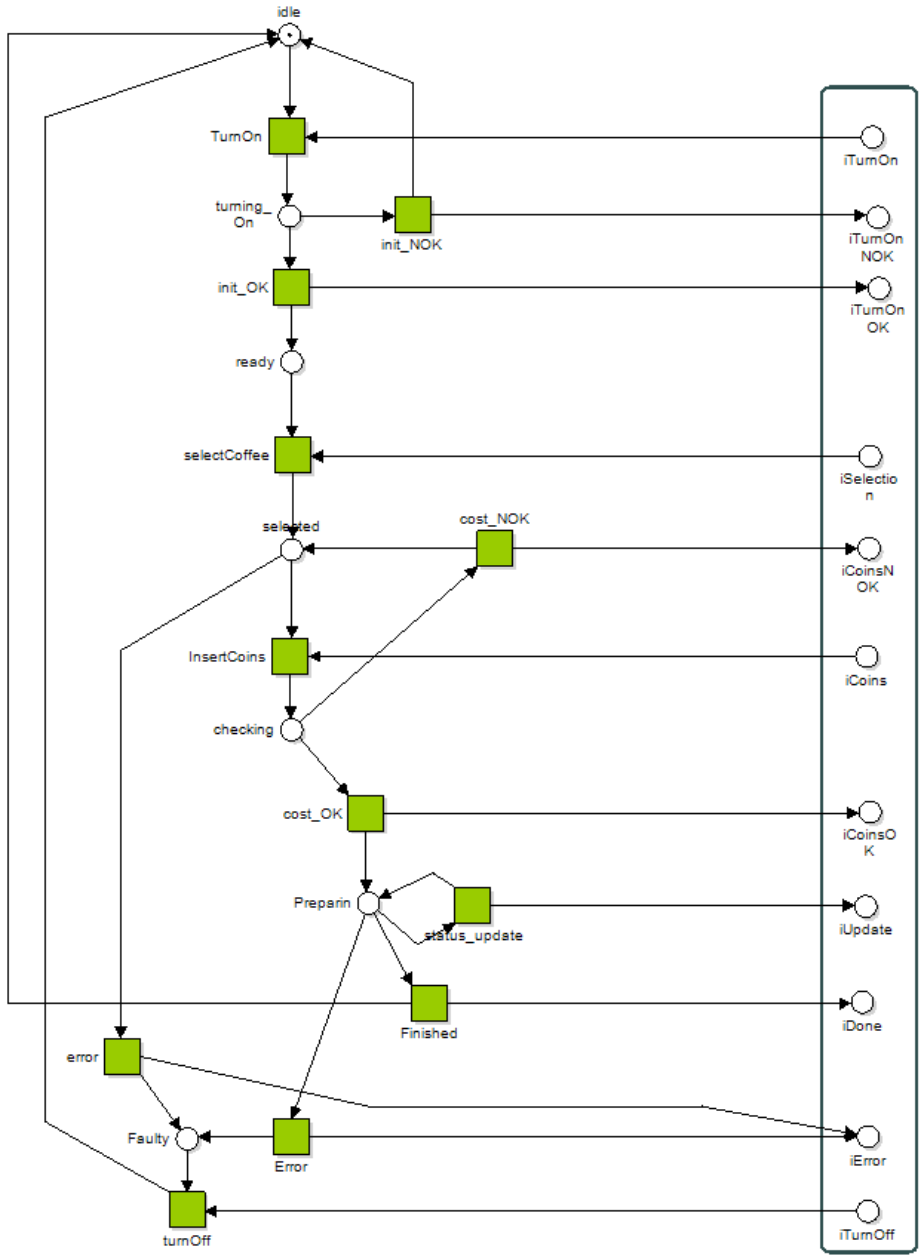
Build server interface models from informal descriptions of coffee machine

Transform the model into communicating client-server OPNs

Check properties with the tool

- Structural: S-Net, WFN, OPN, Skeleton checks
- Behavioral: weak termination

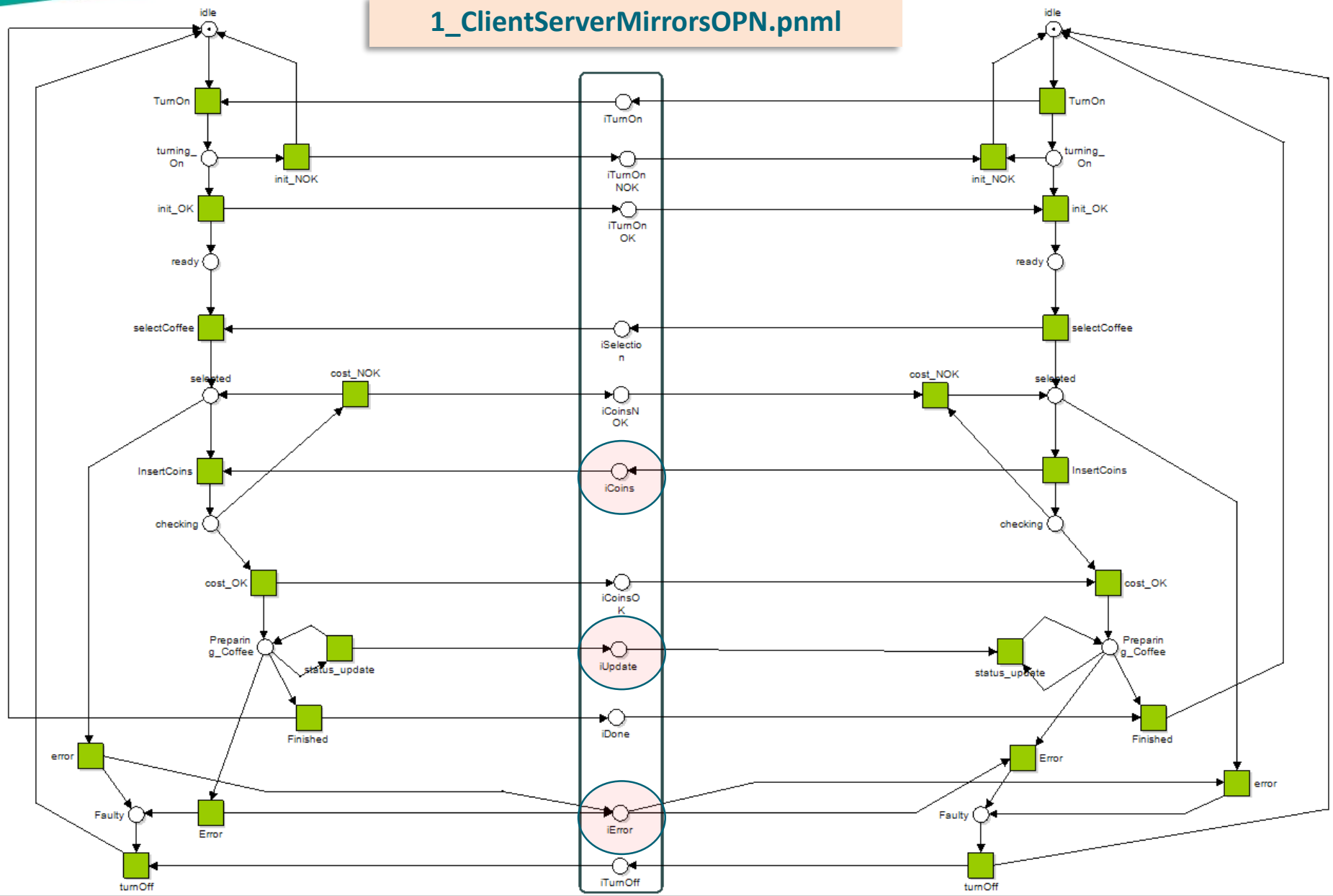
Goal: Make the net weakly terminating with good reasoning



ServerProtocolStateMachine.pnml

Observe: What makes it a server?

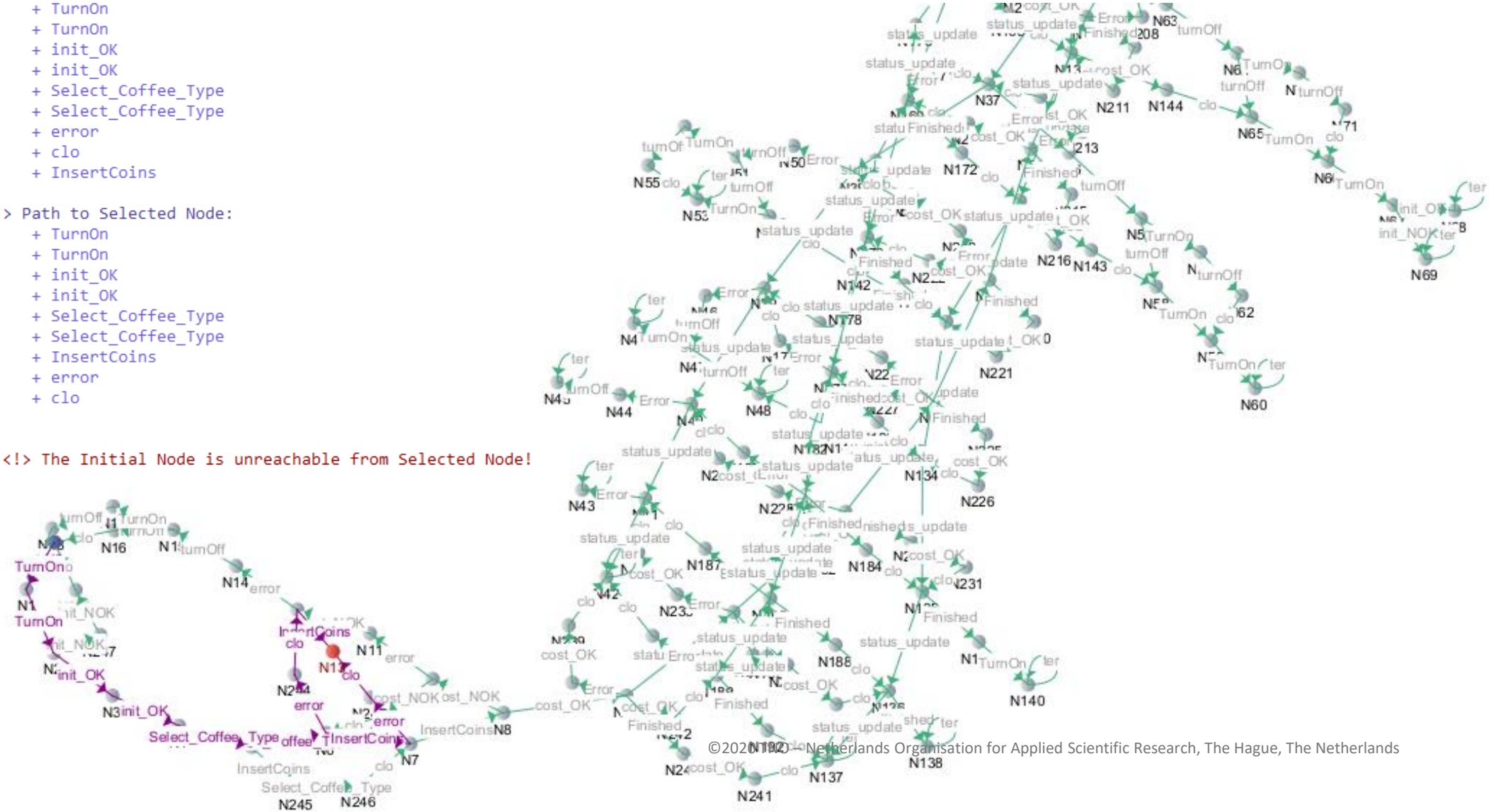
1_ClientServerMirrorsOPN.pnml



Generate reachability graph using PnAT with value 30. Inspect it.

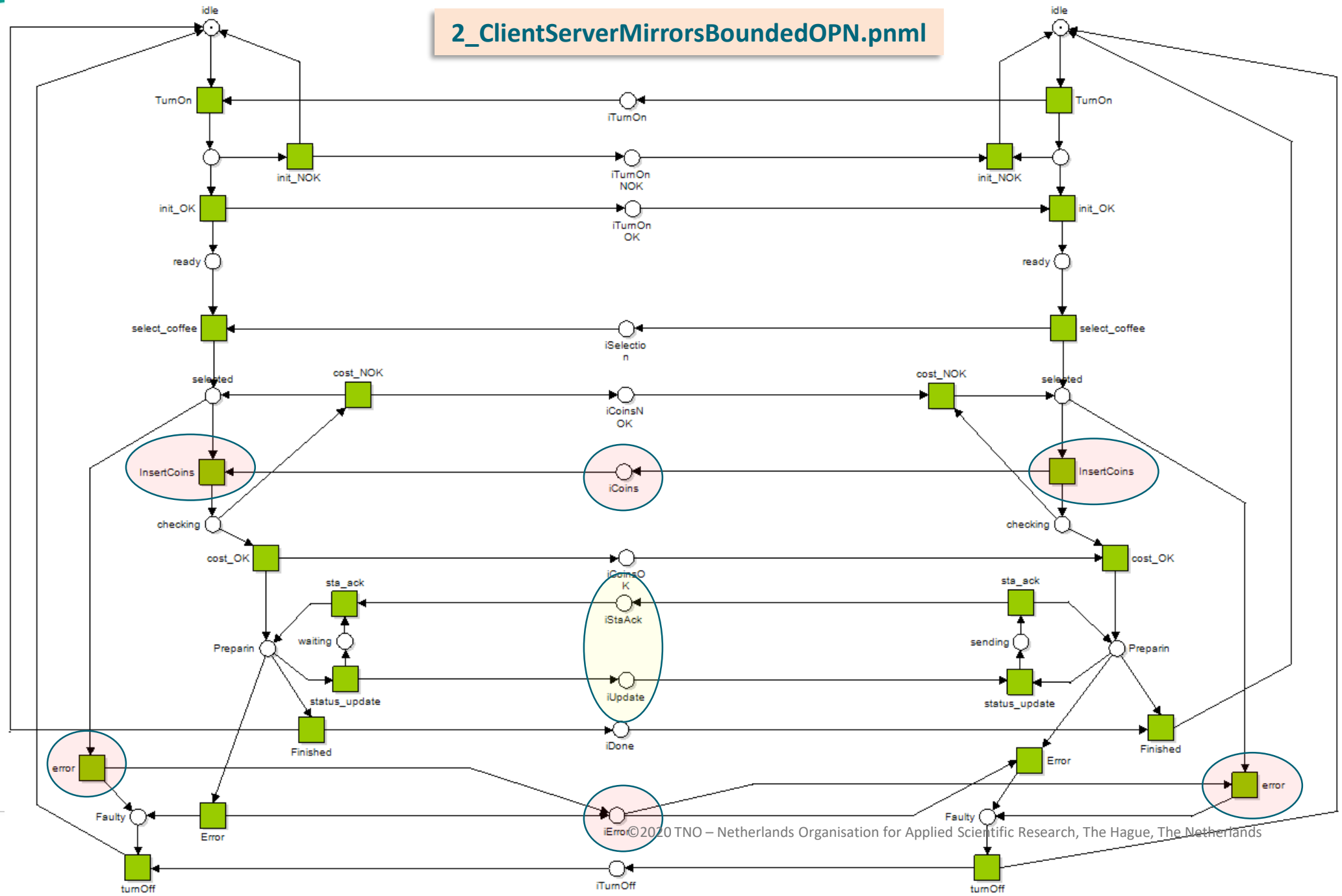
```
> You Selected Node: N13 with Marking:
  + Place: checking has Tokens: 1
  + Place: Faulty has Tokens: 1
  + Place: iCoins has Tokens: 1
  + Place: iError has Tokens: 1

  > Path to Selected Node:
     + TurnOn
     + TurnOn
     + init_OK
     + init_OK
     + Select_Coffee_Type
     + Select_Coffee_Type
     + error
     + clo
     + InsertCoins

  > Path to Selected Node:
     + TurnOn
     + TurnOn
     + init_OK
     + init_OK
     + Select_Coffee_Type
     + Select_Coffee_Type
     + InsertCoins
     + error
     + clo

<!> The Initial Node is unreachable from Selected Node!
```

Observe that in addition to the deadlock there is unbounded behavior



2_ClientServerMirrorsBoundedOPN.pnml

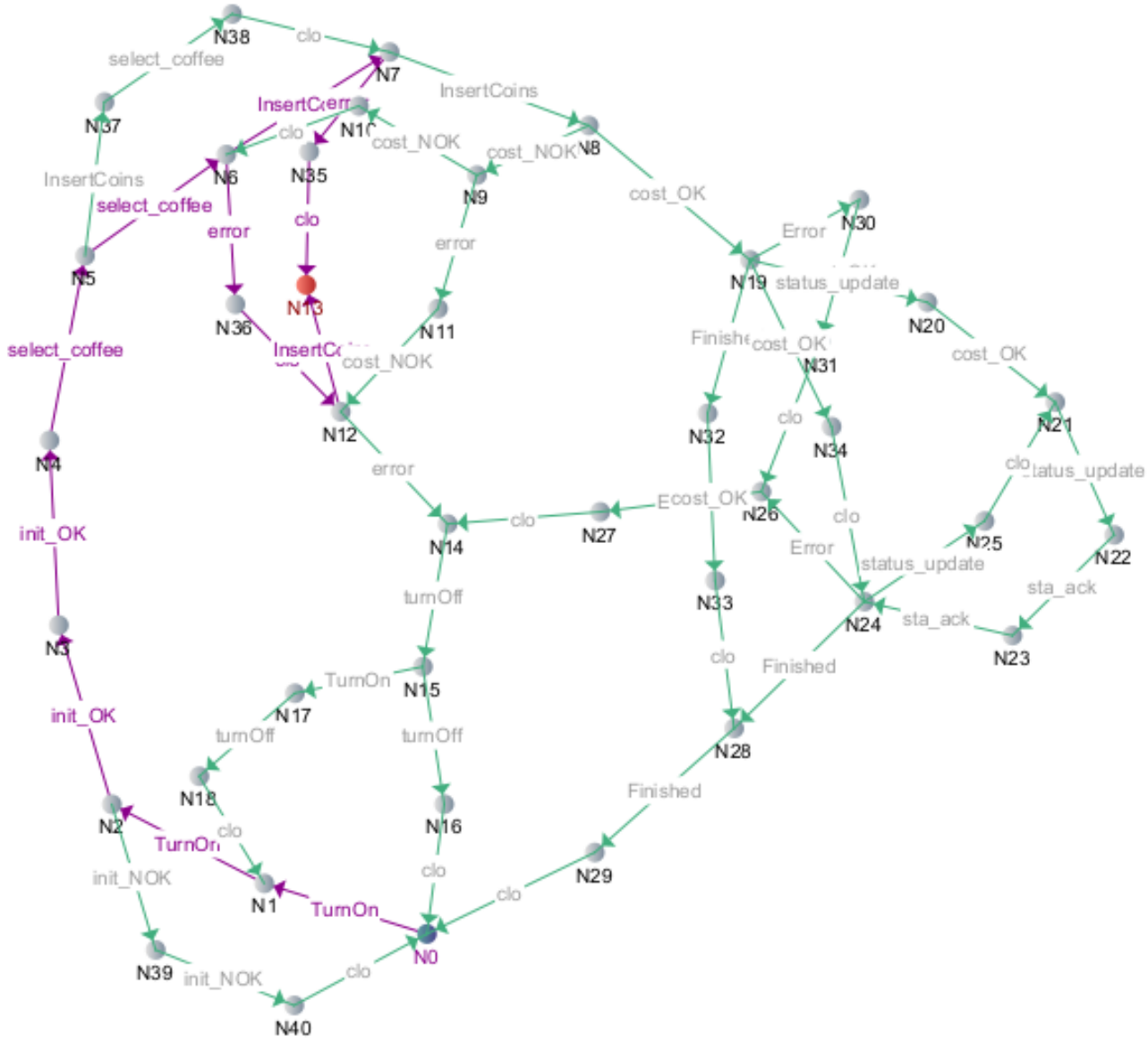
One possible fix for the boundedness issue!



```
> You Selected Node: N13 with Marking:
  + Place: checking has Tokens: 1
  + Place: Faulty has Tokens: 1
  + Place: iCoins has Tokens: 1
  + Place: iError has Tokens: 1

  > Path to Selected Node:
     + TurnOn
     + TurnOn
     + init_OK
     + init_OK
     + select_coffee
     + select_coffee
     + error
     + clo
     + InsertCoins

  > Path to Selected Node:
     + TurnOn
     + TurnOn
     + init_OK
     + init_OK
     + select_coffee
     + select_coffee
     + InsertCoins
     + error
     + clo

<!> The Initial Node is unreachable from Selected Node!
```

Are we losing coins when there is an error?

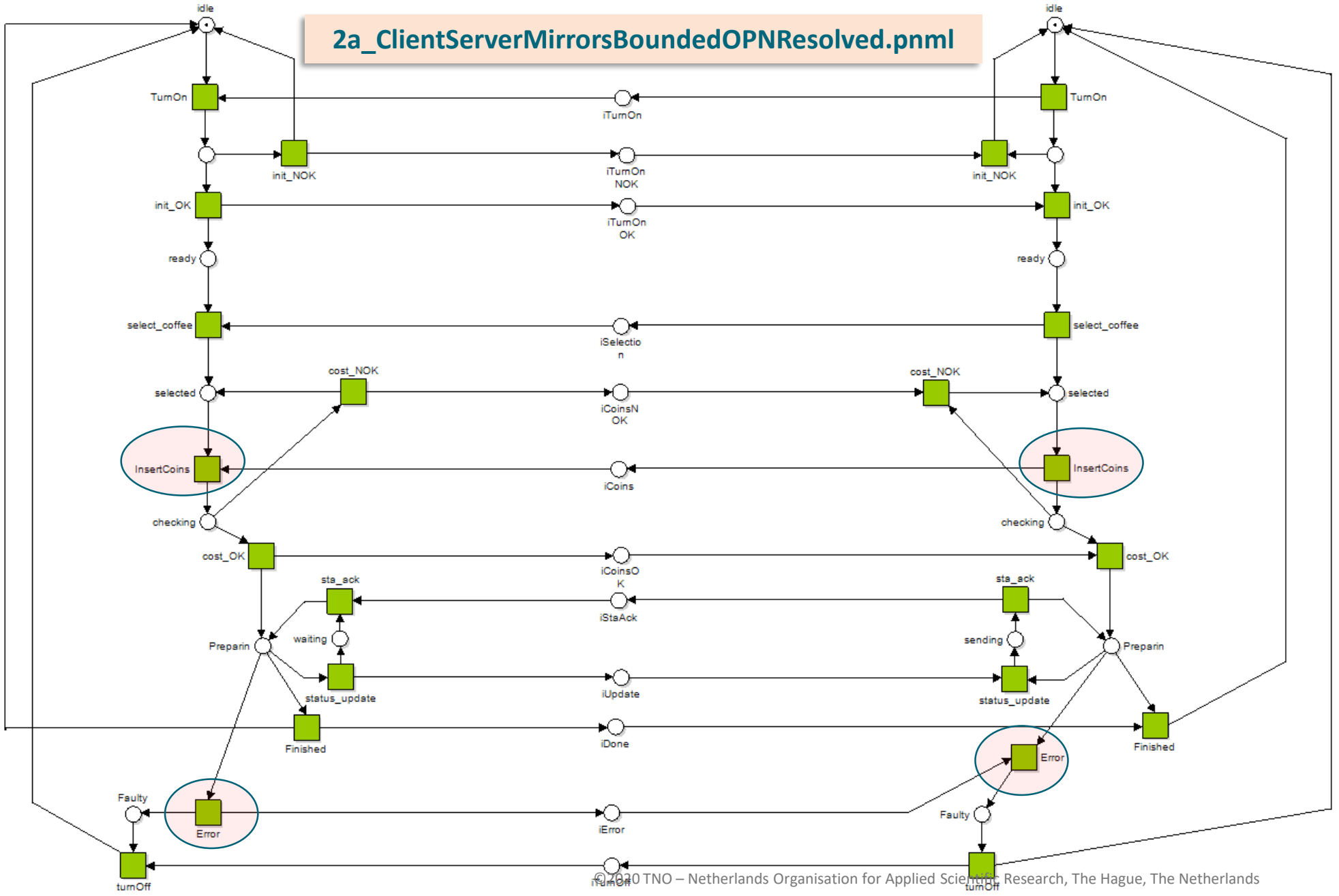
Are we losing coins when there is an error?



Now check by simulating the sequence of events in Yasper. Why are there deadlocks?

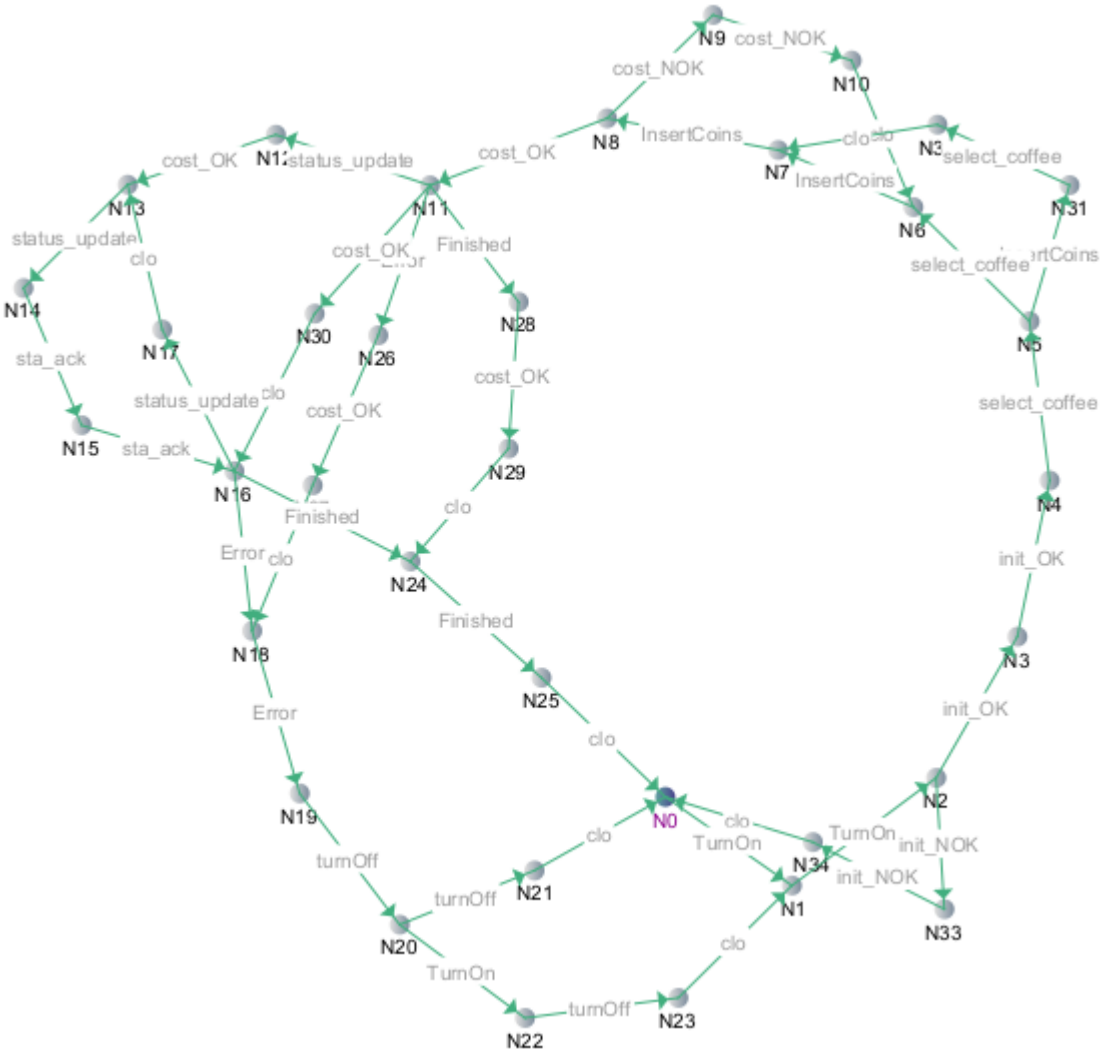
Possible Solution

Remove the possibility to raise an error when coins are inserted!



Possible Solution

```
**********************************************************
  Property Check for Weak Termination
**********************************************************


> Checking for Boundedness..

[ Summary ]

<+> Weak Termination: OK
<+> The net is safe!
**********************************************************
```



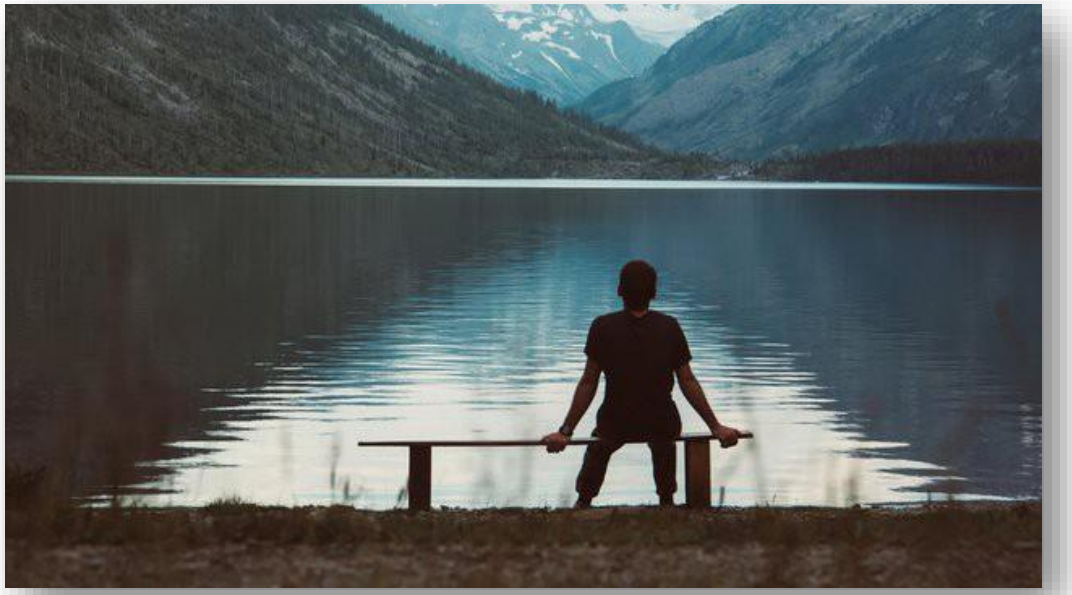
The solution presented is not ideal nor useful. In reality, errors can always be raised and it must be handled properly in every scenario!

We will learn better solutions to handle such problems in the next module.

Reflection

Please take a minute to individually reflect on what you have learned

Write down any insights that you would like to remember after the course on a PostIt note



Relation to the ComMA Framework

Component Modelling and Analysis

ComMA Overview

ComMA provides languages to describe component-based systems

- Specification artefacts: component and interface specifications
- Behavioral aspects in terms of protocol state machine and constraints (incl. data and time)

ComMA Interfaces describe an expectation over a set of events from a server perspective

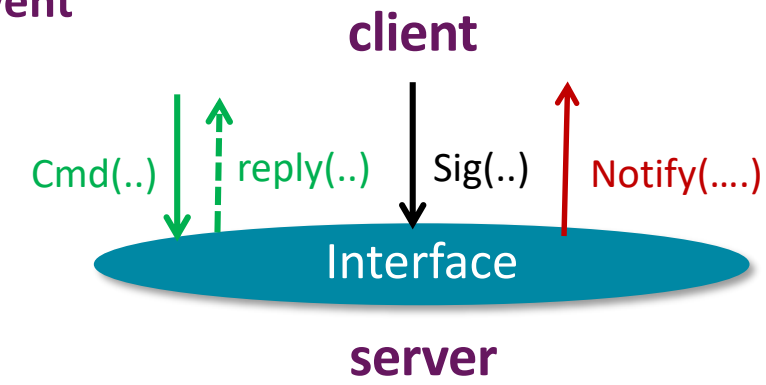
- Event types: command-reply, signal, notification

Transitions of a ComMA state machine may have more than one event

- Atomicity: Runs to completion

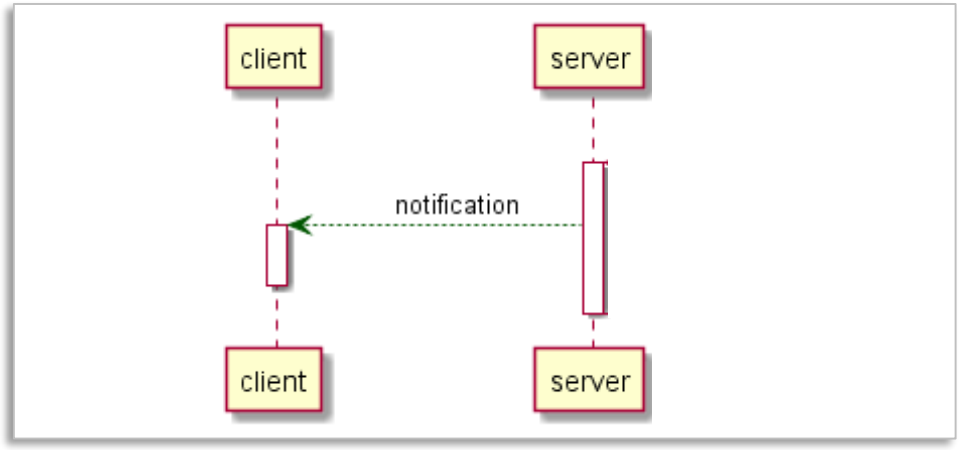
Distinguishes two types of transitions

- **Triggered Transitions**: Initiated by the client
- **Non-Triggered Transitions**: Initiated by the server



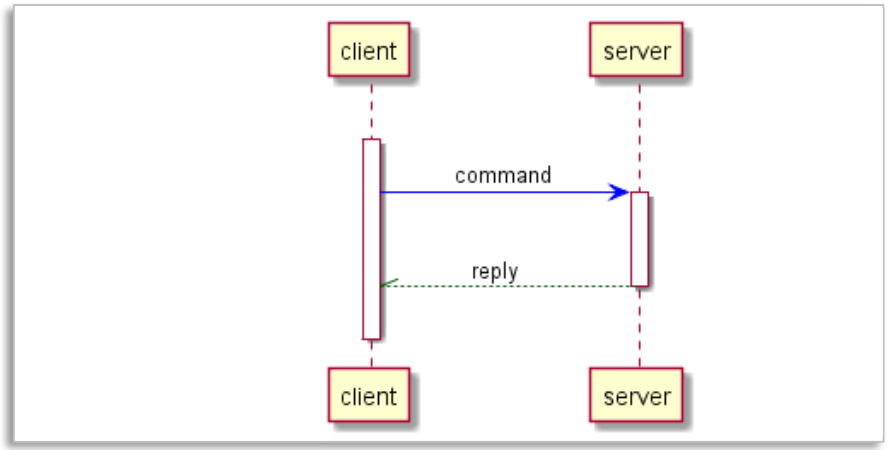
Event Types in ComMA - I

Server can send **Notifications**

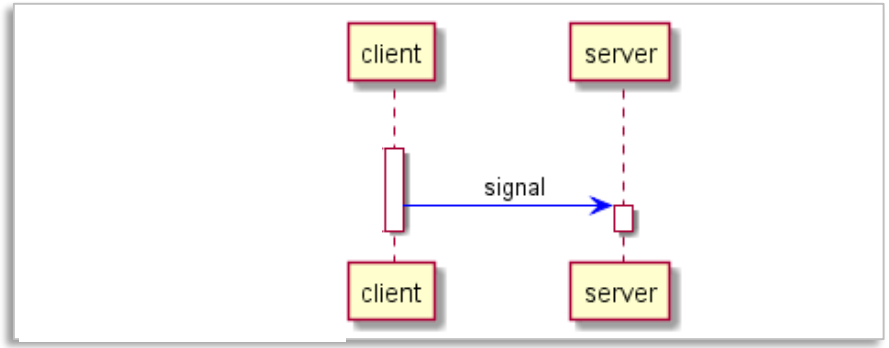


Non-Triggered Transition

Client can trigger **Commands**



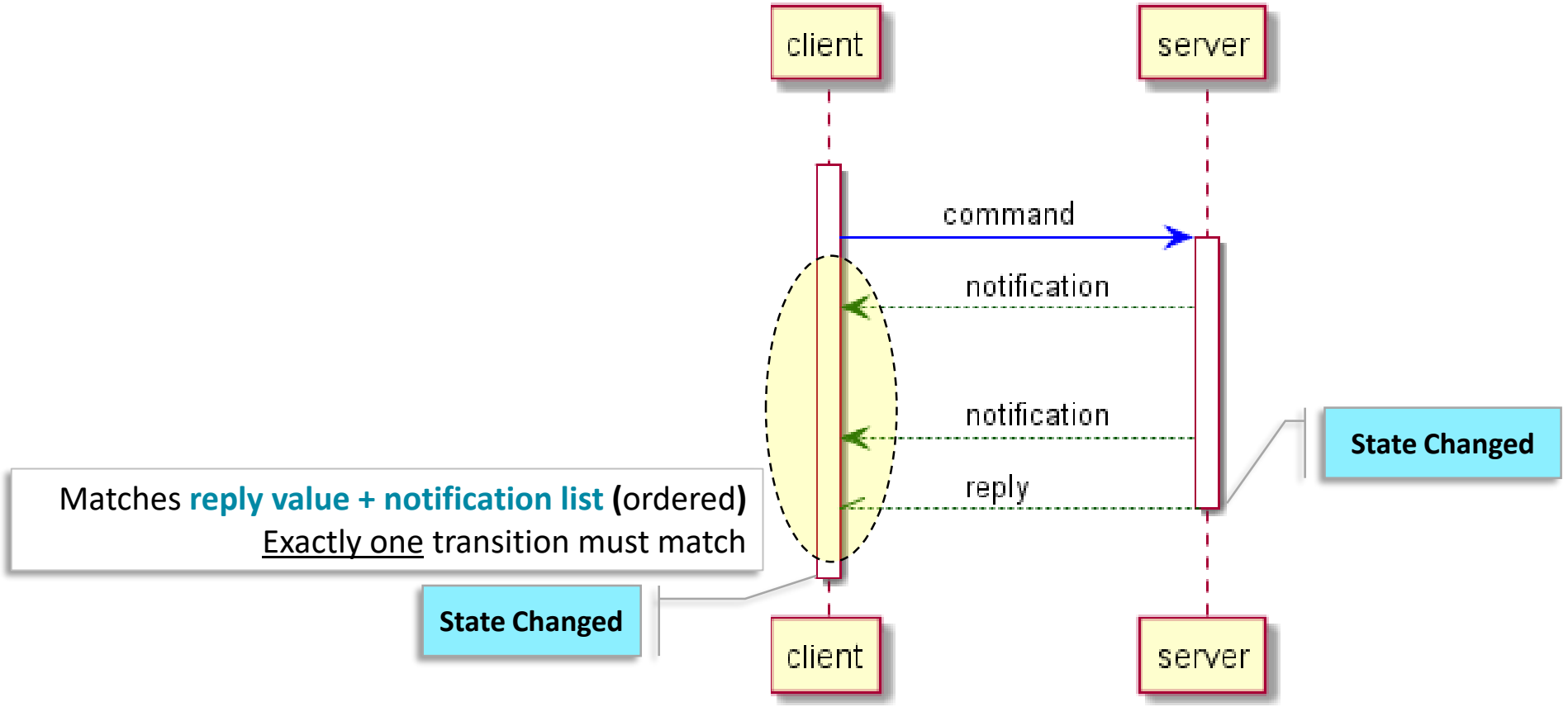
Client can trigger **Signals**



Triggered Transition

there is one more…

Event Types in ComMA - II



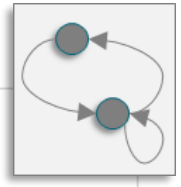
ComMA interface specification

Useful for monitoring and in the future for testing

Constraints

```
timing constraints
coinReq command coinThrowedIn - [ .. 70.0 ms ] -> reply(*)
errorReq in state Error notification outOfOrder then
notification outOfOrder with period 2000.0 ms jitter 200.0 ms

data constraints
variables
int val
coinStatus status

returnReg command returnMoney;reply(val) where val >= 0
observe returned := val

coinreturnReg command coinThrowedIn;reply(status)
where (status == coinStatus::ACCEPTED OR
status == coinStatus::NOT_ACCEPTED)
observe coinreply := status
```

Interface



```
state Operational {

    transition trigger: coinThrowedIn
            do: credit := credit + 1
            reply(coinStatus::ACCEPTED)
            next state: Operational
        OR
            do:
            reply(coinStatus::NOT_ACCEPTED)
            next state: Operational

    transition trigger: orderProduct(productName prod)
        guard: prod == productName::COLA
            do: if colaBottles < 1 then
                reply(result::NOT_ENOUGH_SUPPLIES)
            else
                if credit < price.colaPrice then
                    reply(result::NOT_ENOUGH_MONEY)
                else
                    credit := credit - price.colaPrice
                    colaBottles := colaBottles - 1
                    reply(result::DELIVERED)
                fi
            fi
            next state: Operational
}
```

Types

```
/*
 * Enum with the available products
 */
enum productName {
    WATER
    COLA
    JUICE
}

/*
 * When a coin is inserted in the
 * vending machine it can be either
 * rejected or accepted
 */
enum coinStatus {
    ACCEPTED
    NOT_ACCEPTED
}
```

Signature

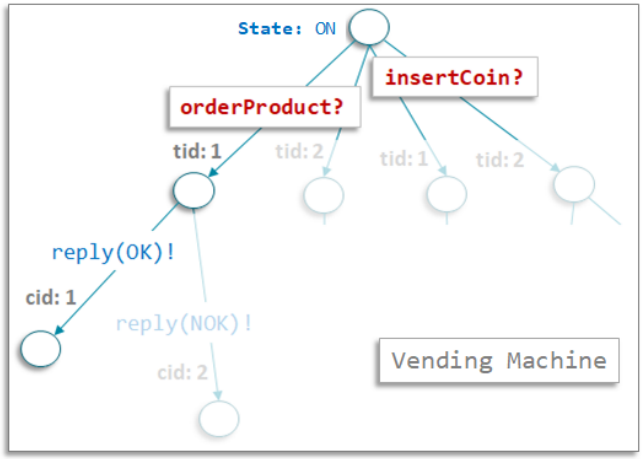
```
commands
void            loadProducts(int colaSupplies,
                             int juiceSupplies,
                             int waterSupplies)
coinStatus      coinThrowedIn
int             returnMoney
result          orderProduct(productName prodName)
Prices switchOn(in int idx, inout string testStr)

signals
switchOff

notifications
zeroTotalSupplies
outOfOrder
inventoryInfo(int items)
```

Example ComMA Specification

Various types of non-determinism may exist:

- Client makes choices of triggering a **command** or **signal**
- Server decides which **instance** and **clause** to execute.
- Server makes choices of generating **notifications**

ComMA interfaces can be expressed by the class of OPN whose skeletons are S-nets!



In State Operational **three transitions** are possible.

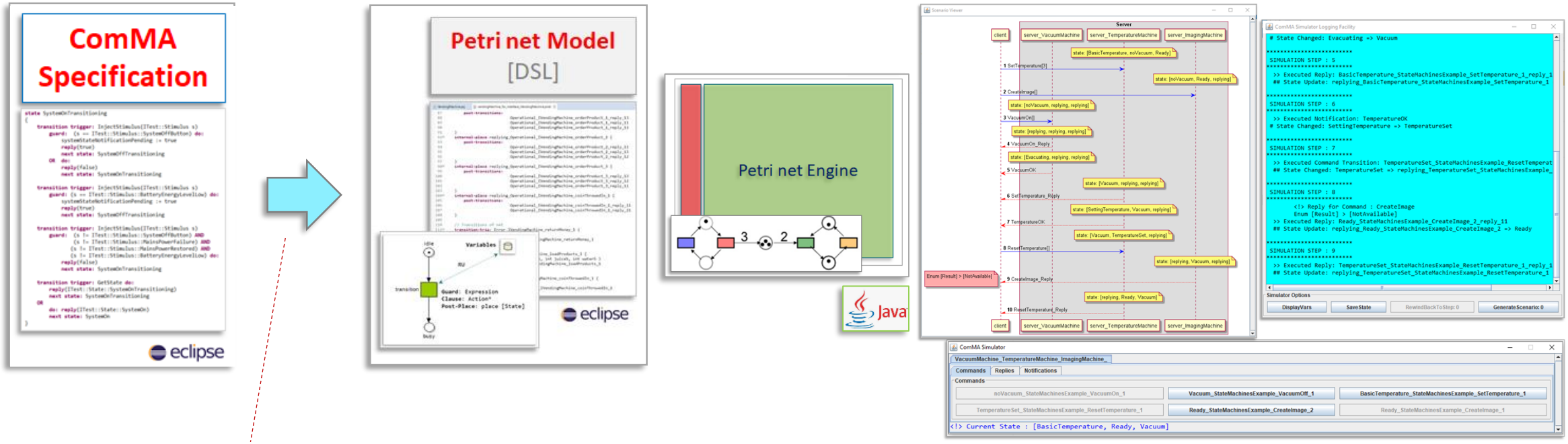
Two instances of the same transition (see **2 & 3**)

```
state Operational {

    transition trigger: coinThrowedIn
        do: credit := credit + 1
        reply(coinStatus::ACCEPTED)
        next state: Operational
    OR
        do:
        reply(coinStatus::NOT_ACCEPTED)
        next state: Operational

    transition trigger: orderProduct(productName prod)
        guard: prod == productName::COLA
            do: if colaBottles < 1 then
                reply(result::NOT_ENOUGH_SUPPLIES)
            else
                if credit < price.colaPrice then
                    reply(result::NOT_ENOUGH_MONEY)
                else
                    credit := credit - price.colaPrice
                    colaBottles := colaBottles - 1
                    reply(result::DELIVERED)
                fi
            fi
            next state: Operational

    transition trigger: orderProduct(productName prod)
        guard: prod == productName::JUICE
            do: if juiceBottles < 1 then
                reply(result::NOT_ENOUGH_SUPPLIES)
            else
                if credit < price.juicePrice then
                    reply(result::NOT_ENOUGH_MONEY)
                else
                    credit := credit - price.juicePrice
                    juiceBottles := juiceBottles - 1
                    reply(result::DELIVERED)
                fi
            fi
            next state: Operational
}
```

1 1 2 2 3

Two clauses

Petri Net Framework in ComMA

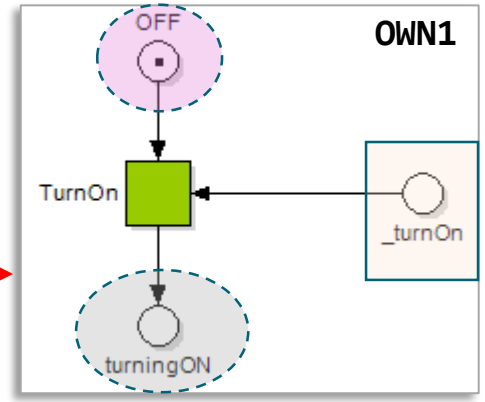


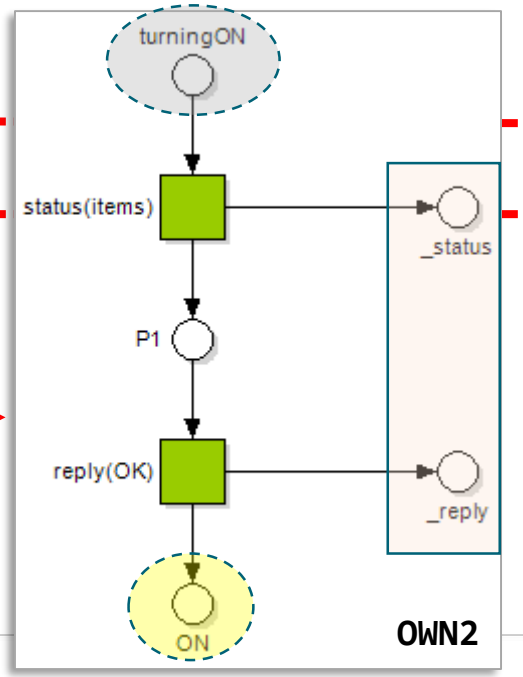
Based on standard place refinement and fusion operations

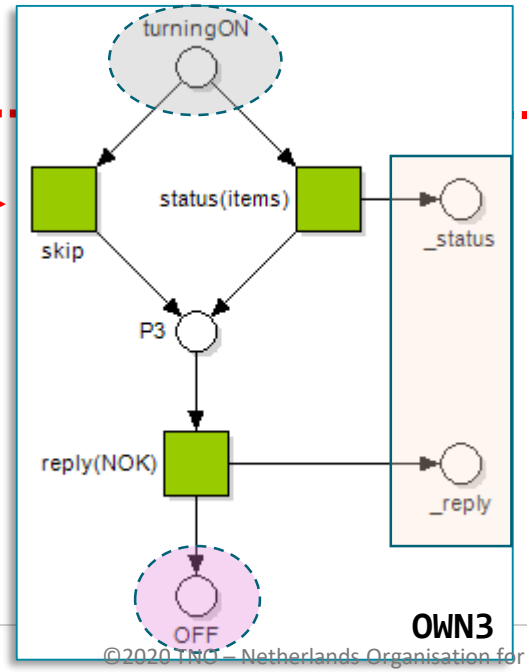
Gen. Trig. Transition

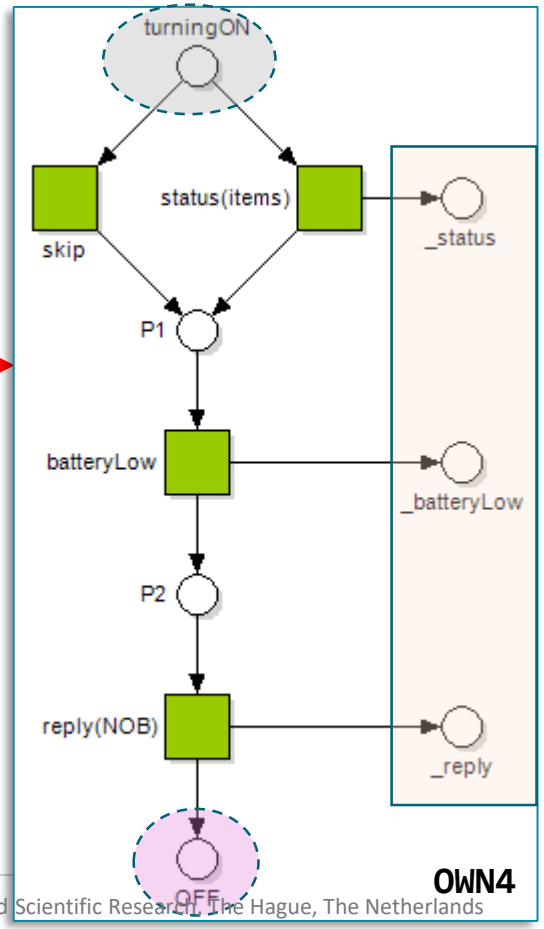
```
initial state OFF {
transition trigger: TurnOn
guard: coins > 0
do:
    if(items > 0) then
      status(items)
    fi
    if(cond) then
       batteryLow
       reply(NOB)
    fi
    reply(NOK)
    next state: OFF
OR
do:
    status(items)
    reply(OK)
    next state: ON
}
```

Fuse Places!



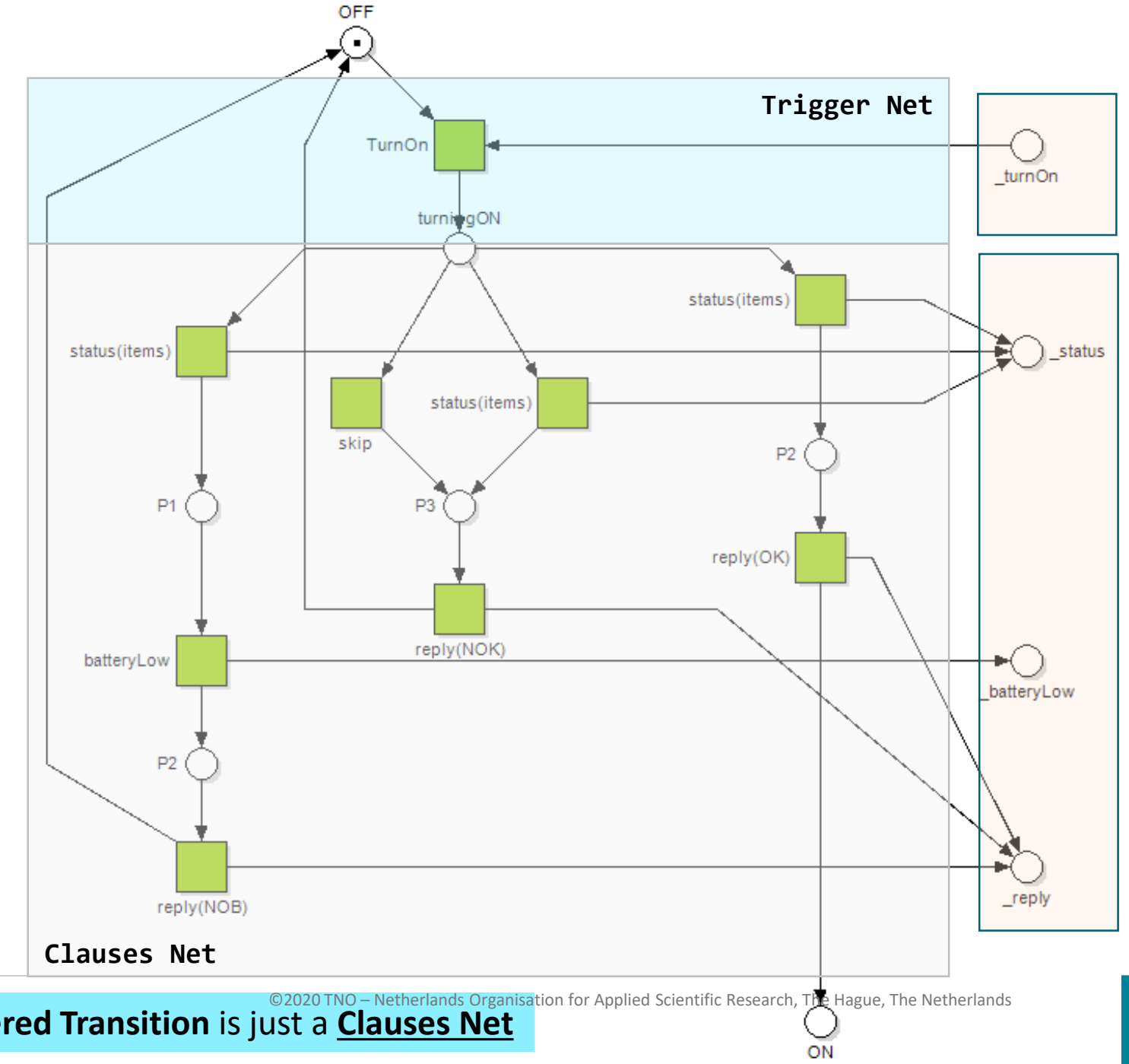






Gen. Trig. Transition

```
initial state OFF {
transition trigger: TurnOn
guard: coins > 0
do:
    if(items > 0) then
      status(items)
    fi
    if(cond) then
       batteryLow
       reply(NOB)
    fi
    reply(NOK)
    next state: OFF
OR
do:
    status(items)
    reply(OK)
    next state: ON
}
```

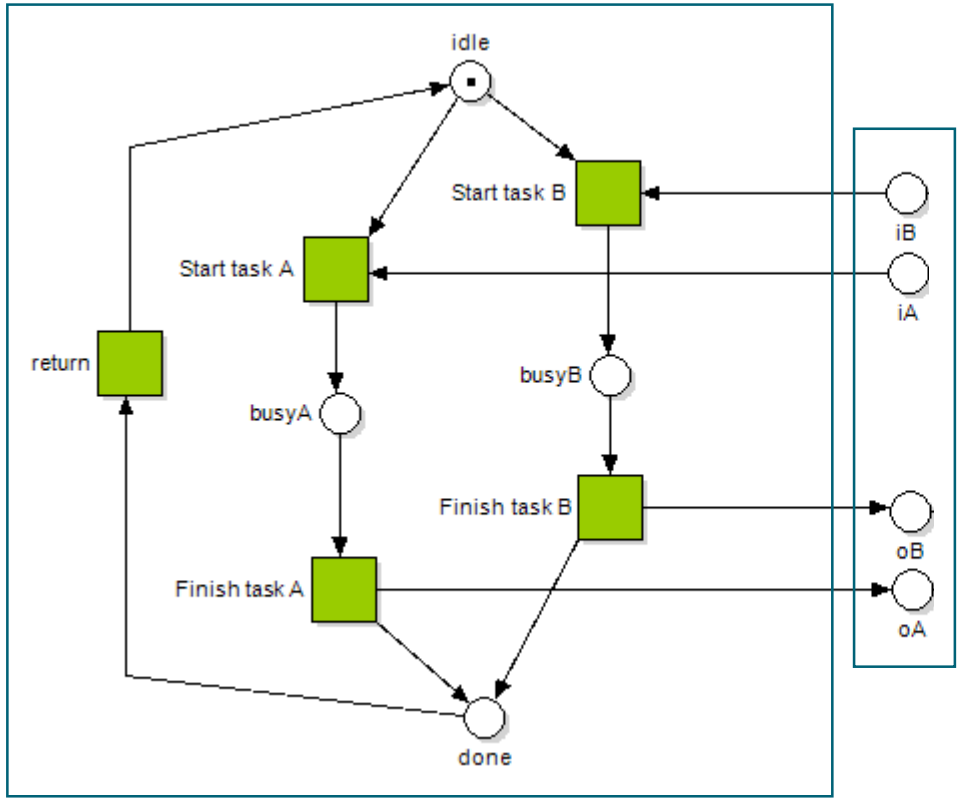


Signal is similar, just no **Reply**

Non-Triggered Transition is just a **Clauses Net**

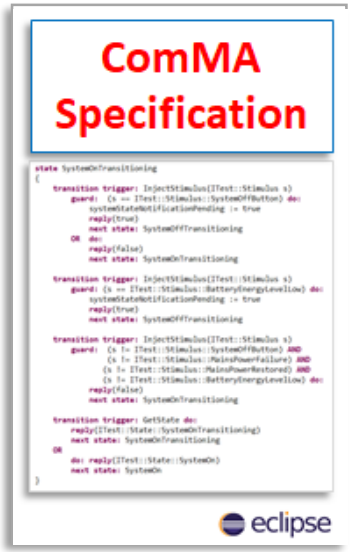
Deriving Client State Machine

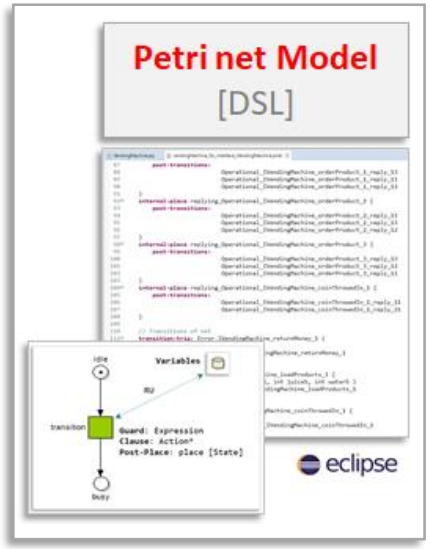
To do any sort of analysis or simulation on an OPN we must close it.
One way to derive a client is to just take the mirror!
This is a well-known class of interface nets called the **mirrored port pattern**

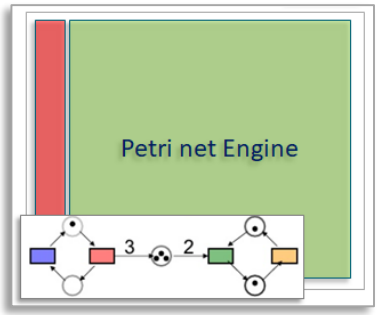


Server State Machine net

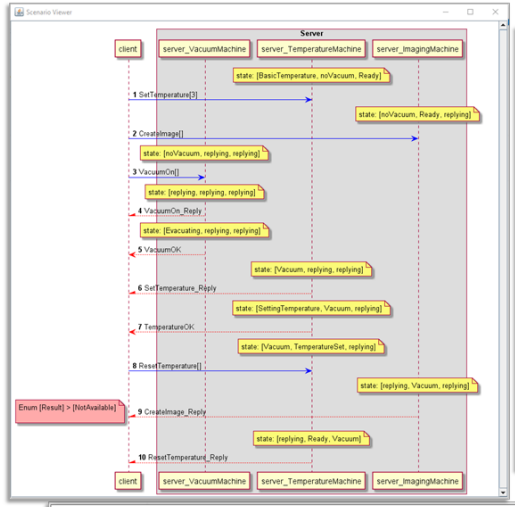
Petri Net Framework in ComMA

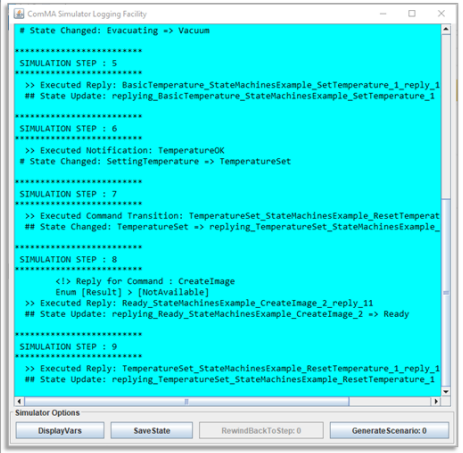


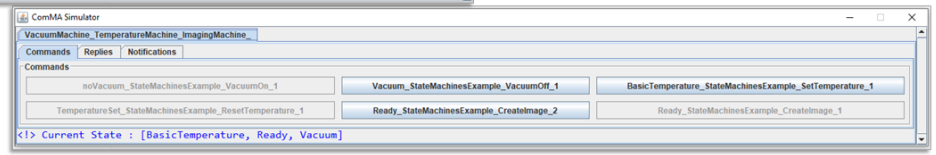




Simulation

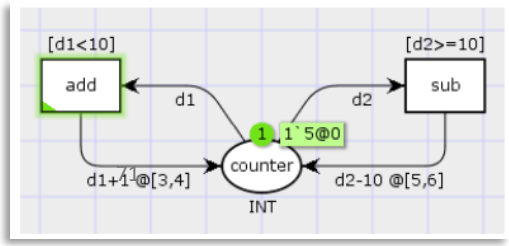




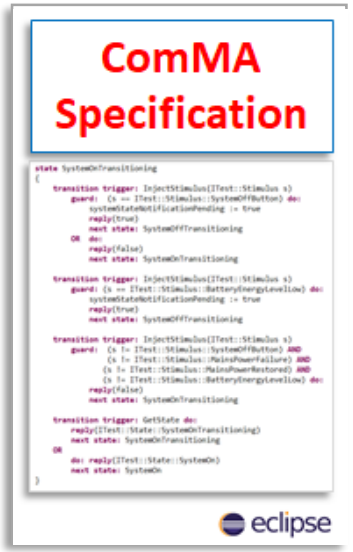


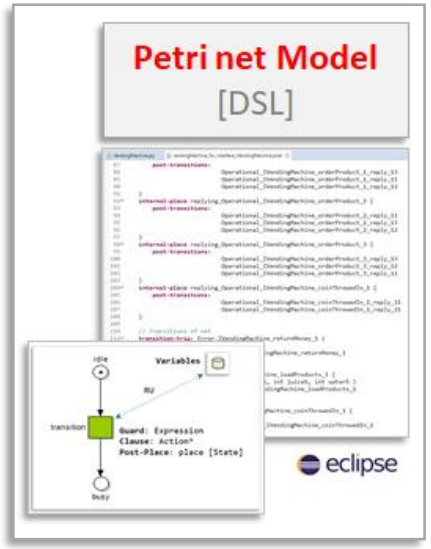
Classes of Petri nets

- **S-net**: state machine net
- **WF-net**: workflow net (Procedures)
- **OPN**: open Petri net (Interfaces, Components)
- **CPN**: colored Petri net (Data)

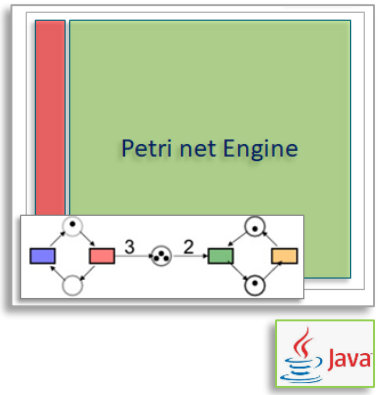


Applications of Petri nets in ComMA

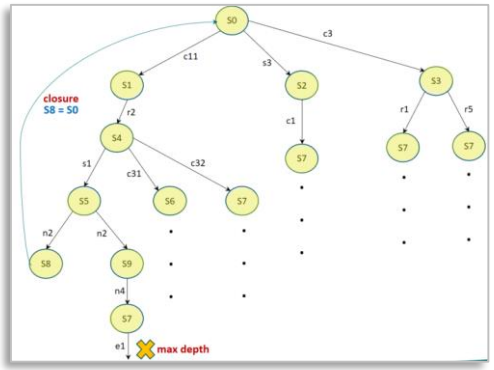




Simulation



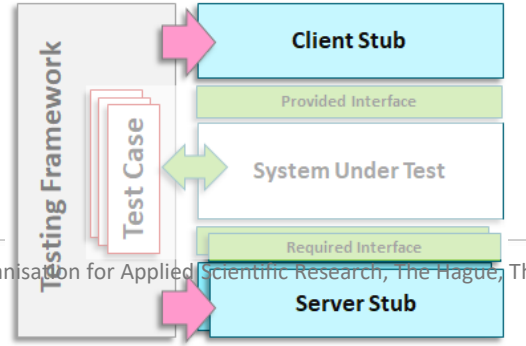
Reachability Graph



Checks on Properties

Dead States, Sink States, Race Conditions
State | Transition Coverage, Unbounded Events,
Functional Constraints, etc.

Testing



Demonstration

Sample Interface modeled in ComMA

Generation of Petri nets from ComMA Specifications

- Generation of Petri nets (skeleton of server OPN)
- Simulation and Generation of Reachability Graphs

Discussion

Go to www.menti.com

Discussion

Discussion in Breakout Sessions

Discuss the following three questions for 30 minutes in groups

1. **Can the interfaces you work with be modelled using Petri Nets?**

2. **What are the limitations of the presented theory in practice?**

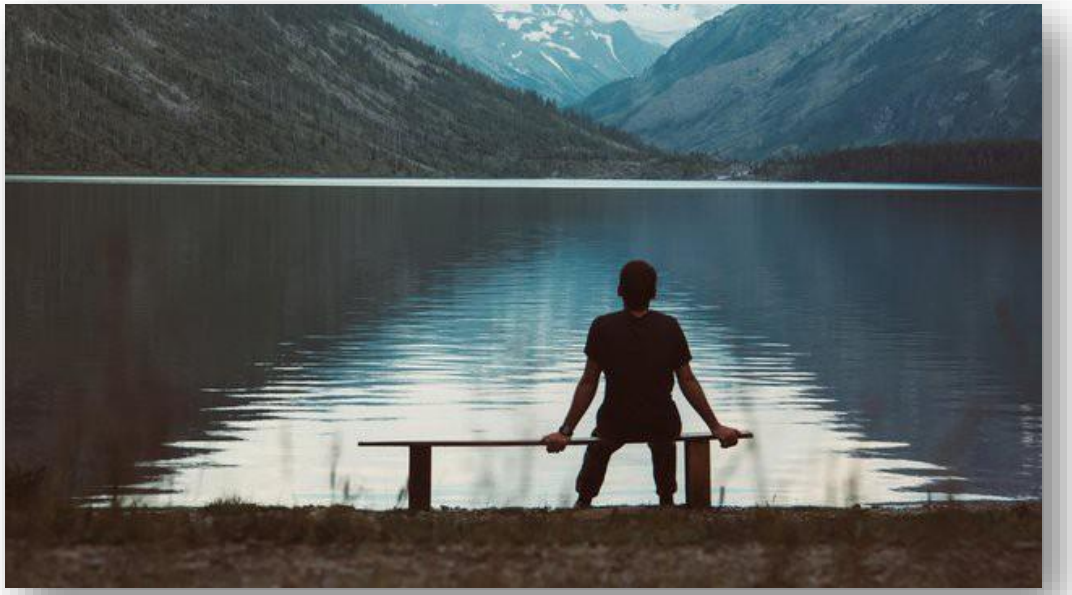
3. **What actionable knowledge have you gained in this module?**

Wrap Up

Reflection

Please take a minute to individually reflect on what you have learned

Write down any insights that you would like to remember after the course on a PostIt note



Analysis support in interface design is helpful. Why do we not have this already?

Sharing Reflections

Module Evaluation

Go to www.menti.com

Mentimeter

Copyright Notice

TNO grants Thales permission to reproduce and distribute this presentation according to article 6.6. of the Project Cooperation Agreement of Thales and ESI(TNO), effective data 1 January 2019, with ESI (TNO) document number 2019-10056.

When Thales wants to organize training courses based on this presentation, this shall be done in consultation with TNO to assure adequate quality of any training course based on this presentation. This presentation is protected by Dutch and International copyright laws. Reproduction and distribution of the presentation without prior written consent of TNO is prohibited.