

A CPN model of the MAC layer

Pablo Hernández Morera, Tomás M. Pérez González
Departamento de Ingeniería Telemática
University of Las Palmas de Gran Canaria,
Las Palmas de Gran Canaria, Spain

e-mail:pablo@cma.ulpgc.es

Abstract

This paper describes the design and validation of a CPN (Coloured Petri Net) model for the MAC layer based on the IEEE 802.3 standard. The main purpose of this system is to teach the TE students at this university how Coloured Petri Nets can be used in the modeling and analysis of real systems (especially those based on concurrency), allowing them to extract net modeling and analysis techniques. Furthermore, it also aims to introduce Design/CPN as an adequate tool for the edition, simulation and validation of CPN models.

A bus LAN has been modelled. The model comprises a set of stations (each of them with the MAC layer implemented inside it) and a common channel which connects them all.

By means of this model it is aimed to analyse how the MAC layer allocates use of the shared communication medium among the competing nodes. The basic idea is to make the stations transmit packets in bursts to the common noisy channel using the CSMA/CD access method and analyse how they acquire the medium and recuperate from collisions. This access method where several stations may try to transmit their packets concurrently creates the perfect environment for the use of Coloured Petri Nets because of the natural way they include concurrency in their structure.

1 Introduction

This paper describes the designing and validation processes for the MAC layer of a bus LAN using the CSMA/CD access protocol. A previous work on modeling LANs using Petri nets can be found in [Goo87]. The Petri nets in this paper were created and analysed by Design/CPN tool [Ref93].

The basic purpose of the MAC layer is to allocate use of the shared communication medium among the competing nodes coordinating the communication among them by means of the CSMA/CD protocol [Jes94] [Tan] [Joh96]. According to this protocol, when a station generates a new packet, it first senses the broadcast channel to detect the presence of any ongoing transmission. If the channel is sensed idle, the packet transmission is started after a short delay called “interframe spacing”. If instead, the channel is sensed busy, the transmission is delayed until the channel becomes idle. During the transmission the channel is monitored to detect the presence of any interference from another transmitting station (collision). If a collision is detected, the transmission is immediately stopped, and the channel is jammed for a short time to make sure that all stations recognize the collision, and the packet transmission is rescheduled at some later time, after a random delay. By using a random delay, the stations which are involved in the collision are not likely to have another collision on the next transmission attempt. However, to ensure that this backoff technique maintains stable, a method known as *truncated binary exponential backoff* is used. In this method a station will persist in trying to transmit when there are repeated collisions. These retries will continue until either the transmission is successful or 16 attempts (the original attempt plus 15 retries) have been made unsuccessfully. At this point (if all 16 attempts fail) the packet is discarded.

The backoff strategy is quite simple. If a packet has been transmitted unsuccessfully n

times, the next transmission attempt is delayed by an integer r times the base backoff time (which is often chosen to be twice the end-to-end propagation delay). The integer r is selected as a uniformly distributed integer in the range $0 \leq r \leq 2^k$ where $k = \min(n, 10)$, that is, k is the minimum of the number of presently attempted transmissions and the integer 10. Thus as the load becomes increasingly heavy, the stations automatically adapt to the load.

The global model has been constructed taking advantage of the hierarchical CPN capabilities [Jen92][Jen94][Jen97a][Jen97b]. Hierarchical CPN allow the construction of a large model as a set of smaller models connected to each other using well-defined interfaces (substitution transitions). In this way, a complex model like this can be reduced to the generation of smaller models which solve certain functions in the global model into which they are integrated. The model aims to be as detailed as possible, i.e., it tries to collect the whole of the functions and aspects included in the MAC layer. The size of the resulting model is proportional to the adopted level of detail.

The main drawback of such a modeling is the risk of a state space explosion. This phenomenon makes difficult the validation of the model forcing to master the size and the level of detail and trying to find an equilibrium between these two opposite factors: detail and model size vs size of the state space and easy validation.

Since the system model is rather complex -its full net specification requires some 23 pages and a total of about 200 places- this paper merely attempts to cover some basic features of its construction and its validation, but it cannot be very specific on all details. In this sense, some of the component pages of the model have not been included and only main pages have been exhaustively explained.

The paper first introduces the nets structure naming all its parts. Each part is described breaking it down into the basic net components necessary to model it. Finally, the last part of this paper is related to the validation process and the several problems found to achieve it.

2 The CPN model

The following assumptions have been adopted in order to set the basic parameters of the system:

- *Size.* The network comprises six similar stations.
- *Burst arrivals.* In each station there is a LLC (Logical Link Control) user who generates packets in bursts. In this way, it is aimed to model the real behaviour of a network information flow where it is possible to find time intervals where no packets are transmitted and other ones where one or more stations may try to transmit a great volume of information.
- *Buffering.* Each node has a buffer containing packets to be transmitted. The buffer capacity has been limited to one packet in order to simplify the model.
- *Collisions and retransmissions.* Each packet which collides with another must be retransmitted. The maximum number of retransmission attempts for a given packet has been set to 16 (the original one plus 15 retries).
- *Noise.* The channel is not ideal and contains noise. This implies that packets are submitted to two different kinds of errors: those produced by the noise and those produced by

the collision among packets from stations which try to transmit concurrently.

- *Propagation delay*. The propagation delay between two adjacent nodes, Tr , has the same value for all adjacent node couples and it is small in comparison with the packet transmission time, Tp (all packets have the same length).

Figure 1 shows the page hierarchy graph for the bus network. It can be noted that the CPN model has a very modular structure. Among its pages it is possible to distinguish the page

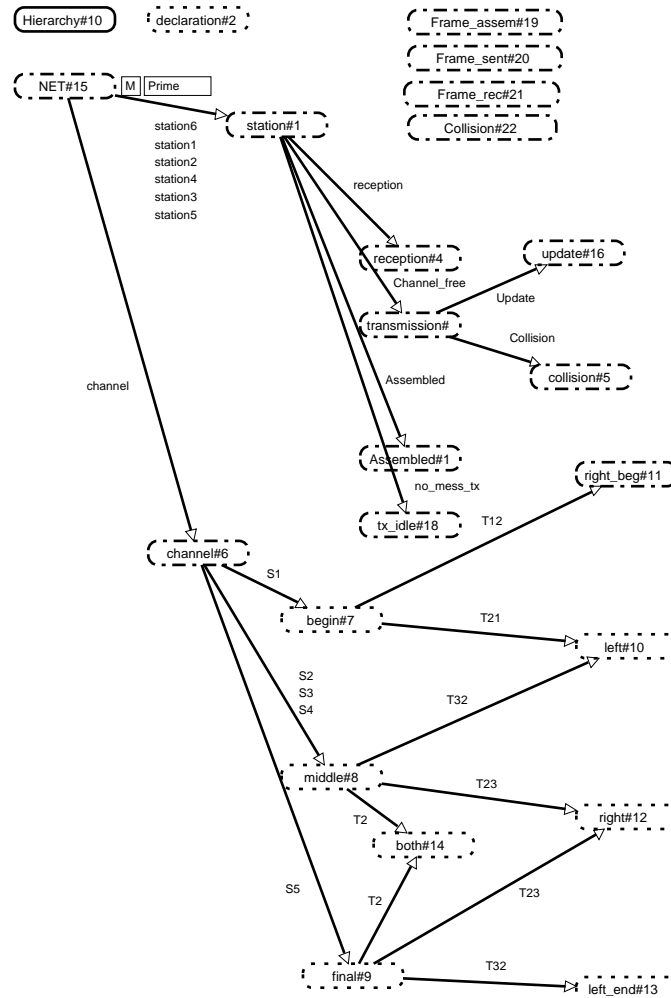


Figure 1. Page hierarchy graph.

NET#15, the prime page. Its content is shown in figure 2. According to the hierarchy, the rest of the net pages depend on it. This forces to a top-down analysis on its hierarchical structure. It comprises six similar stations connected to each other by a common communication channel. Each station is composed of one transition $station_x$ with $1 \leq x \leq 6$ where x is the identifying subindex for each station and three places associated to it: nx , Cx and $interfacex$. Each of these transitions ($station_1, \dots, station_6$) is related to the same subpage ($station\#1$). This means that the hierarchical net has six page instances of $station\#1$, each one with its own marking which is different from the other page instances markings.

As regard to the places:

- *Interfacex* represents the access point of the station x to the channel allowing the transmission and reception of packets. The color *Packet* associated to this place must be of the

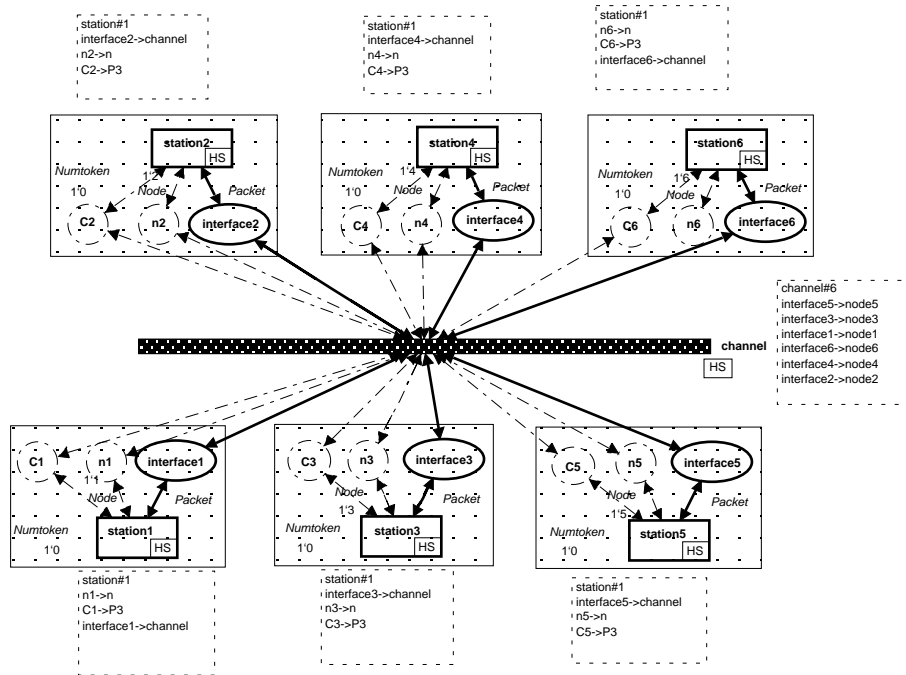


Figure 2. Page NET#15.

general form:

color Packet=product *State***Origin***Destination***CRC* timed;

with

color State=with *STX* | *MTX* | *ETX* | *JAM*;

color Node=int with 0..*maxnodes*¹;

color Origin=*Node*;

color Destination=*Node* declare *ms*;

color CRC=int with 0..1;

According to its declaration, *State* can adopt four different values: *STX* (Start Transmission), *MTX* (Middle Transmission), *ETX* (End Transmission) and *JAM* (collision signal), indicating the part of the packet that is being transmitted. *STX*, *MTX* and *ETX* refer to the head, data and end of the packet respectively. *JAM* is a general value to be used when the token which transports it does not represent a packet but a collision signal.

Origin identifies the station which generates the frame, *Destination* indicates the address of the destination station and *CRC* models the frame check sequence (FCS) associated to the information transported by the data field. The data field, as well as other component fields of the basic MAC frame have not been modelled. This is the case of the preamble, PAD, etc.

The word “timed” included in the declaration indicates that this is a timed color, i.e., the tokens belonging to this color will have a time stamp attached to them, describing the earliest model time at which the token can be removed by a binding element.

- *C_x* is a place whose associated token always indicates the number of tokens in the place

1. *Maxnodes* is a global constant indicating the maximum number of stations. 0 is a broadcast address.

interfacex of the station x . This token belongs to the color *Numtoken*, an integer type. Taking the existing tokens in each of the channel access places (and hence, of the channel itself) into account is a very important feature since it allows to know the number of packets being transmitted along the channel. In this way, it is easy to detect:

- The collisions in the channel. For a given station, a collision will be produced when the station is transmitting and the token in Cx has a value of two or more.
 - The idle channel. This condition must be fulfilled before the transmission starts, and this will happen when the token in this place is equal to 0.
- nx is a place associated to the color *Node* which always stores a token establishing the address of the station to which it is attached. This address is an integer number included in the range $0..maxnodes$.

As in any real bus network, the stations must be connected to each other by means of a common channel in order to communicate. This function is achieved by the substitution transition *channel* which is related to the subpage *channel#6* through the interfaces declared in its hierarchical inscriptions.

3 The channel model

The channel constitutes the physical device which interconnects all the stations in a network. Figure 3 shows the obtained model for the channel. It is made up of 17 places and 5 transitions.

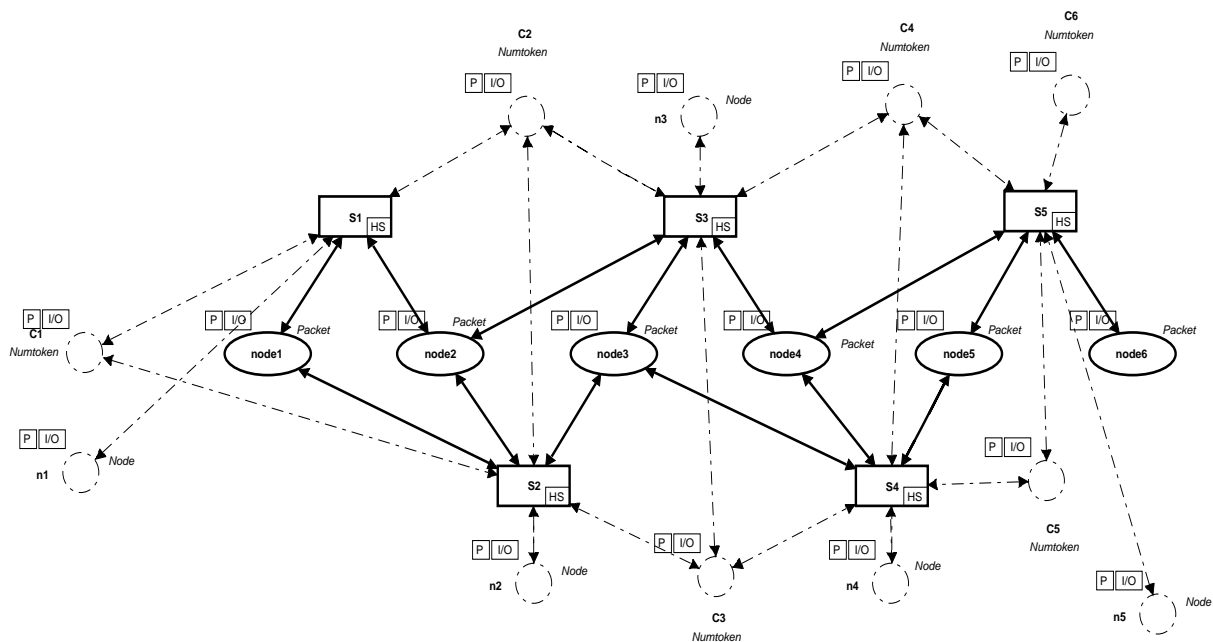


Figure 3. The channel page.

The channel is composed of six main places (*node1*, ..., *node6*) modeling the physical points of the channel through which the stations transmit and receive their packets (modelled by means of tokens of color *Packet*) interconnected among them via the transitions S_i with $i=1..5$ which also connect some control places taking account of the number of tokens in each of these main places and informing about the address of the station to which every one of these

main places is attached ($C1, \dots, C6$ and $n1, \dots, n5$ respectively).

The rest of the existing channel between two consecutive stations has been modelled bearing in mind its time parameters only, i.e., the distance between two consecutive stations has been transformed into propagation delays. This implies that the movement of the tokens from a place called *nodex* to another one, *nodex+1*, must be time controlled in order to get a realistic effect in the propagation process of packets along it. The transitions S_i is in charge of this, at the same time that they update the values of the tokens in the control places. These five transitions, S_i with $i=1..5$, are substitution transitions in such a way that the transition $S1$ is related to the subpage *begin#7*, $S5$ to *final#9* and $S2, S3$ and $S4$ to the subpage *middle#8*. The subpages *begin#7* and *final#9* model the extreme segments of the channel, while the subpage *middle#8* models its intermediate segments.

This channel structure as a set of three possible types of subpages, *begin#7*, *middle#8* and *final#9*, allows us to construct a channel with the desired size by only leaving intact the substitution transitions related to the subpages *begin#7* and *final#9* and their associated places and introducing between them as many places and substitution transitions related to the subpage *middle#8* as needed.

3.1 Propagation of a packet

Once the basic channel structure has been explained, a brief introduction about the packet propagation along the channel must be given. This includes an explanation not only about the propagation of those packets correctly transmitted, but also about their propagation in those situations where collisions have been produced. The way in which the noise affects the transmitting packets and how this fact is modelled by means of the CRC field will also be explained.

One of the way of achieving the propagation of a packet along the channel consists of making the transmitting station to deposit at each time interval (with the same value as the propagation delay between two consecutive nodes) a token which moves bidirectionally, at the same time that the previous tokens propagate simultaneously along the channel in their propagation direction. This process will continue until the transmission of the packet is completed. Figure 4 describes this process for a packet.

According to the described process, all the tokens which propagate along the channel don't need the element *State*, since they are all similar. Although this process is very realistic, it presents a serious drawback: the number of firings required to achieve the transmission of a packet is very high, slowing the simulation down and, at the same time, producing a very large state space. This last consequence leads to a disproportionate growth of the size of the occurrence graph due to the great quantity of firings required and hence, different reachable markings. Furthermore, as this process has been defined, the size of the occurrence graph depends, in a direct way, on the length of both the packet and the channel (as the number of stations per length unit remains constant, this is equivalent to affirm that it depends on the number of connected stations).

This is quite a serious problem, not only because of the simulation slowness but also because a large occurrence graph may be impossible to be analysed (by both visual inspection and standard queries), due to the limitation of RAM memory in our equipment.

The reduction of the state space has been achieved by making its size not to depend on the length of the packets but only on the number of stations connected to the channel (note that it is

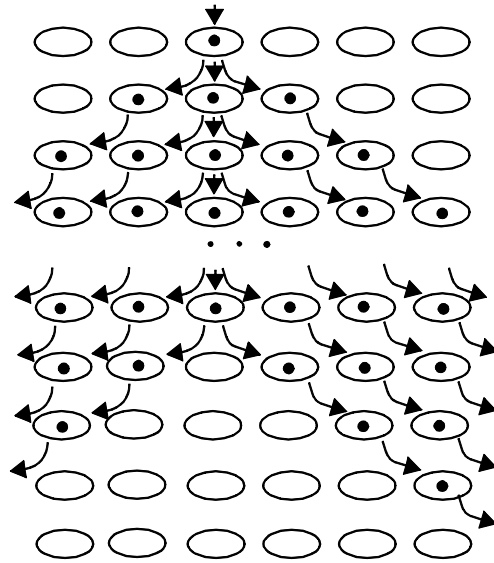


Figure 4. Packet propagation.

almost impossible to make the occurrence graph not to depend on this last factor). Now, firings must not be produced during the whole packet duration but only during the acquisition and liberation intervals, i.e., at the beginning and the end of the transmission of the packet.

Accordingly, it is necessary to distinguish between the beginning and the end of the packet. This is achieved by means of the element *State* in the color *Packet*. Therefore, it is important to emphasize that the introduction of the element *State* in the color *Packet* is not based on the MAC frame structure but on the obtained CPN structure for the channel which forces to differentiate between the beginning and the end of the packets in order to be able to achieve its propagation along the channel in an efficient way, minimizing at the same time the size of the state space.

This forces us to find a new way of performing the packet propagation along the channel. In this way, it is supposed that as soon as the station which wants to transmit senses the channel to be idle, it places in the channel a token with the form $(STX, x, y, 1)$, where *STX* represents the beginning of the transmission (the head is being transmitted), *x* is the origin address, *y* is the destination address and 1 indicates that the FCS is in accordance with the information carried by the packet. After a time period Tr (delay between two consecutive stations), this token will propagate in both directions to the adjacent places, leaving in the place it occupied a token with the form $(MTX, x, y, 1)$ which will remain in this place until the end of the packet is transmitted modeling in this way that in this part of the channel data is being transmitted.

Starting from this moment tokens with *State*=*STX* will move sideways in their direction of propagation leaving in the places where they move from tokens with *State*=*MTX*. When these tokens arrive to the end of the channel, they are lost.

The channel remains untouchable with all its places occupied by tokens with *MTX* until the end of the packet is being transmitted (*State*=*ETX*). In this moment the token with *MTX* placed in the place attached to the transmitting station is consumed and a token with *State*=*ETX* is added to it. This token will propagate in both directions leaving the places it comes from empty, modeling in this way the progressive channel freeing once the packet transmission is ended. This process will continue until the channel becomes idle. As an exam-

ple, figure 5 shows this process for a packet.

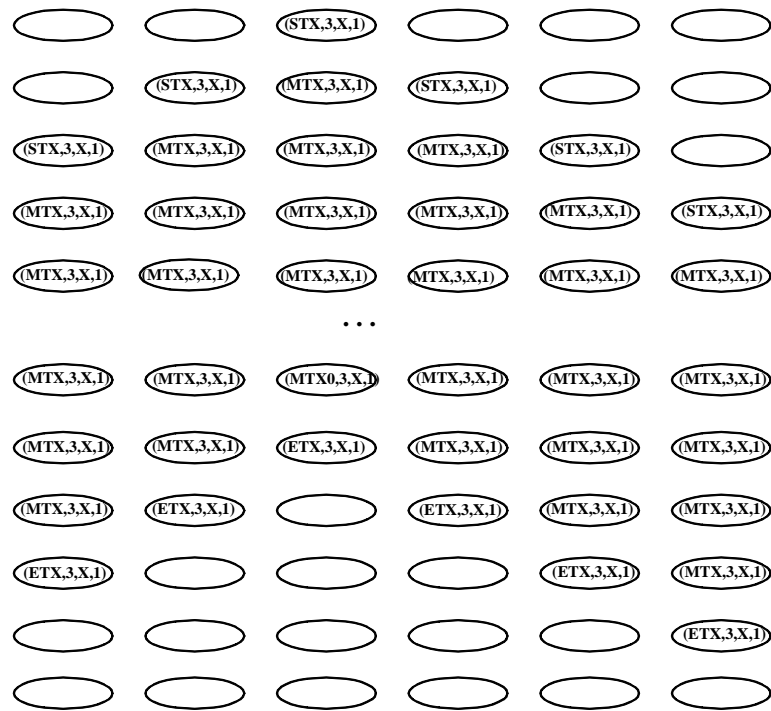


Figure 5. New packet propagation.

This form of propagation minimizes the state space size since it limits the transition firings in the channel to the time intervals needed to acquire and free it. Furthermore, there is another important reason justifying the introduction of *State* in the color *Packet*: it facilitates the modeling of the frame reception in the station reducing it to only detect the first and last tokens of a transmitting packets as it will be later explained.

3.2 Noise effect

As it has been already said, the channel is not ideal and the presence of noise may affect the packets. To model this effect it will be supposed that there is a probability of packets to be affected by this phenomenon and that all affected packets will be discarded. It is only left to be defined the way in which noise will be modelled in the net.

Although any part of the packet may be affected by noise (head, data or end), in order to simplify the model, it will be supposed that such effect will always be noticed at the end of the packet (token with *State*=ETX). It is equivalent to affirm that the probability of any packet to be affected by noise is condensed in its final part in order to be easily detected by the transitions in our obtained model.

A transmission will be considered to be noise free if the FCS received by the receptor during all the transmission is always equal to 1. If its value changes from 1 to 0 it will be supposed that it has been affected by the noise and it will be thrown away.

3.3 Propagation of several packets simultaneously: collisions

Once a packet propagation has been defined, it is only left to establish the way in which col-

lisions are signposted and how the jamming signal (*State=JAM*) will propagate along the channel.

When two or more stations transmit their packets simultaneously to the channel, they will propagate in accordance with the method established in section 3.1 until the moment in which their signals reach the other transmitting stations. In this moment the stations detect the collision, stop their packet transmission and transmit to the channel a jamming signal with the form (JAM,x,0,1) where JAM indicates collision, x is the origin address and 0 is a broadcast address. The collision signal will propagate along the channel.

4 The station model

Figure 6 shows the obtained model for the MAC layer of the station (page *station#1*).

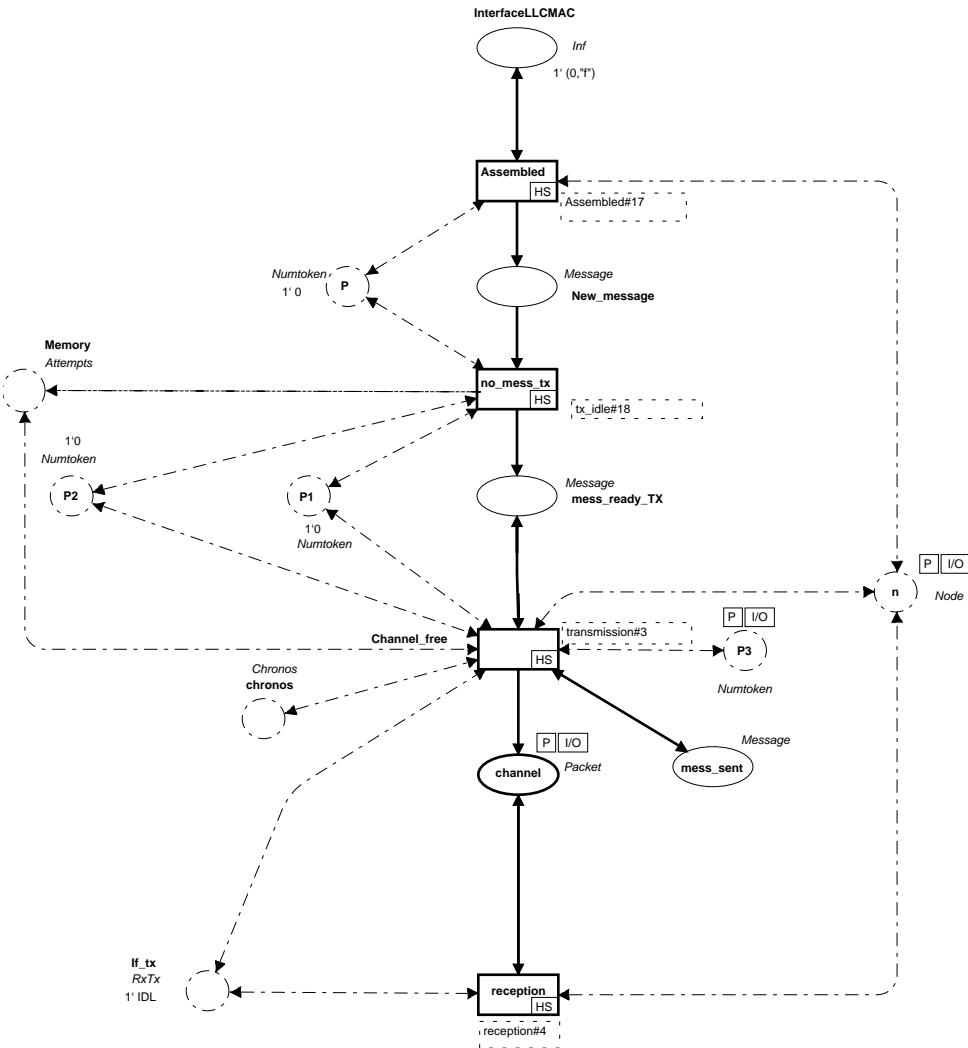


Figure 6. The station (station#1).

Information being passed from the LLC layer to the MAC layer can be found on the place *InterfaceLLCMAC* which acts as an interface between these two layers. This place stores a token of color *Inf* whose declaration is of the general form:

```
color Times=int with 0..5;
```

$color\ Prev = string;$
 $color\ Inf = product\ Times * Prev;$

Their interpretation will be discussed later.

Each firing of *assembled* (related to the subpage *Assembled#17* whose content is shown in figure 7) firstly generates a new message which is stored in the place *New_message* that acts as

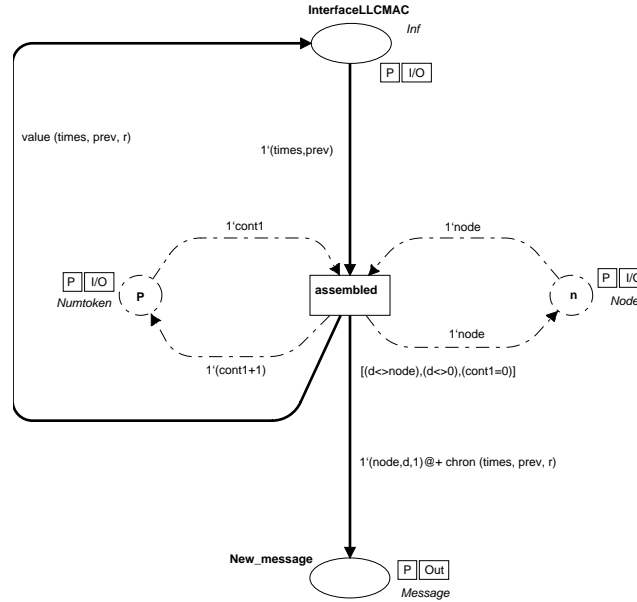


Figure 7. Subpage Assembled#17.

the system buffer with capacity for one message, secondly updates the token in place *P* which acts like a counter of the number of tokens in *New_message* and thirdly sends back to *InterfaceLLCMAC* a token whose value is determined by the function *value*.

Since the buffer capacity is limited to one message, this transition will only fire when the buffer is empty, i.e., the token in *P* is equal to 0.

The first element of the new message is an origin node number which is extracted from the information carried by the token in place *n*. The second element identifies the destination station. Its value is fixed by the variable *d* which does not appear in the input arc inscriptions. This implies that *d* may adopt whatever value of its corresponding type, i.e., from 0 to *maxnodes*. A guard $[(d \neq node), (d \neq 0)]$ has been attached to the transition in order to prevent this variable to adopt the same value as the origin address (a station is not allowed transmit a packet to itself) or the broadcast address (0, reserved for collisions). During the simulation, the simulator will provide this random value automatically. The third element will be fixed to 1 to model the fact that the FCS is in accordance with the information carried by the assembled frame.

This frame generation must follow a burst function. This is done using the information carried by the token in the place *InterfaceLLCMAC* and the time stamp attached to the token added on the place *New_message*.

From a mathematical viewpoint this function has been obtained by combining three different random functions: two negative exponential distribution functions whose intensities, n_1 and n_2 (with $n_1 \gg n_2$) set the time separation among frames and a third function *r* with a uni-

form distribution determining the pass from one exponential distribution function to another. Figure 8 shows the mathematical function structure, where p is the probability for generating packets with intensity n_1 and $1-p$ with intensity n_2 .

In order to obtain a noticeable burst effect, this last variable will be examined each five generated messages. This account will be taken by the first element of the token in *InterfaceLLC-MAC* (*times*).

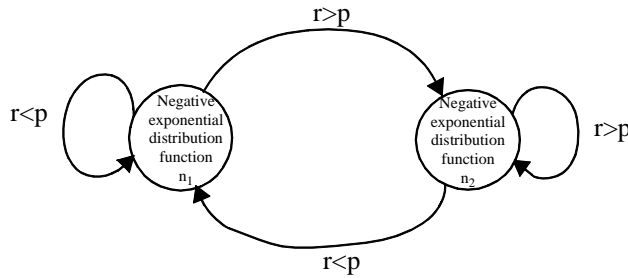


Figure 8. Mathematical structure.

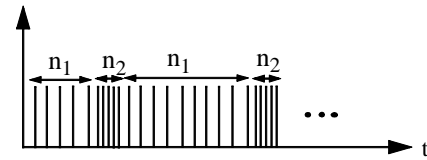


Figure 9. Frame generation.

The random variable determining the pass from an exponential distribution function to the other is the variable r we can find in the output arcs of the transition *assembled* arriving to the places *New_message* and *InterfaceLLCMAC*. Its value is automatically fixed by the simulator in each occurrence of *assembled* and together with the variables *times* (indicating the number of generated frames since the last inspection of r) and *prev* (indicating the exponential function which is being used) fix the time stamp for the token added in *New_message* by the function *chron* and the token added in *InterfaceLLCMAC* fixed by the function *value*.

Chron was implemented using random number generators. The distribution functions used have been implemented on the top of the new random function. This random function gives samples from a standard uniform distribution. The different distribution functions are then implemented by applying different procedures to this random function. In this way, the negative exponential function was extracted from [The97] and used to obtain our functions.

Once the new frame is generated, it will remain in *New_message* until no previous frame is trying to be transmitted. In this sense, the transition *no_mess_tx* in page *tx_idle#18* (figure 6) inspects the stored tokens in *P*, *P1* and *P2* only firing in the case that *P* has a token with value 1 (indicating the existence of a message in the buffer) and the places *P1* and *P2* (which take account of the number of tokens stored in the places *mess_ready_TX* and *mess_sent* respectively) have their respective token with value 0, indicating that the previous message (if existed) has already been transmitted properly or that the maximum number of transmission attempts has been reached and the packet has been thrown away.

Mess_ready_TX stores packets trying to be transmitted and *mess_sent* is a buffer where a copy of the transmitting packet is stored in order to prevent a packet loss when a collision occurs.

The firing of *no_mess_tx* passes the token from *New_message* to *mess_ready_TX* and adds a token with value 1 to *Memory* to indicate that this is the first transmission attempt. *Memory* models a counter of the number of transmission attempts. The message will remain in this new place until the channel is sensed idle. When this happens, the transmission is started after a short delay called “interframe spacing”. This process is achieved by the substitution transition *Channel_free* whose associated subpage is *transmission#3* shown in figure 10.

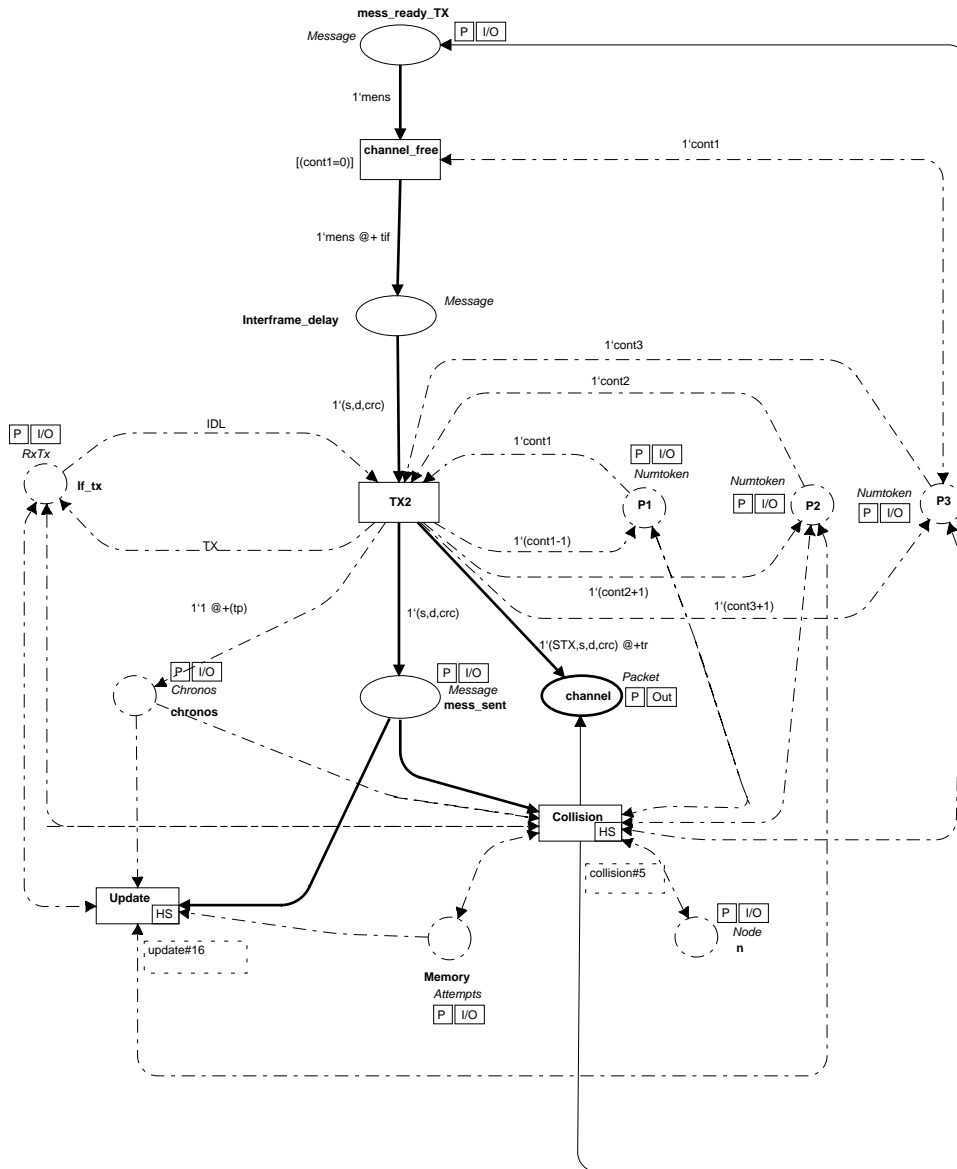


Figure 10. Subpage transmission#3.

In this page, *channel_free* inspects the place *P3* whose token indicates the number of tokens in the place *channel*, the station access point to the channel. As soon as *channel* has no tokens, this transition will fire passing the token from *mess_ready_TX* to *Interframe_delay* with a time stamp “interframe spacing” units (*tif*) greater than the simulated time at which the transition fired. After this time interval, *TX2* is enabled and it fires starting the packet transmission by storing the packet in the place *channel* and adding a token with value *TX* on the place *if_tx* to indicate that the station has passed from being idle (*IDL*) to be transmitting. It also adds a copy of the transmitting message to the place *mess_sent* and stores on the place *chronos* a token with a time stamp “packet length” (*tp*) greater than the current simulated time. This last place acts as a clock indicating the simulated time at which the transmission ends.

The place *if_tx* models the state of the station: whether it is idle (*IDL*), transmitting (*TX*) or receiving (*RX*). Its token ensures the mutual exclusiveness between the transmission and reception processes. In this way, any of these processes is only allowed to be achieved when the system is idle (*IDL*).

Notice that a copy of the transmitting message has been stored on the place *mess_sent*. This is necessary for the retransmission of such a message in the case that a collision is produced.

Once the transmission is started, two possibilities may occur:

- The packet is properly transmitted.
- A collision is produced and the transmission is retried if the number of attempts is smaller than 16.

In the first case, the only thing left to do to start the transmission on a another frame is to erase the copy of the transmitted message stored in *mess_sent*, update the value of the token in *P2*, remove the token stored in *Memory* and set the state of the station to idle (*IDL*).

The removal of such a copy must be time controlled, i.e., it has to be removed after ensuring that the transmission has ended without collision. That's what we use the token on the place *chronos* for. As soon as the global clock reaches the same value as the time stamp of the token stored on it, the transition *update_p2* in page *update#16* is enabled. That subpage is shown in figure 11.

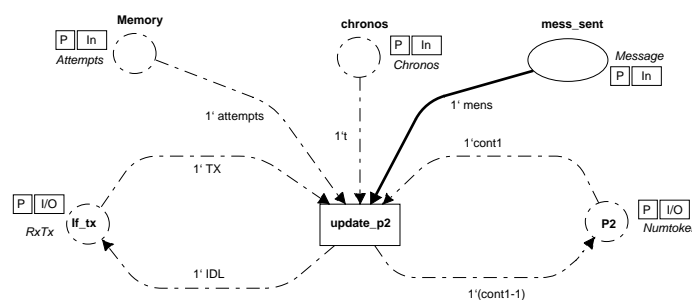


Figure 11. Subpage *update#16*.

In the other case (collision) the transition *collision* in subpage *collision#5* will fire. Figure 12 shows its subpage. A collision is produced when two or more stations try to transmit their packets concurrently through the channel. Since at every moment the number of tokens in each of the places of the channel is taken in account, a collision will be produced when in the access place to the channel related to a transmitting station there are two or more tokens. The detection of such a situation is performed by the transition *collision* which inspects the place *P3*.

Its firing immediately stops the transmission and lets the station idle (*IDL*). It also removes the token in the place *chronos* to avoid a later firing of *update_p2*, adds a token with *State* equal to *JAM* on the place *channel* in order to inform the rest of the stations about the collision and stores on *inter* a token with value null that does not carry information and that can be interpreted as a signal sent to the transition *top_attempts* to force it to fire.

As soon as the collision is detected and the jamming signal is transmitted, the only thing left to do is to investigate whether the message will be transmitted again or it is thrown away because of having overtaken the maximum number of transmission attempts. *Top_attempts* inspects the number of attempts stored on *Memory*. If this number is smaller than 16, the message copy is passed from *mess_sent* to *mess_ready_TX* and the number of attempts is increased by one unit.

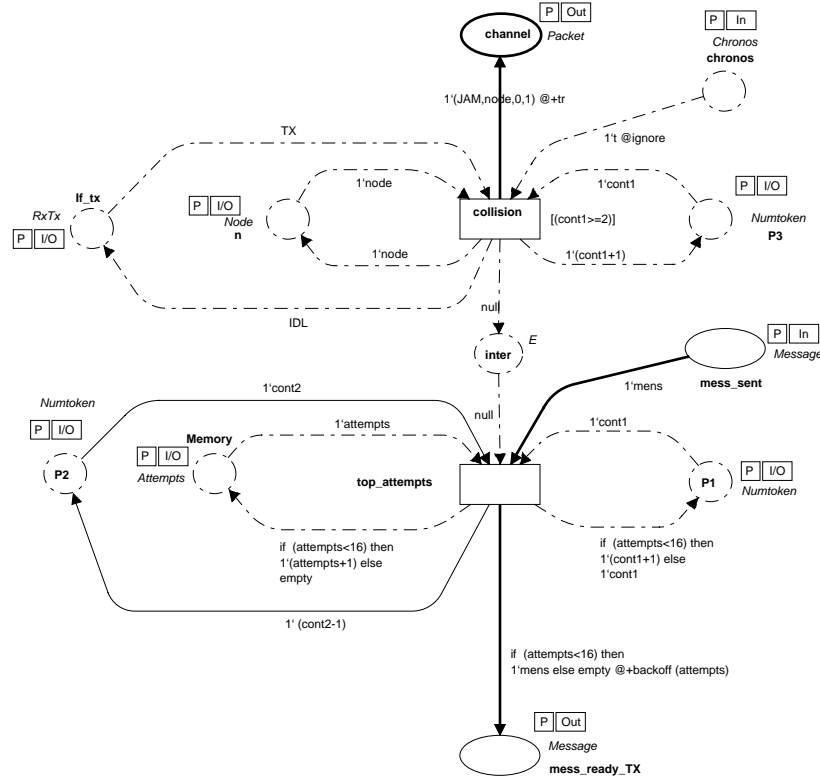


Figure 12. Subpage collision#5.

Before trying to transmit this copy again, it has to wait for a random delay fixed by a function called *backoff*. This is done by giving to the token added to *mess_ready_TX* a time stamp fixed by the function *backoff* which is of the form:

```

infix **;
fun ((x:int)**0)=1 | (x**y)=x*(x**(y-1));
fun backoff (n:attempts)=(real(CPN'randint(0, 2**(min(n,10)))))*ts;

```

where *ts* is the time slot value.

In the case that the number of attempts is equal to 16 both the copy of the message stored on *mess_sent* and the token in *Memory* will be erased.

The modelled station not only takes charge of the transmission process but also the reception one. This is performed by the substitution transition *reception* in figure 6 whose subpage is shown in figure 13. The reception of a frame starts when an idle station detects in the channel the head of a packet (a token with *State* equal to *STX*) appointed to it (*Origin* is equal to the station address). The detection of such a token is performed by *RX1* whose occurrence sends back to the channel a similar token as the consumed one and stores the origin direction on the place *Store_orig*. After this process the station has started to receive a packet, that's why the state of the station is now *RX*.

The station will continue receiving a packet until the transition *RX2* detects a token appointed to it from the same origin station as the first one indicating end of packet (*State=ETX*) or collision (*State=JAM*). If the token is a jamming signal (*State=JAM*), the station ceases from receiving and remains idle. On the other hand, if the received token is an end of packet indicator (*State=ETX*), it is supposed that the packet has been totally received. Since the channel is noisy, the receiving station analyses the FCS (*CRC*). If its value is 1, it is sup-

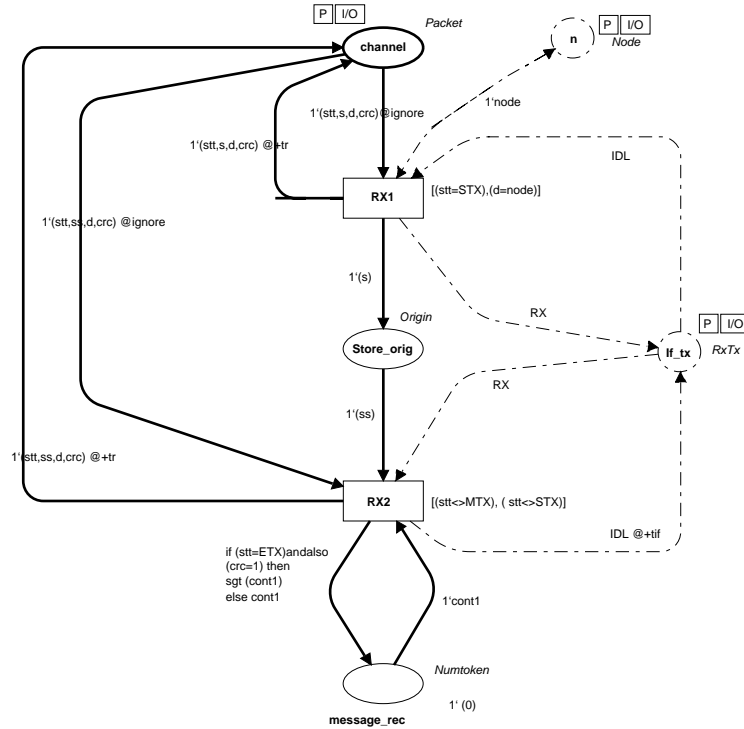


Figure 13. Subpage reception#4.

posed that the received packet is noise free and its information is passed to the LLC layer modelled by the place *message_rec* which also counts the number of well received packets.

In both cases the occurrence of *RX2* stops the reception and lets the station idle. This last fact is modelled adding to *if_tx* a token with value *IDL*. Its time stamp is *tif* units greater than the simulated time to avoid from receiving another packet until this period of time has passed.

5 Validation of the MAC layer

The validation is the work of proving that the obtained model reproduces the real system behavior. The obtained model may contain errors or be incomplete in the sense that it does not cover all the possible situations through which the real system may pass. It is then necessary to verify the model in order to guarantee it is error free and without any kind of anomalous properties.

To verify the model it is necessary to identify first all the properties it has to comply with. After that, it must be proven that the obtained model satisfies them.

Design/CPN supports two different analysis methods to verify the CPN model performance: simulation and occurrence graph analysis.

5.1 Simulation of the model

Simulation is an important instrument for debugging and validating CPN. It gives the developers an improved understanding of the system behaviour.

During the construction of the model, simulation was intensively used. In its early phases, manual simulations were made. This kind of simulation was applied to the individual model

subpages in order to test their performance. Its application helped to detect some errors allowing us to dig down in the CPN model to locate and fix them. In this sense, an iterative approach was used, alternating between modeling and simulation.

The first prototype was gradually refined, and eventually it constituted the final model. Since manual simulations became time cost due to the model size, automatic simulations were achieved in order to check the correctness of the final model. In this last kind of simulation mode it is only possible to know the initial and final markings and the fired transitions sequence (stored in a text file), but it is nearly impossible to know the intermediate markings of the model. It was necessary to find a way of knowing what happens in the net during simulation and this was made using the report facilities.

Report facilities allow the user to apply standard graphical routines to create and manipulate graphical representations of the simulation results. With their help it is possible to obtain graphics relating the occurrence of certain events in the net with the simulated time in which they occur. In this sense, some code segments updating graphics were attached to certain transitions. Analysing the obtained graphics and bearing in mind the initial and final markings, it was easy to establish whether during the simulation the model worked properly.

As an example, figure 14 shows one of the obtained graphic for the packet generation system. It describes the bursts of packets generated by the LLC user.

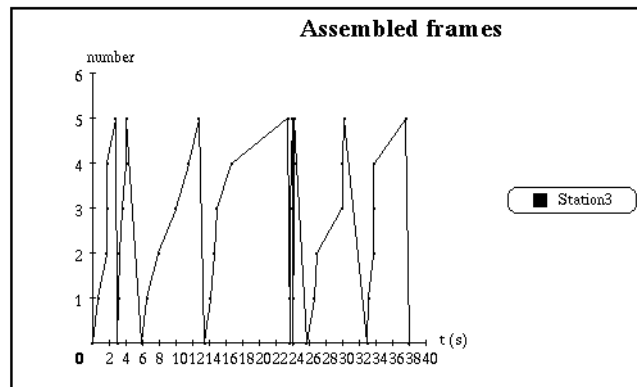


Figure 14. Bursts generation.

Although simulation is extremely useful for the understanding and debugging of a CPN, it is obvious that, by means of simulation it is impossible to obtain a complete proof of its dynamic properties. This forced us to analyse the model using occurrence graphs.

5.2 Occurrence graph analysis

At the end of the design phase, occurrence graph analysis was applied in order to detect as many errors as possible and prove that the obtained model fulfilled some dynamic properties (reachability, boundness, liveness, etc). The basic idea behind occurrence graphs is to construct a graph containing a node for each reachable marking and an arc for each occurring binding element. Obviously such a graph may become very large and in many cases, infinite.

In our case, despite of the adopted decisions in the modeling process in order to minimize the occurrence graph size, it was quite too large to allow its analysis via visual inspection (the obtained graph for a 10 seconds analysis has a partial status and more than 10.000 nodes).

The great size is due to:

- The great quantity of different actions that may occur in the network: each station is able to send packets to the rest of the stations. At the same time, these packets may collide among them a number of times, lose because the maximum number of transmissions has been reached or because of noise, transmit properly, etc.
- The inclusion of time in a cyclic system like this (the transmission-reception process in each station is a cyclic system) makes the timed occurrence graph become infinite, because each repeated appearance of a marking corresponds to a new state and hence implies the creation of a new node.

To obtain occurrence graphs with a manageable size and a full status, it was necessary to simplify the CPN model. In this sense, the size of the network was reduced to only three stations without changing the channel and each of them was allowed to transmit only one new packet. These three packets were forced to collide and later be properly retransmitted.

A part of the occurrence graph (O-graph) for the simplified network is shown in figure 15. It has a full status and 751 nodes.

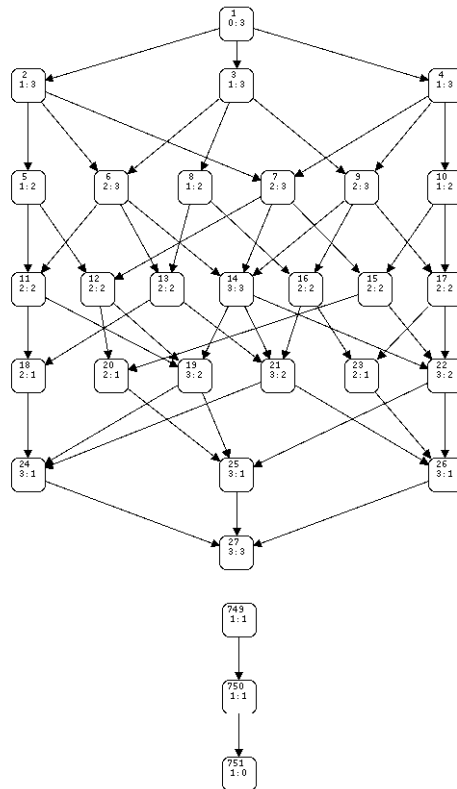


Figure 15. Occurrence graph.

The O-graphs were constructed by means of the OG tool [Occ93]. When an O-graph has been calculated, a set of standard queries makes it possible to investigate reachability, boundedness, home, liveness and fairness properties. For the reduced model a standard report was calculated and a summary is shown in table 1. The report doesn't show deadlocks and the examination of the state [751] indicates the packets have been properly transmitted and the channel is free. This standard report applies some standard queries to the model and saves the

results in a text file. Its analysis revealed no errors in the model.

Table 1: OG results.

Nodes	Arcs	Time	Dead Markings
751	2257	3	1 [751]

6 Conclusions

Two different versions of the tool have been used during the construction of the model (3.0.4 and 3.1.1). The migration from the 3.0.4 to the 3.1.1 version not only depended on time reasons (note that this last version has recently appeared) but also on some problems detected in the previous version. These problems were related to an strange syntax sensibility on the tool that made the simulation and validation of the model by means of the 3.0.4 version nearly impossible. After hard enquiries in the model and Internet pages, the problem could be partially solved. The detected errors were mainly related to the introduction of carry returns in color regions of places. However some problems could not be solved neither by means of the 3.0.4 version nor by means of the 3.1.1 version. This unsolved problem, that in fact still affects our model, is the application crash when some report facilities (bar charts) are used. Despite of our efforts and other people advices it still remains.

In addition to this problem, there are some other areas that have become into serious problems along this project. Some of them have been partially solved by means of the 3.1.1 version they mainly still remain unsolved. To start with, the pass from the editor to the simulator and occurrence graph generator is very time cost. When the model has a reasonable size the time spent to enter the simulator usually makes the user drive to despair, especially during the model debugging when it is necessary to enter the simulator and return back to the editor many times in order to correct some errors. Another important problem is the great memory requirements for the application. In our case, the graduate student is disposed of a Pentium working at 120 Mhz and 48 Mbytes of Ram memory. It was rather impossible to work with this equipment. While these two problems remain unsolved it will be difficult to introduce Design/CPN as a common use tool among our students.

Despite of this, some good impressions have been extracted from the application. It must be brought out the well-suited graphical interface, the possibility of constructing modular models using the hierarchical capabilities and overall, its straightforward analysis methods: simulation and occurrence graphs. Version 3.1.1 has reinforced this last section by the introduction of occurrence graphs with equivalence classes and symmetries. It is a pity that OE/OS graphs with time have not been theoretically developed yet. On the contrary they would had been applied to our model.

Bibliography

- [Goo87] G. Goods and J. Hartmanis. *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer-Verlag. “An accurate performance model of CSMA/CD bus LAN”. 1987
- [Jen92] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1. Springer-Verlag. 1992.
- [Jen94] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 2. Springer-Verlag. 1994.
- [Jen97a] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 3. Springer-Verlag. 1997.
- [Jen97b] Kurt Jensen. *A Brief Introduction to Coloured Petri Nets*. Computer Science Department, University of Aarhus. Denmark. http://www.daimi.aau.dk/~Kjensen/paper_books/. 1997
- [Jes94] Jesús García Tomás, Santiago Fernando y Mario Piattini. *Redes para procesos distribuidos*. Ra-ma. 1994.
- [Joh96] John Freer. *Introducción a la tecnología y diseño de sistemas de telecomunicaciones y redes de ordenadores*. Anaya. 1996.
- [Ref93] *Design/CPN Reference Manual for X-Windows*. Version 2.0. Meta Software Corporation. 1993.
- [Occ93] *Design/CPN Occurrence Graph Manual*. Version 3.0. University of Aarhus. 1993
- [Tan] Andrew S. Tanenbaum. *Redes de ordenadores*. Second edition. Prentice-Hall.
- [The97] Theo van Drimmelen. *Implementation of Statistical Functions in Design/CPN*. <http://www.daimi.aau.dk/designCPN/libs/pdf>. 1997.