Exercise & Case study (Telecom Systems) proposal for solutions

F. Kordon & C. Girault - MATCH School

Installation - if you did not run Exercise 1 on your account.

First, you must get all the information (models for this exercise and formalisms) to be operational. To do so, please type first (when logged under Unix) the following commands :

```
>cd
>cp -r /export/home/profesores/kordon/MACAO .
```

Then, typing "ls", you notice a MACAO folder in your home directory. All required information is there. When you run Macao under your Home Directory and open models, You will get into this MACAO that contains two folders :

- FORMALISMS that contains the AMI-Net description (you go there when you create a new model),
- MODELS in which we will insert (if required) directories containing models for exercises.

Installation - if you did run Exercise 1 on your account.

First, you must get all the information (models for this exercise and formalisms) to be operational. To do so, please type first (when logged under Unix) the following commands :

```
>cd
>cd MACAO/MODELS
>cp -r /export/home/profesores/kordon/MACAO/MODELS/EXERCISE_TELECOM .
>cp -r /export/home/profesores/kordon/MACAO/MODELS/SOLUTION_TELECOM .
```

All should read this.

1) In the MACAO/MODELS folder, you will find the EXERCISE_TELECOM folder containing two models. File identifier starts with a number that corresponds to the question you are required to use the model.

2) In the MACAO/MODELS folder, you will find the SOLUTION_TELECOM folder containing modeling solutions. If you want to check your solution or get tired of looking for it, you can get them. File identifier starts with a number that corresponds to the related question.

3) There is an annex (at the end of this document) that provides you with some basics about PROD queries (menu «Evaluation of the RG with PROD/Expert mode/Build a query...» in CPN-AMI).

1. Design of a safe channel

We would like to design a safe channel based on a single cable line. The usual problem with a unique cable is that electric signals coming from various origins may provoke collisions (message is lost). To ensure a safe communication on the channel, we propose the architecture of Figure 1.

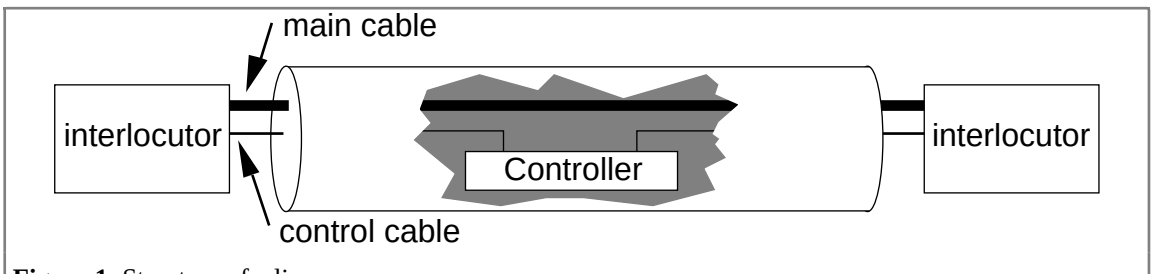


Figure 1: Structure of a line.

The channel relates two *interlocutors* that communicate together. It is composed of a *line* and a *controller* that manages shared access to the channel *main cable* (128 bits

width). The controller is connected two each interlocutor with a discrete *control cable* (3 bits width). There is one control cable per interlocutor. Interlocutors cannot send a signal at the same time : they must ask first the line to the controller that accepts or refuse (according to an implemented strategy).

Interlocutors have to respect the following protocol :

(1) the default state for an interlocutor is listening to the main cable,
(2) when it wants to emit a signal, the interlocutor asks for the main cable,
(3) if the controller provides the main line, then, the interlocutor sends its message and waits for an acknowledge,
(4) if the controller refuses the main line, then the interlocutor listens to the main cable (a message should arrive) and retries later on,
(5) interlocutors only send one message at a time,
(6) when an interlocutor gets its acknowledge, it frees the line for another use,
(7) Only messages passing through the main cable are acknowledged[1].

The table above provides the identification of signals passing through the cables.

Signal name	Signification	Signal direction[2]	
		Interlocutor	Controller
AMC	Ask for main cable		►
RMC	Refuse main cable		◄
PMC	Provide main cable		◄
MSG	Message	◄ ►	
ACK	Acknowledge	◄ ►	
FMC	Free the main cable		►

Signal between two interlocutors are transmitted using the main cable. Other signals are transmitted on the control cables.

Some typical execution scenarii are provided hereafter to illustrate the behavior of a interlocutor according to specific situations.

Figure 2 illustrates the behavior of a interlocutor that initiates a communication when the controller provides the main cable. Then, the answer to AMC (demand) is PMC. The interlocutor (here, 1) then sends the message to the other interlocutor (here, 2) and waits for an acknowledge. When it gets the acknowledge, it releases the main cable (FMC).

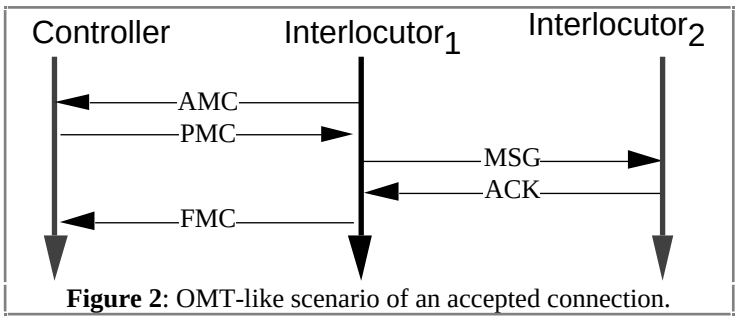


Figure 2: OMT-like scenario of an accepted connection.

Figure 3 shows the behavior of an interlocutor when the controller refuses the main cable (RMC). It means that the other interlocutor (here, 2) obtained the cable and is sending data. The interlocutor (here, 1) then waits for the incoming message and

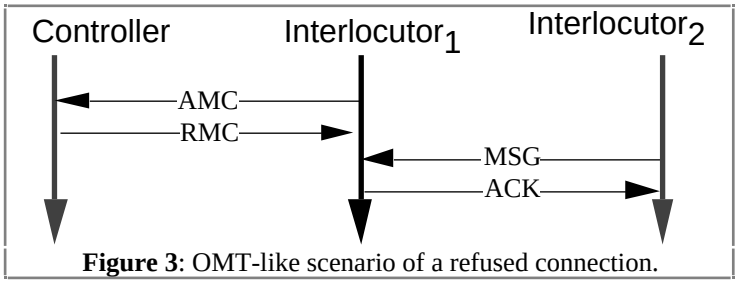


Figure 3: OMT-like scenario of a refused connection.

1 Acknowledgemen will be usefull further in the study.
2 ◄ and ► provides the direction of the signal, ◄ ► means that the signal is exchanged between two interlocutors.

acknowledges it. The interlocutor will try later to get the main cable.

The aim of this study is to model this system (the controller and the interlocutors) with P/T nets in order to validate it. To do so, we model separately the behavior of each components of the safe channel. Then, we connect them together to study the system.

One modeling technique is to let Petri net modules communicate by means of communication places. Signal identified in the table are a good candidate for these communication places. We thus identify six places that will act as interfaces between the two components of the system : AMC, RMC, PMC, MSG, ACK and FMC. The signification of these places is “some signal is in” (signal AMC for place AMC, etc.).

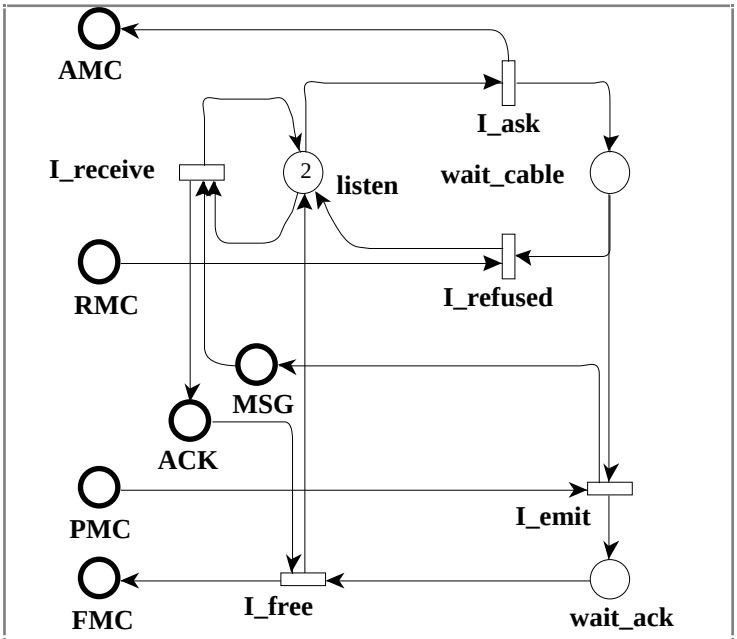


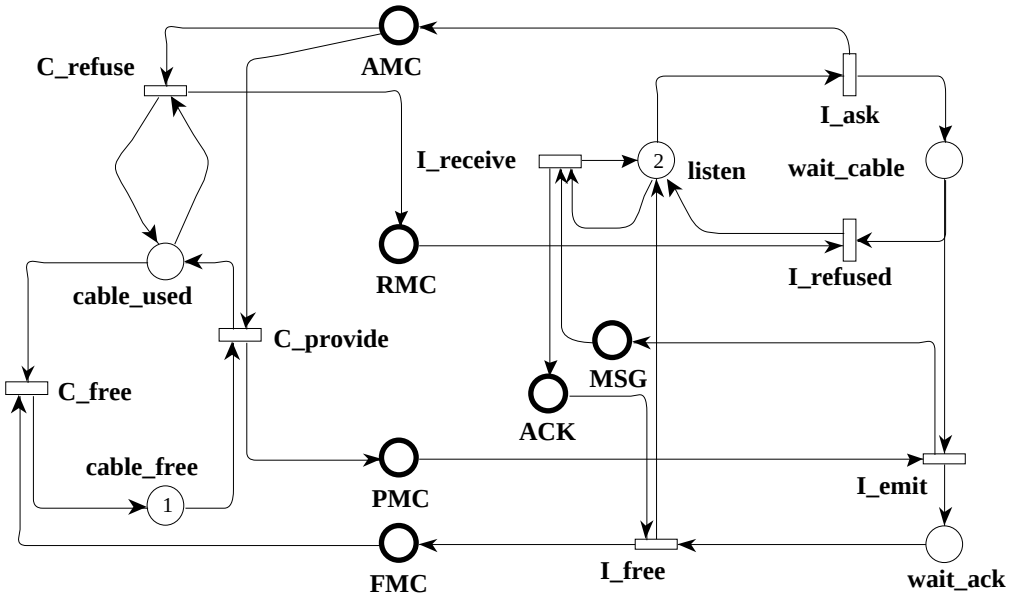
Figure 4: Petri net modeling interlocutors in the system.

We then provide the Petri net that models interlocutors in the system (Figure 4). Interfaces places are bolded. Each interlocutor listen to the main cable (place *listen*). When it asks for the cable (transition *I_ask*), it may get either a RMC (and then goes back to *listen*) or a PMC. It then emit the message and waits for its acknowledge. When it comes, the interlocutor frees the line (transition *I_free*). There are two interlocutors (on each end of the channel) and thus two tokens[3] in *listen*.

Question 1

Complete the Petri net model by inserting the behavior of the controller.

Answer: The Petri net model should be.

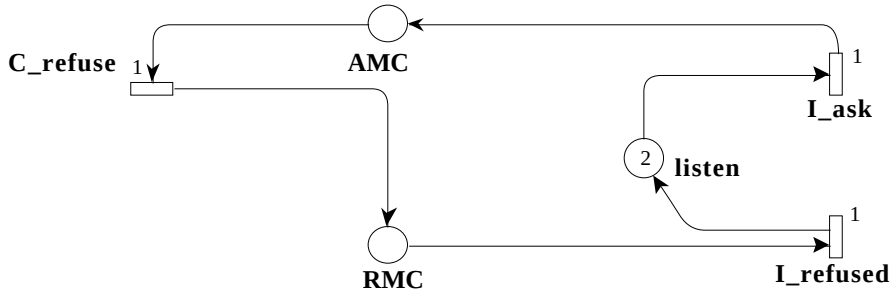


[3] This works only because there are two interlocutor.

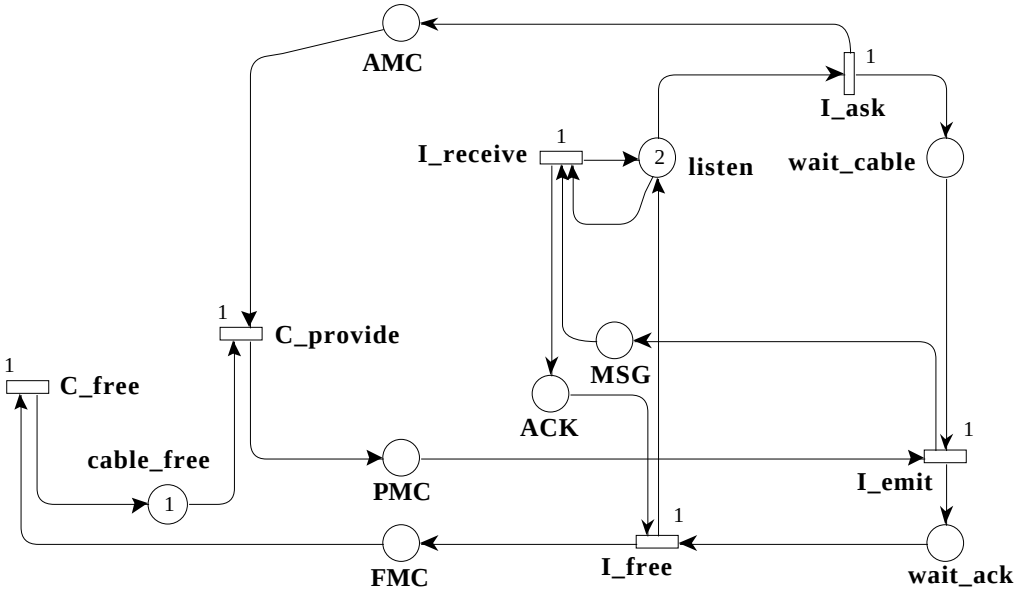
Question 2

Check the T-invariants with GreatSPN services (under CPN-AMI) on the complete model. Can you provide a firing sequence and an interpretation for them?

Answer: The first T-invariant corresponds to the behavior of an interlocutor who does not get the main cable.



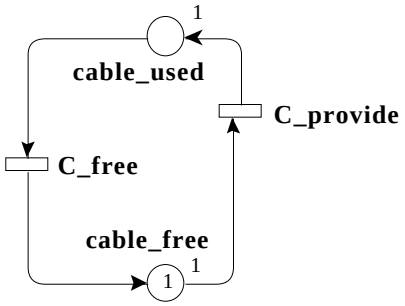
The second T-Invariant corresponds to the scenario where an interlocutor gets the line, sends its message and gets the acknowledge.



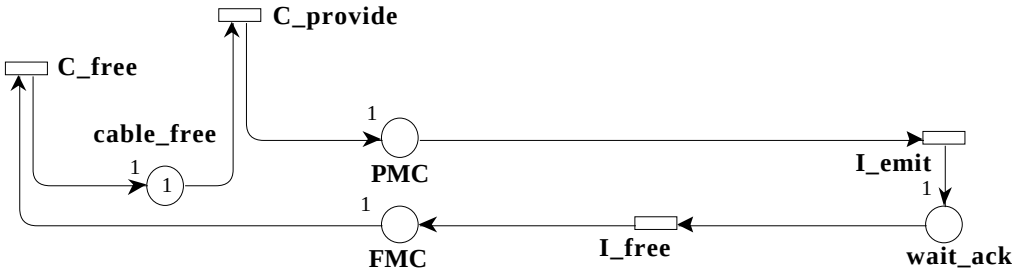
Question 3

Check the P-invariants with GreatSPN services on the complete model. Some of them can be interpreted. Can you provide an interpretation?

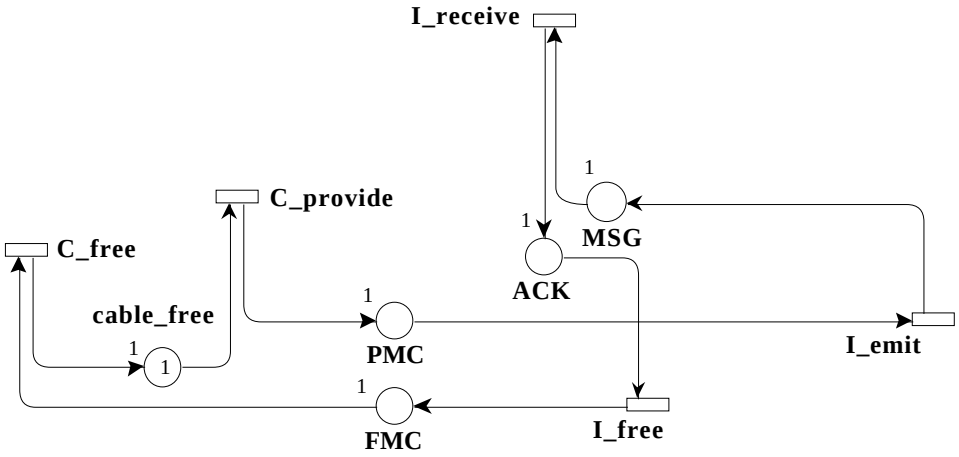
Answer: The first one corresponds to the status of the main cable from the synchronizer point of view (free/reserved).



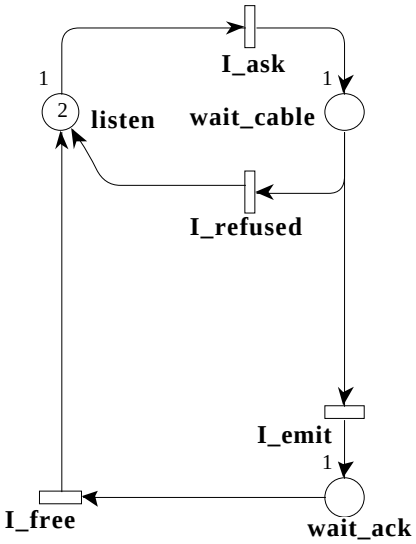
The second one corresponds to state of the main cable : Free, reserved, in use, to be released.



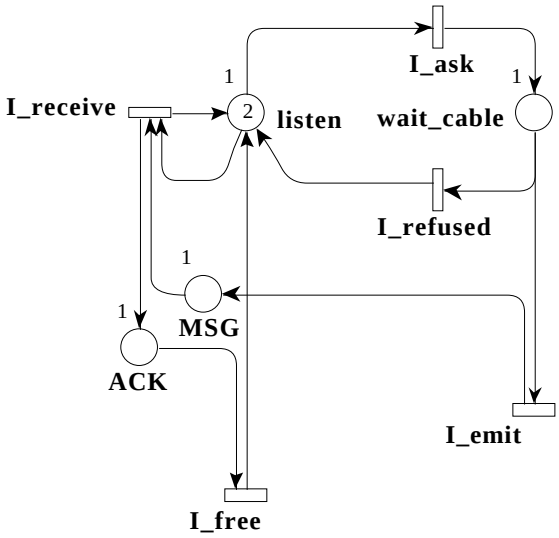
The third corresponds to the state of the main cable from the user point of view..



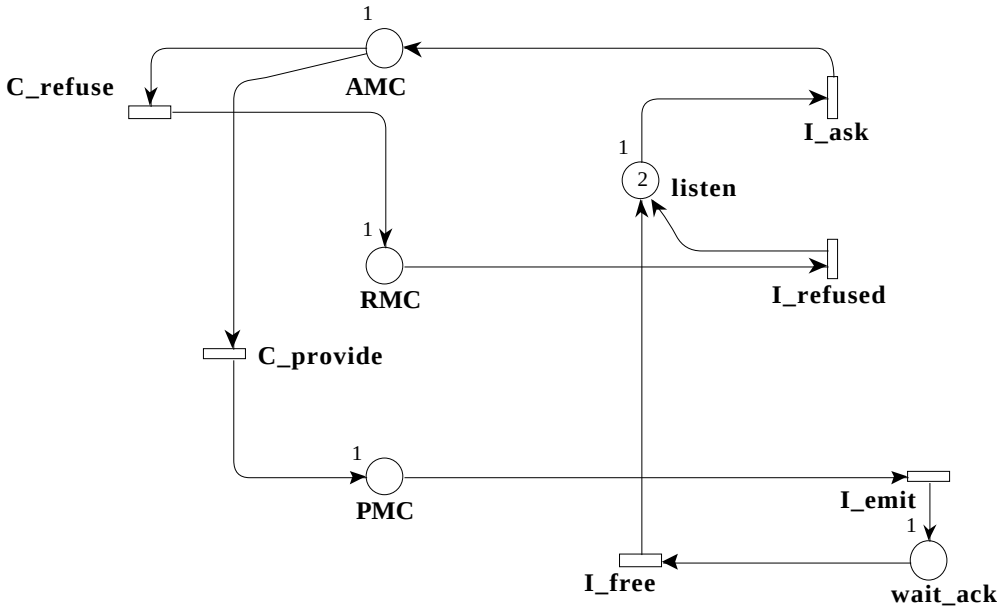
The fourth one corresponds to the automata an interlocutor.



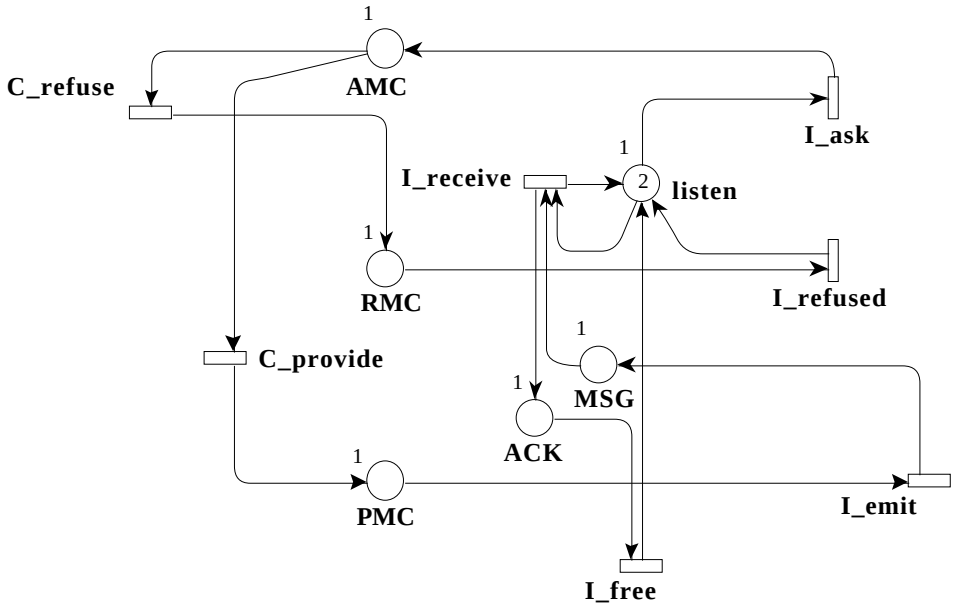
The fifth one is difficult to interpret.



This sixth one is difficult to interpret.



The seventh one shows the signal flow exchanged in the protocol (AMC/PMC/MSG/ACK/RMC or AMC/PRMC)



Question 4

Subtract P-invariants two to P-invariant three. What can you deduce? do you observe something similar with others couple of invariants?

Answer: Place wait_ack is implicit. Some invariants are thus replicated 2/3, 4/5, 6/7 these three couples are made of the same invariant, once with wait_ack, once without it.

Question 5

Is the net bounded? use the corresponding tool to check it.

Answer: The bound place service provides the following result :

cable_free :	[0 ... 1]
cable_used :	[0 ... 1]
FMC :	[0 ... 1]
AMC :	[0 ... 2]
wait_ack :	[0 ... 1]
wait_cable :	[0 ... 2]
listen :	[0 ... 2]
RMC :	[0 ... 2]
ACK :	[0 ... 1]
PMC :	[0 ... 1]
MSG :	[0 ... 1]

You may observ that RMC is 2-bounded. Using PROD, you easaly find a path that lead to this state. It occures when an interlocutor releases the line and ask for a ew line immediatly before the controller takes the FMC signal into consideration.

Question 6

Some properties are stated on the Safe Channel. They are listed below :

- **P1** There is only one electric signal at a time in the main cable.
- **P2** There can never be two interlocutors emitting in the main cable at the same time.
- **P3** If the interlocutors do not crash elsewhere, the protocol between components of the channel does not introduce communication problems.

Translate these textual properties in terms of Petri net properties to verify.

Answer:

P1: places MSG and ACK should never been simultaneously marked.

P2: Place wait_ack is 1-bounded.

P3: The Petri net is deadlock-free.

Question 7

What formal properties can you use to validate properties P1 and P2?

Answer:

P1: The third P-invariant (cable_free+FMC+ACK+PMC+MSG = 1) insures that the two places cannot be simultaneously marked.

P2: The tool computes that the place waits_ack is 1-bounded.

Question 8

Generate the reachability graph. Using model checking techniques (with PROD), try to verify the corresponding properties. You should use the expert mode and type your queries using the information provided in annex.

Answer:

P1: the query «query verbose node ((card(MSG)== 1) and (card (ACK) == 1))» provides no result.

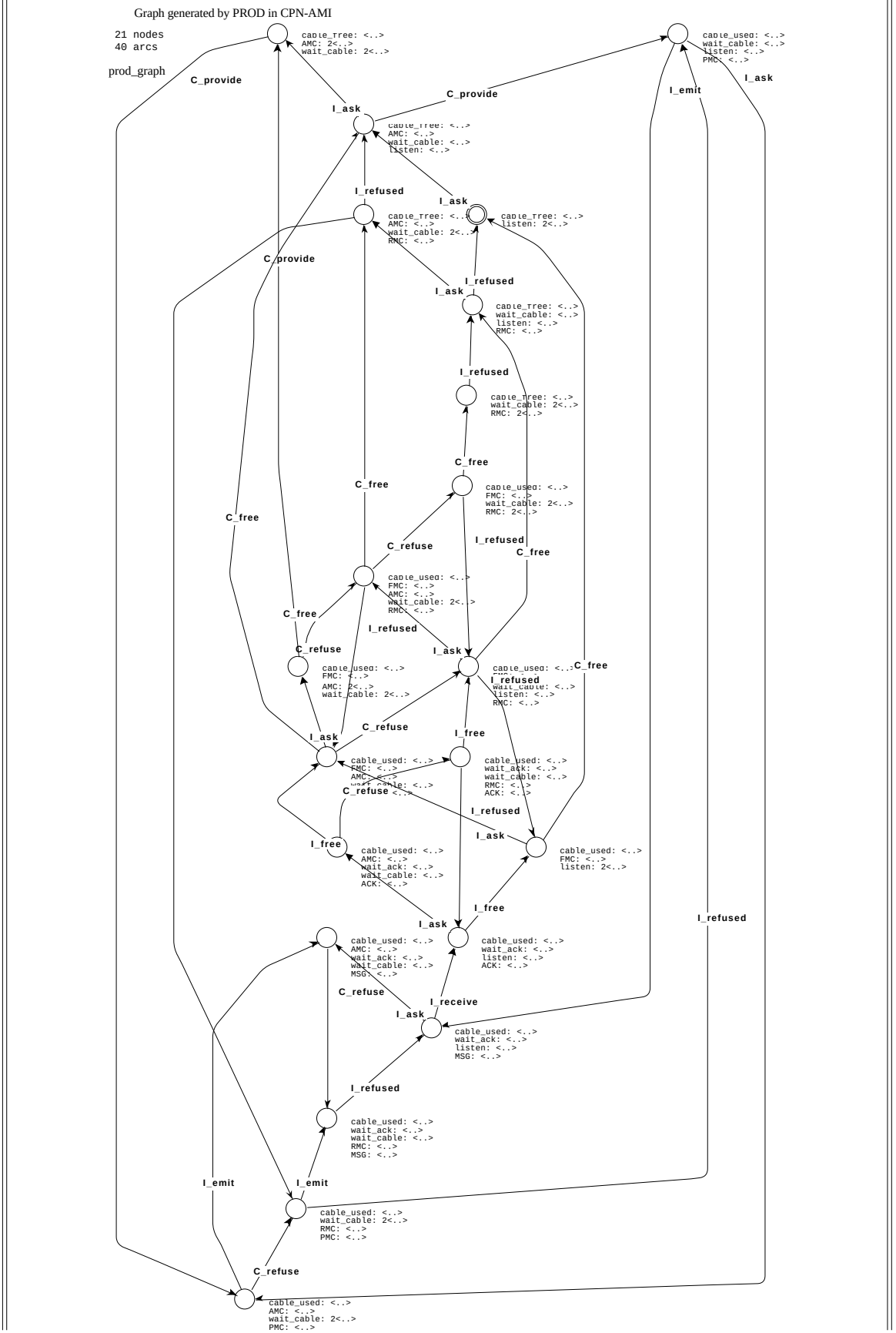
P2: The query.«query verbose node (card(wait_ack)> 1)» provides no result.

P3: PROD says that there is no terminal node.

Question 9

You can have a look on the Reachability graph if you want.

Answer: Here is the reachability graph computed by PROD and displayed with dot. The initial state is noted :



2. Managing loss in the main cable

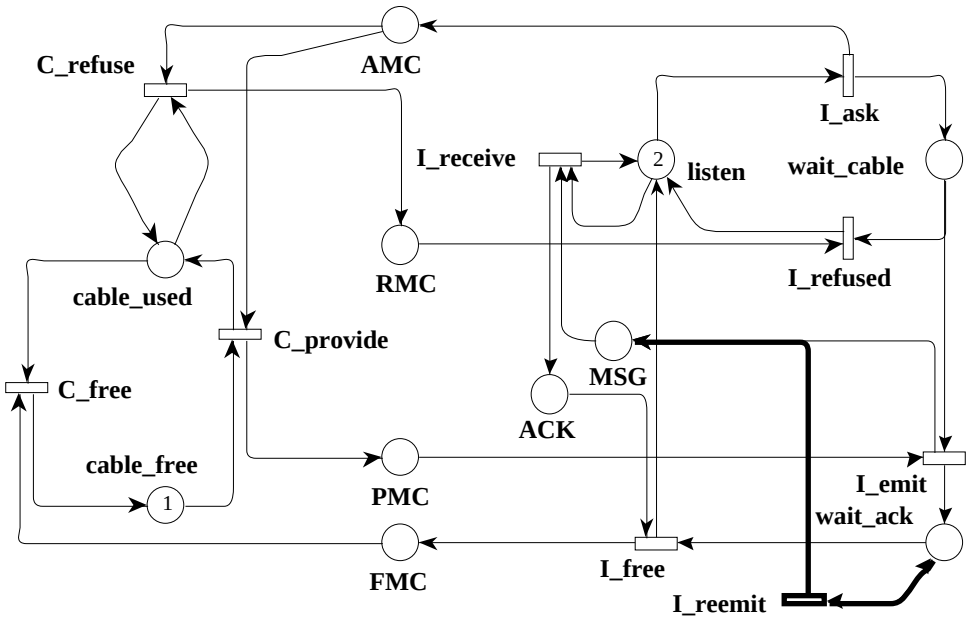
It appears that the 128 bit width main cable is not as safe as the 3 bits width control cables. Thus, the electric signal can be loss (i.e. not interpreted as a message by a

interlocutor). It is thus decided to introduce a time-out. After that time-out, the emitting interlocutor will send again his message.

Question 10

By adding one transition in the previous P/T net, model the fact that an interlocutor can re-emit a message in the main cable. So far, only consider the loss of messages (MSG signal).

Answer: The model is easy to modify, just add the transition I_reemit as shown in the model above



Question 11

Compute bounds of this model. What effect do you observe?

Answer: The model is not bounded any more... thus, the reachability graph should not be computed. The result of bound-places is:

cable_free :	[0 ... 1]
cable_used :	[0 ... 1]
FMC :	[0 ... 1]
AMC :	[0 ... 2]
wait_ack :	[0 ... 1]
wait_cable :	[0 ... 2]
listen :	[0 ... 2]
RMC :	[0 ... 2]
ACK :	[0 ... +oo]
PMC :	[0 ... 1]
MSG :	[0 ... +oo]

Question 12

What do you deduce about the reachability graph?

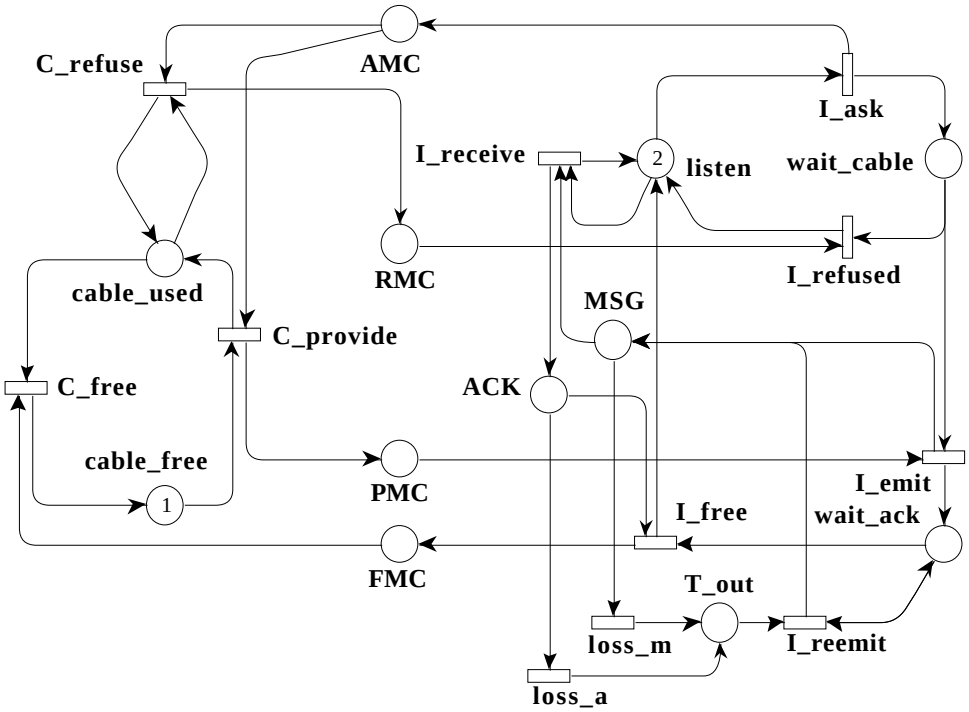
Answer: It is infinite. We should be advised not to compute it;-)

Question 13

Where is the problem, Modify the P/T net model to solve it. Please consider now all loss on the main cable (MSG as well as ACK). Like in TCP/IP, the emiter re-emit the message when te Acknowledge is lots. The receiver then handles it.

Important: *we assume that there is not "bad time-out" (i.e. a time-out is generated only if there is a message loss).*

Answer: The problem is due to the fact that we have modeled the possibility to re-emit a message but not in the appropriate condition (i.e. when this message is lost). We thus have to modify the model by generating errors (i.e. taking out tokens from MASG or ACK). You the get this model :



Question 14

Use PROD queries to list states of the reachability graph that correspond to a time-out generation. How many states do you identify?

Answer:

The PROD query is «query verbose node (card(T_out) == 1)» (i.e. a time-out is generated when place T_out is marked).

PROD tells that three states corresponds to time-out in the Graph. Here is what PROD displays :

```
Result from custom query:
PATH
Node 8, belongs to strongly connected component %%0
  cable_used: <..>
  wait_ack: <..>
  listen: <..>
  T_out: <..>
-------------------------------------------------
PATH
Node 13, belongs to strongly connected component %%0
  cable_used: <..>
  AMC: <..>
  wait_ack: <..>
  wait_cable: <..>
  T_out: <..>
-------------------------------------------------
PATH
Node 16, belongs to strongly connected component %%0
  cable_used: <..>
  wait_ack: <..>
  wait_cable: <..>
  RMC: <..>
  T_out: <..>
-------------------------------------------------
3 paths
Built set %1
```

Question 15

For each identified state, use PROD queries to get a scenario that leads to the generation of a time-out.

Answer: The PROD query is:

«goto <x>
query true
goto 0
query verbose bspan (true) %1»

where <x> is the identification number of the node. For node 8, it provides you the following result:

```
-----------------------------------------------
8#PATH
Node 0, belongs to strongly connected component %%0
  cable_free: <..>
  listen: 2<..>
Arrow 0: transition I_ask, precedence class 0
Node 1, belongs to strongly connected component %%0
  cable_free: <..>
  AMC: <..>
  wait_cable: <..>
  listen: <..>
Arrow 0: transition C_provide, precedence class 0
Node 2, belongs to strongly connected component %%0
  cable_used: <..>
  wait_cable: <..>
  listen: <..>
  PMC: <..>
Arrow 0: transition I_emit, precedence class 0
Node 4, belongs to strongly connected component %%0
  cable_used: <..>
  wait_ack: <..>
  listen: <..>
  MSG: <..>
Arrow 2: transition loss_m, precedence class 0
Node 8, belongs to strongly connected component %%0
  cable_used: <..>
  wait_ack: <..>
  listen: <..>
  T_out: <..>
-----------------------------------------------
```

It corresponds to the scenario “I_ask->C_provide->I_emit->loss_m”.

Question 16

To check if the protocol is safe despite message loss, it is stated that when a message is sent, an acknowledge has to be received. Can you express that using a PROD query and run it? Is this property respected?

Answer: The PROD query is:

«query verbose AG(IfThen(card (wait_ack) > 0, AF (card(FMC) > 0)))»

or

«query verbose AG(IfThen(card (wait_ack) == 1, AF (card(FMC) == 1)))»

If you run it, PROD finds no path. This means the property is NOT verified.

Question 17

Somebody suggests the problem comes from message loss that could generate infinite repetitive sequences of particular transitions. How can you detect such a sequence using structural properties? run the appropriate tool to demonstrate this idea

Answer: T-invariants should outline this problem. If you run GreatSPN, you get :

(1) loss_m + I_reemit

(2) I_receive + I_reemit + loss_a
(3) I_ask + I_refused + C_refuse
(4) I_ask + I_emit + I_free + I_receive + C_provide + C_free

Invariants (1) and (2) demonstrate the hypothesis if the corresponding firing sequences can be found (this is the case, you can use PROD to show that a reachable state may lead to this sequence).

Question 18

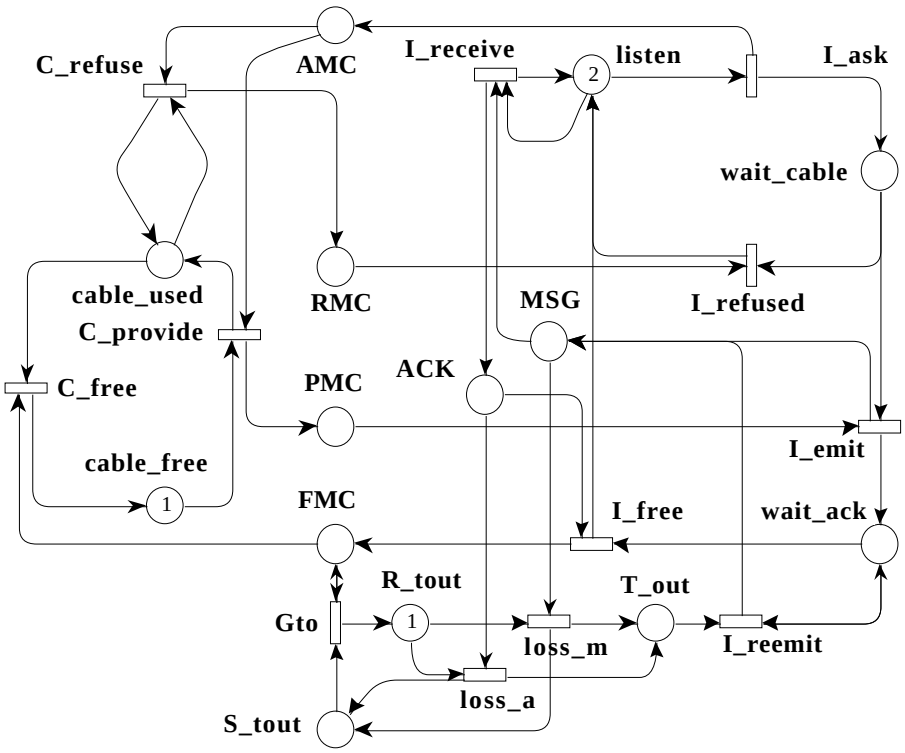
Can you provide an interpretation of these two invariants? what hypothesis can we reasonably suggest on the main cable?

Answer: It means that the protocol supports loss of messages on he main cable but cannot run when the line is broken (no message at all). We can suppose that the number of time-out is bounded (i.e. we suppose the line is not broken).

Question 19

Modify the Petri net model to handle that new hypothesis

Answer: The solution is provided in the model above.



We have introduced place R_tout which initial marking sets the maximum of time-out to be generated when a connection is initiated (here, one). Firing loss_m or loss_a consume a token from this place and thus another time-out cannot be generated. Transition Gto may then mark again place R_tout. Place S_tout is there to bound the reachability graph and limit the maximum number of Gto firings.

Question 20

Process the query of question 16. Is it now verified? Can you use a similar way to the one of question 17 to find an explanation?

***Hint**: have a look on the behavior of interlocutors.*

Answer: We proceed like for question 17 and compute T-invariants with GreatSPN. We get these ones :

(1) Gto + loss_m + I_reemit
(2) Gto + I_receive + I_reemit + loss_a
(3) I_ask + I_refused + C_refuse

(4) I_ask + I_emit + I_free + I_receive + C_provide + C_free

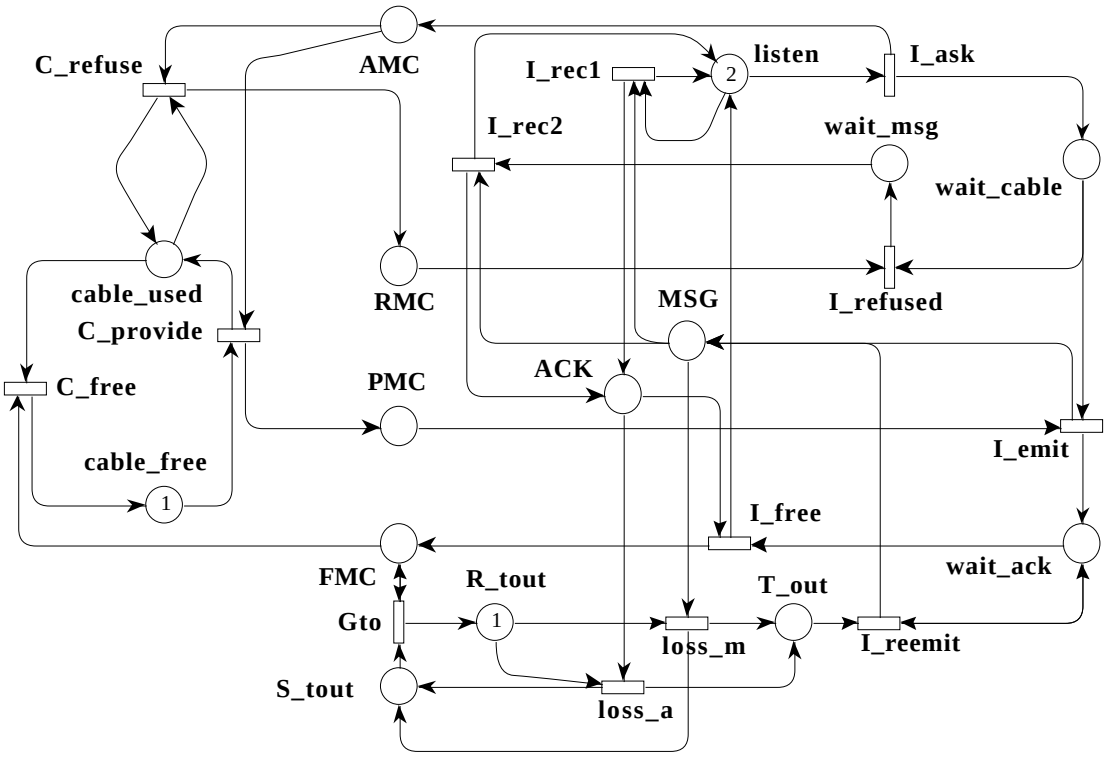
The invariant (3) already noticed and interpreted in question 2 shows that, even if the main cable can be freed, the Petri net may still loop on the T-invariant (3).

Question 21

Does that comes from the modeling of message loss on the main cable? Can you propose a modification on the previous model?

Hint: *you have to change the behavior of interlocutors by preventing this infinite loop.*

Answer: No. To be convinced, process the query of question 16 on the model. It is still not respected. It does not mean that it does not work but starvation may occurs because some point of the requirements have not been properly modeled : when the main cable is refused, the interlocutor has to listen and wait for a message. The new model is shown below



We introduces place wait_msg that is marked when the connection is refused by the controller. We must also duplicate transition I_receive (into I_rec1 and I_rec2) in order to be able to consume a message even if the listening interlocutor does not ask for the main cable.

Question 22

Use PROD to process the query of questions 16 and 20. Is it now verified?

Answer: Yes (ouf;-).

Question 23

Use now PROD to display statistics on the reachability graph What particular states can you observe?

Answer: The reachability graph contains two terminal states.

Question 24

Use PROD to identify path that lead to these particular configurations and propose a solution to the problem.

Answer: the firing sequence: I_ask->C_provide->I_emit->I_rec1->I_ask->I_free ->I_ask->C_refuse->I_refused->C_refuse->I_refused->C_free leads to the marking 2wait_msg+R_tout+cable_free. This marking is dead.

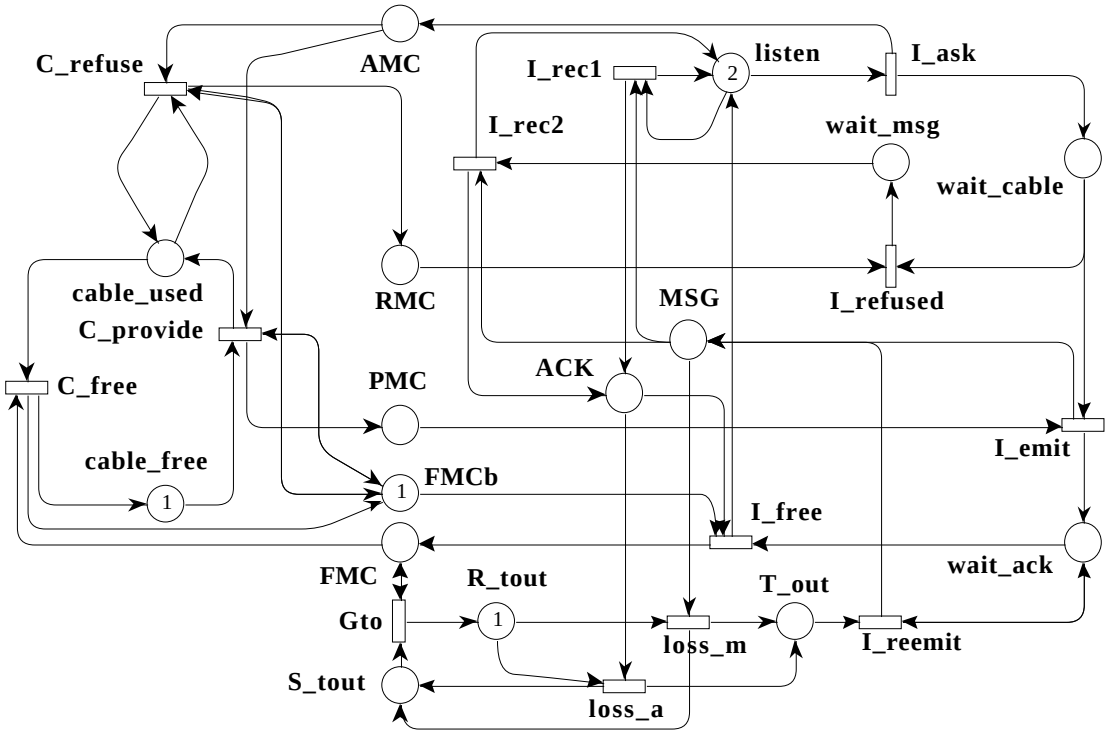
It corresponds to the case where the controller refuses two queries before taking into account the FMC message. Thus the two interlocutors are waiting for each other.

A solution should be to prevent the controller to answer queries when a FMC message is coming in (its like providing a higher priority to transition C_free).

Question 25

Modify the model in order to take into solve this problem.

Answer: The modified model is proposed above.



The simplest way to solve the problem should the use of an inhibitor arc. However, they are not supported by many tools in CPN-AMI and we have to find another way to express this modification.

Place FMC has been doubled into FMC (means that FMC is incoming) and FMCb. FMCb enables the firing of transitions C_provide and C_refuse. It is emptied when I_free is fired and filled again when C_free fires.

Question 26

Check the bounds of places with the appropriate service in CPN-AMI. Do you observe something suspicious? if yes, check with PROD and propose an explanation.

Answer: the service bounds of place provides :

```
cable_free : [0 ... 1]
cable_used : [0 ... 1]
FMC : [0 ... 1]
AMC : [0 ... 2]
wait_ack : [0 ... 1]
wait_cable : [0 ... 2]
listen : [0 ... 2]
RMC : [0 ... 2]
ACK : [0 ... 1]
PMC : [0 ... 1]
MSG : [0 ... 1]
```

```
T_out : [0 ... 1]
R_tout : [0 ... 1]
S_tout : [0 ... 1]
wait_msg : [0 ... 2]
FMCb : [0 ... 1]
```

It says that place wait_msg is 2-bounded where it should not be. However, PROD on the reachability graph finds no node (expert mode, the query is : «query verbose node (card (wait_msg)== 2)»).

The explanation is that structural bounds may not be reached in the reachability graph. They are an upper bound for places. It insure that the marking will never exceed the bounds provided but it may not reach all the proposed range.

3. Designing a bus

Having successfully designed the safe channel, we would like to use a similar strategy to design a safe bus that can connect up to N users. The architecture of this bus is similar to the one presented in section 1. Each interlocutor is connected to the controller using a dedicated control cable and have a unique identification number. Each one is also connected to the maim cable. When an interlocutor sends a message, it provides the identification of its corespondent. Acknowledge does not requires identification while only the sender is listening to it in the main cable. The model is similar to the one of the safe channel.

The Petri net of Figure 5 models such a behavior. Interlocutors are identified using the *It* class. To evaluate the design, we set the maximum of connected interlocutors to 4.

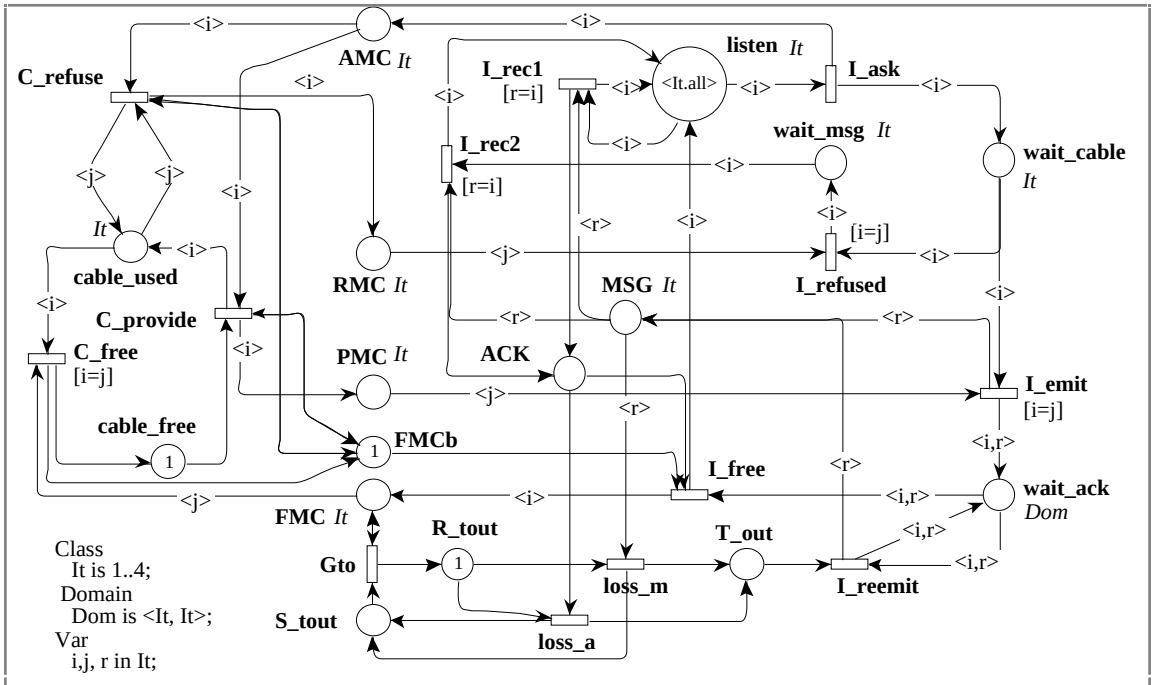


Figure 5: Petri net modeling the controller and four interlocutors.

Question 27

Generate the reachability graph for this model and have a look on the statistics (do not think one second at downloading it;-). Is the Petri net deadlock-free?

Answer: Here are the statistics provided by PROD :

```
Number of nodes: 6624
Number of arrows: 23360
Number of terminal nodes: 4
Number of nodes that have been completely processed: 6624
Number of strongly connected components: 769
Number of nontrivial terminal strongly connected components: 0
```

No.

Question 28

Use the tools to propose an explanation to what you observed in the previous question.

Answer: PROD proposes to view the list of dead markings. Here is the answer :

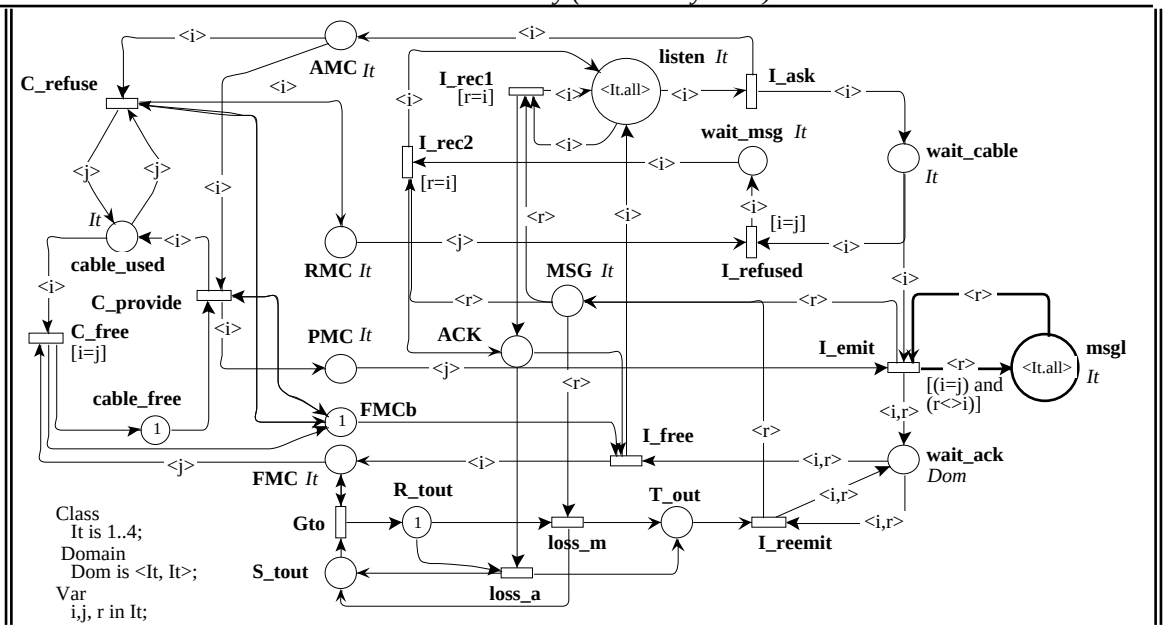
```
Node 6060, belongs to strongly connected component %%0
  FMCb: <..>
  wait_msg: <.2.> + <.3.> + <.4.>
  S_tout: <..>
  MSG: <.1.>
  wait_ack: <.1,1.>
  cable_used: <.1.>
-----------------------------------------------
Node 6065, belongs to strongly connected component %%162
  FMCb: <..>
  wait_msg: <.1.> + <.3.> + <.4.>
  S_tout: <..>
  MSG: <.2.>
  wait_ack: <.2,2.>
  cable_used: <.2.>
-----------------------------------------------
Node 6070, belongs to strongly connected component %%226
  FMCb: <..>
  wait_msg: <.1.> + <.2.> + <.4.>
  S_tout: <..>
  MSG: <.3.>
  wait_ack: <.3,3.>
  cable_used: <.3.>
-----------------------------------------------
Node 6075, belongs to strongly connected component %%220
  FMCb: <..>
  wait_msg: <.1.> + <.2.> + <.3.>
  S_tout: <..>
  MSG: <.4.>
  wait_ack: <.4,4.>
  cable_used: <.4.>
-----------------------------------------------
4 path end nodes
```

You may observe that, in all states, an interlocutor is sending a message to himself. He thus cannot get any answer.

Question 29

Propose a correction that solves the problem.

Answer: To prevent self sending of message, you may create a “message container” (here called msgl) and associate it with the appropriate condition in transition I_emit. Here is the modified model.



Question 30

How can you express in PROD that, if a client <i> sends a message to a given client j ≠ i, it has to get an acknowledge.

Answer: We can provide, for a given i, the set of queries that fit the different values for j. Here is an example for i= 1:

«query verbose AG(IfThen (wait_ack == <.1,2.>, AF (FMC == <.1.>)))
query verbose AG(IfThen (wait_ack == <.1,3.>, AF (FMC == <.1.>)))
query verbose AG(IfThen (wait_ack == <.1,4.>, AF (FMC == <.1.>)))»

For all queries, PROD finds that it is OK. While there is no deadlock. This property should be OK.

Question 31

Somebody is asking about the fairness of the protocol : it means that no interlocutor gets into starvation. Use PROD to demonstrate that it can happens.

Answer: if a client decides to send a message, it should send an AMC signal to the controller and then eventually get into the state FMC (he frees the main cable). The PROD query is :

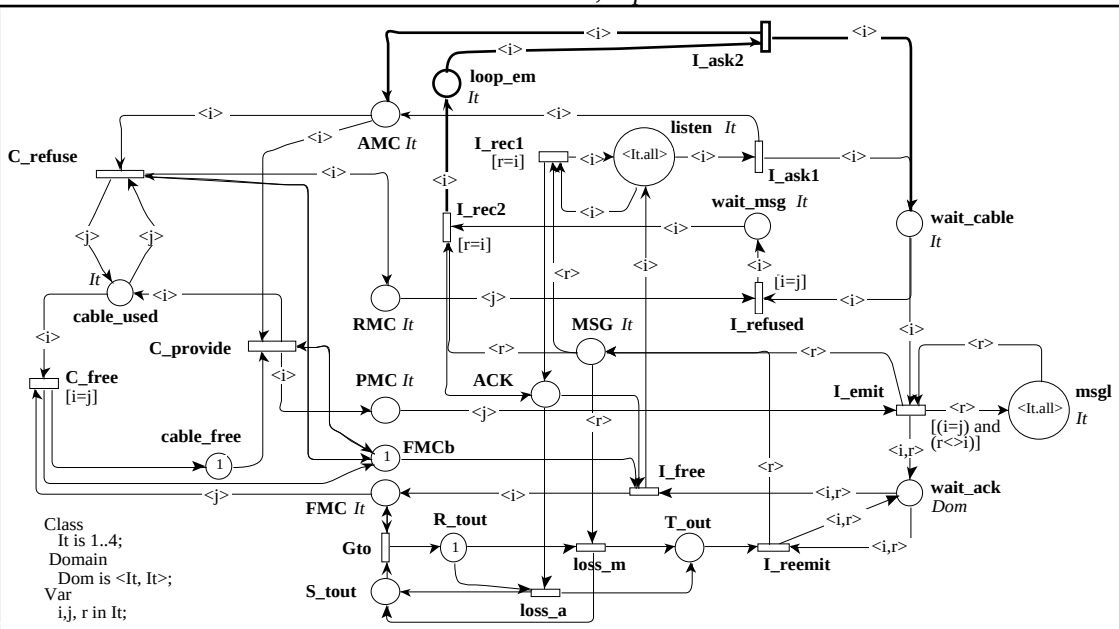
«query verbose AG(IfThen (AMC == <.1.>, AF (FMC == <.1.>)))»

The answer is NO. The process is not fair.

Question 32

It is suggested that interlocutors are not correctly modeled. When a request for the main cable is refused, it should systematically try to re-emit it (it may currently change its mind, which appears to be strange). Change the model to take into consideration this new aspect and apply the same query to prove that this protocol is really not fair.

Answer : You have to double the transition I_ask into I_ask1 and I_ask2. I_ask1 corresponds to the decision. I_ask2 correspond to systematic retries when the main cable is refused. Please note place loop_em as a postcondition of I_rec2.

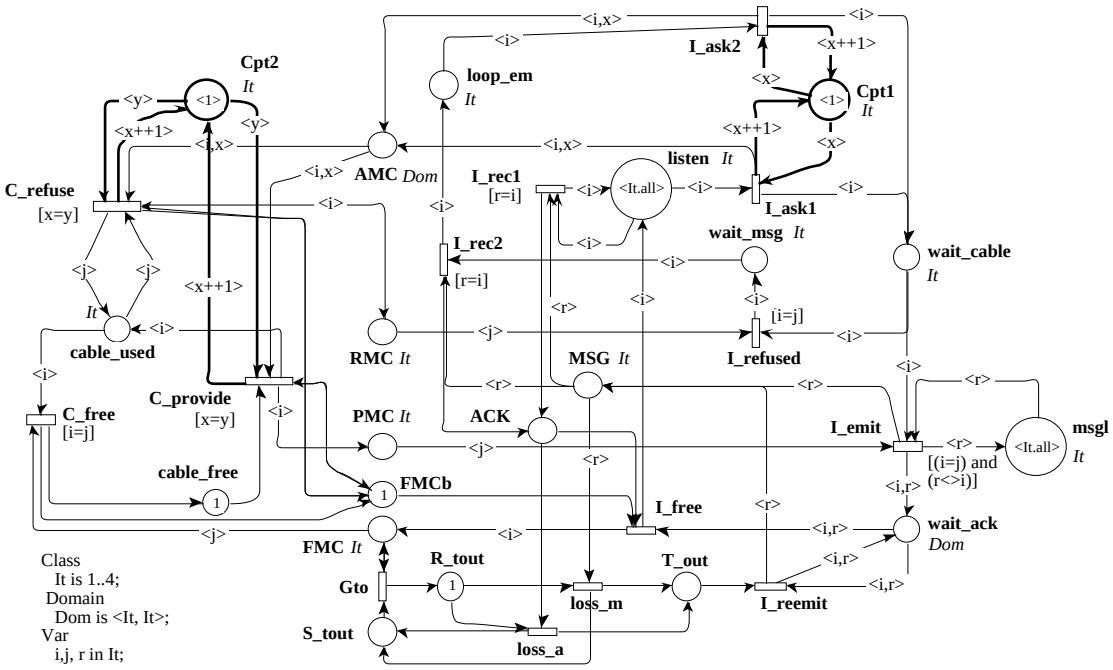


The model is not fair while the «query verbose AG(IfThen (AMC == <.1.>, AF (FMC == <.1.>)))» does not provide a path.

Question 33

To be fair, the controller must sort demands. Model this new controller behavior and apply the previous query to check the fairness of the system.

Answer: You add two places Cpt1 and Cpt2 that emulate a round robin strategy. Please have a look on the guard of transitions C_free and C_refuse.



In this model, the safe bus controller is safe.

Annex :

some information about the way to express PROD commands

The table above summarize some useful queries to play with PROD (expert mode). Please remind that all nodes in the reachability graph are numbered. The initial marking is node #0. When PROD results are displayed on the historic window, you get the identification number of the corresponding states in the reachability graph.

The "verbose" option provides you with detailed information. If you omit it, you will only get the identification number of nodes in the reachability graph.

PROD Query	Meaning
`query [verbose] node (`**`$1`** `=` **`$2`**`)`	provides all the states in the reachability graph for which place **$1** has marking **$2** . Remind that a non colored token is noted <..> in prod.
`query [verbose]node (card(`**`$1`**`) >` **`$2`**`)`	provides all the states in the reachability graph for which place **$1** has more than **$2** token(s). You can also use other operations (<, ==, >=, <=, !=).

`goto` **`$1`**
`query true`
`goto` **`$2`**
`query [verbose]bspan (true) %1`
	provides the shortest state between nodes **$2** and **$1** in the reachability graph.

`query [verbose]node ((`**`c1`**`) and (`**`c2`**`))`	provides all the states in the reachability graph that respect conditions **c1** and condition **c2**. You can do the same with or.
`query [verbose]AG`[4] `(IfThen (`**`$1`**`, AF`[5] `(`**`$2`**`)))`	Are all states in the reachability graph that respect condition **$1** eventually leads a states that respect condition **$2**.
`look` **`$1`**	Display the marking corresponding to node **$1** in the reachability graph.

Tokens in PROD are represented as follow :

- <..> for non colored tokens
- <.x.> where x is a value for colored tokens
- <.x_1,...,x_n.> for composed colored tokens (x_i are values of the appropriate color classes)

4 AG can also be written HenceforthOnAllBranches in some PROD configurations.
5 AF can also be writtent EventuallyOnAllBranches in some PROD configurations.