Using SCPN for Modelling a Crossbar Switched Fabric CAN Network

Mohamed Mazouzi[1], **Ihsen Ben Mbarek**[2], **Oussama Kallel** [3], **Mohamed Abid**[4]
[1]Computer and Embedded Systems Laboratory, ENIS, Sfax, Tunisia
[2] Communication Systems Research Laboratory, ENIT, Tunis, Tunisia
[3] Communication Systems Research Laboratory, ENIT, Tunis, Tunisia
[4]Computer and Embedded Systems Laboratory, ENIS, Sfax, Tunisia

Abstract – *In recent years, the number of Electronic Control Units (ECU) steadily increases which require higher communication bandwidth. Switched fabric has become an active area of research because of its wide uses in industry. In fact, its uses can be a fast and reliable hardware solution for existing CAN-Bus problems like limited bandwidths and throughput.*
In this paper, we proposed and modeled a switched fabric CAN network Architecture based on CAN Controllers and switched fabric by the use of timed colored Petri nets (CPNTools).

Keywords*: Higher communication bandwidth, CAN Bus, Switched fabric, Switched fabric CAN Network, timed colored Petri nets, CPNTools.*

1 Current bussed Network problem and Switched Fabric CAN Network benefit

During last decades, the demand for sophisticated embedded systems requires the use of many connected equipments. A dedicated network bus [1] is used for connecting sensors, actuators in vehicles, robots and industries. Many serial buses were developed by car makers like MOST, J1850, SAE J1708, Byteflight, LIN... and CAN(controller Area Network). Most of them are specific to manufactures and not standardized.CAN is one of the most popular fieldbuses [2,3,4]. More than 400 million nodes were sold worldwide. It is used in those applications that require fast and reliable communication [5]. Nowadays, more sophisticated buses are concurrent to CAN networks like FlexRay [6], recently appeared, and RTethernet. They offer higher speed to satisfy the high bandwidth required for modern vehicles, suitable for x-by-wire application. In contrast the usage of FlexRay [7] is not widely used due to its complex specification and high cost.
Current parallel bus-based [8] solutions present some problems. In fact, it's well known that the physical separation of cards is limited to usually less than 3 feet. There are also limited bandwidths, high protocol overhead and no deterministic performance.

The limitations of a bussed network [2] are eliminated with crossbar switch network. A switched-fabric bus is unique in that it allows all CAN Controllers on a bus to logically interconnect with all CAN Controllers on the bus. The switching fabric is the physical connection within a switch between the input and output ports; it can be proved that all switches need a crossbar inside their switching fabric which allow them to operate at very high speed. Crossbar switches are widely used because of their simplicity and their high-performances [9] which promise to greatly simplify efforts and to add better capability and availability. Crossbar switch [8] can support simultaneously multiple messages. This greatly increases the aggregate bandwidth of the system. Because of the broadcast nature of the CAN protocol (ie: messages are not sent to a specific destination address, but rather as a broadcast), the chosen crossbar switch (as it is shown in Figure 1) is configured by closing all its crosspoints to ensure that the CAN message will be sent at the same time [3,4] for all outputs nodes as it is defined in CAN protocol [5].

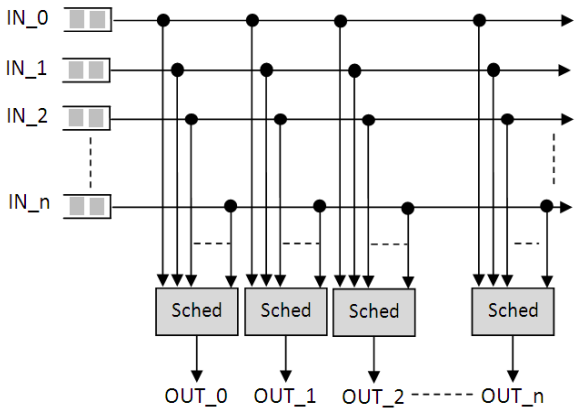

Fig. 1: NxN crossbar Switched Fabric CAN Network supporting broadcast

Each Electronic Control Unit (CAN Controller Node) produces a class priority of messages. For example, ECU_0 produces high level of priority and ECU_n (n in our model is equal 3) produce low level of priority. In fact, Produced CAN messages will be queued in the input queue of the incoming interface (If the input queue is full, the packet is dropped.). Therefore, to respect the CAN protocol philosophy, CAN messages will be broadcasted for all output port through crossbar Switched Fabric [10]. Furthermore, to reduce congested output port and to respect

the priority policy, each CAN message will be queued in the suitable output queue of each outgoing interface according to his level priority. (If the output queue is full, the packet is dropped). Then, each output port scheduler will select the message to be sent among the existing CAN message in accordance with his priority.
In our work, we modeled the switched fabric CAN Network using stochastic colored Petri. Our major contribution is to raise the lack of the bus solutions by proposing switched Fabric CAN. In fact, CAN based Networks using crossbar Switched fabric [11] have yet a well period before its replacement and it can compete the new sophisticated buses.

Our paper is organized as follow:

• The section 2 gives a short overview of Stochastic and Colored Petri Net SCPN [12].

• Based on the proposed architecture and CPNTools software, we model, in the last section, the most important Switched fabric CAN Network modules.

In the fourth section we give some conclusions of our work.

2 Short Recall of Coloured Petri Net

Coloured Petri Nets have been developed by K. Jensen in course of his PhD thesis (Jensen, 1980) to expand the modeling possibilities of classical Petri Nets. Like other forms of Petri Nets a CPN consists of places, tokens, transitions and arcs.
The primary feature unique to CPNs is the inclusion of evolved data structures into tokens [13,14]. These data structures are called coloursets and resemble data structures in high level programming languages; they can range from simple data types such as integers to complex structures like structs or unions in C/C++. Similar to programming languages it is possible to define variables associated with these coloursets such as linked list and queue.
Some examples of colourset and variable definitions are shown in Fig.2. Tokens as well as places of a CPN are always associated with a colourset and a place may only contain tokens of the same colourset as its own. To well understand the SCPN models of our Switched CAN controller, we give a short recall of CPN concepts.

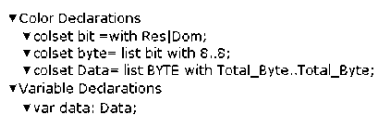

Fig. 2: Coloured and variable definitions

The places in a CPN are depicted as ellipses (Fig.3) with the name of the place written into it and the associated colourset (Id) below. A token in a CPN is represented by a small circle (Fig. 3). Its value (the data stored in the token) is shown in a rectangle attached to the circle. A number in the circle denotes the number of tokens with the same value. Figure 3 for example shows a place called *Buffer_Node_1* associated with the colourset CAN_Messages and holding one token with a value of{ {ID=[Dom,Res,Dom,Dom,Res,Dom,Dom, Dom,Res,Res,Dom,Dom,Dom,Res,Dom,Res,Dom,Dom,Do m,Res,Res,Dom,Res,Res,Res,Res,Res,Dom,Res],DATA=[b yte(4),byte(6),byte(5),byte(1),byte(5),byte(6),byte(6),byte(1)],TS=0}].

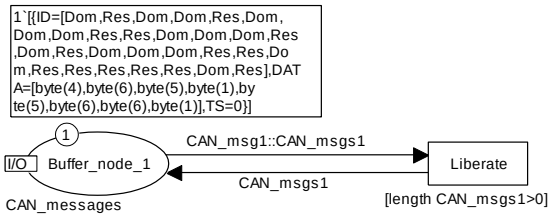

Fig. 3: Graphical representation of a place in CPN

Transitions in a CPN are represented by rectangles (Fig. 4) and can access the data stored in tokens by mapping tokens to variables. There are two possibilities to access this data:

-Guard conditions: The transition is enabled only if a specific condition – called a guard condition – regarding one or more variables is met. Guard conditions are encased in brackets and written above the transition (Fig.4).

-Transfer function: The transition reads and writes variables according to a specified function that can range from simple addition of values to complex conditional commands.

-Transfer functions consist of the definition of input () variables, output () variables and the commands to be carried out (action ()) and are attached below the transition (Fig.4).The example depicted in Figure 4 shows a transition that only fires if the length of variable *CAN_msgs* is less than the value *FIFO_length* and generates an output variable *CAN_msg* without taking any input variables (Fig. 4), the variable *CAN_msg* is filled with the return value of the function defined in the action part, *new_MSG_0*, which in this case is defined in the CPNtools area Declarations.

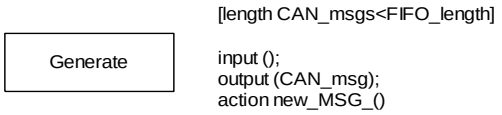

Fig. 4: Transition Generate with guard condition and transfer function

Places and transitions in a CPN are linked by arcs. Arcs in a CPN can be unidirectional or bidirectional. Unidirectional arcs transfer tokens from a place to a transition or vice versa.
Bidirectional arcs transfer the same token from a place to a transition and back. Arc inscriptions define the mapping of tokens to variables. An inscription can either be a constant value or a variable of the colourset that is associated to the place the arc is connected to. If all places connected to a transition by unidirectional input arcs or by bidirectional arcs hold tokens and its (optional) guard condition is met, the transition is said to be enabled. In case of more than one enabled transition in a CPN the one to fire is chosen randomly. Later on, we will add more places to our controller models to avoid arbitrary transitions.

For an analysis of clocked systems it is possible to define timed colourset, defined by the keyword timed and transition or arc delays marked by the characters @+.
If a colourset is defined as timed, a timestamp is added to the tokens of this colourset. The timestamp cannot be accessed by guard conditions or transfer functions. When

using timed colourset the firing of transitions depends on a global clock counter. Transitions can only fire if the clock value is the same as the largest timestamp of its input tokens. When a transition fires with a timed arc, the timestamp of its output token is the sum of the current clock value and the arc delay, in the example in Figure 5 this delay is *Time_sched_delay* clock cycles.

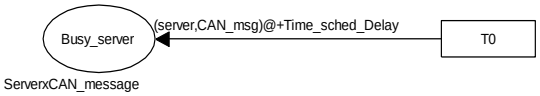

Fig. 5: Timed arc inscription.

3 SCPN Based Switched Fabric CAN Network model

In order to facilitate modeling of Switched CAN Controller, a modular approach was taken making use of hierarchical CPN models. The model of the Switched Controller is built following a hierarchical and modular architecture.

The root of the hierarchical representation of the model is shown in the figure 6.

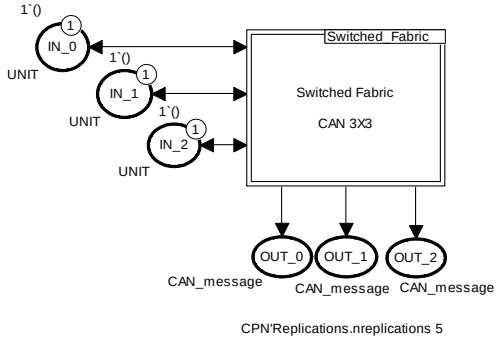

Fig. 6: Root level of the model

The Switched Fabric CAN 3x3 whose activity is modeled by the transition *Switched_Fabric* transmits the CAN message via the Switch Fabric. The places *IN_i* and *OUT_i* (i can be a value between 0:2) play the role of inputs/outputs for sub-models.

Nodes in CAN are identified by their identifier (in this model, colourset *Id* is a list of 29 bits). The coloursets and variables used in this model are shown in Figure 7.

Messages sent through the Switched Fabric CAN are represented by tokens of the colourset *CAN_message*. This colourset is a record of the colourset *Id* that designates the message priority and the colourset Data which represent the data field to be transmitted and the colourset *TS* for saving the time stamp for the birth of the message.

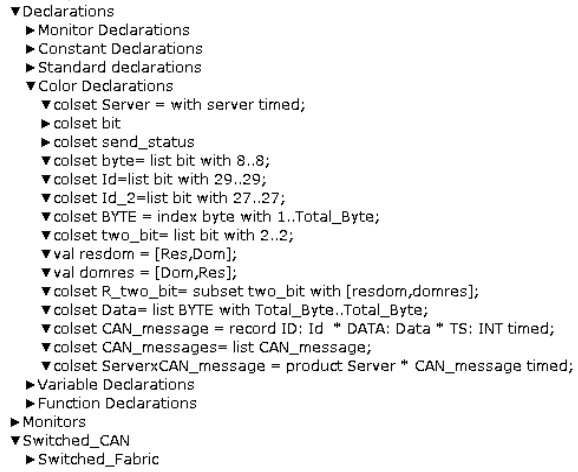

Fig. 7: Coloursets for CAN Network model

The variables (*CAN_msg*, *CAN_msg1* and *CAN_msg2*) are of type of the colorset *CAN_message*. This variable models the messages which cross the different sub-models of Figure 8 (*Node_i*, *Broadcast_i*, *FiFo_i_j* and *Scheduler_i*).

The Switched CAN network model in Figure 8 is composed of three nodes. Each node is represented by a transition and two places. The transition called *Node_i* (i can be a value between0:2) is a hierarchical transition which describes the messages generation within the node, how the messages are stored in buffer. The place *Buffer_Node_i* is used to store the messages already generated.

This place is configured with colourset *CAN_messages* which is a list of colourset *CAN_message*. When a token is present on this place (*Length CAN_msgs* >0) a message is ready for sending. This last fires the hierarchical transition *Broadcast_i*. The originated message is duplicated in three places, one for each output port of the Switch fabric. According to the priority which is associated to the messages (defined by their ID), the messages are stored in the FIFO queue (there is as many queue as of priorities). In this model, three levels of priority are defined: 0: high level of priority; 1: medium priority and 2: low priority.

FiFo_i_j is a substitution transition which presents the queue of *input_i* for the *output_j*. Finally, the scheduler processes the different messages according to its scheduling policy.

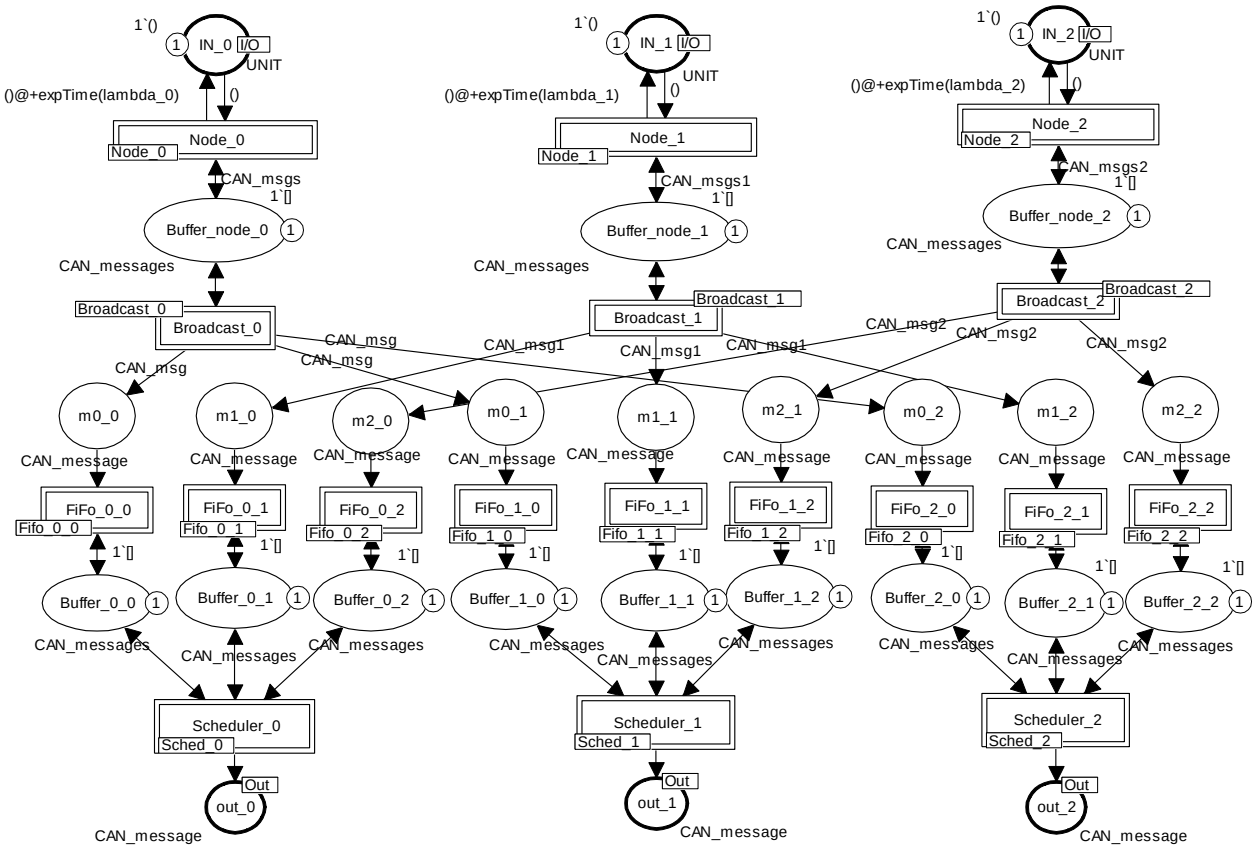

Fig. 8: SCPN Switched CAN Networks model with three nodes having three different priority classes

3.1 Generating CAN Messages (Node_i Transition)

As shown in Figure 9, the load source is modeled by the place *Next* and the transition *Generate*. Initially the place *Next* contain one token and is connected via two arcs to the transition *Generate*, the arc from *Generate* to *Next* is timed using the exponential random function. Thuswe can have a random message with parameterized inter arrival period using the value *lambda_i* (*lambda_0* for node 0: Figure 9).

The place *Buffer_node_i* is used as messages buffer, sized of 4 in our case, to decouple the source message from the Switched CAN controller. When the load source increases and no room is available in the buffer, an overflow occurs and the transition *FIFO_FULL* fires leading to lose the last generated message (due to a congestion or excessive load).

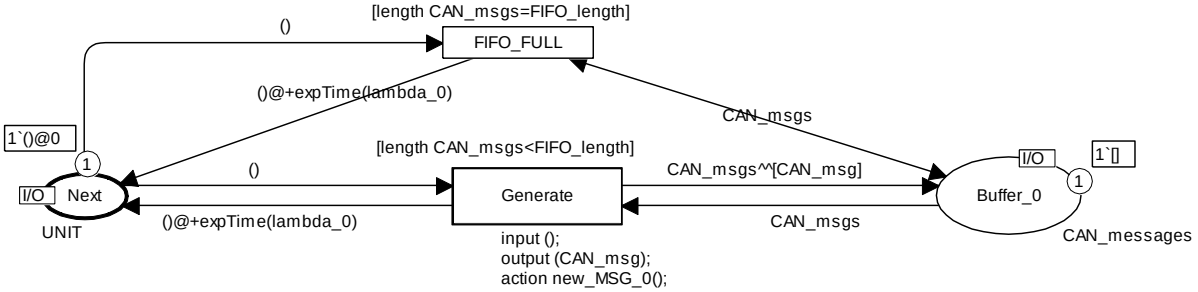

Fig. 9: Generation of CAN Message (Node_0)

3.2 Broadcasting of CAN Message (Broadcast_i Transition)

The set of broadcasting message is represented by the model described in the figure 10. Transition *Liberate* models a message coming from *Buffer_node_i*.

The *Buffer_node_i* place is a list of CAN_Message. When the list length is not null (i.e there is at least a message to send), the *liberate* Transition can be fired if the line is free (there is a token in *Line_Free* place). Otherwise, message coming from *Node_i* has to be delayed *Transfer_Data_Delay* until the previous message will be liberated. If the message is liberated, the *Server* token will be moved from the place *Line_Free* to *Line_Busy*. Then the messages will be duplicated in the right place (queues)

according to their priorities. For example, message of medium priority (*CAN_msg1*) will be routed to the places *m1_0*, *m1_1* and *m1_2*. The CAN message *m1_i* will be queued in the queue 1 of the output port i (*OUT_i*).

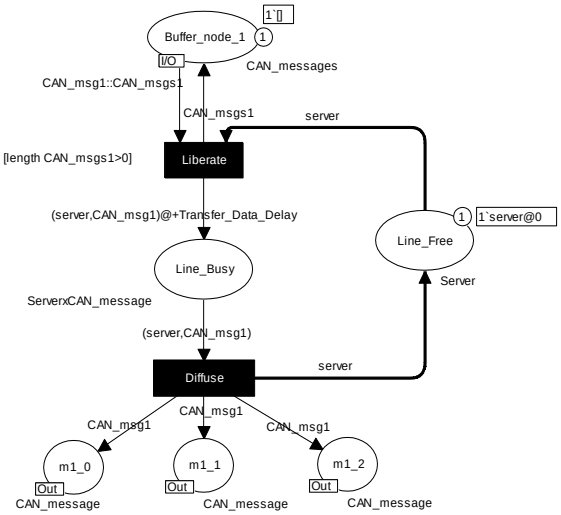

Fig. 10: Broadcasting CAN Messages generated by Node1

3.3 Storing CAN Messages (FiFo_i_j)

FIFO model is represented in the figure 11. It processes the messages in the order of their arrival. The function of the transition *Arrive* is to concatenate incoming message (*CAN_msgi*) to the *Buffer_i_j*. *Buffer_i_j* is a place having *CAN_messages* as colourset: i indicates the level of the message as it was explained previously and j indicates the output port of the model. Thus *Buffer_i_j* is the queue of Message i of the output port j (*OUT_j*). When the buffer is full (in our model *FiFo_length= 4*), the transition *Fifo_i_j_Full* is fired and the incoming message will be rejected.

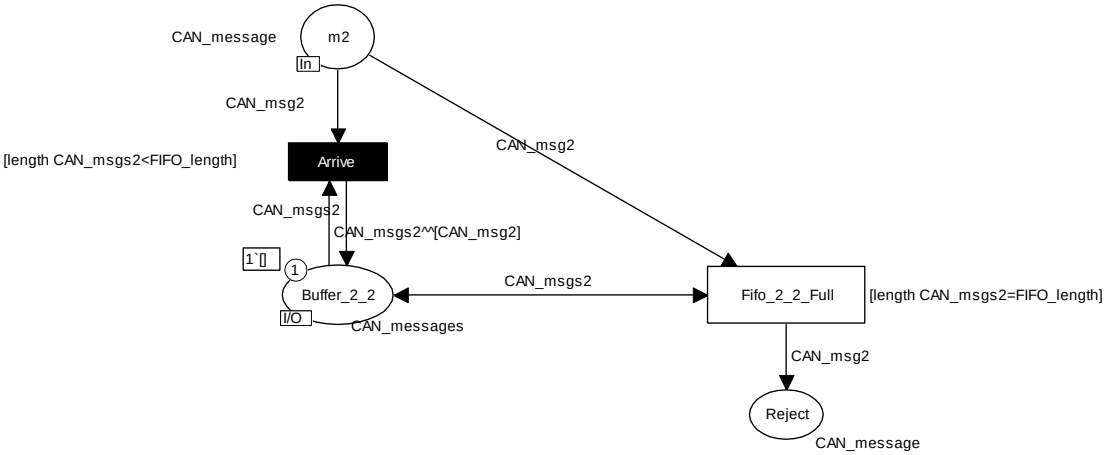

Fig. 11: Storing CAN Messages generated by Node 2 in the queue 2 of output 2

3.4 Scheduling CAN Messages (Scheduler_i)

The model of the figure 12 describes the behavior of a static priority scheduling. The type of messages is classified in three groups:

- High priority messages: These messages are generated by Node_0 and are modeled in the place *Buffer_i_0* as a list of *CAN_Message* (*CAN_msgs*) in the Scheduler of the output port i (*OUT_i*).
- Medium priority message: Those are generated by *Node_1* and are modeled in the place *Buffer_i_1* as a list of *CAN_Message* (*CAN_msgs1*) in the *Scheduler_i*.
- Low priority message; those are generated by *Node_2* and are modeled in the place *Buffer_i_2* as a list of *CAN_Message* (*CAN_msgs2*) in the *Scheduler_i*.

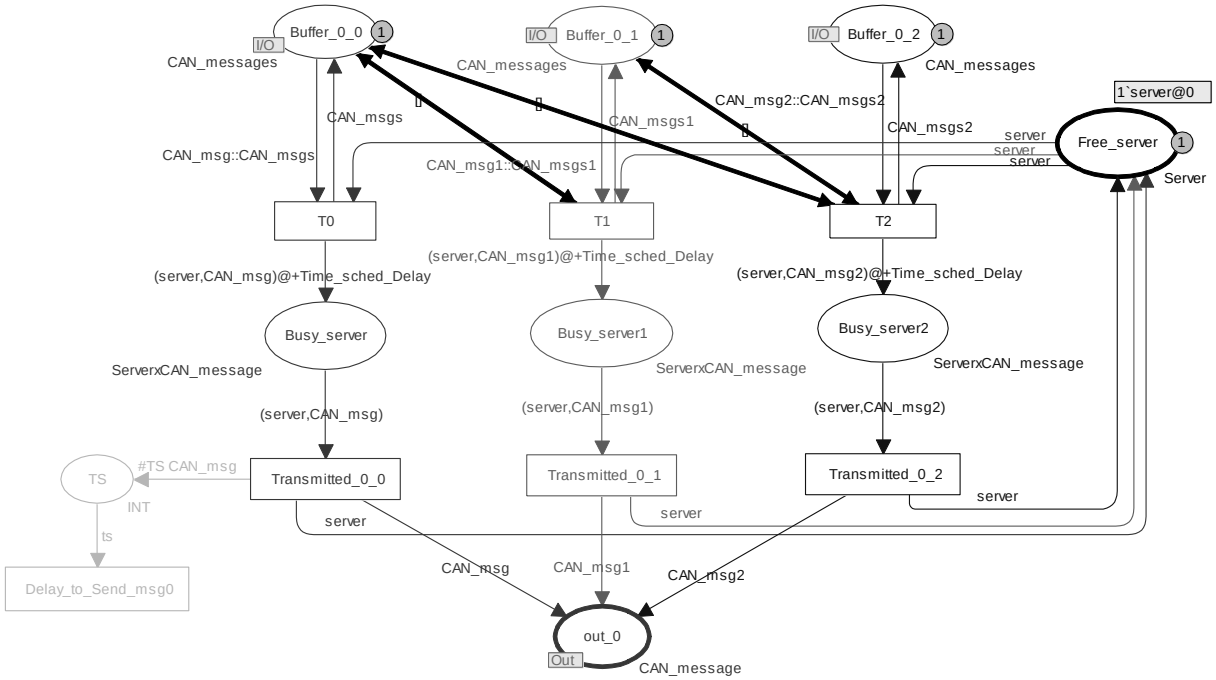

Fig. 12: Scheduling CAN Messages at the Scheduler of Output 0

A message with lowest priorities can be delayed by the other packets due to the non-preemptive characteristics of this kind of message scheduling algorithms [15].
For example, in the case of Scheduler_0, the messages of Buffer_0_2 (low priority messages) has to wait until the messages of Buffer_0_0 and Buffer_0_1 (high and medium priority messages) are fully transmitted. Then, the messages of Buffer_0_1 (medium priority messages) has to wait until the messages of Buffer_0_0 (high priority messages) are fully transmitted.

This scheduling policy is modeled using bidirectional arcs between buffers places and the transitions *T1* and *T2*. These arcs are inhibitor arcs. The method of usage of inhibitor arc is described in more details in [13].
When there is at least a message to transmit, the Ti transition can be fired if the server is free what it means that there is a token in *Free_server* place. Otherwise, the following message (according to the algorithm described above) have to be delayed *Time_Sched_Delay* until the previous message is fully transmitted. After the transmission of the message, the Server token will be moved from the place *Busy_sereri* to *Free_server*. The interest of the static priority algorithms is that it is easy to implement. Other Scheduling algorithms can be studied in future works.

4 Conclusion

In this paper, Switched Fabric CAN architecture is presented and modeled by CPNTools. The SCPN model of the Switched Fabric Controller is presented with three nodes at transmitter side using a switch fabric (3x3). For that, three message priority classes were treated with a clear representation on ID field (high, medium and low priority messages).
The model focuses on queueing, broadcasting and scheduling mechanisms which are the keys factor for the proposed architecture.
The evaluation of throughput, latency and loss probability of the proposed architecture and a comparison with Bussed CAN controller [2] will be studied in the future works to demonstrate that CAN with a crossbar switched fabric has yet a well period before its replacement.

5 References

[1] Salem Hasnaoui, Oussema Kallel "A proof-of-concept Implementation of Modified CAN Protocol on CAN Fieldbus Controller Component"; Accepted oct. 2004, Revised Jan. 2005; AMI. Journal, Ref 03/08.
[2] Oussama KALLEL, Sofiene DRIDI, Salem Hasnaoui “Modeling and Evaluating a CAN Controller Components Using Stochastic and Colored Petri Nets” IRECOS International Review on Computers and Software, Vol.4 N.1, pp 142-151; January 2009.
[3] Marko Bago, Siniša Marijan, Nedjeljko Perić; Modeling Controller Area Network Communication, proceedings of the Ninth Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, October 20-22, 2008
[4] CUI Lian-cheng, ZHAO Zheng-fang, XU Xiao-ju2, WU Fang-ming, SHAN Wei-zhen “Real Time Performance Analysis

of CAN Bus Based on TimeNET" The 3rd International Conference on Innovative Computing Information and Control (ICICIC'08), 2008.
[5] Prodanov, W.; Valle, M.; Buzas, R.; Pierscinski, H."A Mixed-Mode behavioral model of a Controller-Area-Network bus transceiver: a case study" Behavioral Modeling and Simulation Workshop, 2007. BMAS 2007. IEEE International
Volume, Issue , 20-21 Sept. 2007 Page(s):67–72
[6] Heller, C. Reichel, R. "Enabling FlexRay for avionic data buses" Digital Avionics Systems Conference DASC '09,Oct 2009.
[7] FlexRay Communication System. [Online]. available: http://www.flexray.com
[8] Brett Murphy, Emmanuel Eriksson. "Fabrics and Publish-Subscribe Schemes: A Net-Centric Blend" COTS Journal Oct. 2009 <http://www.cotsjournalonline.com/articles/print_article/100148>
[9] Rojdi Rekik, Tarek Guesmi, Salem Hasnaoui "Challenges in the Implementation, the Configuration and the Evaluation of a QoS-enabled Middleware for Real-Time Embedded Systems";
IRECOS International Review on Computer and Software vol. 3, n°3, May 2008.
[10] Arshad, Nauman, Stewart Dewar, and Ian Stalker. "Serial Switched Fabrics Enable New Military System Architectures." COTS Journal Dec. 2005 <www.cotsjournalonline.com/home/article.php?id=10043>
[11] Dr. Rajive Joshi," Using Switched Fabrics and Data Distribution Service to Develop High Performance Distributed Data-Critical Systems" The Journal of Defense Software Engineering. April 2007.
[12] K. Jensen , "Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts" (Monographs in Theoretical Computer Science, Springer-Verlag, 1997).
[13] A.V. Ratzer, L.Wells, H.M.Larsen, M.Laursen, J.F.Qvortrup, M.S.Stissing, M. Westergaard, S. Christensen, K.Jensen. CPN Tools for editing, simulating, and analysing Coloured Petri Net. LNC, 2679:450-462, 2003.
[14] Design/CPN Manuals. Meta Software Corporation and Department of Computer Science, University of Aarhus,Denmark. On-line version: http://www. daimi.aau.dk/designCPN/.
[15] John D. Pape, "Implementation of an On-chip Interconnect Using the i-SLIP Scheduling Algorithm". (The University of Texas at Austin Chap 3 pp 11-15: December 2006)