ESI

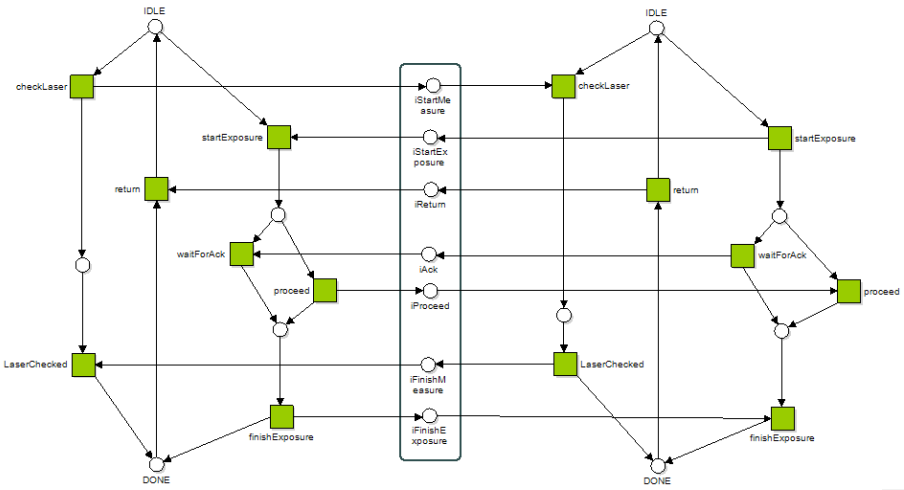
An initiative of industry, academia and TNO

Component-based Systems Modeling and Analysis

Module 2: Assignments

1 Understanding Choice and Leg Properties

1) In folder Module2Exercises, you will find a file named *ChoicePropertyViolations.pnml*. Open it with Yasper.



2) Study the model (model on the left is server and model on right is client) and spot the violations of the choice property. Check by simulation or PnAT that they indeed do not weakly terminate because of deadlocks. How many deadlocks are possible?

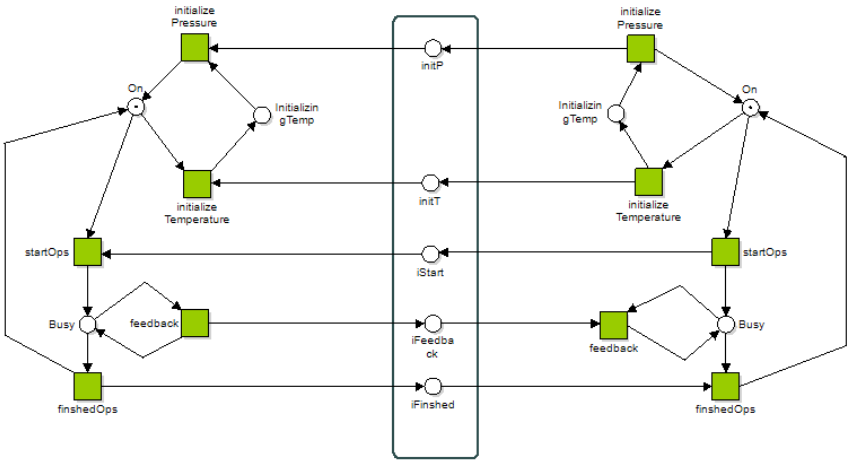
3) Try to fix the problem by adhering to the choice property. The constraint you have is that you can only add new events but not modify the existing ones. After making the changes verify that the problems are indeed no longer present and that the model is weakly terminating.

Commented [AK(1]: It should be easy enough to spot the choice property violations. It is also easy to fix by just reversing some arrows. However, this probably violates the function of this system. Probably, a few lines of text describing the system and outlining constraints is necessary. It would be good if it was indicated who is the server and who is the client.

Based on my assumptions about what this system does, I do not feel right taking away the possibility for the net on the left to check the laser. If it feels it needs to check, I want to let it check (If I knew more about the system I may feel differently). I would hence add a transition before start exposure that sends a message readyForExposure. The net on the left can hence decide if it wants to check the laser and if not, it gets ready for exposure.

It is not easy to decide what to do with the second violation, since I do not know what the ack and the proceed are for. It is hence not obvious that I cannot just reverse on of them.

4) In the folder Module2Exercises, you will find a file named *LegPropertyViolations.pnml*. Open it with Yasper.



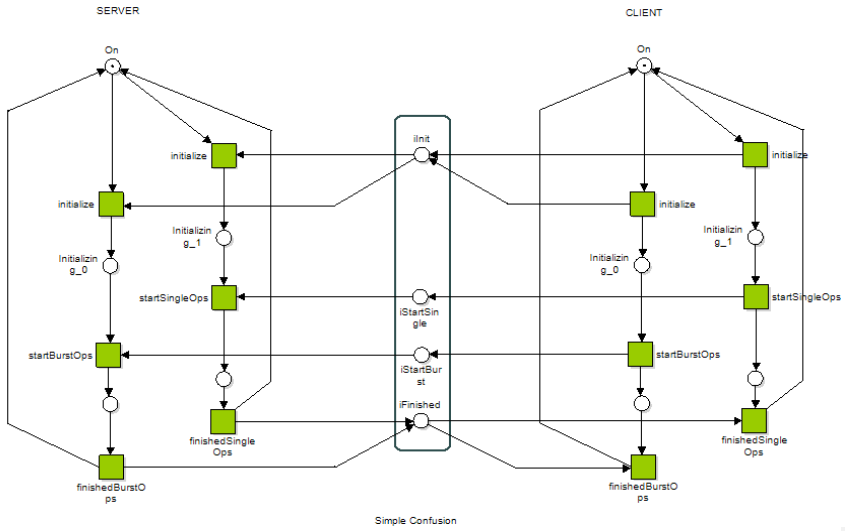
5) Study the model and spot the leg property violations. How many violations are there?

6) How would you resolve the leg violations?
Hint: Are acknowledgements missing? What are the other possibilities?

Commented [AK(2]: I see two leg property violations, init loop and feedback loop. I would resolve this by inserting at least one message going in the opposite direction. Maybe that is an acknowledgement for every message, or just an indication that the loop is finished, e.g. a single init complete in the top loop. I do not see that I have a lot of options here, so if this is the answer you are looking for then you do not need much more information about the system to solve the assignment.

2 Understanding Confusion

1) In Models folder, you will find a file named *SimpleConfusion.pnml*. Open it in Yasper.



Simple Confusion

2) Observe the confusion caused by the transition *initialize*. This leads to a deadlock. How would you resolve this? There are two possible solutions.

First try to solve the problem without modifying the interface places.
Hint: What do you think about the location where the choice is made?

Next reason about how modifying (adding or removing) existing interface places can resolve the problem?
Hint: Remember one of the Port net restrictions of the Portnet structure?

Commented [AK(3]: When discussing port nets, we should probably mention that it is prevents confusion by not allowing multiple transitions to consume from the same interface place. The problem would have to be resolved by splitting into two message types. If this is the solution you have in mind, we should probably mention it as a design guideline to prevent confusion in the slides.

When starting on the next I assignment, I read your hint again and realized that you probably wanted to merge the initialization and then introduce non-determinism. That works well for this case. Is this solution general enough to be considered a design guideline?

If we want to steer people in this direction, perhaps we should say that they have to solve the problem without modifying the interface.

Commented [BD(4R3]: This was very useful, have addressed it now by changing the way the exercise flows.

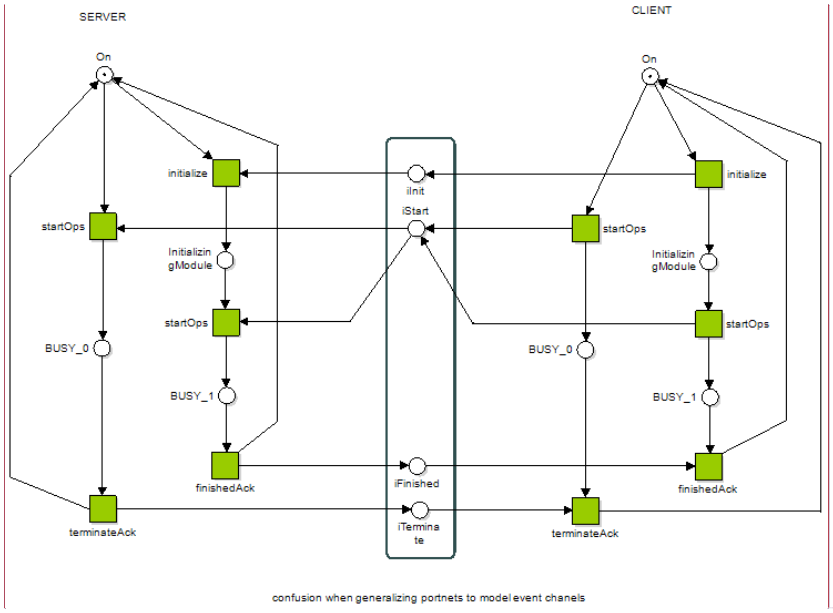
Commented [AK(5]: I was thinking about whether it is obvious that you can merge the initialize transitions. First, I was not sure whether they did exactly the same thing. That raises a question about semantics. Does having the same label mean they do exactly the same thing?

I then thought about it some more, and I realized that both computations have exactly the same input (an init message) so there cannot be any single or burst specific behavior in there. This suggests they can be merged.

On the other hand, this means that it is not a very good idea to split up the init message into two distinct messages for the second case, because they contain the same information.

It is possible to deal with this as a part of a discussion with participants. On the other hand, I can see a critical ESI audience saying that the assignment is ambiguous and under-specified and that they cannot continue without clarity. Sometimes, they overthink it! 😊

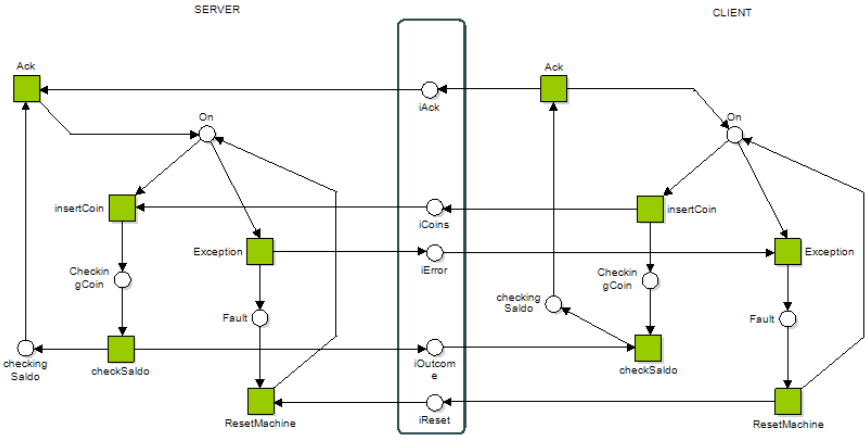
Should we add something to the description and in the first case say that they should imagine that transitions with the same label perform the same computation and not to make this assumption in the second case.



3) Now open a file named *Confusion.pnml* in Yasper. This is yet another example of a confusion leading to a deadlock. How would you resolve the problem?
Hint: Can an acknowledgement at the right place save the day?

3 Identify and Resolve Problems with the Coffee Machine

1) Open the file *SimpleCoffeeMachineWithProblems.pnml* in Yasper.



Commented [AK(6]: In Module 1, we showed participants how to make open nets with subnets. We do not seem to use that in this module. Should we take this out of Module 1 and save it for when it is needed?

Commented [BD(7R6]: It seems to be the case..

Commented [AK(8]: Based on the text, my first instinct would now be to split up the start message into two different types. However, that is the same solution as I thought of above when you talked about port net properties. Do you want me to run with the same solution again?

If I understand this correctly, perhaps the first problem focuses only on the merging solution and clearly specifies that the interface cannot be touched and here we indicate that it is possible to modify the interface and refer to structural properties of port nets

This is currently my feeling about the design guideline for confusion. We have to create the best possible description for how to spot a confusion problem. The simplest solution is to split up messages. If this is not possible, you have to think very carefully about the structure of your net (application specific) and make sure that consuming from the same source does not cause problems. A drawback of this solution is that it can easily reappear when the application is modified. The only simple rule that always work is to not consume the same message from many places.

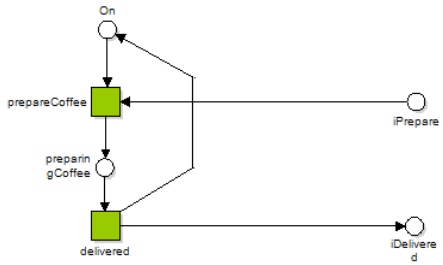
Commented [AK(9]: It is a bit tricky to verify termination of this net since it cannot come back to its initial state. The answer to the question is that it is not weakly terminating, but you have to check the reason to see if it is something you can live with, e.g. that it is only node 0 that is unreachable. In that sense, it is easier if there is an off button that brings it back to the initial state.

Commented [BD(10R9]: Indeed. Fixed the problem now

2) Study the model and try to spot the problem by visual inspection. This is a commonly occurring pattern in practice! How would you resolve it?
Hint: Diamonds? Think about the solution taking into context the functionality as well. Should new events be introduced or existing ones further generalized?

3) If all went well, you managed to make the model weakly terminate. Otherwise, open the model *ResolvingChoiceWithDiamonds.pnml*.
Now delete the transition Ack. Why does it not weakly terminate anymore? What class of problem does it belong to? Resolve the problem by adding the transition Ack back to the model again.

4) We will now add the possibility to order coffee in our machine!
Open model *AddFeature.pnml* using Yasper. The model describes a feature: order coffee, which should be possible from the state ON.



Reason about how you will add this feature to the coffee machine model without disturbing the weak termination property.
Hint: What additional possibilities should be considered in the states: preparingCoffee and Fault?
Validate your reasoning by adding this feature to the coffee machine model and check that it is weakly terminating again.

Commented [AK(11]: This model is in the solutions folder. We probably have to move it. My client had the initial token in On instead of the initial unlabeled state (should probably be labelled OFF or _OFF).

Commented [AK(12]: In which machine is this? We conclude the previous step by asking participants how they would fix the problem. Should we require them to actually fix it and make sure it is weakly terminating, or should they use an unmodified version of ResolvingChoiceWithDiamonds?

Commented [AK(13]: This easily results in trial and error and cluttered solutions. It would be nice if we could provide design guidelines that can be systematically applied to solve the problem. It should be clear that the problem is hard, but the guidelines make it easy. Then we have added value.

Commented [BD(14R13]: We will have a slide on this now, to be presented just after the exercises.