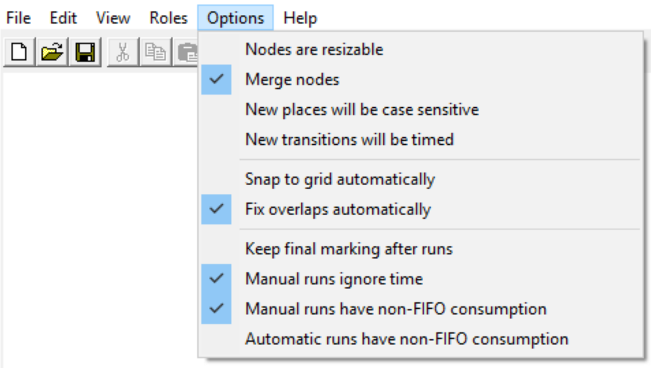
An initiative of industry, academia and TNO

Component-based Systems Modeling and Analysis

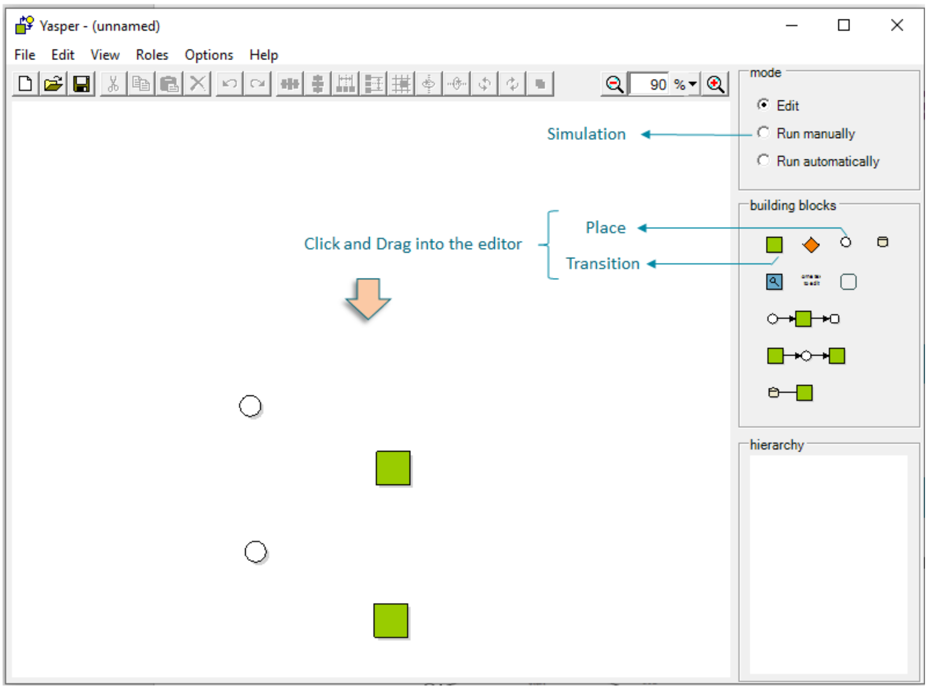
Module 1: Assignments

1 Familiarizing yourself with Yasper

1) Start Yasper and check configuration as shown below

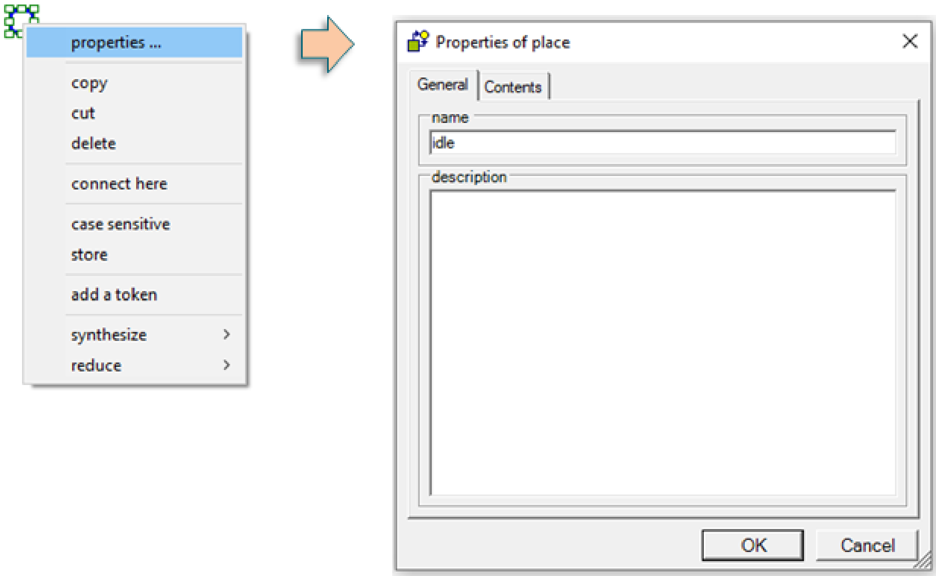


2) Explore the tool layout and try adding places and transitions to the editor.

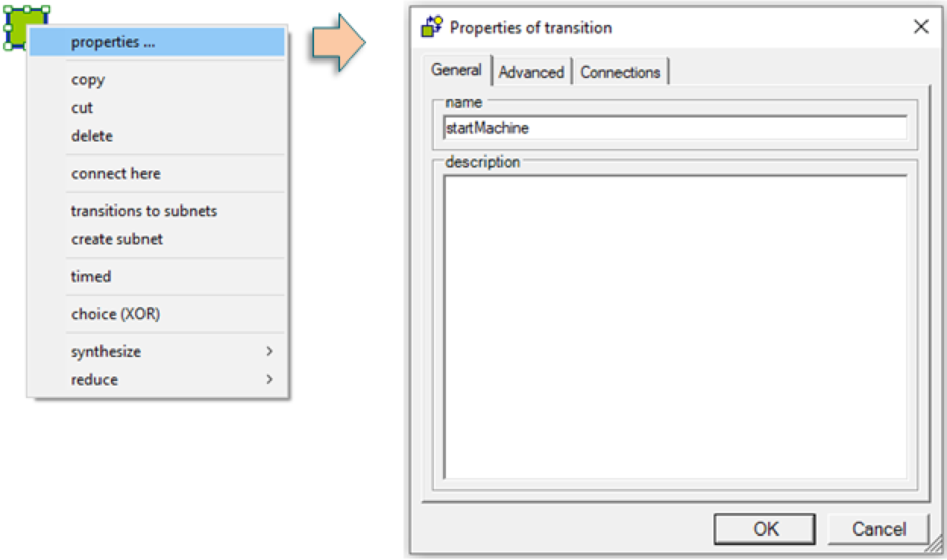


3) Give names to these places and transitions

Right click on a place



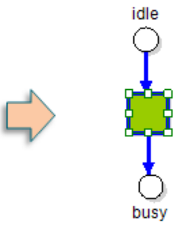
Right click on a transition



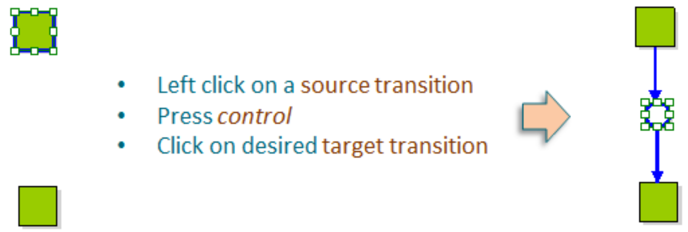
Alternative to right-clicking, is to just double-click on a node.

4) Now connect two places

- Left click on a source place
- Press *control*
- Click on desired target place

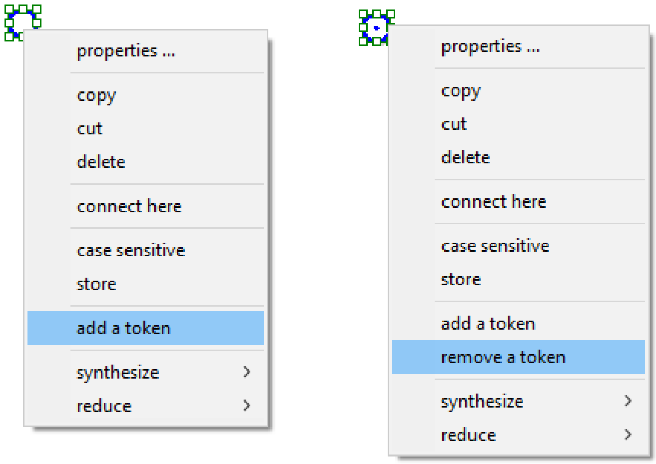


5) Now connect two transitions

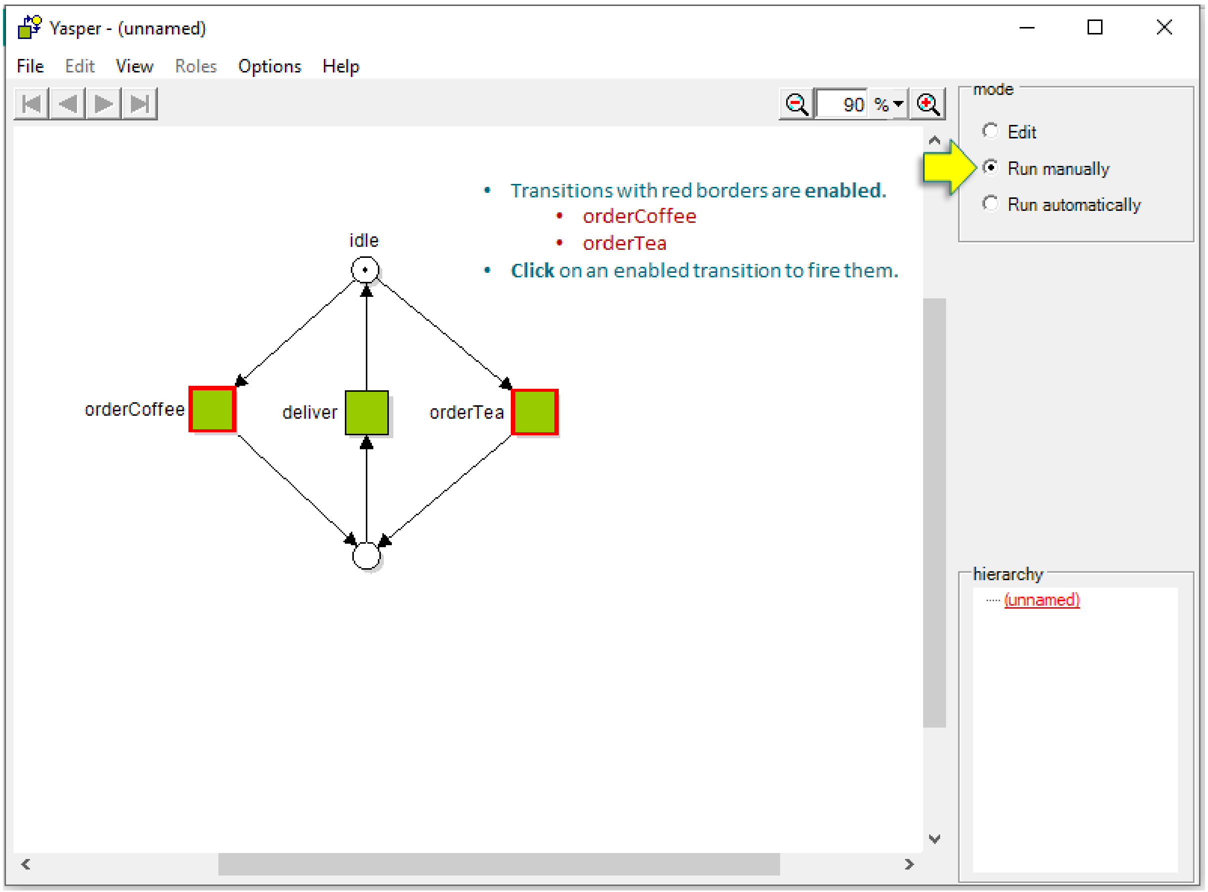


6) Use the same strategy to connect a pair of place and transition.

7) Now try adding and removing tokens from a place.



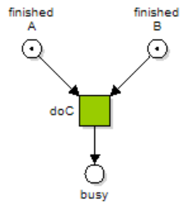
8) Now try simulation. Select option: Run manually and click on enabled transitions to fire them



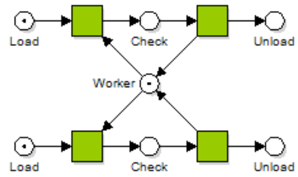
9) Open Yasper and try to make the following models and simulate them.

Sequential

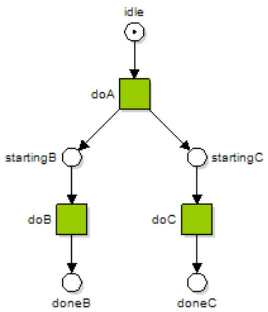
Synchronization



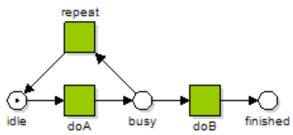
Mutual Exclusion



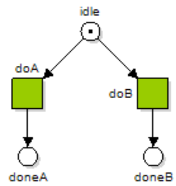
Concurrency



Loop

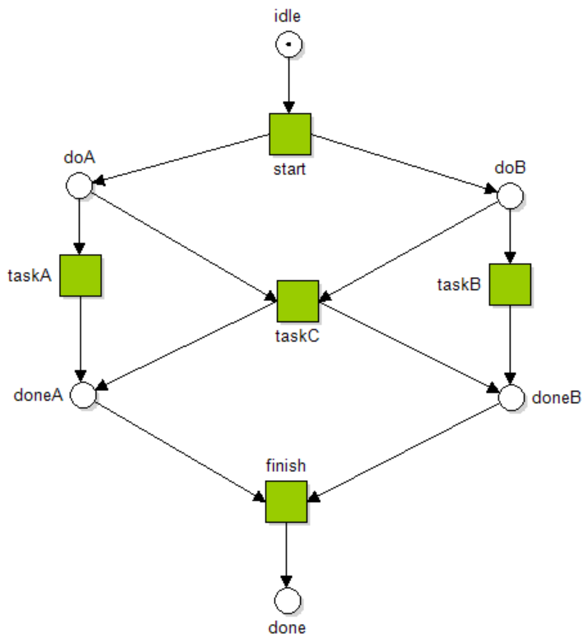


Choice / Non-determinism



2 Understanding Petri Net Syntax

1) In **Models** folder you will find a folder named **Module1Exercises**. In this folder, you will find a file named *BasicPetriNet.pnml*. Open it with Yasper.



2) Study the model and try to understand how it behaves. Identify the presence of non-determinism and concurrency.

3) Change mode in Yasper to Run Manually and play the token game. Validate your understanding of the behavior.

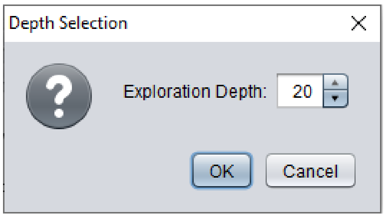
4) Change the mode in Yasper back to Edit and add a transition with name *repeat* to make the net cyclic, i.e. it consumes from place *done* and produces into place *idle*. We refer to such transitions as *closure*. Run the model manually and verify that it is possible to get back to the *idle* place.

5) Suppose we have a change request! Instead of a single task C, we need to split it up into three tasks (taskC1, taskC2, taskC3), such that taskC2 and taskC3 can be done only after taskC1 has been completed. The taskC2 and taskC3 have no dependencies, so they can be executed concurrently. Run the model and verify that it behaves as intended.

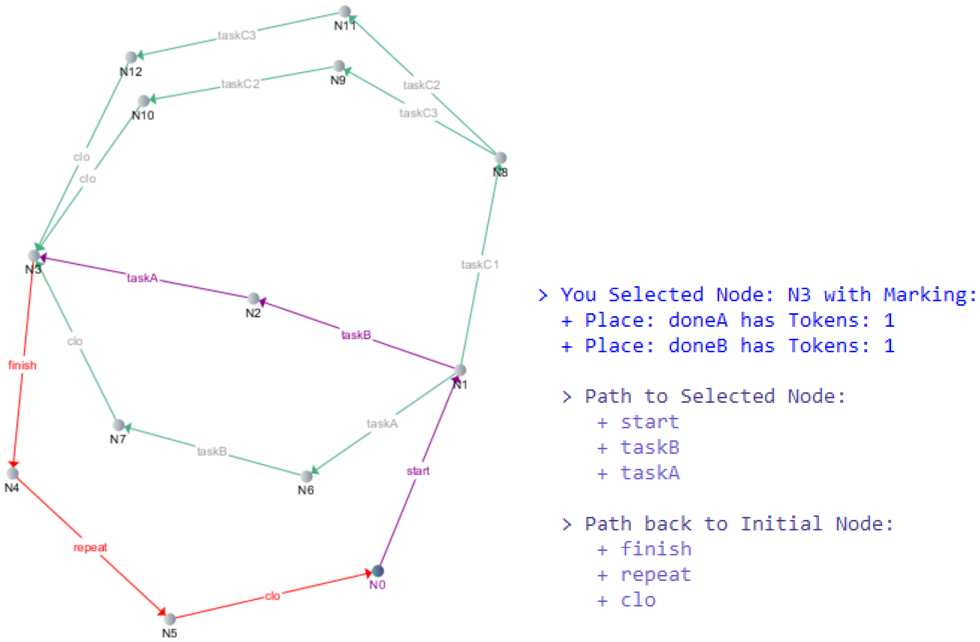
3 Understanding Petri Net Semantics

1) In folder PnAT, double click on *LaunchPetriNetAnalysis.jar* and select the model from the previous step, i.e. *BasicPetriNet.pnml.*

2) You will see many options for analysis. For now, click on compute-reachability-graph. You will be prompted to enter the depth of search, i.e. the maximum number of steps to explore from the initial marking. Enter value 20, and click OK. The displayed graph is a visualization of the reachability graph.



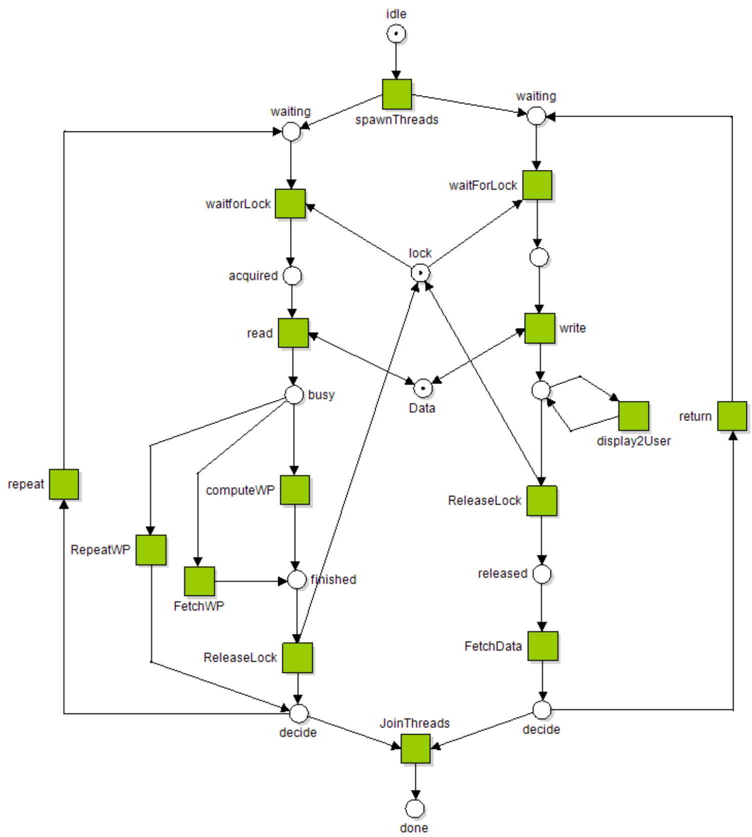
3) Inspect the nodes of the graph by clicking on them



4) Now check for weak termination by clicking on button *check-WT*. Analyze the results. Note that boundedness checks with k-value = 1 (i.e. safe) are also reported.

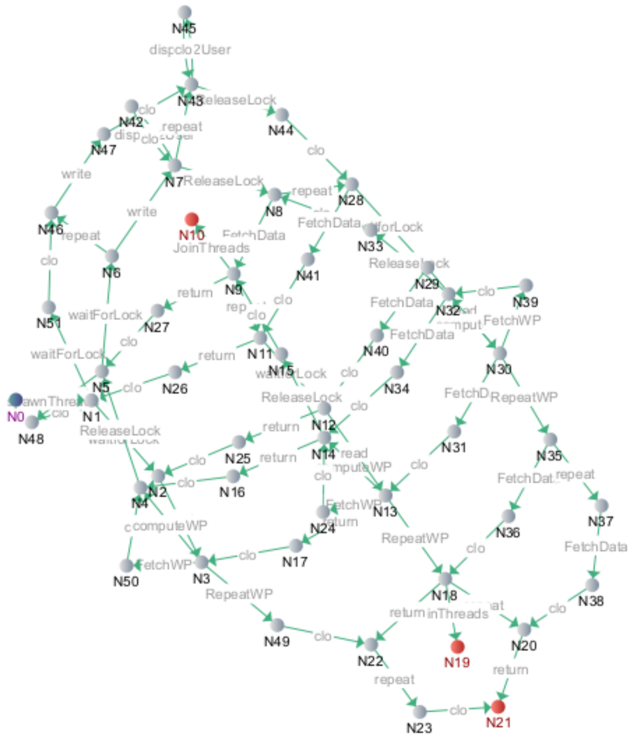
4 Resolving Problems in a Model

1) From the folder **Module1Exercises**, open the file *ThreadingAndDeadlock.pnml* in Yasper. The model contains a typical example of threading and synchronization. Look at the model and try to understand what it does. Think about whether this a workflow net and verify your conclusion with PnAT.



2) The model contains a deadlock. If you cannot see the reason, run the model in Yasper and try to find out.

3) Check if you found all the deadlocks using PnAT tools by generating the reachability graph. The visualization of this graph will already indicate if there are deadlock markings by coloring them in Red. Click on these nodes and inspect the graph. Note that a marking with one token in place done is also a deadlock marking, although this is not harmful since it is a final marking of a workflow net.



4) Analyze and resolve the deadlock by modifying the net.

5) Add a transition named *finish* connecting the place *done* to *idle*. Verify that there are still no more deadlocks in the model.

5 Coffee Machine

5.1 Model Server Interface Protocol of a Coffee Machine

Objective: Model an interface from the perspective of a server, understand OPN

1) Model the given description of a coffee machine as a Petri net in Yasper.
Recall that in an OPN a transition is either a send or receive, determined by the direction of the arc from the transition to its corresponding interface place.

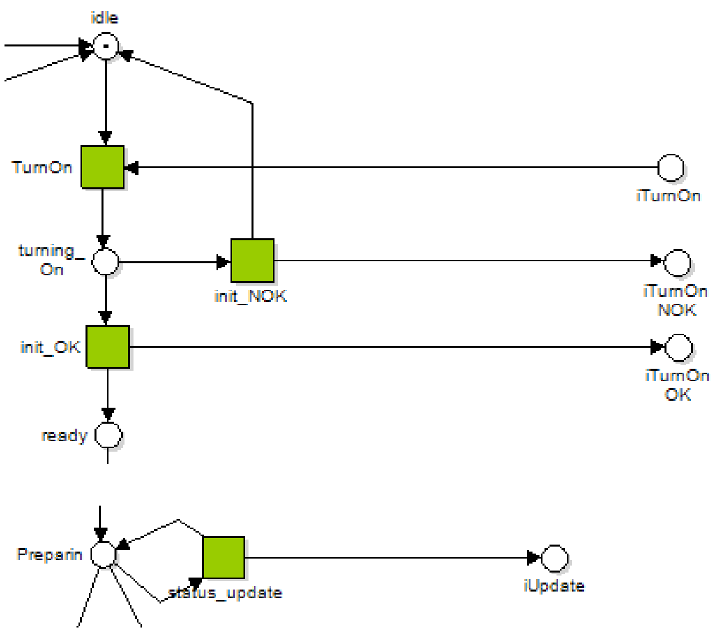
   In the idle state, it is possible to turn on the machine and receive either a positive acknowledgement and the machine goes to the operational state, or a negative acknowledgement and the coffee machine goes back to the idle state.

   In the operational state, it is possible to select the type of coffee. Once the selection is made, the machine goes to the selected state. In the selected state, the machine can accept coins until the amount is sufficient leading to the preparing state. In the preparing state, the machine gives periodic status updates until the coffee is prepared and the machine returns to idle state.

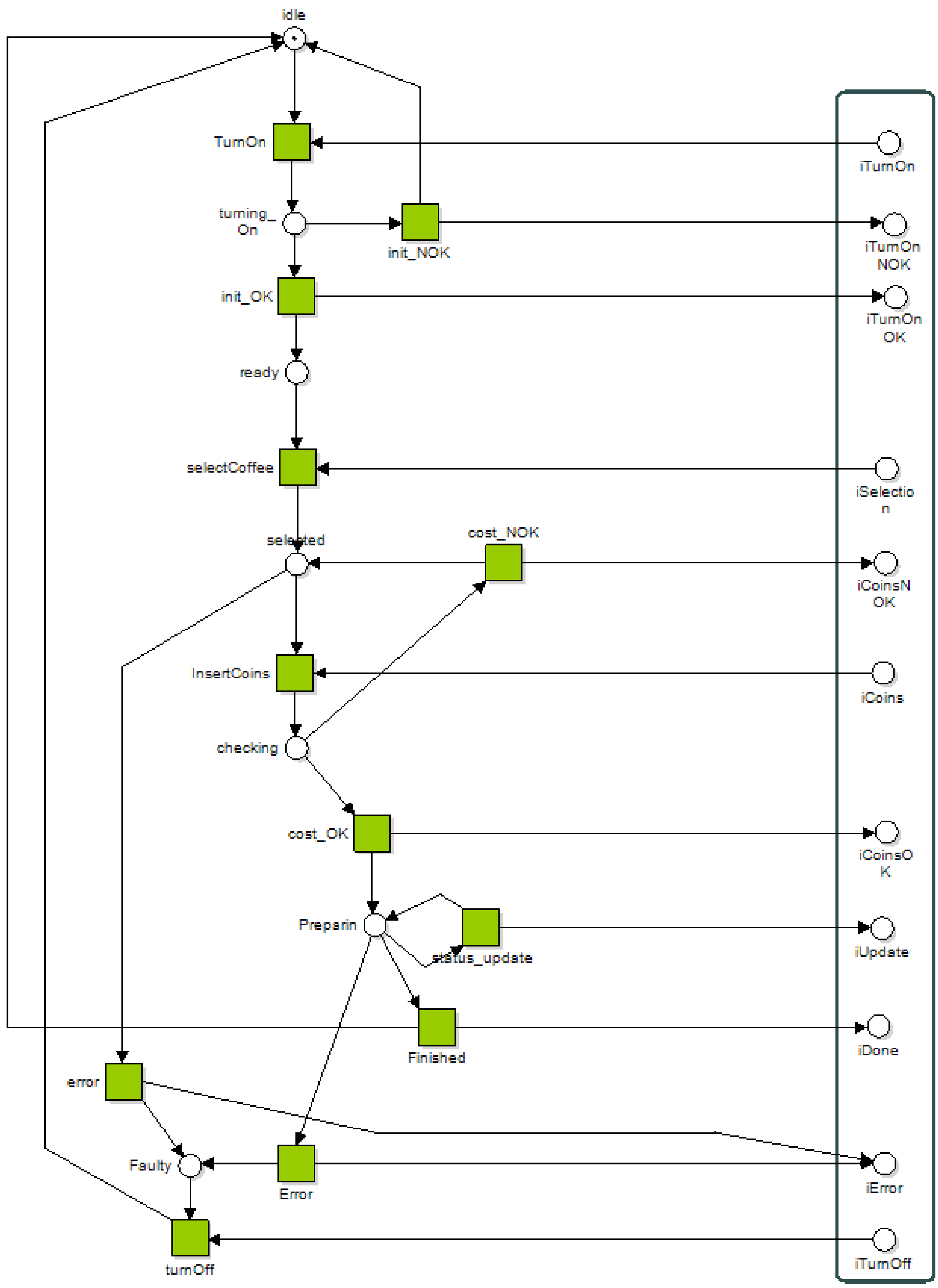
From both states selected and preparing, it is possible that the machine raises an error and goes to the fault state. In the fault state, it is only possible to switch off the coffee machine and return to idle state.

Hint: First identify the states of the coffee machine, e.g. idle, turningOn, operational etc. Then add transitions corresponding to an event of the server.

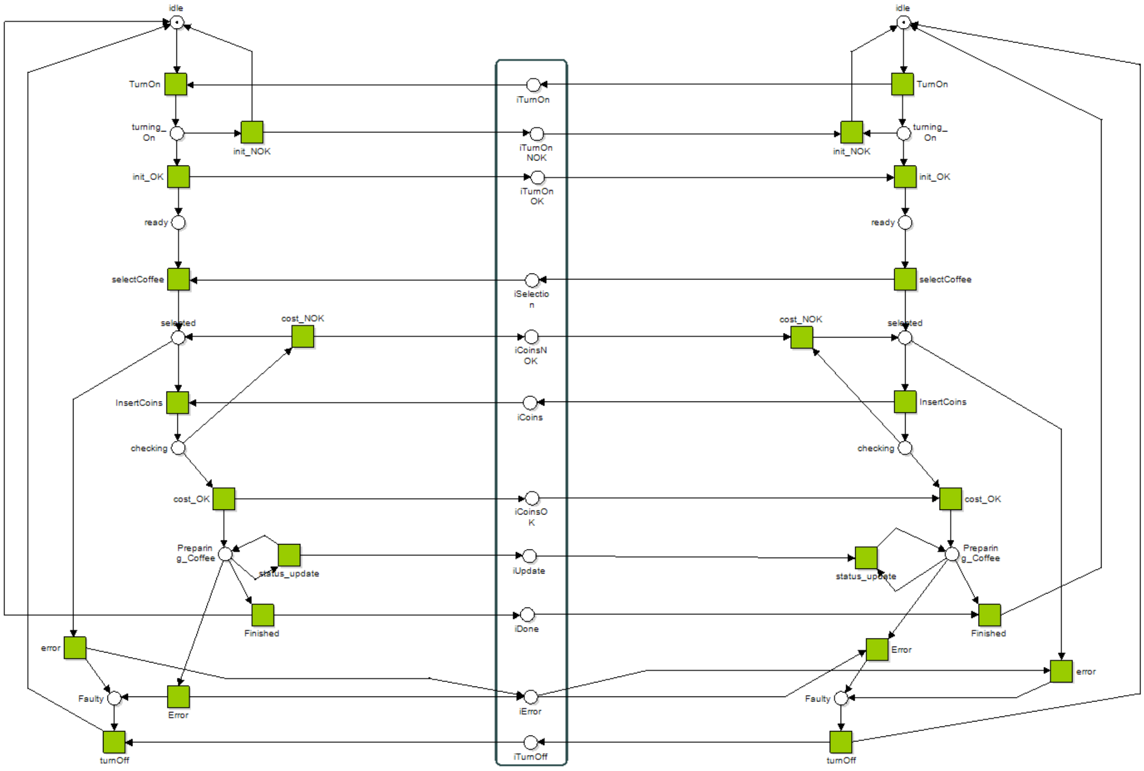
Two sample constructs are shown below to get you thinking and started:



2) If everything went well in the previous step then you must have a model that looks similar to *"ServerProtocolStateMachine.pnml"*. Check with PnAT that the model is weakly terminating.



3) Now make a mirrored client net for this server net. This done by copying the current net (select all nodes and then press: control + c) and pasting (press: control + v) it next to it (i.e. nodes are not overlapping). While the pasted net is still selected, flip it horizontally to create a mirrored structure (Edit menu -> Flip horizontally, or control+8). After that fuse matching interface places by dragging and dropping interface places with matching labels on top of each other. Lastly, invert the arcs from interface places to transitions of the client net by right clicking on the arc and choosing invert. The resulting net should look like this:



In the **solutions** folder: *1_ClientServerMirrorsOPN.pnml*

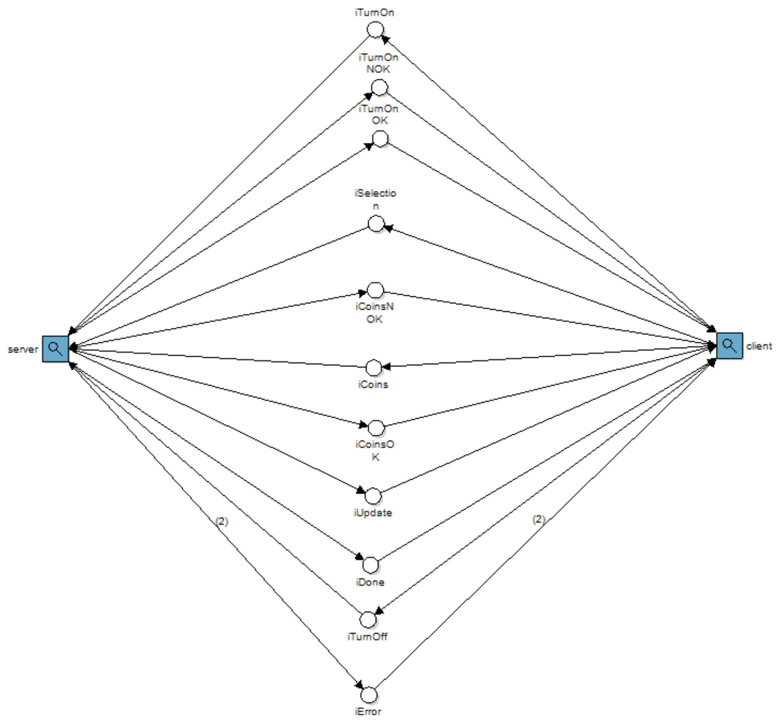
4) Check that the model no longer weakly terminates, even though the skeleton was weakly terminating! Resolve the problem(s).

   Hint: First resolve the boundedness problems caused by status updates. This results in a much smaller reachability graph where you can much easily find the deadlock.

Optional

5) To create a client subnet in Yasper, select all the transition of client, right-click and select create subnet. Repeat the same to create a server subnet.

6) Check using PnAT (i) The model is an OPN, (ii) Client-Server skeletons are S-Nets


iTurnOn
iTurnOn NOK
iTurnOn OK
iSelection
iCoinsNOK
server
client
iCoins
iCoinsOK
iUpdate
(2)
(2)
iDone
iTurnOff
iError