Evaluation des performances temporelles d'architectures d'automatisation distribuées sur Ethernet par simulation d'un modèle eb réseau de Petri de haut niveau.

Gaëlle Marsal, Gaëlle Poulard (épouse Marsal)

ENSC-2006#26

PhD thesis of the
ÉCOLE NORMALE SUPÉRIEURE DE CACHAN
and of the
UNIVERSITY OF KAISERSLAUTERN

Defended by

Ms. Gaëlle Marsal

to obtain the grades of

DOCTOR OF THE ÉCOLE NORMALE SUPÉRIEURE DE CACHAN
and
DOCTOR OF THE UNIVERSITY OF KAISERSLAUTERN

In :

ELECTRICAL AND AUTOMATION ENGINEERING

Evaluation of time performances of Ethernet-based Automation Systems by simulation of High-level Petri Nets

Defended at Cachan on December, the 11th, 2006 in front of the committee composed of:

Hassane Alla	Professor - INPG - LAG	Examiner
Stefan Kowalewski	Professor - RWTH Aachen University	Reviewer
Eric Rondeau	Professor - University of Nancy - CRAN	Reviewer
Jean-Marc Faure	Professor - ENS Cachan - LURPA	Advisor
Georg Frey	Junior Professor - TU of Kaiserslautern - JPA^2	Advisor
Bruno Denis	Assistant Professor - ENS de Cachan - LURPA	Advisor
Patrick Salaun	Researcher - EDF R&D	Invited

LURPA

ENS Cachan
Université Paris 11
61 Avenue Président Wilson
94235 Cachan Cedex – France

JPA^2

Dep. of Electrical and Computer Engineering
University of Kaiserslautern
Erwin-Schrödinger-Str. 12
67653 Kaiserslautern – Germany

The work that has resulted in this thesis could not have been performed without the support of several people who must be mentioned here.

First of all, I would like to thank my advisors Professor Jean-Marc Faure, Professor Georg Frey and Assistant Professor Bruno Denis for the opportunity to perform my work at their institutes. I thank them for all encouragement, guidance and the necessary intellectual freedom they have given me to fulfil this work. Working in their teams was a privilege and a great experience I will never forget.

I would like to thank the reviewers of this thesis, Professor Stefan Kowalewski from the RWTH Aachen (Germany) and Professor Eric Rondeau from the University of Nancy (France) for the interest they took in my work and the effort they invested in reviewing deeply this thesis. I would also like to thank Professor Hassane Alla and Patrick Salaun for agreeing to participate to the Evaluation Committee.

These three years spent in the LURPA have been a pleasant time thanks to all the colleagues who have always been very helpful either for technical, scientific or personal problems. In particular, I thank Françoise for its efficiency and its kindness, Marc for its philosophical talks, Sylvain for its support, the Mexican team for the "quesadillas", Steve for its gossip, Vincent for its coding capacities.

I also would like to thank all the colleagues of Kaiserslautern, in particular Jürgen and Mohammed, for their kindness each time I came there.

Finally, I want to thank my parents and my brothers who always encourage me in the way I have chosen. I don't forget my husband, David, who has coached and supported me during all these three years, and of course my daughter, Clara, who enlightens my life every day.

Cachan, December 2006
Gaëlle Marsal

Contents

Introduction

To ensure correct operation of industrial production systems, all microprocessor-based equipments must communicate and cooperate at any level, from control to management, including monitoring and planning. To provide communication at the field level, when distributed automation systems appeared, automation solutions providers have developed fieldbuses based on specific communication hardware and protocols, such as Modbus developed by Modicon in the late seventies. Up to now, these fieldbuses have been designed to comply with automation requirements of different industrial domains (process control, batch or manufacturing systems automation), and each well-established automation solutions vendor went on with its particular product (Profibus for Siemens, Unitelway for Schneider Electric...). As a consequence, no standardisation compromise was ever found and interoperability of Networked Automation Systems, distributed automation systems whose components are linked by a fieldbus, was really weak. In this thesis, all these solutions for communication at the field level developed during the 80s and the 90s will be referred to as "classical fieldbuses".

To reduce design and implementation costs and to improve interoperability of Networked Automation Systems, the protocols Ethernet, TCP and IP are considered as the future universal standards for fieldbuses; in fact, industrial solutions claiming to rely on one or several of these protocols, and labelled 'Industrial Ethernet', are already available. The introduction of common Local Area Network (LAN) protocols at field level provides indeed great prospects for both interoperability and flexibility of Networked Automation Systems. They should ease communication over all Computer Integrated Manufacturing (CIM) levels and permit to use Commercial Off The Shelf (COTS) devices for automation.

However, these LAN protocols have not been designed specifically for automation applications and thus own features that are a priori inappropriate for a fieldbus. The two main features that seem not to fit Networked Automation Systems are, firstly, the non-deterministic access to media method (CSMA/CD) and, secondly, the local management of network resources provided by the client/server communication model, which is classically implemented in Ethernet/TCP/IP LANs. Indeed, the non-deterministic access to medium method can lead to possible frame collisions, and the local management of network resources to long waiting times for availability

of shared resources. These two phenomena can slow down strongly communication between equipments, which is not acceptable in automation.

To overcome these two problems, a first solution is to implement producer/consumer or master/slave cooperation model at the highest communication layer, as in classical fieldbuses. Indeed, these cooperation models guarantee both deterministic access to medium and global management of network resources. Hence, delays caused by a network using these cooperation models are bounded and easy to compute. However such fieldbuses are not really Ethernet-based.

A second solution consists in introducing switches within the network, to reduce collision domain, and using a rigorous design method, that shall include adequate assessment or verification methods for checking compliance to requirements. This solution, which offers a total interoperability with other COTS components, enables automation engineers to develop really Ethernet-based automation systems which contain products from well-established automation solutions providers (as Modbus/TCP protocol from Schneider Electric), or soft PLCs.

The overall goal of this work is to show the interest of this second solution. To reach this objective, it matters first to define accurately relevant time performances. Then, the method and tool for time performances evaluation of Ethernet-based automation systems are to be provided. It will be possible, in a third step, to compare client/server cooperation model to the two other ones (master/slave and producer/consumer) and to analyse finely time consumption mechanisms in Ethernet-based automation systems.

The first chapter of this report starts by the definition of the limits of this study and of the time performances that shall be evaluated: network cycle time and response time. Then, a technological overview of classical fieldbuses and of Ethernet and TCP/IP protocols is presented. At last, a detailed, while not formal, analysis of the time consumption mechanisms in the different hardware and software components of Ethernet-based automation systems enables us to pinpoint that the overall response time is the sum of three delays: processing time, waiting time for synchronisation of asynchronous processes and waiting time for availability of shared resources.

The second chapter contains a survey on academic works that addressed evaluation of time performances of Networked Automation Systems. Both "a priori" (before implementation) and "a posteriori" (after the system has been designed and implemented) methods are considered: analytic calculus, formal verification methods, simulation and experimental methods. It appears that to determine the time performances we focus on, the most relevant method is the simulation of a dynamic model of the automation system.

The third chapter is mainly devoted to the presentation of the modelling formalism that we have chosen for the dynamic model. Principles for designing the dynamic model are first stated: generic modelling, ability to represent three time consumption mechanisms, suitable time and data representation. Given these principles, Hierarchical Timed Coloured Petri Nets have been chosen as the modelling formalism. Once the concepts of this formalism presented, the chapter closes on the description of the structure of the generic model of Ethernet-based automation systems.

In the fourth chapter, first, some components of this generic model (Ethernet Modbus client, PC-based controller and switch), which describe the behaviour of hardware or software components of Ethernet-based automation systems, are detailed. In the following section, the instantiation process, which permits construction of particular models from the generic model, is presented and finally an example of one particular Ethernet Modbus client is given.

The fifth chapter deals with simulation of one particular model so as to obtain its time performances of the corresponding automation systems. Particular attention is paid to set up of the model in order to obtain meaningful simulation results.

At last, comparison of the three cooperation models is addressed in the first part of the sixth chapter. The network cycle times of systems using these cooperation models are compared and discussed. Once this comparison achieved, focus is put on evaluation of response time of Ethernet-based automation systems and on analysis of the relative importance of the three delay causes that were pointed out above.

From the results which have been obtained in this PhD work, several research and technical development prospects are drawn up in the conclusion.

Chapter 1

Fieldbuses in Automation Systems

This chapter presents the context of our work. The first section introduces Networked Automation Systems. Then, the second section defines the Networked Automation System performances, focusing on the time performances studied in this work, the network cycle time and the response time. A third section presents some characteristics of specific purpose fieldbuses. Finally, a fourth section presents the use of general communication technologies in fieldbuses, in particular, Ethernet, TCP, IP and the client/server cooperation model. In this section is also details the impact of these technologies on the networked Automation System time performances.

1.1 Networked Automation Systems

A Networked Automation System is composed of automation components, controllers and Remote Input Output Modules (RIOMs), interconnected by a network, often called fieldbus, and exchanging data thanks to communication protocols.

In such systems, controllers must accomplish two functions: the first one is to treat data for plant control and the second one is to collect from and transmit to the RIOMs data via the communication medium. In this thesis, only mono-task controllers will be considered. Then, the first function, to treat data for plant control, corresponds to the cyclic execution of the control program: first reading inputs, then executing the user program, and finally writing outputs. The second function is obtained by scanning regularly the RIOMs through the fieldbus.
Two hardware solutions are currently encountered. The first one is to have one dedicated module with its own processor for each function which concerns most of the Programmable Logic Controllers and some Industrial PCs with special board added. In the following, we choose to call this solution either modular controller or PLC-based controller. In this case, the two functions are implemented in the two different modules, they run in parallel and communicate via a backbone bus.
The second hardware solution is to have only one processor unit to carry out both

functions which concerns small controllers and some industrial PCs. In the following, we choose to call this solution PC-based controller. In this case, the two functions are executed in the unique module, where the processor of the controller is shared for control and communication tasks that are performed sequentially.

RIOMs are interfaces between the plant and the network. Each RIOM is connected to sensors and actuators. Some RIOMs are able to treat inputs and outputs locally. These ones are called intelligent RIOMs. They are more often used in continuous process control. In the case of automation systems, mainly regular RIOMs are used which only send input values to controllers and set up output values received from controllers. In this work, only regular RIOMs are studied that are called RIOMs in this thesis.

In the following, the term "device" is used for controllers, RIOMs and switches, while "components" is used for sub-parts (hardware or software) of devices, such as communication modules, CPU modules and user program in controllers.
Figure 1.1 shows a Networked Automation System with three controllers and three RIOMs (RIOM).

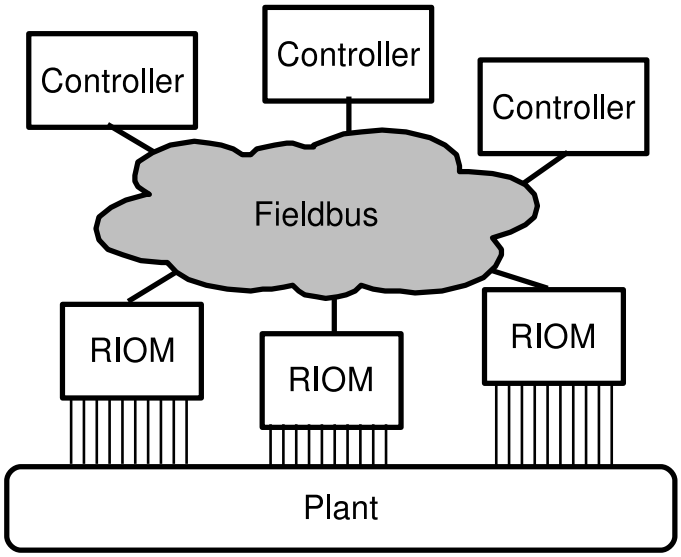


Figure 1.1: Example of a Networked Automation System

In the Networked Automation Systems, one controller generally communicates with several RIOMs and one RIOM can be scanned by several controllers. All these devices share the same communication medium. As a consequence, the fieldbus is a shared resource to manage. In the following, different solutions proposed are studied, both with specific purpose fieldbuses (page 10) and with general purpose communication technologies (page 15).

The next section details the time performances of interest in Networked Automation Systems.

1.2 Time performances of Networked Automation Systems

1.2.1 Networked Control Systems

The expression Networked Control Systems (NCS) is reserved for continuous closed-loop control, as indicated by the term "feedback control" in the definition of the University of Maryland (Intelligent Control Engineering Laboratory [1998]): "When a traditional feedback control system is closed via a serial communication channel, which maybe shared with other nodes outside the control system, then the control system is called a Networked Control System (NCS)".
A major user requirement in Networked Controlled Systems is then the stability of the system. This stability can be obtained by having a constant delay (Juanole [2002]). An other approach is presented in Vatanski et al. [2006]: robust control based on delay compensation thanks to the evaluation of upper-bound delays.
Networked Control Systems concern a large scientific community with many advances. However, the scope of Networked Automation Systems can not benefit of all the results found for Networked Control Systems because user requirements and time performances needed are completely different, as explained in the next subsections.

1.2.2 Classification of Networked Automation System performances

Generally speaking, the performances of a system can be put into two categories: performances related to value correctness and performances related to time correctness (Kopetz [2003]). Value correctness performances analysis focuses on correctness of data values produced by the system, while time correctness performances analysis considers correctness of dates and of time delays between events whatever the values of data associated to the events.

Regarding value correctness in Networked Automation System, two causes can damage values: errors in the user program or disturbances on the network. To verify correctness of the user program is another field of investigation based for instance on model checking techniques (Kowalewski et al. [1999b,a], Mertke and Frey [2001]). Disturbances on the network, that can damage data, are important in wide area networks or in WiFi networks but not in local area networks such as fieldbuses. Hence,

this study does not address the issue of value correctness.

Regarding time correctness performances in Networked Automation Systems, they are detailed in the two next subsections.

1.2.3 From user requirements to response time

Performances of a Networked Automation System are driven by the requirements of the system that it controls (Greifeneder and Frey [2006]). To illustrate the link between time performances of automation and user requirements, the example of filling a water tank to a desired level is pictured figure 1.2. The system considered is composed of two tanks TA1 and TA2. The tank TA1 fills TA2 until a required level included in the range Lreq. For this, when sensor S1 detects water (level Ldet), the input I1 on RIOM6 becomes TRUE (at time t0). Then it is treated by PLC2 to emit the output O1, at time t1, that closes the valve V1. The obtained level in TA2 is noted Lef. The performance of this system, in term of water level, can be expressed as $(Lef - Ldet)$, taking Ldet as the reference level. If we consider the entire automation system, we can notice that $Lef - Ldet$ depends on $(t1 - t0)$, the response time of the control system.

Whatever the system that is controlled, its performance is always linked with the response time of the Networked Automation System defined as the delay between the occurrence of a cause event (input I1) and the corresponding consequence event (output O1) (Marsal et al. [2006c]).

It is essential to note that the response time is generally not constant. Actually, the system being time-driven by the user-program execution, even if there is no network, the new input values are taken into account only at the next reading phase, i.e. between zero and one cycle time. As a result, the response time, that includes processing of the data, lasts between one and two cycle times. In a Networked Automation System, delays due to the network appear. These delays may be constant or not depending on the communication protocol used.

The end user is then interested not only by one value of the automated system performance but by the distribution of all possible values. This distribution is highly linked with the distribution of response time. *Therefore, one goal of this study is to evaluate the distribution of response time.*

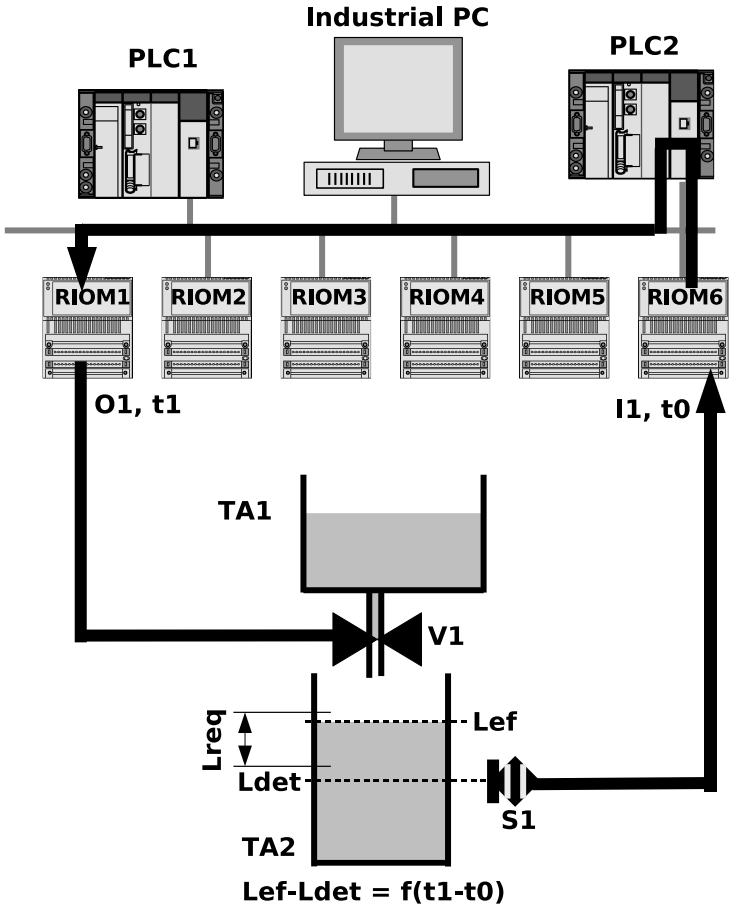


Figure 1.2: Example of performance for filling a tank

1.2.4 From designer requirements to network cycle time

The response time is the major performance for the end user but it may not enable to discriminate different fieldbus solutions. For this, the commonly used property is the Network Cycle Time. It is defined as the time between two consecutive sendings of message from one controller to one RIOM. The network cycle time can also be variable, depending on the network protocols chosen. Here, as for the response time, it is essential to evaluate the distribution of the network cycle time.

This work is going to investigate these two time performances, response time and network cycle time of Networked Automation Systems for fieldbuses using general purpose communication technologies. Before to study them, the two next sections details the specific and general communication technologies that are available for fieldbus.

1.3 Using specific purpose fieldbuses

Before to detail the features of special purpose fieldbuses and the mainly used co-operation models, a first subsection gives general concepts of networks.

1.3.1 Network topology and OSI reference model

Network topology

The topology of a network refers to the layout of devices connected on a network. The different existing topologies are based on six basic types presented in figure 1.3: linear, ring, bus, star, tree, mesh. On this figure, the squares are stations, either networking or automation devices (controllers and RIOMs).

Whatever the network topology has, protocols are needed to communicate from one station to others. For this, a framework, named OSI model, has been defined and standardised.

OSI reference model

The Open Systems Interconnection Reference Model (OSI model) (Zimmermann [1980]) is an ISO standard which defines a layered, abstract description for communications in computer networks. This model has been designed to simplify interconnection of systems from different manufacturers who have defined their own protocols.
Each layer of the model depends on the immediate lower layer and is isolated of the higher layer. A layer adds value to the services provided by the lower layers and the highest layer offers the services to run applications. The seven layers defined in

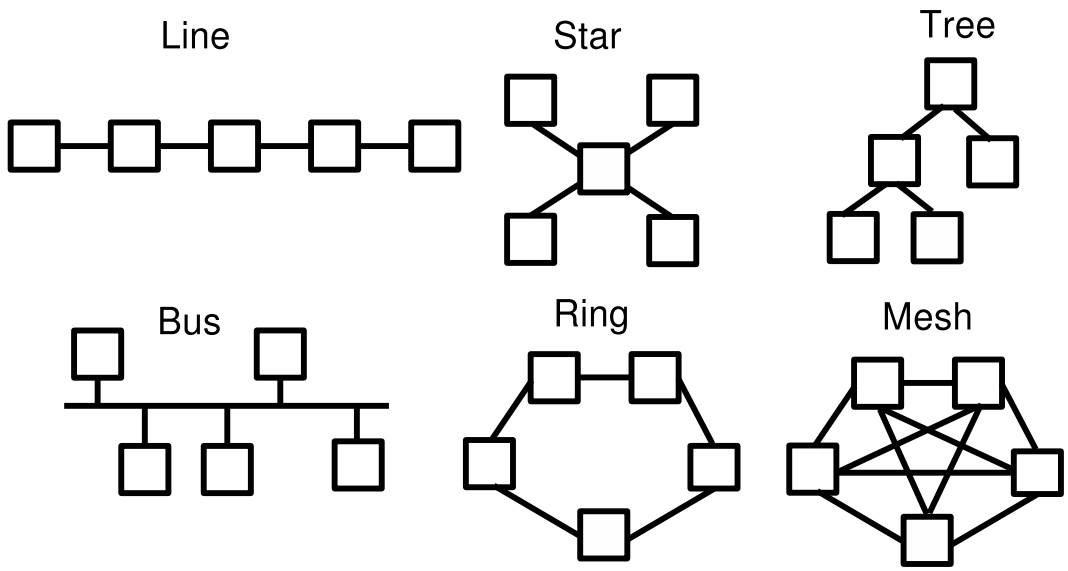


Figure 1.3: Basic network topologies

this model are given figure 1.4, Physical layer being the lowest one and Application layer the highest one.

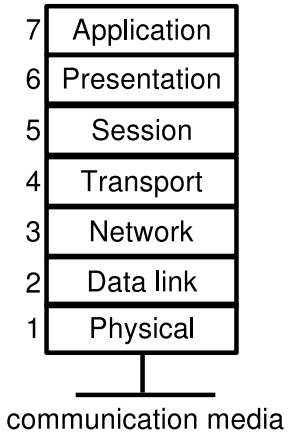


Figure 1.4: OSI model

The application layer provides to the user with means to access network information through an application. As examples, two well-known protocols of application layer are FTP (File Transfer Protocol) and HTTP (HyperText Transfer Protocol). The presentation layer provides to application layer a standard interface to interpret data. This layer is used with cryptographic protocols to decode data. The session layer controls the connection between hosts. The transport layer provides a transparent transfer of data between hosts. Two widely used transport protocols are TCP

(Transmission Control Protocol) and UDP (User Datagram Protocol). The network layer provides with means to exchange data between two hosts over a network connection. The most used network protocol is IP (Internet Protocol). The data link layer provides with means to establish, maintain and release data link between hosts. Media Access methods are included in this layer. For instance, CSMA/CD in Ethernet. Finally, the physical layer defines the mechanical and electrical characteristics to establish maintain and release data link between two network entities. For instance, RS485 and 10Base-T are two different physical links.

Based on these general concepts, the following subsections present fieldbuses with the mainly used cooperation models.

1.3.2 Fieldbuses history

Fieldbuses appeared in the late seventies in the form of dedicated networks developed by automation devices manufacturers. These developments have been carried out independently and lead to different non-interoperable networks. In the area of automation, the first fieldbus, Modbus (Modbus-IDA [2002]), designed by Modicon has been developed for RS232 and RS485 physical layers. These serial links enable only a bus topology. As a consequence, the access to the media must be controlled to allow only one message on the bus at each time.
Then, academic works and consortia have begun to work on the definition of new fieldbuses in a goal of standardisation. Three well-known results are FIP, evolving in WorldFIP (WorldFIP organisation [2003]), Profibus in Germany around 1985 (PROFIBUS Nutzerorganisation e.V. [2006]) and Fieldbus Foundation, gathering a number of companies that decided in 1994 to define a global specification for all the layers to be standardized. These fieldbuses have been initially designed also for serial links RS232 and RS485, with a bus topology. So, the access to media must be controlled, too.
For a more detailed history of fieldbuses the reader can refer to Thomesse [1999] and Felser [2002].

The needs in automation systems to control the medium and to have a cyclic or periodic refreshment of data in controllers and RIOMs is obtained by the cooperation models. The two that are mainly used are master/slave and producer/consumer. The figure 1.5 shows how the cooperation models take place in the OSI model. They are providing both the media access control, which is part of the layer two (data link layer), and the refreshment of data, which is the application layer.

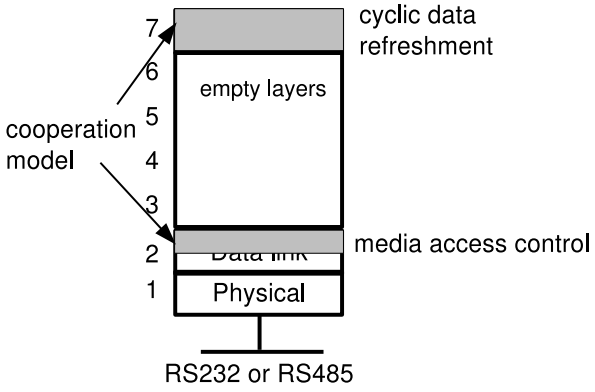


Figure 1.5: Cooperation models in the OSI model

Master/slave cooperation model

The master/slave model consists in polling all slaves (RIOMs), one after the other by one or several masters (controllers). In the case of multi-master system, the masters must be coordinated to not proceed in parallel in order to have always just one communication taking place on the fieldbus. For each slave, the master sends a request, waits until it gets the response and skips to the next slave. When it has scanned all its slaves, the next master can proceed. The sequence diagram on figure 1.6 illustrates this behaviour with two masters and four slaves. The black rectangles corresponds to communication activities linked with Master1 while the light grey ones corresponds to communication activities linked with Master2. In the first cycle (Cycle1), Master1 begins by requesting Slave1; when Master1 has received the associated response, it sends a request to Slave2. Then, when Master1 has received its second reply, no more slaves are to be requested and so it is the turn of the next master, Master2. In the same way, Master2 scans its two slaves, and then a new cycle can begin. To allow only one master to transmit, a third device, called bus arbiter, must be used (not represented on the figure 1.6). Its role is to distribute time slots to access the media for each master. In the studied automation architectures, masters are controllers and slaves are RIOMs.

Producer/consumer cooperation model

In the producer/consumer model (Miorandi and Vitturi [2004]), a producer is a component that sends its data on the network to all consumers interested in. A bus arbiter schedules the media access, enabling only one producer to transmit in order to have always just one communication taking place on the fieldbus. As in master/slave model, in each cycle, all producers must have time to send data to consumers. In this model, devices are both producers and consumers depending on the time. For instance, a controller is producer when it sends new output val-

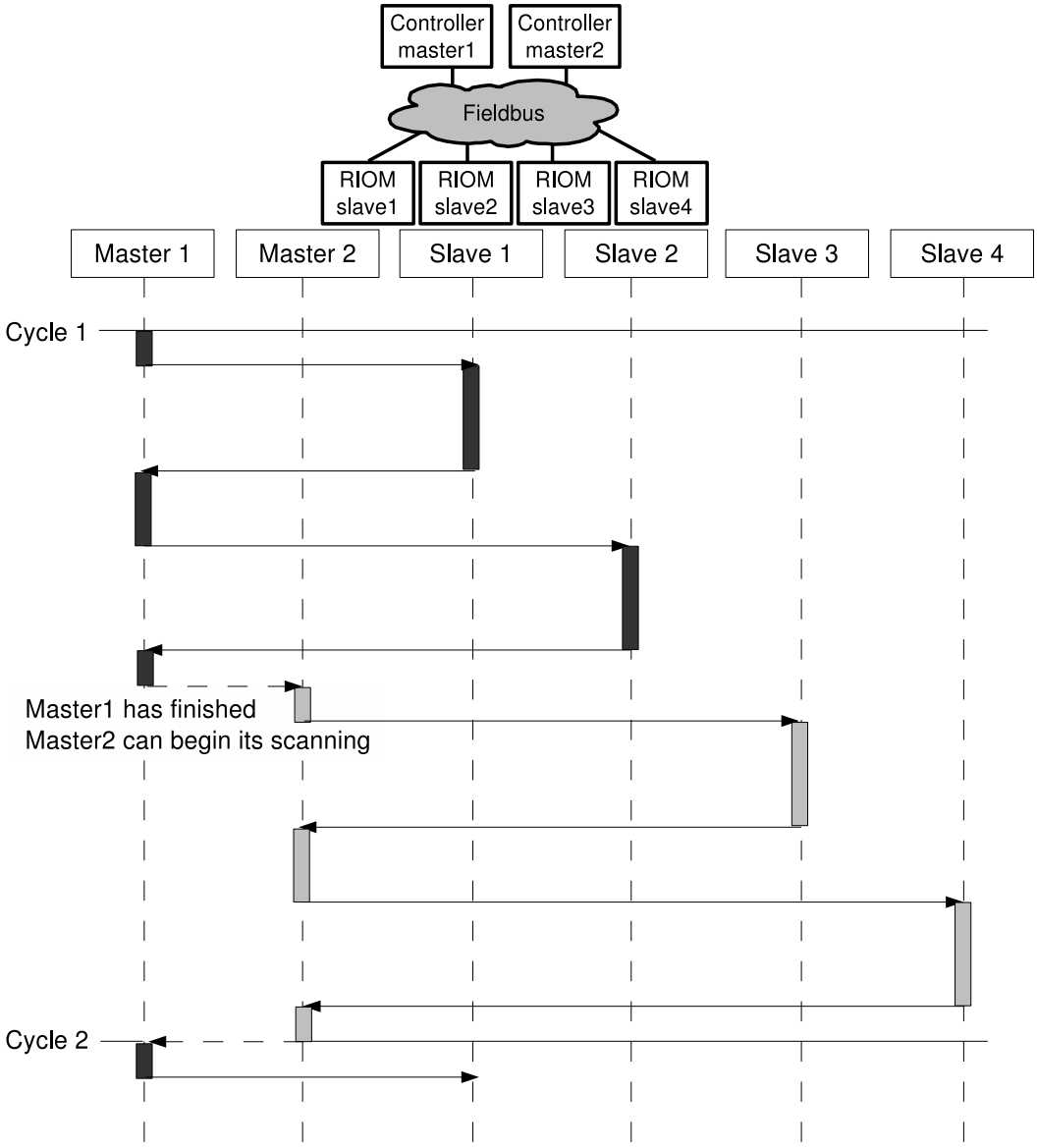


Figure 1.6: Sequence diagram illustrating Master/Slave model with two masters and four slaves

ues to RIOMs, while it is a consumer when it waits for input values coming from RIOMs. The sequence diagram on figure 1.7 shows an example with four devices which are two controllers and two RIOMS. In this example, each controller scans the two RIOMs by only one broadcast message (on the figure 1.7, arrows number (1) and (2) for controller producer1 and (3) and (4) for controller producer2) which are sent to all devices. The sending of data from RIOMs (producer3 and producer4) to controllers uses also one broadcast message for all the controllers that need their data (here 5, 6, 7 and 8). Both broadcast and unicast communication can be used in producer/consumer model. In this study, broadcast is always used because it reduces communication delays.

1.4 Using general purpose communication technologies for fieldbus

The general purpose communication technologies refer to the ones developed for office automation networking. First, the more matured technology for fieldbus, Ethernet, is detailed. Second, the use of the protocols TCP and IP is discussed. Finally, different cooperation models are considered and their time performances are explained.

There are different technologies currently available to communicate on a network. Three major ones are:

- Ethernet, defined in IEEE 802.3 [2002], with different media (copper cables, optic fibre),
- WiFi for Wireless communication in LAN, defined in IEEE 802.11 [1999] and
- Bluetooth for short distance Wireless communication, defined in IEEE 802.15 [2005].

Some academic studies are already working on the introduction at field level (Miorandi and Vitturi [2004]) of the last two standards specifying wireless communication. However, they stay exceptions for fieldbuses due to the problems of magnetic interferences and non negligible rate of lost frames. As a consequence, these solutions are not studied in this work.

1.4.1 Using Ethernet protocol

General points on Ethernet

What is commonly known as Ethernet is a standard of IEEE society (IEEE 802.3 [2002]) named ”Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

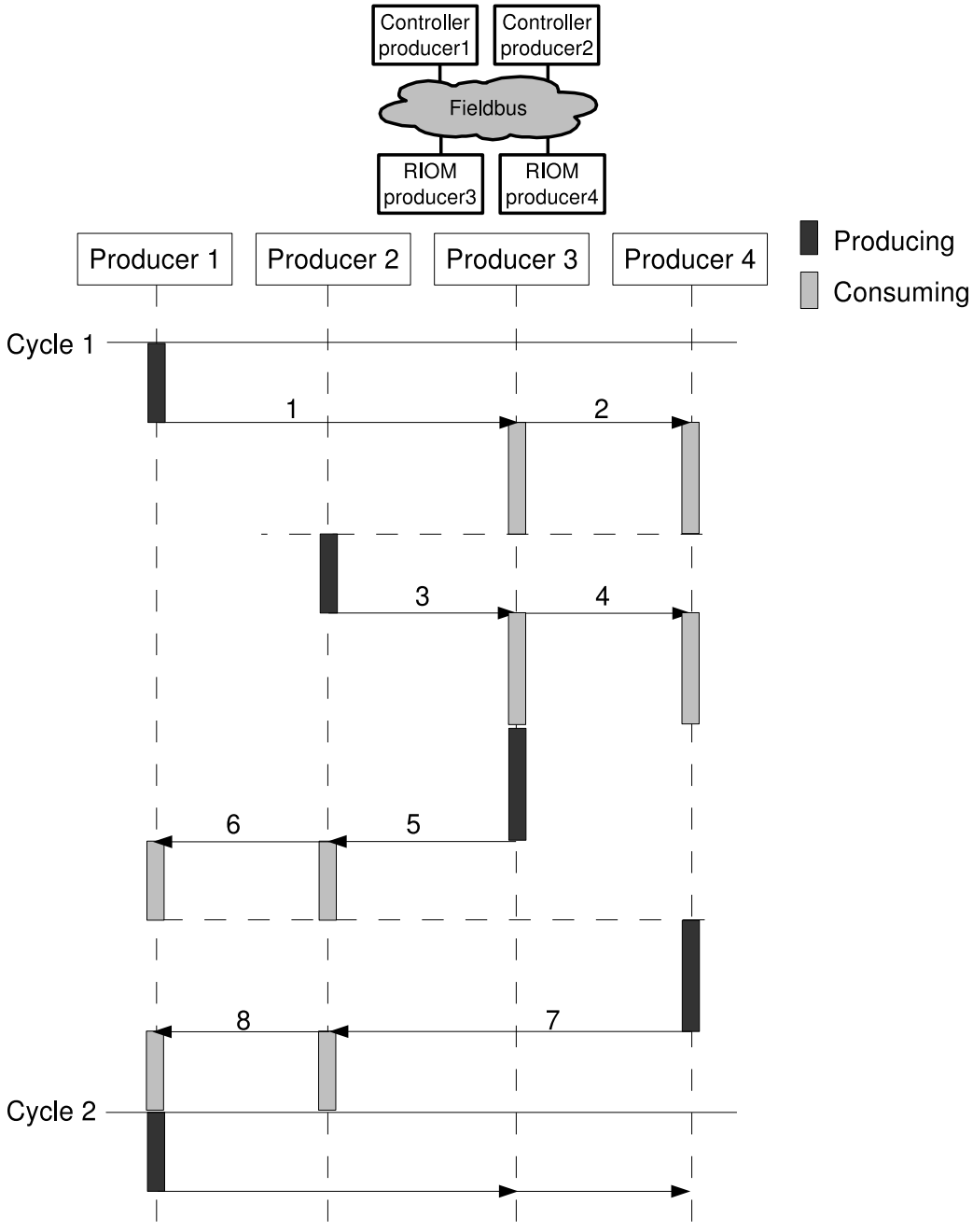


Figure 1.7: Sequence diagram illustrating Producer/Consumer model with four producers, which are two controllers and two RIOMs

access method and physical layer specifications". As mentioned in the title, Ethernet has two functions corresponding to the Data Link (access method) and the Physical layers. There are several physical layers specified in the standard. In this study, only the most mature technologies are considered: 10Base-T and 100Base-T which are respectively 10 MBytes/s and 100 MBytes/s transmissions on copper cables with RJ45 connectors.
The use of Ethernet implies the introduction of new components: hubs and switches. These devices have several ports to connect devices and transfer frames in the network.
Hubs transfer a frame arriving on one port to all its other ports whatever is the recipient. Consequently, everyone can listen to all frames which is useful when sending one frame to several recipients, i.e. when using broadcast. However if the communication takes place between only pairs of hosts (unicast), the use of hubs overloads the network. When using hubs, collisions can occur over the complete network which means that the collision domain covers the complete network.
Switches transfer a frame arriving on one port only to the ports linked with the recipients of this frame. To know what device is connected on which port, a table is built at the first exchange with each device. Consequently, only the recipients of a frame can received it which is useful when using unicast. It is also possible to use broadcast mechanism even if it is more time consuming. When using only switches (fully-swutched architecture), collisions can occur only in one link and if using full-duplex connections, i.e. if there is one wire in the cable for transmission and another for reception, collision cannot occur. There are two different switch technologies: "cut through" and "store and forward". The first type begins to forward a frame when it arrives without checking its validity. The second type, "store and forward", first stores the entire frame that arrives, then checks it and, if correct, forwards it to the next device (Lee and Lee [2002]). As a consequence, "cut through" switches are faster than "store and forward" ones but these latter ones enable not to overload the network with damaged frames.

Ethernet in fieldbuses

For the use of Ethernet in fieldbuses, most solutions now available on the market are only using the physical medium with specific protocols developed that are called "real-time". These protocols implement on-line traffic control and centralised resource management without using the media access control of Ethernet (figure 1.8). Actually, they use a master/slave or a producer/consumer model, with coordination between masters or producers, as in specific purpose fieldbuses presented in section 1.3.2, page 12. As a result, there is always only one message on the medium and hubs are prefered to switches. This kind of solution is chosen by some companies, such as Siemens with ProfiNet and Rockwell Automation with EtherNet/IP, or by associations, such as Ethernet Powerlink Standardisation Group with Ethernet

PowerLink.
However, these specific protocols are neither open nor interoperable with general purpose networks. Moreover, they often need dedicated hardware to be really efficient.

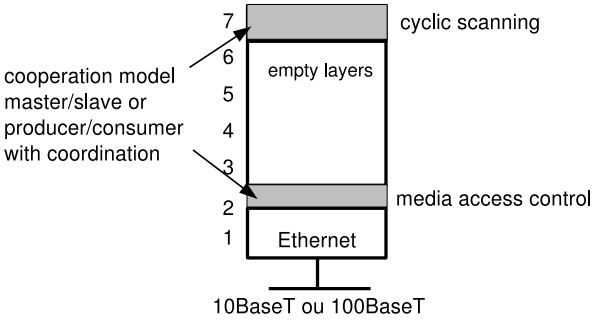


Figure 1.8: Master/slave and producer/consumer models represented in the OSI model

In the following, the term "Ethernet-based" concerns only the systems that are totally conform to the Ethernet standard both in Physical and in Data Link layers. This work focuses only on Ethernet-based Automation Systems.

1.4.2 Using TCP and IP protocols

The general purpose networks on Ethernet mainly use the Transport Control Protocol (TCP) and the Internet Protocol (IP). The function of TCP (University of Southern California [1981b]) is the transport of data by establishing a connection for each transmission between two hosts and corresponds to the Transport layer of the OSI model. TCP ensure reliable communication by the use of acknowledgement messages. These messages tend to increase the traffic on the network and can be a cause of congestion. To avoid this problem TCP integrates algorithms to reduce congestions.
The function of IP (University of Southern California [1981a]) is to communicate in interconnected systems by means of fixed length addresses and correspond to the Internet layer of the OSI model. Presentation and session layers are not implemented in general with TCP and IP while the application layer protocol depends on the user application. The succession of the protocols is performed with the encapsulation of frames, adding at each layer a specific header. The figure 1.9 shows the mapping of Ethernet, TCP and IP on the OSI model .

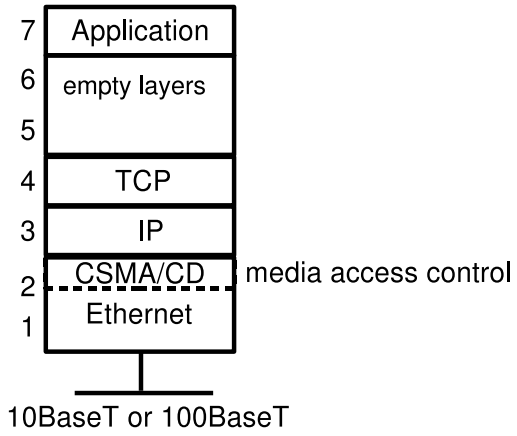


Figure 1.9: Ethernet, TCP and IP represented in the OSI model

1.4.3 Using the client/server cooperation model

The cooperation model mainly used with TCP and IP protocols is the client/server model. In this model, client devices send requests to server devices, thereafter servers can reply. This cooperation model takes place commonly in the application layer, and the media access control is ensured by CSMA/CD in Ethernet (figure 1.10).

The client/server model begins to be used in fieldbuses. This is the case when using Modbus-TCP protocol (Modbus-IDA [2004]), an open implementation of Modbus fieldbus protocol on Ethernet and TCP/IP using a client/server cooperation model. It uses classic Modbus function codes consisting in "Read", "Write", "Read and Write" and an indication on adress and number of bytes to transmit. This solution is already used by one manufacturer (Schneider Electric [2006]) and can be implemented on any standard TCP/IP implementation.
It matters to note that the client/server model does not ensure the regular refreshment of data in devices. For this, another protocol must be added in the application layer to ensure a cyclic scanning of the RIOMs by the controller, called IO scanning.

As our interest is to apply this model in automation systems, this behaviour is now illustrated for the particular case of automation system. The client/server model implements controllers as clients and RIOMs as servers. Thus the controllers request data from Remote IO Modules and then these latter can answer. This behaviour is illustrated on the sequence diagram 1.11, with two clients implemented in controllers (Client1 and Client2) and two servers implemented in RIOMs (Server1 and Server2). The different grey levels on the diagram are present to enable an easy reading.

This illustration enables us to understand both the advantage and the shortcoming of this model. Compared to master/salve and producer/consumer models,

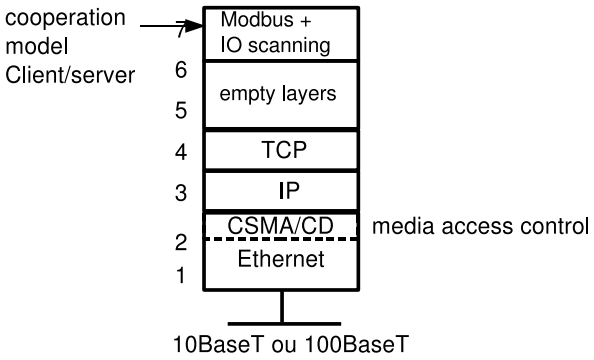


Figure 1.10: Modbus on Ethernet, TCP and IP with client/server model represented in the OSI model

which schedule and order all sent messages without enabling any parallelism, in the client/server model, all clients and servers can send messages at the same time. Therefore, in best-case, the cycle time of each client is minimal, whatever the other ones are doing. This is shown on the figure with the two clients that do not begin their first RIOM scanning cycle (noted "Cycle1 Client1" and "Cycle1 Client2") one after the other. The two cycles of clients are independent. However, the consequence is that if resources are shared, as the switch SW1 and the RIOMs on figure 1.11, messages can be delayed by waiting in buffers. On the sequence diagram, there is first the request from Client2 (arrow number 2) that must wait first of all in the switch before to be treated (rectangle number 2), for the request of Client1 (arrow number 1) to be forwarded (arrow number 5). Then after being forwarded to Server1 the request from Client2 (arrow number 6) has to wait once again for the one from Client1 to be treated (rectangle number 5).
As a consequence, a dilemma of automation system architect is to choose between a well-known and deterministic but probably slow architecture and a new and non-deterministic architecture but probably faster.

To help architects, in the chapter 6.2 page 91, we will perform a comparative study of the three cooperation models:

- master/slave
- producer/consumer
- client/server

on the basis of the network cycle time. The aim of this study is to determine whether the client/server model can lead to shorter network cycle time than the two other models or not.

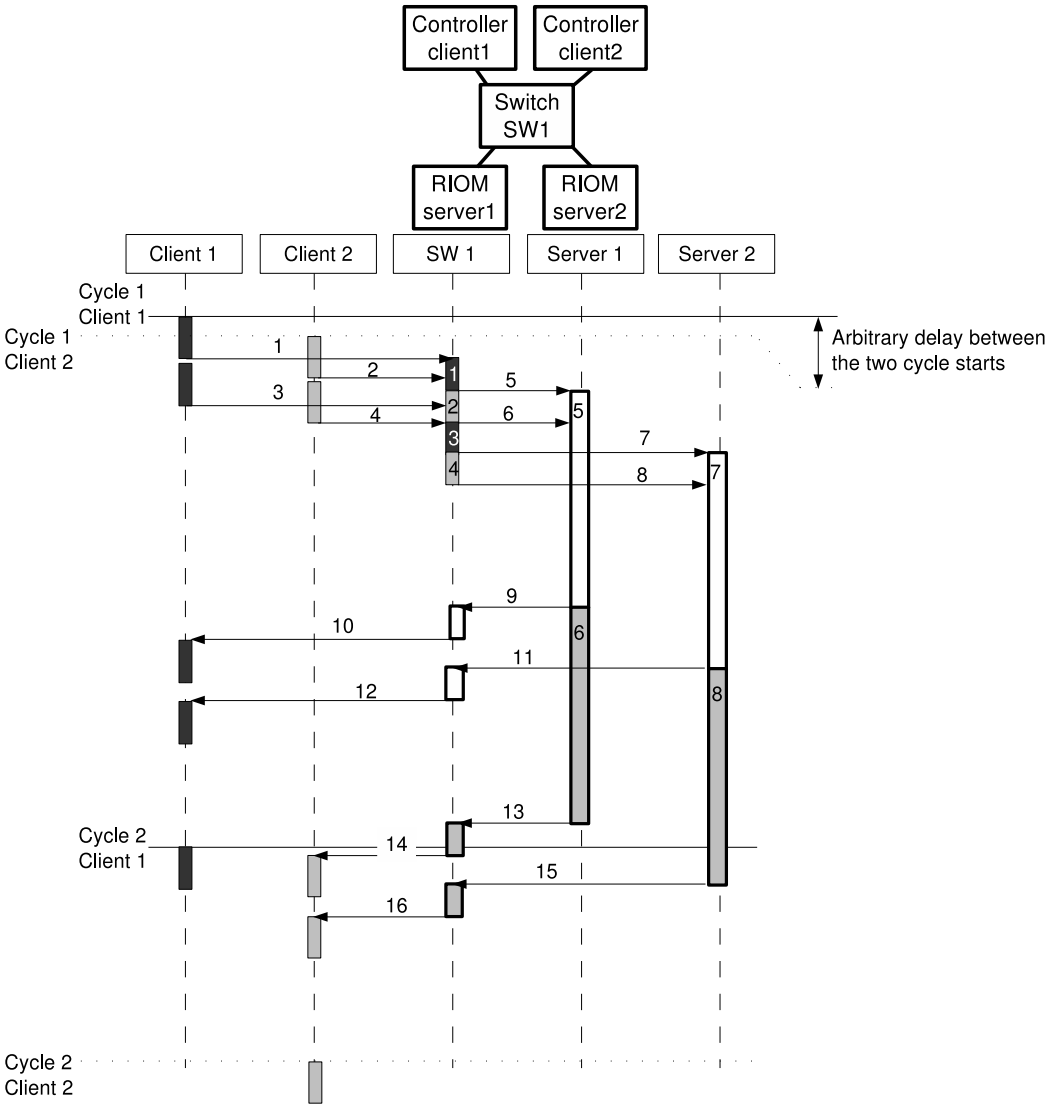


Figure 1.11: Sequence diagram illustrating client/server model with two clients and two servers

1.4.4 Impact on time performances of switched Ethernet-based Automation Systems with a client/server model

Before concluding this chapter, it matters to study with more details the different time consumption mechanisms in switched Ethernet-based Automation Systems with a client/server model and to show how they impact the response time and the network cycle time.

Response time

When using a client/server cooperation model, the response time can be defined more precisely, as in Jasperneite and Neumann [2001], as the round trip time from client to server and back to client including a thinking time in the server. Here, we have already defined the evaluated response time as the delay between the occurrence of a "cause" event (I1) and the occurrence of the "consequence" event (O1). It is also a round trip time but from server to client and back to server including thinking time for each client and server. In general this time is not constant because of parallelism between cyclic processes: occurrence of event, user program execution and cyclic scan of remote IOs.
If the fieldbus has non-deterministic access to media and local resource management, as in our case, another non-constant source of delay is the waiting time for availability of resource.

On the architecture given figure 1.12, the analysis of the response time is proceeded from the occurrence of a cause event I1 to its effect on the process O1 after being processed by a PLC (CPU1).

Network cycle time

It matters to well define the network cycle time when using client/server architectures. When using master/slave, with coordinating masters, or producer/consumer models as in specific fieldbuses, in one network cycle time is taken into account communication among all devices. For instance, in a system composed of two masters and four slaves, as on figure 1.6, Cycle1 is one network cycle which includes both Master1 and Master2 scanning. In this case, the network cycle time is a static and intrinsic data of the architecture.
At the contrary, when using client/server model, there is one network cycle time per client. On the figure 1.11 which presents a system with two clients, there are two network cycles, "Cycle1 Client1" and "Cycle1 Client2".

As a consequence, the network cycle time should then be shorter when using client/server model compared to the one when using master/slave or producer/consumer

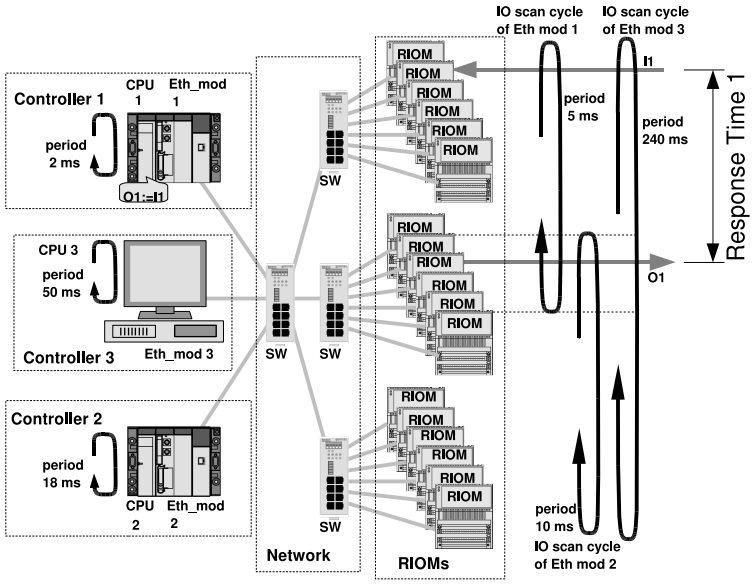


Figure 1.12: Example of response time studied

model. However, the parallelism of network cycles implies to share the communication medium and so it introduce the waiting time for availability of resource, which is a variable and dynamic delay. *In this case, the network cycle time is not a static and intrinsic data of the architecture but a dynamic parameter.*

To help architects, in the chapter 6.2 page 91, we will perform a comparative study of the three cooperation models:

- master/slave
- producer/consumer
- client/server

on the basis of the network cycle time. The aim of this study is to determine whether the client/server model can lead to shorter network cycle time than the two other models or not.

Time consumption mechanism in switched Ethernet-based Automation Systems when using client/server model

Figure 1.13 illustrates this complex behaviour, showing for each component on a time diagram the time spent by information in components.
In the client/server model, the information "occurrence of I1" waits for a request

of the PLC to be transmitted to this last one. Then the information is encapsulated in a Modbus-TCP-IP-Ethernet frame in response to the PLC. After crossing all necessary switches, here two, the frame is received in the Ethernet module of the PLC. There, the treatment of the information is possible at the next Input reading phase of the PLC cycle. Next, at the end of the cycle, consequently to I1 occurrence, O1 state changes. This information "occurrence of O1" is encapsulated in a Modbus-TCP-IP-Ethernet frame that is sent to the recipient RIOM at next scan cycle. Finally the frame crosses the switches until the RIOM and after being processed, the state change effectively occurs on the plant.
The previous description of the behaviour to be exact must be enhanced with the delays due to shared resources. Indeed, using client/server protocol on TCP/IP and Ethernet results in the local management of shared network components. For instance, two RIOMs can send their answers to the Ethernet module of PLC at the same time. In this case, these messages are first queued in the switch, that can only send messages one by one on a port. So, it sends a first message to the Ethernet module. Then, if the second message arrives before the end of the processing of the first one, it is queued in a buffer of the Ethernet module. The message is thus delayed by waiting for availability of Ethernet resources.
Figure 1.13 gives an example of possible evolution and distinguishes the three different causes of delay:

- processing data,
- waiting for synchronisation and
- waiting for availability of resources.

Therefore, the response time should be modelled as the sum of these three types of delays in each component. As a consequence, the first attempt to evaluate the response time is to perform analytically this sum, and for that to evaluate the delays in each component. Concerning processing delays, the evaluation is easy because they are static in steady state, i.e. for the same parameters, at any time, the processing delay is equal. For instance, the execution of the user program in PLC is always the same for a given set of data in a given state, as well as the forwarding of an Ethernet frame in a switch is constant for a given frame length if the destination address is known. The two other delays, waiting for synchronisation or for availability of resources, are more complex to determine because they are dynamically depending on the history of the system. Figure 1.14 details the delay of waiting for synchronisation in the RIOM coming from the Ethernet module of the PLC. This synchronisation takes place when the request for the value of the input corresponding to the cause event arrives. This arrival date depends on the date of sending the request and the delays spent in switches. These delays depend on the rest of the system. On figure 1.14, to illustrate this dependency, we suppose that three messages have just arrived in one switch before the request and are currently forwarded or waiting for. The problem is to determine how many messages are waiting in this queue, given that it

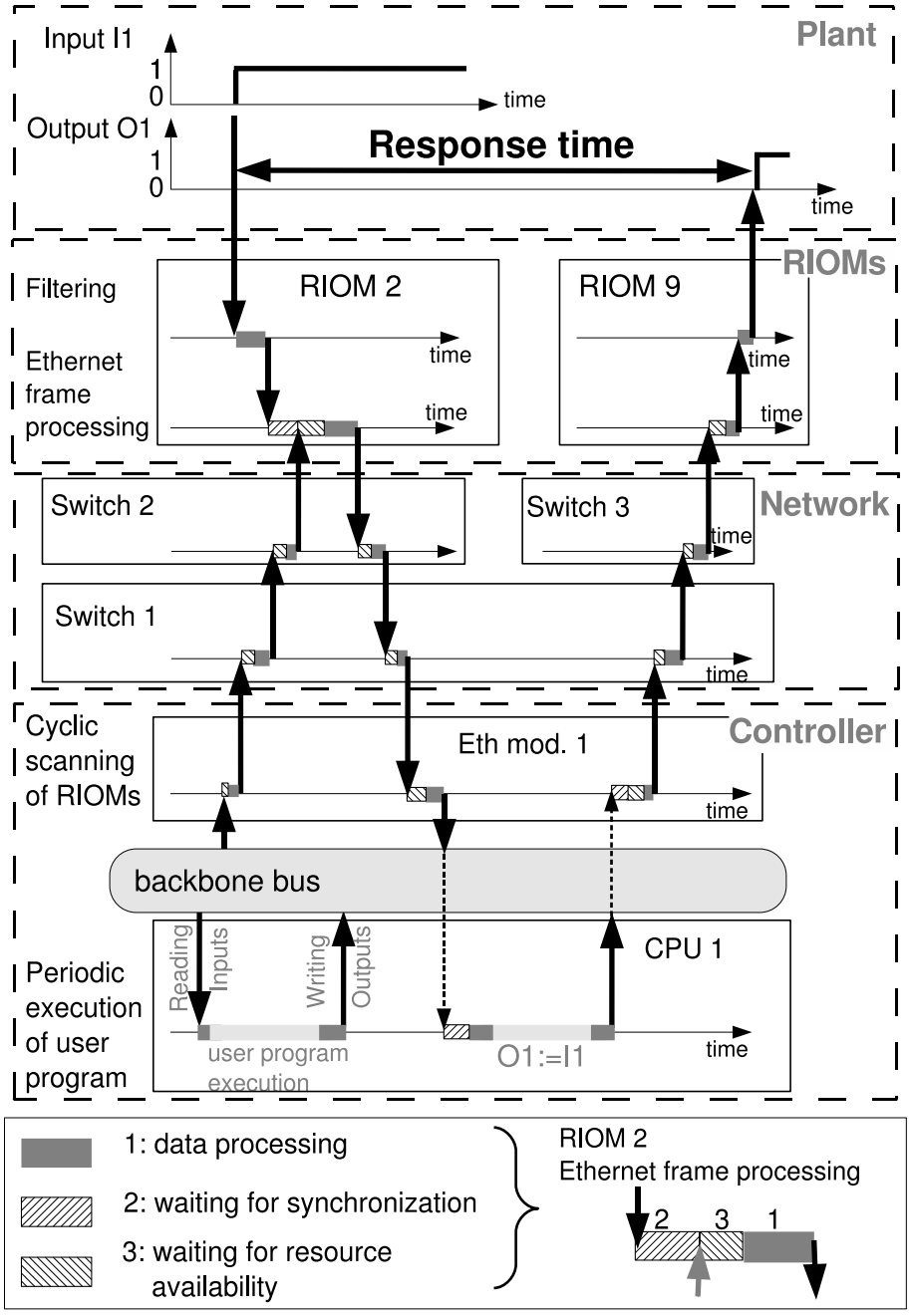


Figure 1.13: Decomposition of response time in components represented on time diagrams

depends on the traffic generated by the other components, its amount and its time characteristics. In network calculus, this corresponds to determine the arrival curve of the switch. This task is rather hard because a message arrival date in the switch is dependent on other previous message arrival dates and also on delays in other components. The larger the system is, the trickier is this analysis. Therefore, we can not assume to conduct it manually on automation systems.

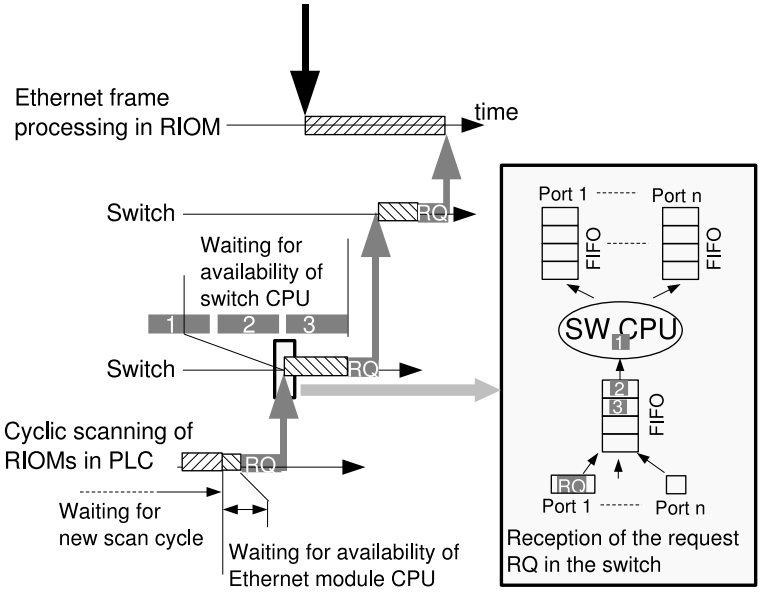


Figure 1.14: Example of delay of waiting for synchronisation in RIOM

Conclusion

In this chapter, we have presented the hardware devices studied (controllers, switches and RIOMs), Ethernet,TCP and IP protocols, and three existing cooperation models that are available for Ethernet-based Automation Systems. The first two, master/slave and producer/consumer, are inherited of common fieldbuses. They have the advantage to give a fully deterministic behaviour to the system because parallel communications are forbidden. The third model, client/server, is widely used for Local and Wide Area Network. It does not provide a fully deterministic behaviour because parallel communications that involve shared resources are possible.

Two main time performances of Networked Automation Systems have been defined: the network cycle time and the response time (figure 1.15). The first one is a criterion to compare different networking solutions. Consequently, the three presented cooperation models are going to be compared on their Network Cycle Time

in chapter 6.2. The second one, on which focuses the end user, indicates the performance of the whole automation system. This performance will be evaluated on the solution using Ethernet, TCP/IP and client/server model of cooperation.

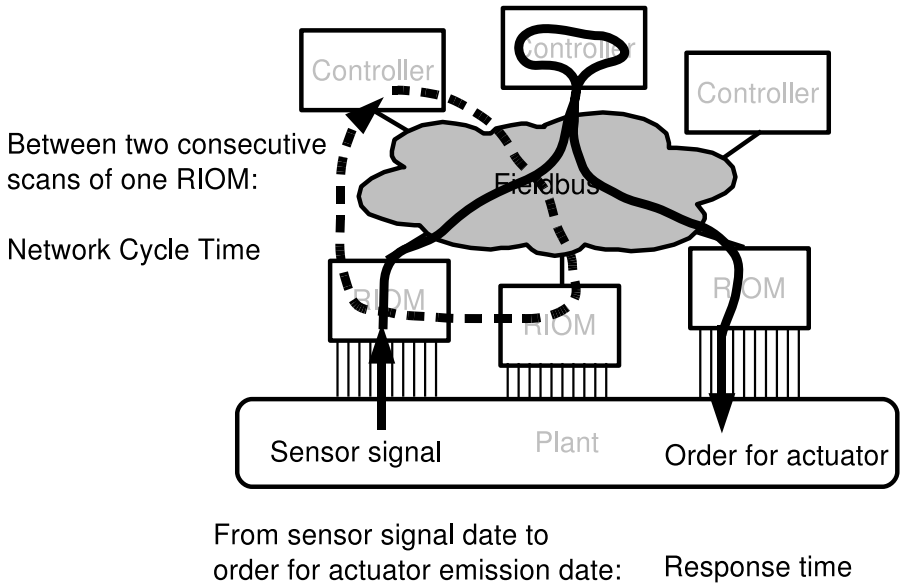


Figure 1.15: Networked Automation Systems Time performances that are going to be evaluated in this thesis

The next chapter, first, details several major scientific works about the determination of time performances of Networked Automation Systems and, based on these explains the approach we have chosen to assess these performances.

Chapter 2

Methods for evaluation of time performances in Networked Automation Systems

This chapter first details existing works to evaluate time performances in Networked Automation Systems. These works can be distinguished following the kind of method used: either ”a priori” or ”a posteriori”. The first kind of methods aims to evaluate the performances of a system during the design phase, before it exists, and consequently it requires models of the system. The second kind of evaluation methods aims to evaluate the performances of a system during running phase, when it already exists, and consequently it consists in measurements. Then, based on the analysis of these works, the proposed approach to evaluate time performance in Networked Automation Systems is presented.

2.1 A priori evaluation

These methods can be sorted in three types: analytic calculus, formal verification and simulation.
Analytic calculus consists in solving a set of equations that contain delays introduced by the different components. In the case of time performances of Automation Systems, this method is mainly used to determine upper-bound delays and use static equations, i.e. equations where the delay values are invariant. For the calculation of a value, the evaluation of the worst-case values of each term need to be performed preliminary.

The mainly used formal verification technique is model-checking. A model-checker is a model-checking tool that verifies that a property is true or false when exploring exhaustively the state space of the model. For this, one must design a

model of the system and a model of the property to verify. Most tools use a class of automata for the system model and timed or untimed temporal logic for the property model. For evaluation of time performances, timed automata (Alur and Dill [1994]) and Computational tree logic (CTL) are mainly used. The advantages of model-checking are first to give bounds on complex system that are difficult to model analytically. The problem of model-checking is the state-space explosion that drastically limit the size and the complexity of treated problems.

Simulation is a widely developed method. It consists in a partial exploration of the state space of a model. The modelling languages and available tools are numerous and in general adapted to a domain or a class of problem. For automation systems modelling and simulation, it is common to work with Petri Net models. Actually, this language is appropriate to model Discrete Event Systems with synchronisation and time aspects. The advantages of this method are to be not subject to state space explosion compared to model-checking, and to be able to produce a distribution and not only bounds. The main forthcoming of simulation is that it gives only a partial view of the real system behaviour. Indeed, if it is quite easy to have a precise estimation of mean values, that is more difficult for the bounds. An attempt to evaluate the precision of the bounds obtained by simulation and a calculus of the number of simulations to obtain these bounds are presented in Meunier [2006]

2.1.1 Methods based on analytic models

Analytic calculus is often used for determination of Worst-Case Delay of part of architectures. Tovar and Vasques [1999, 2001] and Vitturi [2001], propose analytic calculus of network cycle time on master/slave and producer/consumer models. The calculus is obtained by the sum of elementary delays related to the protocol. Only synchronisation between cycles is a source of non-constant delays.
In Tindell [1993], the author provides an analytical method to determine response time for distributed systems based on Worst-Case Execution Time analysis in processor (Puschner and Koza [1989], Colin and Puaut [2000]), resulting in a Worst-Case Response Time analysis. This method is finally adapted by Pereira et al. [2004] to response time (called by the authors end-to-end delay) in control system with remote IOs using producer/consumer cooperation model. It is a static method, very similar to the one proposed by Tovar and Vasques [1999, 2001] and Vitturi [2001], that relies on the sum of Worst-Case Delays in each device and on the network. Thus, results obtained could be very pessimistic because Worst-Case Delays of the components are not independent, and so there is low probability that they all happen simultaneously. In all these cases, analytic calculus gives realistic results because the authors consider networks in which the media access method is driven by master/slave or producer consumer models and therefore there is no shared resource. This not the case when using client/server cooperation model with which shared resources can be a non-negligible source of delays. Actually, to determine a worst-case network cycle

time with client/server model, analytic calculus as proposed in the previous works implies to evaluate a maximum value of the delay due to waiting for availability of resource, which often gives very pessimistic results.

For switched Ethernet using CSMA/CD, Lee and Lee [2002] propose a calculus based on analysis of time diagram of the Data Link Layer. The method employed consists in summing the elementary delays during the communication, presented on the time diagram on the figure 2.1:

- DPS is a processing delay for transmission at the source,
- DPR is a processing delay for reception at the destination,
- DT is a frame transmission delay,
- DPROP is the propagation delay for the electrical signal to propagate from the source to the switch and from the switch to the destination,
- DIF is the interframe delay, and
- DQ is the delay spent by the frame in the queue of the switches.

The authors conclude that switched Ethernet delay is small and has little variation. Finally they carry out an experiment to check the effect of switched Ethernet on stability of Networked Control System. The response is similar than for a point-to-point control system. As a consequence, the authors consider switched Ethernet as a promising network for industrial application.

In a different way, Song [2001] uses a more complex analytic method, queueing theory, for determining delays in switched Ethernet. The author focuses on determining buffer delays in switches. Indeed, in switched Ethernet, the switches are shared resources and so when a frame arrives, it possibly has to wait for the CPU to end its current task. The contribution of this work is mainly the modelling of delay in switch (figure 2.2). On this figure, we can notice that in this model the switch has only output buffers. Only the output ports are shared resources and not the CPU. In our switch modelling, the switch CPU is a shared resource of the switch, and so input buffers will be modelled.

More recently, Miorandi and Vitturi [2004] also used queueing theory to evaluate time performance for producer/consumer model implemented on wireless network. This work takes into account time jitter and packet error probability. The authors conclude on the good performances of the system, enabling industrial application. Queueing theory is adapted to model shared resources but to use it one must describe the behaviour with probability law and it yields only mean values. So, this method is more adapted for systems with local non deterministic behaviour, such

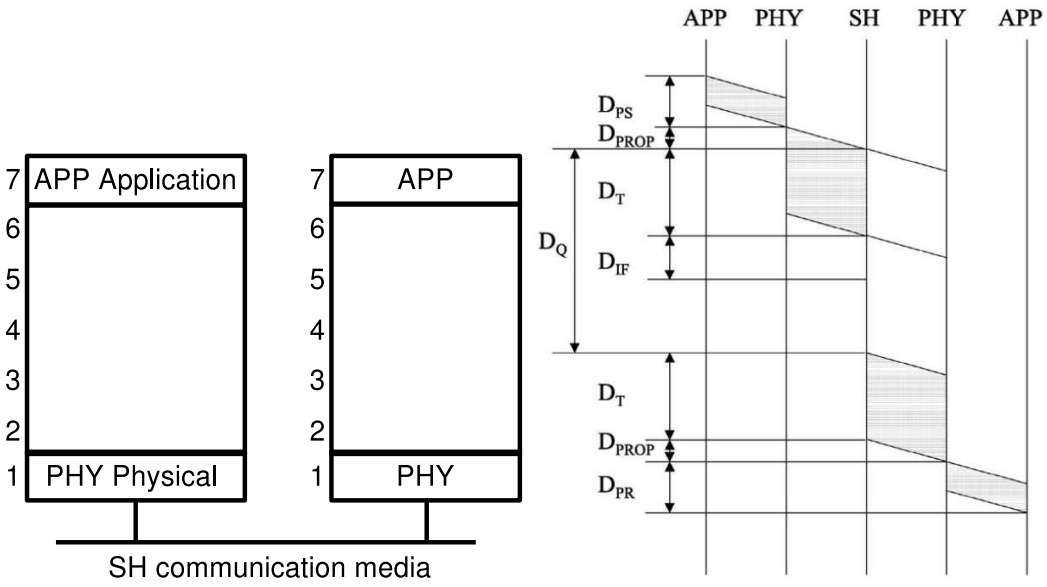


Figure 2.1: Time diagram of the Data Link Layer in Lee and Lee [2002]

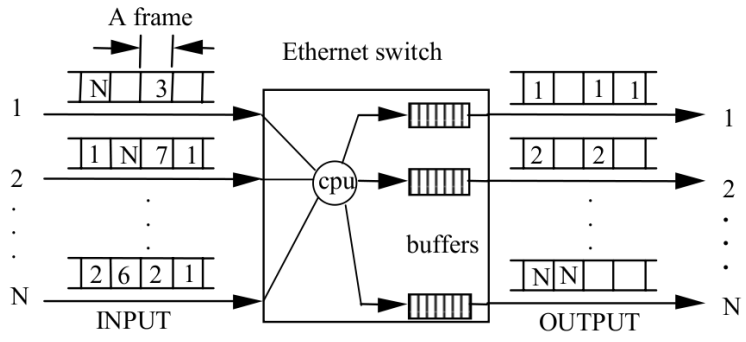


Figure 2.2: Model of switch proposed in Song [2001]

as packet lost probability in wireless network, which is not the case for the studied system.

The most recent approach to calculate analytically network device delays is the network calculus introduced by Cruz [1991] and developed by Le Boudec and Thiran [2001]. This analytic approach aims at obtaining maximum and minimum delays in network components (buffer, multiplexer, demultiplexer, queue) modelled as leaky buckets using Minplus algebra (figure 2.3). Network calculus implies to define for each component an arrival curve R(t), which models the arrivals in the bucket, and a maximum level of the bucket b, and an output r.

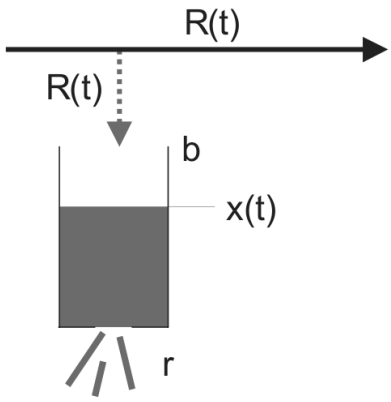


Figure 2.3: Leaky bucket with R(t) the arrival curve, b the maximal level and r the output, from Le Boudec and Thiran [2001]

An application to calculate the maximum end-to-end delays in a cascade of switch is presented in Georges et al. [2002], considering both periodic and aperiodic traffic of automation system. The figure 2.4 presents their modelling of switches, composed of one multiplexer, one buffer, one ASIC and one demultiplexer. In this work the arrival curve in the switch is defined by the traffic, which must then be known, the maximum level is defined by the buffering capacities in switches and the output by the output ports throughput. A large part of this work has been to formulate in Minplus algebra how to calculate a upper-bound delay not in only one component but with several components.

Jasperneite et al. [2002] also use network calculus to obtain the upper-bound of worst-case transaction time and conclude that in this point of view Ethernet with CSMA/CD access method can be used for real-time systems. Network calculus gives accurate results compared to experimental ones, however it deals only with sharing resources and not with synchronisation between parallel processes. So it cannot be used directly to determine response time in Networked Automation System.

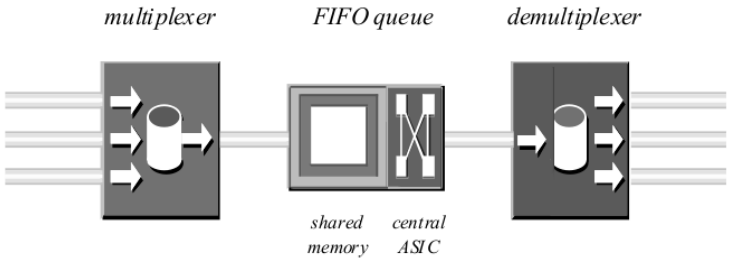


Figure 2.4: Model of switch proposed in Georges et al. [2002]

2.1.2 Exhaustive state-space exploration of models

If many researchers have addressed the issue of time performances evaluation by using analytic approaches, few of them have investigated the possibility of formal verification techniques. Nevertheless Ben Hédia et al. [2005] propose a method based on formal verification of timed communicating automata models for data acquisition systems with parallel processes. Indeed, this technique enables to have a exhaustive exploration of the state-space of the model, and so it should cover the whole real behaviour. However the authors point out that state space explosion problem appears promptly when including time intervals on transition even if the models are small (figure 2.5). The system studied is composed of one sensor and one Real-Time Operating System, which contains a communication interface, a device driver and a real-time application. The communication interface connects a sensor to the software part of the system while the device driver is a dedicated software integrated into the operating system and independent of the application.

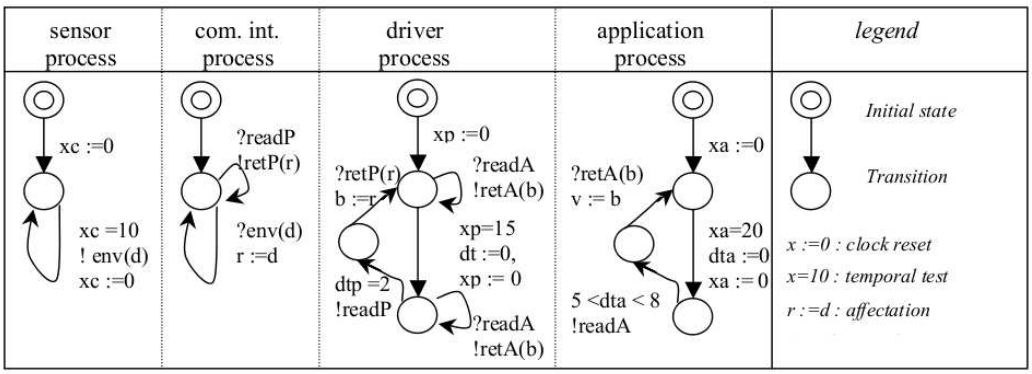


Figure 2.5: Models proposed in Ben Hédia et al. [2005] for the time model checking of acquisition systems

Timed model checking is also used in Krakora and Hanzalek [2004] to verify temporal and logic properties of distributed real time control system over a CAN bus. In this study, authors consider two types of shared resources: processors and bus.

However, the processes in each processor are synchronised and there is an arbitration to schedule the access to the bus for the different processors, as a consequence, the delay of "waiting for synchronisation" between asynchronous processes does not exist. The authors determine on a case study, with four processors on a CAN bus, the worst case response time. For this, they must repeat the verification iteratively for different values of deadline. The authors conclude that it is possible to design a whole distributed system model by a modular approach. However, the study case presented is very small and even if the model is designed easily, it would be interesting to know the maximum number of components before occurrence of state-space explosion.

In Greifeneder and Frey [2006], Probabilistic Model Checking is used to determine the effective stop position of a trolley when it is controlled via a Networked Automation System. The chosen Probabilistic Model Checker implements an extension of Computation Tree Logic called Probabilistic Computation Tree Logic to specify properties over systems described by Markov models.
The system studied is composed of two inductive sensors, one controller, one trolley and one motor. The sensors detect the trolley moving to the stop position, and are connected to the controller via a wireless TCP/IP Ethernet-based network, without packet losses. The controller sends the order to slow down the trolley motor also via the network. When the controller is aware that the trolley has been detected under the first sensor, it must send a signal to slow down the trolley motor from normal speed to slow mode. Then, when the controller is aware that the trolley has been detected under the second sensor, it must send a signal to slow down the trolley motor until it stops.
The authors have considered several pairs of sensor positions and have find out the optimum pairs to minimise the probability of stopping outside predefined tolerance bounds. This work has shown that Probabilistic Model Checking can be a promising tool to optimise Networked Automation System features to comply end user requirements. However, the problem is always the state space explosion, limiting the size of problem that can be treated.

To sidestep the state-space explosion problem, a solution consists in designing a minimal equivalent model or by decomposing the problem in smaller independent ones. If both solutions are currently under investigation, there is not yet an effective one.

2.1.3 Partial state-space exploration of models

For networked and automation systems analysis, two kinds of simulation tools are distinguished:

- software tools for network analysis, such as OMNet, and

- general purpose simulation tools based on Discrete Event System theory.

A use of network simulation tools in the analysis of networked automation system is proposed by Pereira et al. [2004]. It aims to determine to determine by simulation realistic assumptions for analytic model. This first work presents evaluation of response time by simulation of an automation system on Ethernet-IP (Rockwell Automation) using a producer/consumer model, and compares the results to the analytical ones obtained with Worst-case Analysis. The authors choose to implement a model in a network simulator, here OMNet++ (Varga [2001]), to have detailed models for protocols. Indeed many versions of TCP, IP and Ethernet are already modelled in these tools designed for their analysis. In this work, the authors want to highlight the importance of simulation approaches to complete worst-case analysis because simulation yields less pessimistic results.

The use of general purpose simulation is more developed in scientific community. We focus here on works dealing with networks and protocols.
Simulation is also used for evaluating response time in switched Ethernet LAN, as in the recent works of Zaitsev [2004a,b], who proposes to model this network with Timed Coloured Petri Net in the simulation environment Design/CPN University of Aarhus [2006c], and in its successor CPNtools (University of Aarhus [2006a]). Petri Net-based model is chosen for its ability to represent dynamic behaviour such as client/server cooperation. The model of the system is composed of different submodels representing switches and workstations. Evaluation of response time is performed by an additional Petri Net called measurement workstation. The figure 2.6 presents the model of switches proposed by the author. The colour is used here to model several switches on this Place-Transition structure. Each switch is composed of several ports, of one buffer and of a switch table. However, if we simulates this switch, we can notice that when the switch table is treating a message from an input port, it stays available for messages from other ports. *As a consequence, the switch table is not considered as a shared resource.* In this model, two messages that arrives at the same time on two different input ports, that have to be forwarded on the same output port, will be forwarded at the same date. This behaviour is not possible, indeed, two frames cannot be emitted at the same time on the same link.

In Brahimi et al. [2006], Coloured Petri Nets are also used to model switches, but the goal is here to evaluate the impact of different scheduling policies. This work is carried out in the followings of the one of delay in switches by network calculus (Georges et al. [2002]). As a consequence, the switch model is split into elementary components: one input FIFO queue, one demultiplexer, one output FIFO queue per priority level and one scheduler. In a studied case, the switch is associated to six message producer and six consumers. The traffic load in the switch is given by the producers and is defined periodic to represent sensor or actuator sampling period. This study enables to highlight that depending on the scheduling policy, messages can be lost.

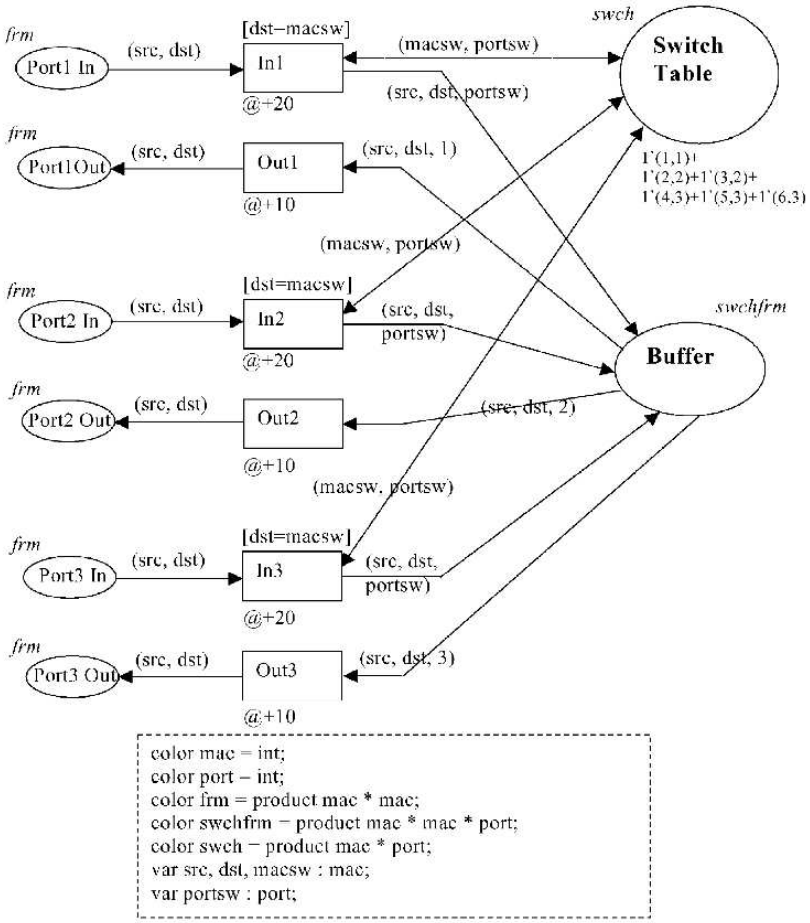


Figure 2.6: Model of a switch proposed in Zaitsev [2004a]

Other studies concern protocols modelling using Petri Nets. As an example, de Figueiredo and Kristensen [1999] propose a Hierarchical Coloured Petri Net model for TCP protocol in order to evaluate the behaviour of different TCP version in presence of frame losses. A more recent work by Bitam and Alla [2003, 2005], also on TCP protocol, proposes to use Hybrid Petri Nets to model this protocol (figure 2.7). In this modelled system, two emitters (E1 and E2) have to share a resource (Rt). The Hybrid Petri Net models effectively this shared resource thanks to the place C. Actually, when transition T3 or T10 are fired a token is removed from C and will be available only when T5 is fired, i.e. when Rt is free. This modelling of shared resources is possible with any class of Petri Net and shows that this language is well-suited to model this mechanism.

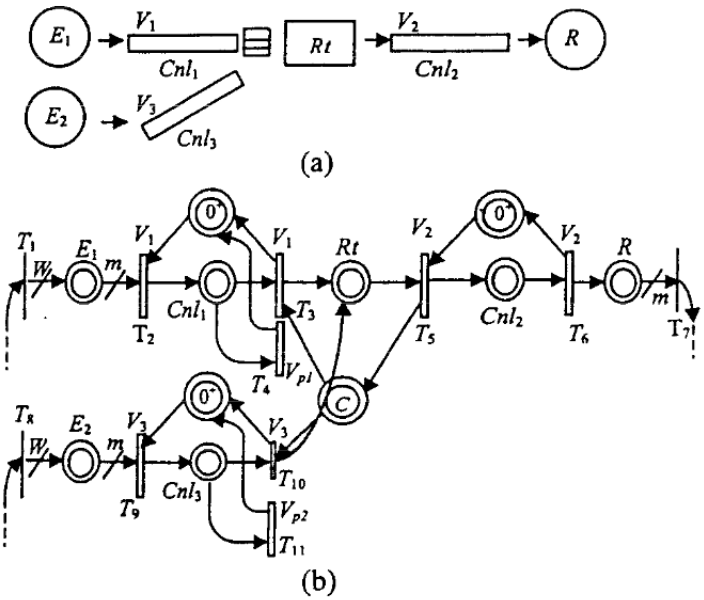


Fig. 9. (a) Two emitters connected. (b) Hybrid PN model of a line of communication with two emitters connected.

Figure 2.7: Hybrid Petri Net for modelling TCP protocol in Bitam and Alla [2003]

The goal of these two previous works is to show the interest of Petri Nets to model networked systems that present shared resources and parallel processes such as Ethernet-based Automation Systems with client/server cooperation model.

2.2 A posteriori evaluation: Experimental methods

Concerning Networked Control systems, Seuret et al. [2006] shows how measurement of communication delays using GPS can be used to stabilise the system on-line. Concerning networked automation systems, if many works have addressed a priori evaluation of time performances, either using analytic models, formal verification or simulation, very few researchers have considered a posteriori evaluation, i.e. use of experimental methods to obtain time performances. To our knowledge, only three research teams investigate deeply this way.

First, Ferrari et al. [2004, 2006], Depari et al. [2006] from University of Brescia (Italy) are developing their own instrument to be able to measure accurate time in Ethernet networks. Their instrument captures frames at different points in a network with different probes, that are being synchronised by an external source. It enables to measure any kind of delays in a network, whatever are the protocol and the cooperation model used.

Second, Cena et al. [2006] from University of Torino (Italy) proposes an experimental method to measure latencies in Ethernet. The goal is to measure common COTS Ethernet boards latencies. The authors show the difficulty to analyse the results due to the different behaviours of each driver. If the results are different according to the hardware tested, it gives a good order of magnitude for delays in Ethernet boards that can be used in other works.

Third, Parrott et al. [2006] from University of Michigan (USA) investigates more deeply the delays induced by the application layer in three types of communication:

- UDP, User Datagram Protocol, a simple transport layer protocol,
- OPC, OLE for Process Control, an open application level communication protocol, and
- VPN, virtual private networks, which create secure "tunnels" to transfer data between networks.

The authors focus on this delay because, with the network speeds that increase, application layer delays can become more important than network delays. As a consequence, application layer delays must be taken into account but they are often not specified.
The authors conclude that application layers are effectively more important than network delays. Indeed, the application layers add significant delay when OPC is used and VPN delays are more than double the UDP delays.

2.3 Proposed method for time performance evaluation of switched Ethernet-based Automation Systems using client/server model

In the previous chapter, two time performances of interest have been identified: the network cycle time and the response time. The first one enables us to compare different solutions and the second one enables us to validate a solution with regard to end user requirements. Both performances are very interesting to determine during design phase so we choose an "a priori" approach. The experimental methods will be used to determine some elementary delays that are nor documented.

Within the three kinds of "a priori" approach, the previous survey has shown that the various analytic calculus methods are not adapted to obtain a delay distribution, but only bounds. Even for bounds, if simple analytic calculus can give a good evaluation of best-case and worst-case response time for master/slave and producer/consumer models, it is not the case for client/server model. In the last one is introduced shared resources, where several clients and servers can communicates with each other at the same time, as it has been presented in the previous section. In Marsal et al. [2006b] and Marsal et al. [2006a], we have shown that neglecting the shared resources mechanism to have a formulation near Tovar and Vasques [1999, 2001] and Vitturi [2001] can lead to false bounds. Evaluating a maximum value for the delay induced by waiting for availability of shared resources can be performed but it gives highly pessimistic results.

To use directly other more accurate formal methods, like model-checking, is still not reasonable given the size of the system and the granularity required. Indeed, if we want to model with same granularity all the phenomena (user program execution, IO scanning cycle, switch behaviour and RIOM behaviour) state-space explosion is easily reached. This has been shown in Poulard-Marsal et al. [2005] and Witsch et al. [2006], where the model checking is limited to network cycle time verification on small-size architectures.

Consequently, the solution chosen for client/server model is to evaluate time performances by simulation. Indeed, this last one is the only method that could give the required distribution of delays.

Within the simulation of a dynamic model, i.e. a model that can represent all possible evolution of the modelled system, two approaches have been mentioned previously in this section page 35. One is to use a network simulator tool and to take benefit of very detailed protocol models, the other one is to develop a new model based on Petri net formalism. Our first interest is to validate the architecture in normal working, using full-duplex links and switched network, so there is no lost

packets. This two points imply that we do not need to model all detailed protocol mechanisms, such as establishment of connection with TCP. At the contrary, the synchronisation and shared resources mechanism are essential to model.

Therefore, we choose to design a Petri net model of the whole architecture aiming to determine the influence of the different delays on the response time. As our second interest is to compare different solutions, this model must be generic for a class of architectures.

As a result, our approach (figure 2.8) consists in evaluating time performances in Ethernet-based Automation Systems using client/server model is to design a generic Petri Net model. This model is going to be instantiated for each architecture to test.

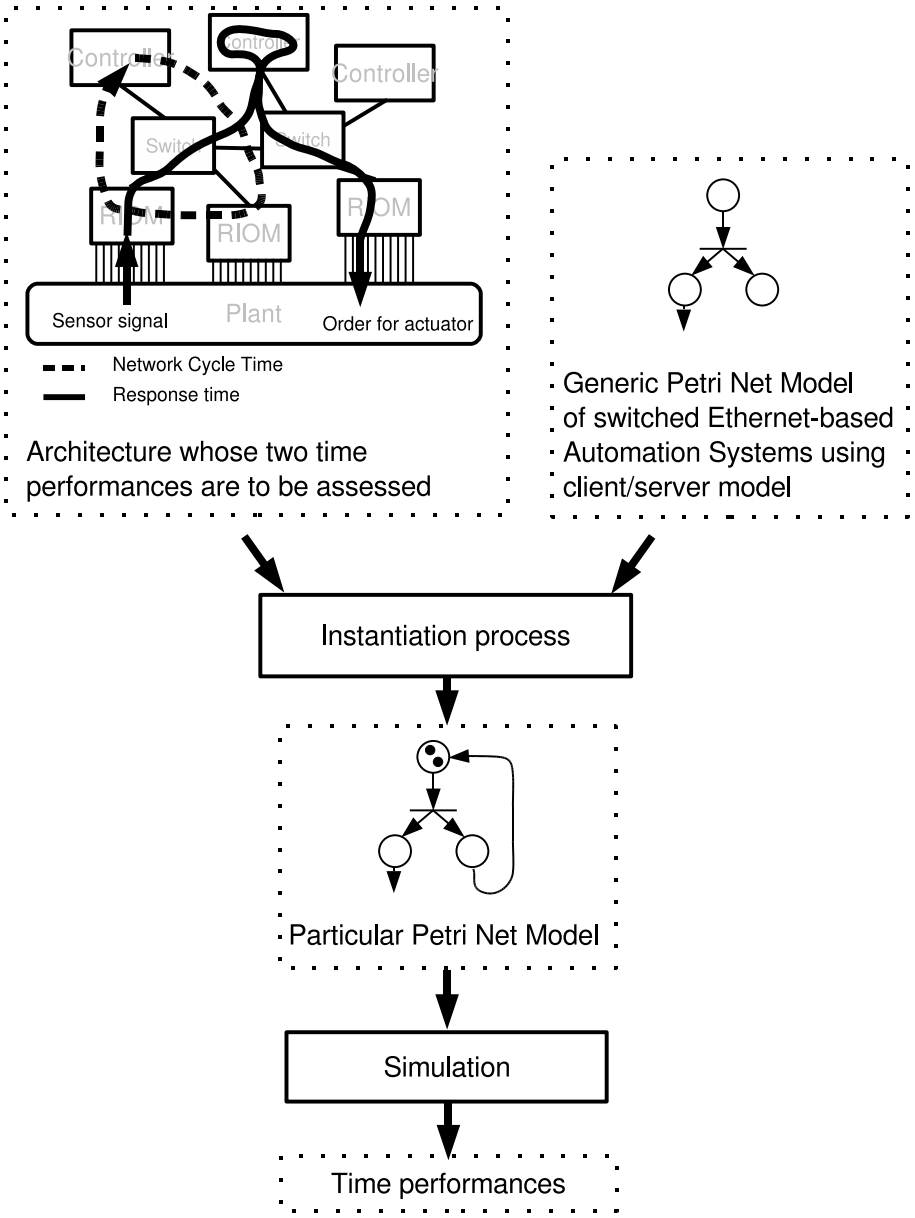


Figure 2.8: Proposed approach for evaluation of time performaces of switched Ethernet-based Automation System using client/server model

Chapter 3

Principles for dynamic model construction

This chapter explains the principles followed for designing the dynamic model of Ethernet-based Automation Systems. First, a static modelling of the system enables us to define clearly the class of architectures studied, including hardware and software components and data types. Second, the four major characteristics of the system lead to the choice of the formalism for the dynamic model. Third, this formalism is briefly presented and finally, the structure of the generic model is detailed.

3.1 Static models of switched Ethernet-based Automation Systems using client/server cooperation model

A Ethernet-based Automation System is a set of hardware and software components where data are exchanged and treated. The hardware components compose a physical architecture as the example presented on figure 3.1, including controllers, switches and Remote IO Modules. There are two kinds of controllers: modular controllers composed of one processor module and one Ethernet module, and PC-based controllers. The software components that are considered are user programs in controllers, clients also in controllers and servers in RIOMs.

Data flowing in the architecture are presented on figure 3.2 by the arrows between components. Two aspects are to be taken into account:

- the structure of data and
- the flow of data in the architecture

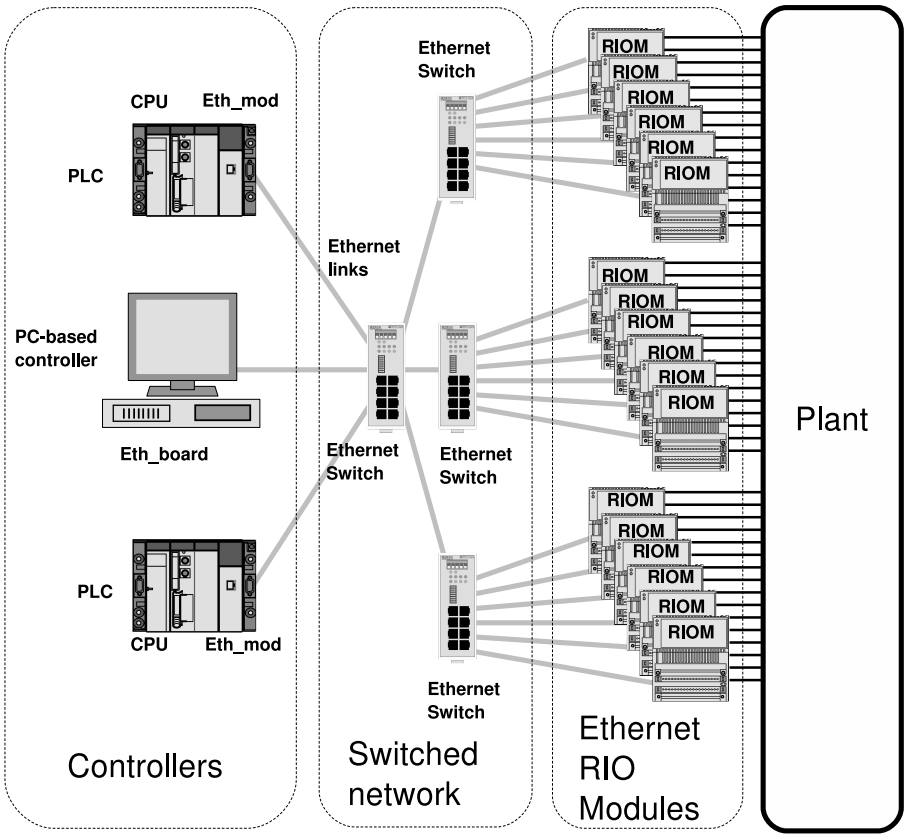


Figure 3.1: Physical architecture of switched Ethernet-based Automation systems

Static modelling models the structure of data while the flow of data is obtained directly when simulating the dynamic model and will be addressed later in the subsection 3.4.2, page 60.

Two generic static models, one for components (figure 3.3) and one for data structure (figure 3.4), have been designed to help structuring the final generic dynamic model.

The class diagram of components in UML formalism describes their relationship. Each Ethernet module, or PC-based controller, must be linked with one and only one switch, as well, each RIOM is linked with one and only one switch. Each switch can be linked to several devices, either Ethernet modules, PC-based controllers, RIOMs or other switches. Each Ethernet module and each PC-based controller implements clients as well as each RIOM implements servers. The model of a given architecture is one instantiation of this class diagram.

The class diagram of data structure in UML formalism describes relationship between data. From top to bottom, data of type "event" represent the change of state of a discrete signal. In this work, the term event also represent the value (zero or one) of a variable after a change of state. The events are included inside Modbus frames, which are also included inside Ethernet frames. Several events are included in one Modbus frame. That Modbus frame contains, first, a code representing an action (read and/or write), an address and a size of data, and second, the value of data to transmit. The Modbus frame is itself included in one Ethernet frame, indeed the size of the Modbus frame is not large enough to need to be cut into several Ethernet frames. The TCP segments is not represented in our case because the Modbus frame does not need to be cut and the IP datagram is neither represented because we study only switched networks and not routed networks. Then Ethernet, TCP and IP protocols add only to the Modbus frame headers and tailers.

These two static models have presented the objects to model and their relationships. In the dynamic model, the behaviour of software and hardware components, their time features and the data flows between components are going to be modelled. In the next section, the choice of the formalism of the dynamic model is explained.

3.2 Choice of the formalism for the dynamic model

3.2.1 Time consumption mechanisms representation

The three causes of delays, or time consumption mechanisms, taken into account are (section 1.4.4 page 22):

- time for processing data within components,

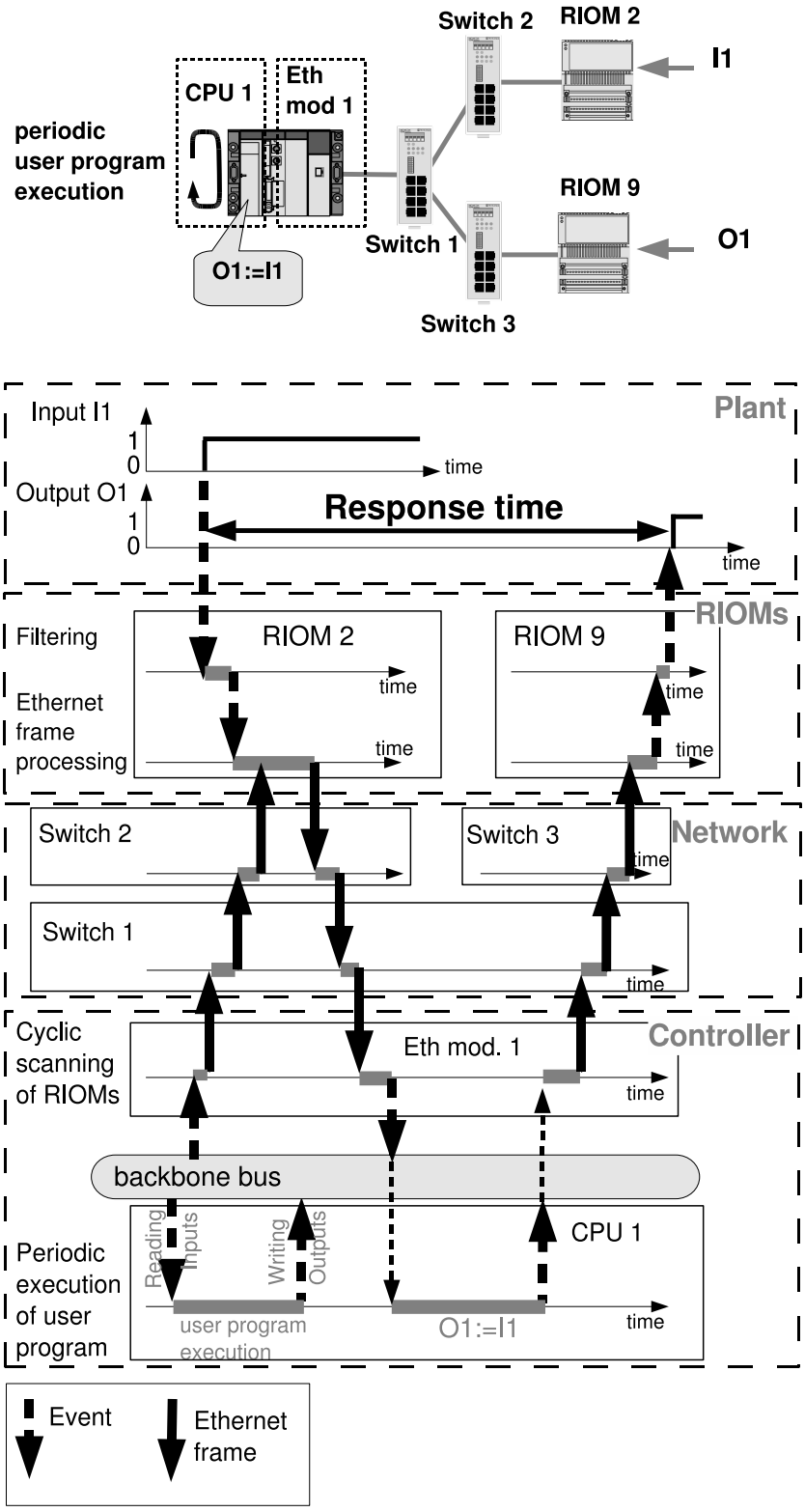


Figure 3.2: Data flows in Ethernet-based Automation systems using client/server cooperation model

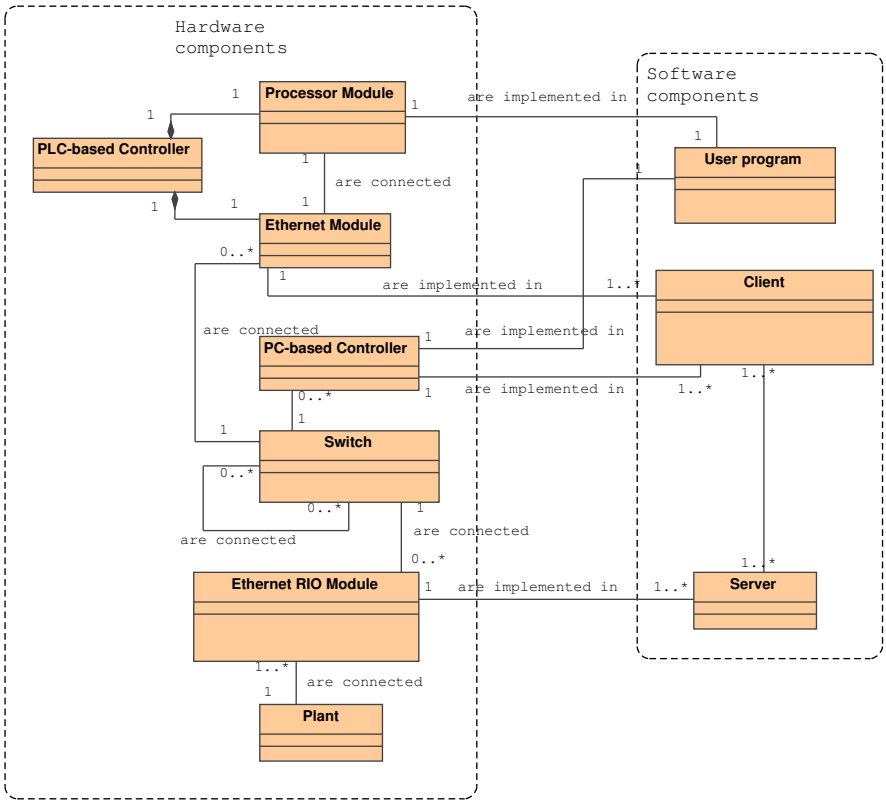


Figure 3.3: Class diagram of component structure in switched Ethernet-based Automation systems using client/server cooperation model

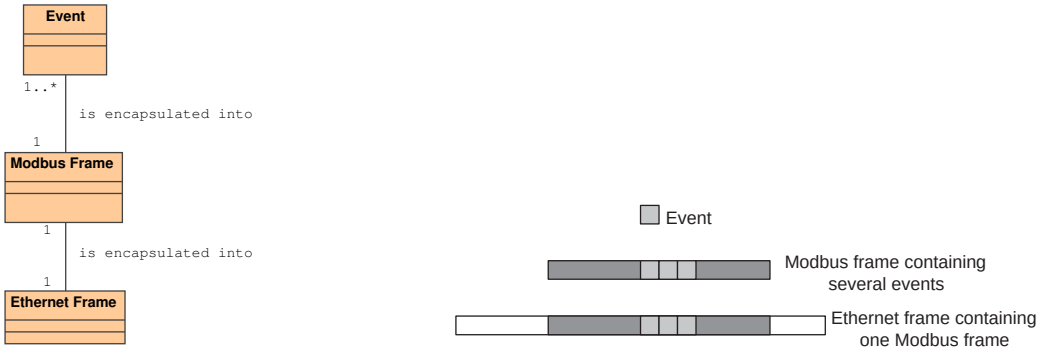


Figure 3.4: Class diagram and illustration of data structure in switched Ethernet-based Automation systems using Modbus protocol

- waiting time for synchronisation between parallel processes and
- waiting time for availability of resources.

From the survey of scientific works given in the previous chapter, the best-suited modelling language to model the three previous mechanisms is Petri Net.

3.2.2 Representation of architecture structure

One of the goals of the dynamic model is to evaluate time performances of Ethernet-based automation systems in order to compare different architectures. Consequently, the model must be generic to the class of architecture defined in the previous section figure 3.3. This generic model will give rise to particular models representing the different architectures to compare thanks to an instantiation process that will be described in chapter 5.
The structure of the generic model is issued from the class diagram figure 3.3. This means that the generic model of Ethernet-based Automation Systems shall be composed of independent generic models of components (Plant, Ethernet RIO Module, Switch, PC-based controller, Ethernet module and Processor module). Some of these generic models are presented in chapter 4.
These two modelling principles, component-based structure and genericity are obtained respectively by two Petri Net extensions:

- compound-component hierarchy and
- colour

Compound-component hierarchy enables us to build the overall generic model as the composition of generic models of hardware components.
Colour enables us to fold a large particular model including several several particular components models and to obtain a component particular model describing the behaviour of a given architecture.

3.2.3 Physical time representation

In the context of evaluation of time performances, the time representation that is chosen for the model is essential and is driven by both the type of performances to evaluate and the assumptions made on the system.
We remind the reader that this study focuses on the variability of time performances that are due to the conjunction of the three time consumption mechanisms with complex relationships, but not the variability due to variable processing time inside components. *As a consequence, the time for processing data is modelled by a constant value.* Therefore, the modelling formalism should based on Timed Petri Nets, that deal only with constant duration, and not on Time Peri Nets, that allow

representation of variable duration.

It matters at this point to recall also that, as we consider switched network and full-duplex connections, there is no frame loss. As a consequence, the use of stochastic classes of Petri Net, as in Juanole [2002], is not needed in our case.

3.2.4 Data representation

The figure 3.4 shows the three types of data (event , Modbus frame and Ethernet frame) which are included one in the other. The place-transition structure of the model represents the behaviours of components and their connections, so data flowing in the network are represented by tokens flowing from place to place. To represent data structure of figure 3.4, colours will be used.

To sum up, colours will enable us both to fold model of architectures and to distinguish the different kinds of data flowing in the network.

3.2.5 Formalism for dynamic modelling

In order to model Ethernet-based automation systems in a generic way for evaluation of time performances, the chosen formalism, from the above requirements, is the Hierarchical Coloured Timed Petri Net, which is detailed in the next section.

3.3 Presentation of the chosen Petri Net class

3.3.1 Colour feature

Coloured Petri Nets have been introduced in the beginning of the nineties thanks to academic papers and books like David and Alla [1994] and Jensen [1992]. They have been used in particular to model manufacturing processes where both synchronisation and shared resources mechanisms are present, with several identical functional and behavioural units. It is also the case of Ethernet-based Automation Systems.

Coloured Petri Net is standardised by IEC as High Level Petri Net (HLPN) (ISO Standards [2004]). Its formal definition in the standard is a tuple $(P, T, D, Type, Pre, Post, M_0)$ where:

- P is a finite set of elements called Places
- T is a finite set of elements called Transitions disjoint from P $(P \cap T = \phi)$
- D is a non-empty finite set of non-empty domains where each element of D is called a type

- $Type : P \cup T \rightarrow D$ is a function used to assign types to places and to determine transition modes
- $Pre, Post : TRANS \rightarrow \mu PLACE$ where are the Pre and Post mappings to with

$$TRANS = \{(t, m) | t \in T, m \in Type(t)\}$$

$$PLACE = \{(p, g) | p \in P, g \in Type(p)\}$$

- $M_o \in \mu PLACE$ is a multiset called the initial marking of the net with $\mu PLACE$ the set of multisets over the set $PLACE$.

A Colour in HLPN is a type, and D is also called colorset. Concerning the arcs function Pre and Post and the transition firing rules, figure 3.5 is the example taken from the standard, with two places p1 and p2 of different types (or colours) A and B, defined in the declaration under the Petri Net. The initial marking (M_0) is noted in the places concerned. Here, the place p1 have three tokens, one token whose value is two (1‘2) and two tokens whose value is three (2‘3). There are two arcs, one from place p1 to transition t1 and the other from transition t1 to place p2. When the transition t1 is fired, one token is removed from p1, noted ”x” on the arc, and one token ”y” is put in place p2. The guard ”x < y” on transition t1 implies that the value of the token x must be smaller than the value of the token y. The transition t1 can be fired as soon as there is one token in place p1 that can satisfy the condition x<y.

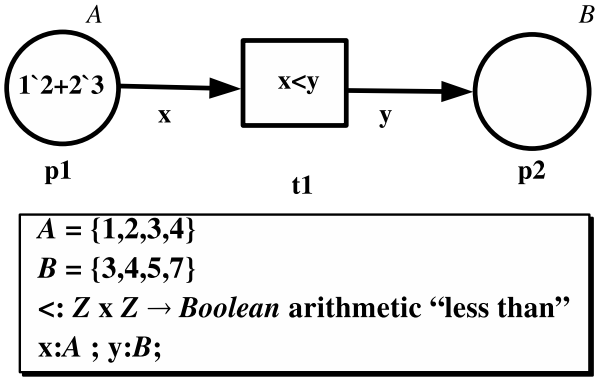


Figure 3.5: Example of HLPN from standard

3.3.2 Hierarchy

A hierarchical Petri Net is a model where one part can be abstracted by a substitution transition and this part is detailed in apart. The hierarchy is used to give the possibility to cut the model into different parts, that makes it easier to design in a

modular way. This is explained on our model in the next section 3.4.
If the Coloured Petri Net is standardised by IEC as High Level Petri Net, this is not the case of the Hierarchical extension. Therefore, we choose to use the semantic defined in Jensen [1997], which is supported by a simulation software, CPNTools (University of Aarhus [2006a]). Hence, a Hierarchical Coloured Petri Net is a tuple $(S, SN, SA, PN, PT, PA, FS)$ where:

- S is a non-empty finite set of pages where each page is a non-hierarchical Coloured Petri Net
- SN is a set of substitution nodes included in T (transitions)
- SA is an assignation function of pages defined from SN into S where no one page is its own sub-page
- PN is a set of port nodes included in P (places).
- PT is a function port type PN in $in, out, i/o$.
- PA is an assignation function of port which defines relation between nodes, between pages and sub-pages
- FS a set of fusion sets included in P where all elements have the same colour and initialisation function

The use of hierarchy will be illustrated on the designed model in section 3.4.

3.3.3 Time feature

The Timed extension of Coloured Petri Net is not standardised in the IEC as High Level Petri Net. Once again, we choose to use the semantic defined in Jensen [1997], which is also supported by a simulation software, CPNTools. In this case a Timed Coloured Petri Net is defined as the tuple (CPN, R, r_0) where:

- CPN is a High Level Petri Net with the Post condition that can be timed
- R a set of time values included in $\Re^{+0}$
- r_0 initial value included in R

The previous example (figure 3.5) is adapted on figure 3.6 to represent a timed CPN. The two colours have a new attribute, "timed", and the marking of place with timed token is noted with a new operator ++. When the transition t1 is fired, a time-stamp of 10 ($@ + 10$) is added to the token produced in place p2. As a consequence, this token is available in p2 only ten time units after firing the transition t1. Compared with the standard notation, here the names of transition and places are noted

inside the symbol while the initial markings and the guards, in brackets, are noted aside.

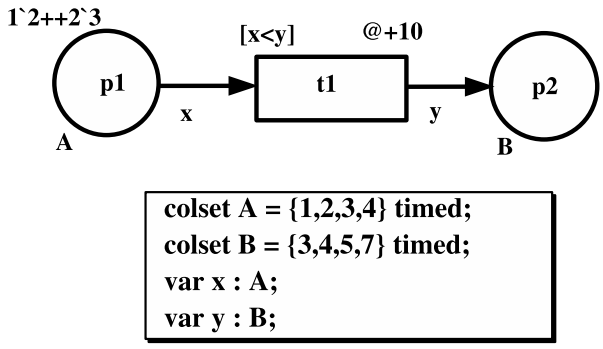


Figure 3.6: Example of CTPN with CPNTools syntax based on the example of standard

This definition differs from that often used in litterature for two reasons:

- time is not only linked to transitions but is also associated to tokens that carry each one a "time-stamp",
- there is no token reservation prior to transition firing, but a "token unavailability" in the upstream place after firing the transition.

3.3.4 Use of complex colours and functions

In this subsection, some specific features of the software tool CPNTools often used in the generic model are explained. In this modelling and simulation software, the declaration of colours and variables is supported by a programming language called CPN ML University of Aarhus [2006b] which is based on the Standard ML programming language Milner et al. [1997], Ullman [1998]. With CPN ML language, it is possible to use complex colours and functions.
The colours corresponds to CPN ML types, the simple ones are unit, integers, Boolean, string, enumerated and index. Based on these types, it is possible to build complex colours. In the model, the two following are largely employed:

- product of several colours sets which is a set of variables of type tuple,
- list which is a set of ordered variables of the same type,

As shown in the previous chapter, based on these features, colour enables to represent different concepts. It is possible to mix different types by defining composed colours. In our case, tokens can represent either components resource or data.

Concerning data structure, there are some defined as list, an ordered set of one data type:

```
colset A = INT timed;
colset B = list A timed;
```

The list is defined without any size, and so elements can be removed or added during model simulation. As a result, this colour type enables to stack several data or tokens into one token and keep their arrival order in a given place. In this way, it is possible to model buffers as FIFO queues very easily.
To manage the lists, two operators are essential: :: and ^^. The first one adds one element at the beginning of a list, and the second one concatenates two lists. Their use to design FIFO queues is illustrated on figure 3.7. Place p2 represent a buffer containing a list of value of colour A. This list is filled by transition t1 with an element e1 coming from place p1. The list is emptied by the transition t2 that remove one element e2 from the head of the list.

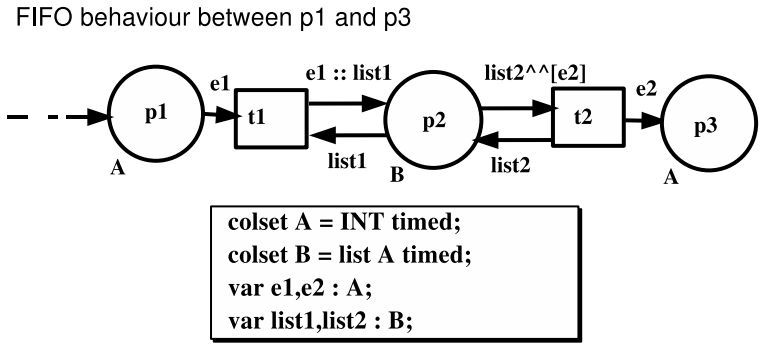


Figure 3.7: Design of FIFO queues in CPNTools

CPN ML also provides a syntax and structures to design functions such as *if then else* or *case of*. This feature is used to define guard functions and initial marking. In both cases, the goal is to have a parametrisable Petri Net structure and to obtain its instantiation by only changing textual declarations.

$$\begin{aligned} fun\ example(i) \quad &= \quad case\ i\ of \\ &\qquad 1 \Rightarrow true \\ &\qquad |2 \Rightarrow false; \end{aligned}$$

In this function, if the i variable is 1, then the function returns the Boolean value true, else if i is equal to 2, the function returns false. This function can be associated to the guard of a transition. In this case the transition is fired only if the variable i is equal to 1.
For initial marking functions, the functions used returns only one result which is the set of token desired for initial marking. It is going to be illustrated on the instantiation of component models in the next chapter, section 4.3.

3.4 Structure of the generic model

3.4.1 Hierarchical structure

The structure of the model is represented as a tree on Figure 3.8, where the higher level *Global*, on the left, contains the generic model of the whole architecture class. The second level, contains the generic models of hardware components of the diagram class: Ethernet RIO Module (RIOM), Switch, PC-based controller and PLC-based controller composed of one Ethernet module and one processor module.

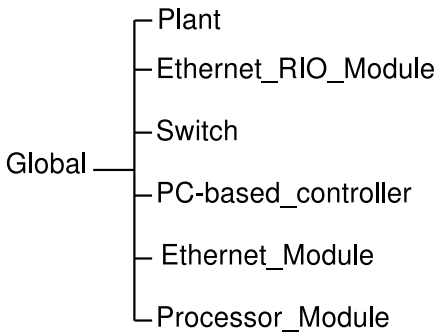


Figure 3.8: Structure of the HTCPN model

Figure 3.9 shows the HTCPN representation of the highest hierarchical level *Global*, in which each component is modelled as a substitution transition (*Plant*, *Ethernet_RIO_Module*, *Ethernet_Switch*, *PC-based_module*, *Ethernet_module* and *Processor_module*). On the picture, a substitution transition is differentiated from a transition by the tag rectangle containing the name of the sub-model at the bottom of the transition; in general, this is the same name as the transition name. The sub-model of the Ethernet Module is shown at the right in the middle. Some of these sub-models, like the model of Ethernet Module, may include substitution transitions.

The structure of this Petri Net remains the same whatever the structure of the particular Ethernet-based Automation System is. To distinguish one architecture, that includes a set S1 of physical components, from another one, that includes another set S2, we will use component colours presented in the next section. The initial marking of the particular model describing the behaviour of one architecture will feature the structure of this architecture.

To model the information transport from one component to another, tokens modelling data are ”sent” from one component model to another via the interface places *Events*, *Ethernet_frames* and *Variables*. The three associated colours, EVENT, ETHFRAME and VAR, model the different data types and are going to

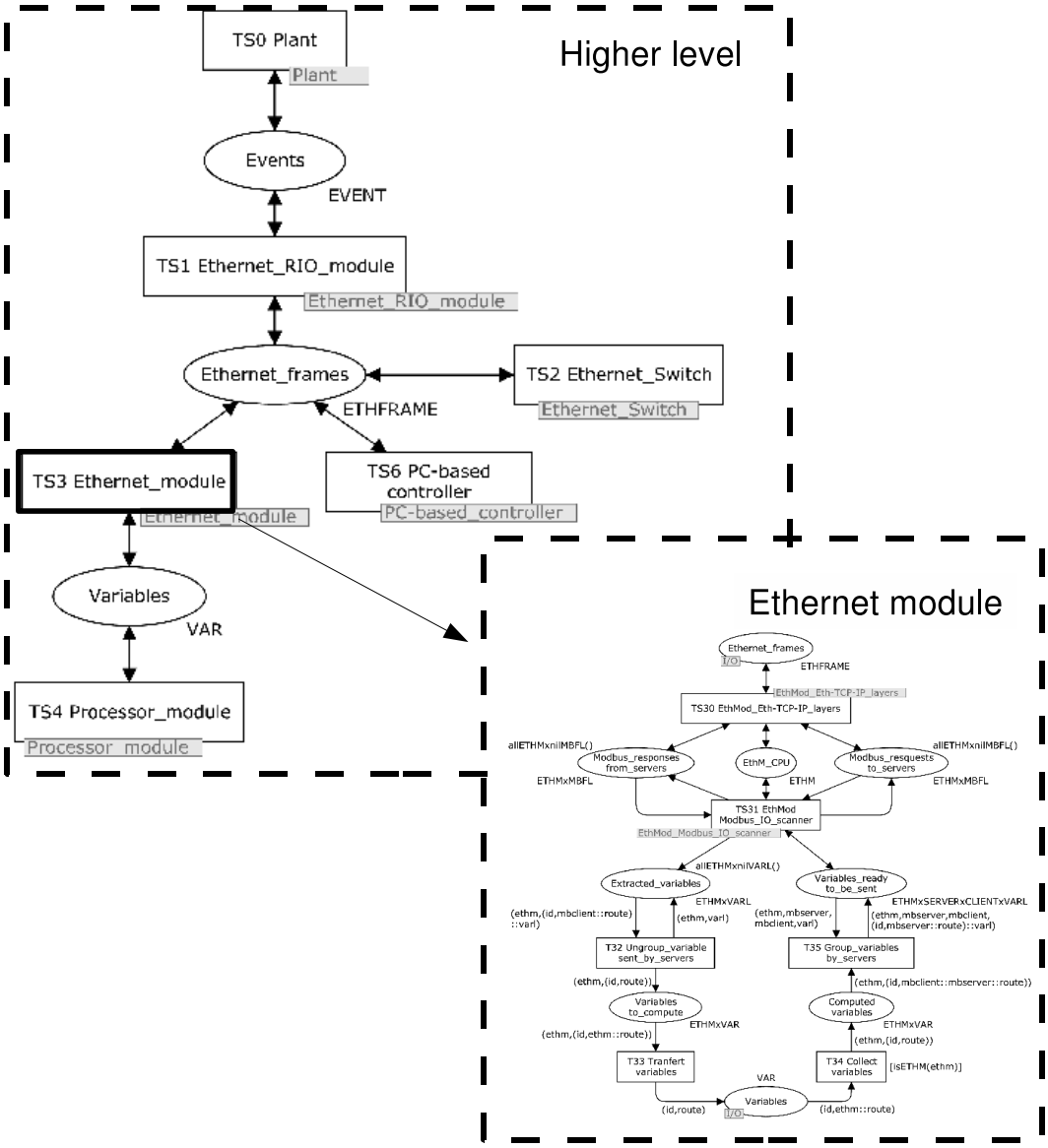


Figure 3.9: Global HTCPN model

be explained in the next section.

3.4.2 Colours definition

Colours, in our modelling, are used to distinguish components and data.

Colours representing components enable a generic representation. Indeed, by defining a Colour for each kind of component, one structure of Petri Net can represent multiple components that have a similar behaviour. For instance one structure of RIOM model is defined and each new RIOM instance is represented thanks to a new token, carrying the RIOM colour. The second function of colour is to model four information kinds, events, variables, Modbus frames and Ethernet frames. In the definition of these colours must be modelled the encapsulation of events into Modbus frames and of Modbus frames into Ethernet frames, as shown on the data structure (figure 3.4).

Concerning colours associated to components, the core concept is to identify each component by one unique and absolute identifier (positive integer) in the architecture, called a location. The locations are grouped by type of components. Hence the following colours have been defined:

```
LOCATION = INT timed;
DEST = LOCATION timed;
SRC = LOCATION timed;
SERVER = LOCATION timed;
CLIENT = LOCATION timed;
PROC = LOCATION timed;
ETHM = LOCATION timed;
RIOM = LOCATION timed;
SWITCH = LOCATION timed;
ETHLINK = LOCATION timed;
PLANT_ELEMENT = LOCATION timed;
```

Timed means that tokens of this type have a time-stamp which evolves with model time. DEST and SRC are for destination and source of a Modbus frame, that could be a SERVER or a CLIENT. These last two represent software components, clients are in controllers and servers are in RIOMs. Colours PROC, ETHM, RIOM, SWITCH, ETHLINK and PLANT_ELEMENT represent the hardware components of the same name (PROC for processor module) and they are used to model resource availability. PLANT_ELEMENT is located in the plant model and is used to generate or receive events; this part is going to be explained later in subsection 4.2.3.

Concerning data, there are two different kinds in the system and they are dependent. In the plant side (Plant model), there are logic signals, while in the Ethernet network side, there are Ethernet frames. The logic signals are transformed into input and output binary values, called "event", then they are encapsulated into a Modbus frame. Finally each Modbus frame is encapsulated into one Ethernet frame to be sent to another automation device via the network. The following colours have thus been defined:

```
ID = INT timed;
ROUTE = list LOCATION timed;
EVENT = product ID * ROUTE timed;
EVENTL = list EVENT timed;
MBFRAME = product SRC * DEST * EVENTL timed;
MBFL = list MBFRAME timed;
ETHFRAME = product ROUTE * MBFRAME timed;
ETHFL = list ETHFRAME timed;
VAR = EVENT timed;
VARL = EVENTL timed;
```

The colours VAR and VARL have been introduced in controllers because this one cannot treat events but variables that represent input and output values after an event.

In the list below, only the first Colour ID is simple data type while the others are complex. The identifier of colour ID is the events'one, it enables us to follow easily events all along their routes in the architecture. This route is modelled by variables of coloured ROUTE, defined by a list of locations. This colour is used in EVENT which is a product of ID and ROUTE and in ETHFRAME which is a set of products of ROUTE and MBFRAME. The colour EVENT is used to follow each event in the system, so as to know the time delay between the occurrence of an input and the corresponding output. The route contained in EVENT is a list of automation components, i.e. specific components for automation such as controllers and RIOMs. The route contained in ETHFRAME is a list of Ethernet components, i.e. general purpose communication components such as switches.

To illustrate this concept of route, supported by the colour ROUTE, an example is given on the response time assessment presented on Figure 3.10. This architecture is composed of two controllers, each one composed of a processor module (CPU) and an Ethernet module (Eth_module), four switches (SW) and eighteen RIOMs (RIOM). The identifiers of hardware components are indicated by the numbers, such as 0 and 1 for the processors of controllers and 60 and 61 for the Ethernet modules. The response time to evaluate here is the delay between input I1 and output O1 occurrences, with processing in CPU 0 to set O1 equal to I1. The image

of I1 state in the controller as well as the state of O1 in the plant are updated thanks to a cyclic IO scanning performed every 5 ms by the Ethernet module 60.

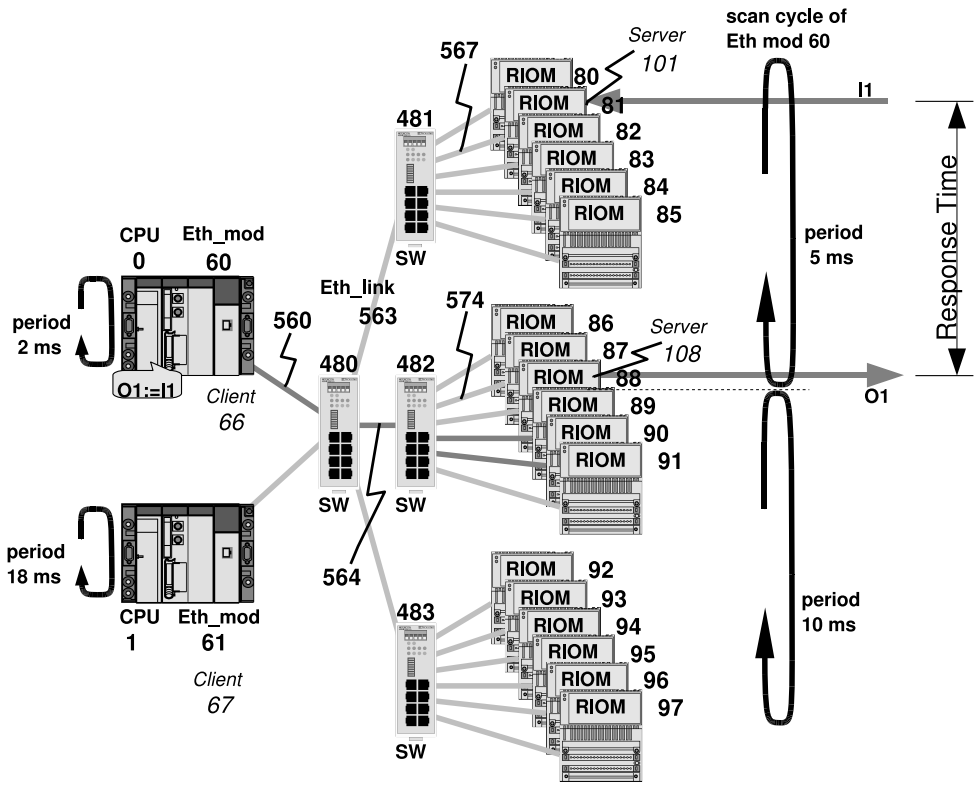


Figure 3.10: Example of architecture and response time

The token carrying the colour EVENT models the event occurrence and also the causality chain from I1 to O1 by the route, including all automation components, as given on the Figure 3.11 b, with the RIOM (81), the Modbus server (101), the Modbus client (66), the Ethernet module (60), the processor (0), once again the Ethernet module (60), the Modbus client (66) and finally the Modbus server (108) and the RIOM (88) where is linked the output O1. The routes between a pair of client and server are given only by colours associated to tokens representing Ethernet frames, these routes are written Figures 3.11 c and 3.11 d for our example (Poulard et al. [2004]). The first one, [101,81,567,481,563,480,560,60,66], is for the first network crossing, from RIOM 81 to the Ethernet module of the controller. The numbers 567, 563 and 560 model Ethernet links. The first one is between the RIOM (81) and the switch (481). The second route is [66,60,560,480,564,482,574,88,108], corresponding to the second network crossing, when data is sent from controller to RIOM. It is to notice that both Modbus server and client are just software pieces and the hardware components are connected to Ethernet link. Therefore, both RIOM and Ethernet modules appear as interfaces between Modbus software and Ethernet

link. That is why in the frame route 1, the RIOM identifier reappear after the server identifier. The whole sequence in the RIOM is:

- treatment of the digital signal by the RIOM hardware component 81,
- treatment of the information ”event” by the RIOM software component, server 101,
- sending of the frame built by a hardware physical link of RIOM 81.

The two first step appears in the event route, the second appears both in event and frame route and the third appears only in frame route.

We have discussed colours components and colours representing data or data routes. It remains to model data in components, for this it is necessary to define composed colours, such as:

```
ETHMxETHFL = product ETHM * ETHFL timed;
RIOMxEVENT = product RIOM * EVENT timed;
PROCxVARL = product PROC * VARL timed;
```

These three compound colours are associated to tokens that represent one set of one component and one data. They have been introduced to be able to know, at any moment, which Ethernet frames are waiting in or processed in an Ethernet module, which event are waiting for a request from a client in a given RIOM, and which variable list is currently processed by the processor of a controller.

Conclusion

Timed Petri Nets provide constructs for modelling the three causes of delay encountered in Ethernet-based Automation Systems:

- time for processing data within components,
- waiting time for synchronisation between parallel processes and
- waiting time for availability of resources.

To facilitate time performances evaluation and comparison of several architectures, we chose to develop a generic, component-based model. This leads to select a high-level Petri Net class: Hierarchical Coloured Timed Petri Net. Given this formalism, we proposed a structure and defined colours so as to derive easily particular models from the generic one.

The next chapter focuses on a detailed presentation of the generic model.

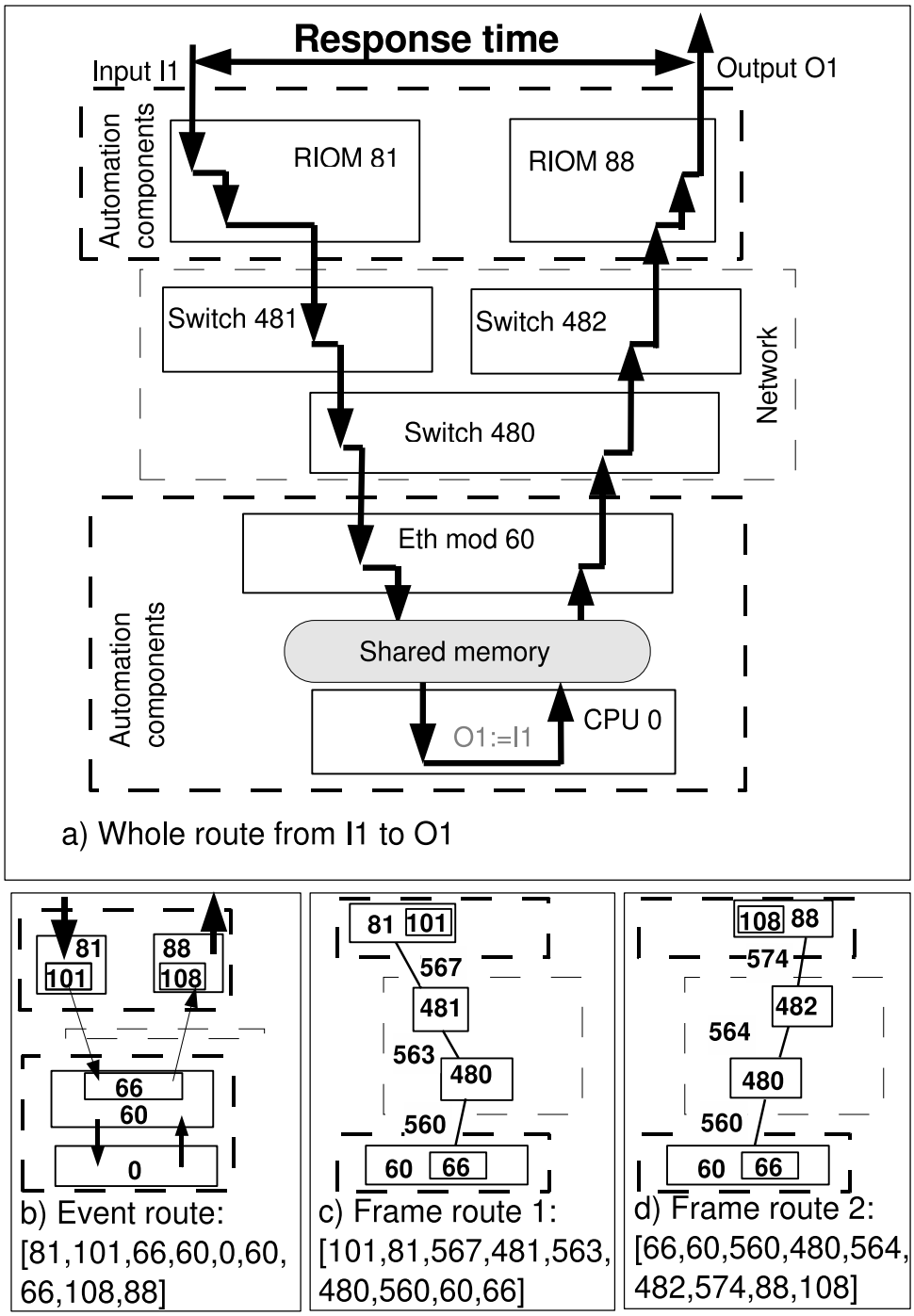


Figure 3.11: Routes for event and frame, presenting simplified versions of figure 1.13 without distinction between delays in components and with location numbers of figure 3.10

Chapter 4

Dynamic model description

This chapter opens with specific features of the implementation language often used in the model. Secondly, four extracts of the generic model are presented. Thirdly, a section details the instantiation process and finally one particular model is explained.

4.1 Generic model

4.1.1 Global generic model

In the previous chapter, section 3.4, the component-based structure have been explained. To obtain component generic models of manageable size, some of them contain sub-models. The tree of the complete hierarchy is given on figure 4.1.

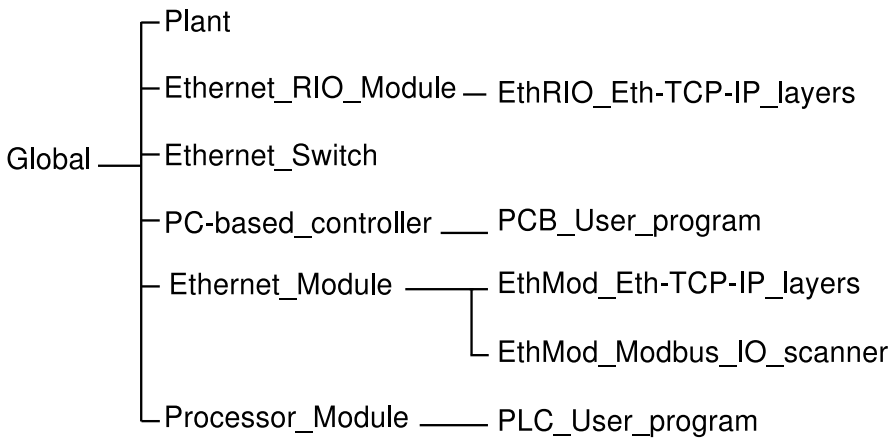


Figure 4.1: Complete hierarchy of the Ethernet-based Automation Systems generic model

4.1.2 Modelling data flows

The modelling of data flows inside the system with the flow of tokens implies to manage token route between the component models. For instance, the place *Ethernet_frames*, we have seen in the global Petri Net, contains all tokens modelling an Ethernet frame which is between two components. To know which transition can be enabled by these tokens, i.e. in which component model the token will be put, it is necessary to use guards on transition that are only true for the desired tokens. The guard on the transitions that enables input in the Ethernet module is *[isETHM(loc)]* with the function defined:

$$\begin{aligned} fun\ isETHM(loc) \ = \ & case\ loc\ of \\ & 60 \Rightarrow true \\ & |61 \Rightarrow true \\ & |62 \Rightarrow true \\ & |\_ \Rightarrow false; \end{aligned}$$

If the *loc* variable of an available token is 60 or 61 or 62, the transition can be fired, else it cannot. This function can have different results, true or false. This modelling induces to have a unique identifier for each component.

4.2 Components: extracts of the generic model

As the generic model is large, we prefer here to explain in detail four parts of model that have a particular interest in the architecture studied: the Ethernet Modbus client, the PC-based controller, the source of events and the switch.

4.2.1 Ethernet Modbus client

The Timed Coloured Petri Net (TCPN) modelling the Ethernet Modbus client, which is a part of the Ethernet module, is presented on the figure 4.2.

Its function is to:

- build cyclically Modbus requests with variables computed by the controller and
- to extract variables of Modbus frames coming from the Ethernet-TCP-IP layers sub-model of the Ethernet module.

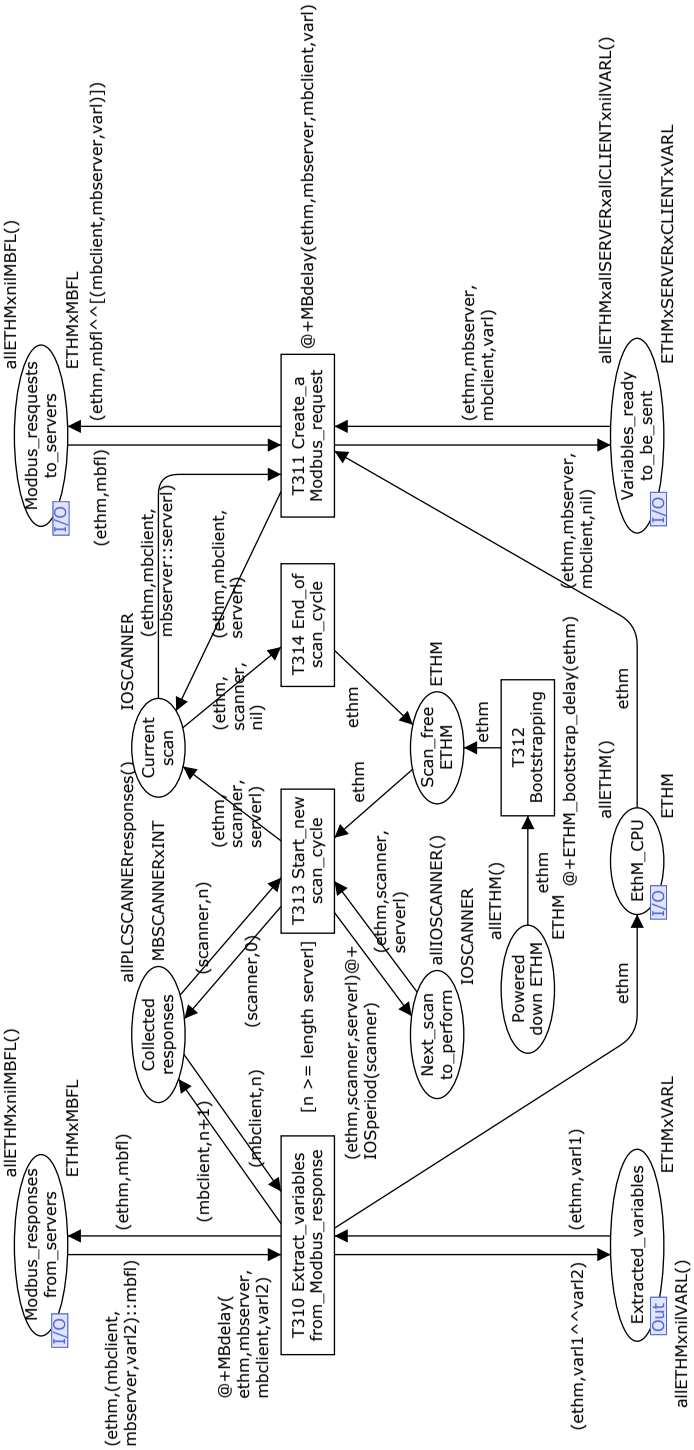


Figure 4.2: Generic HTCPN model of IO scanning cycle

This cyclic requesting of RIOMs is also called IO scanning. The figure 4.3 presents the PLC-based controller with its two asynchronous processes, the user program cycle and the IO scanning cycle, studied here.

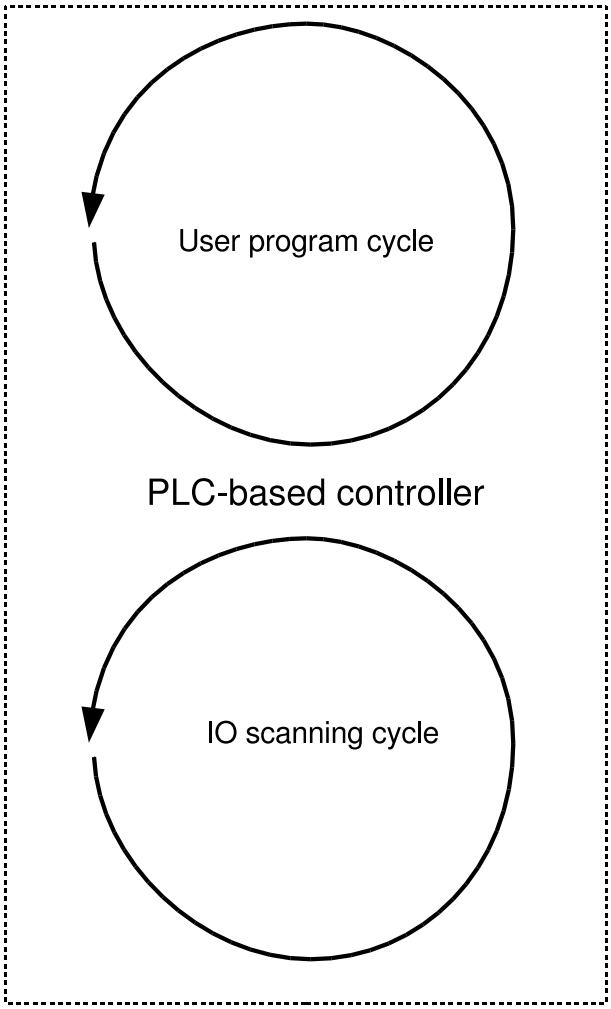


Figure 4.3: PLC-based controller

This model can be split in five parts: interface with the models of higher and lower level, the IO scanning cycle, the buffering mechanism, the processor resource and the bootstrap mechanism.

The interface with the higher model Ethernet Module is composed of the four places that are in the corners. These places are called interface places and enable token flowing to and from the higher level model which is Ethernet Module. These interface places are different of the one presented in the previous chapter (Events, Ethernet frames and Variables), section 3.4, because these previous places were designed to enable token flowing between the global model and sub-models as Ethernet Module, but not between the sub-models (as Ethernet Module) and the

sub-sub-models (as Ethernet Modbus client).

The IO scanning cycle is modelled by the following set of place:

- Next scan to perform
- Collected responses
- Current scan
- Scan free

and transition:

- T313 Start new scan cycle
- T311 Create a Modbus request
- T314 End of scan cycle
- T310 Extract variables from Modbus response

The place *Next scan to perform* must contains tokens that represent the next list of servers to scan for each client. These tokens are of colour IOSCANNER , which is a product of the colours ETHM, CLIENT and SERVERL. The time-stamp of this token indicates the date of the next cycle. A new cycle is started by transition T313 at this date only if:

- the corresponding scan is free, i.e. the corresponding token is available in place *Scan free* and
- if all corresponding responses have arrived, i.e. the corresponding token in place *Collected responses* has its variable "n" equal to the number of servers to scan (guard [n >= length server] on T313).

When transition T313 is fired, a token of colour IOSCANNER is created in the place *Current scan*. Then, for each server of the list the transition T311 is fired to create a Modbus frame with the variables available modelled by tokens in the place *Variables ready to be sent*. The processing time for creating the Modbus frame is modelled by the time result of the function $MBdelay(ethm, mbserver, mbclient, varl)$ associated to the transition T311. The data are finally extracted from response received when transition T310 is fired.

The four interface places models also buffers, using list colours as presented in the previous chapter, section 3.3.4. In the places *Extracted variables* and *Variables ready to be sent*, list of variables are stored in the tokens, while in the places *Modbus responses from servers* and *Modbus requests to servers*, list of Modbus frames are

stored in the tokens.

The processor resource of Ethernet Modules is modelled by the tokens in place *EthM CPU*. Concerning the sending of request to RIOMs, the resource is taken from the creation of one Modbus frame (transition T311) to the creation of the corresponding Ethernet frame (in the model EthM_Eth-TCP-IP_layers). Concerning the receiving of responses from RIOMs, the resource is taken from the extraction of Modbus frame from one Ethernet frame (in the model EthM_Eth-TCP-IP_layers) to the extraction of the variables from the corresponding Modbus frame (transition T310).

The bootstrap mechanism is modelled with the place *Powered down ETHM* and the transition T312. This place contains all tokens representing the Ethernet module that have not yet booted. The time to boot is given by the function $ETHM\_bootstrap\_delay(ethm)$, which is different for each module and for each component. This mechanism enables to avoid non required synchronisation between all components.

Concerning the Modbus client, we begin by detailing initial markings for each place. The initial markings of *Modbus responses from servers*, *Extracted variables*, *Modbus requests to servers* and *Variables ready to be sent* prepare empty lists which models empty buffers. The initial marking of *Collected responses* is an easy way to know by parametrisation the number of server to scan for each client and consequently to know if the current scan cycle is finished or not. The initial marking of *Next scan to perform* models all the next scans to perform with one token for each client taking the form of the tuple (Ethernet module, client, list of server to scan).

It is very important to notice that the IO scanning behaviour is time driven. Actually, the Ethernet module generates requests and therefore Ethernet frames for the RIOMs cyclically. The RIOMs answer whatever is the evolution of inputs and outputs. *As a consequence, traffic load on the network depends only on the number of RIOMs to scan and of the period of RIOMs scanning, and then is constant for a particular architecture. It does not depend on the values of events coming from the plant.*

In the next subsection, we detail the model of the PC-based controller and of its Modbus client, without going so far in details, but enhancing the differences between the two models of Modbus client in *Ethernet module* and in *PC-based controller*.

4.2.2 PC-based controller model

The figure 4.4 shows the part of HCTPN model for PC-based controller.

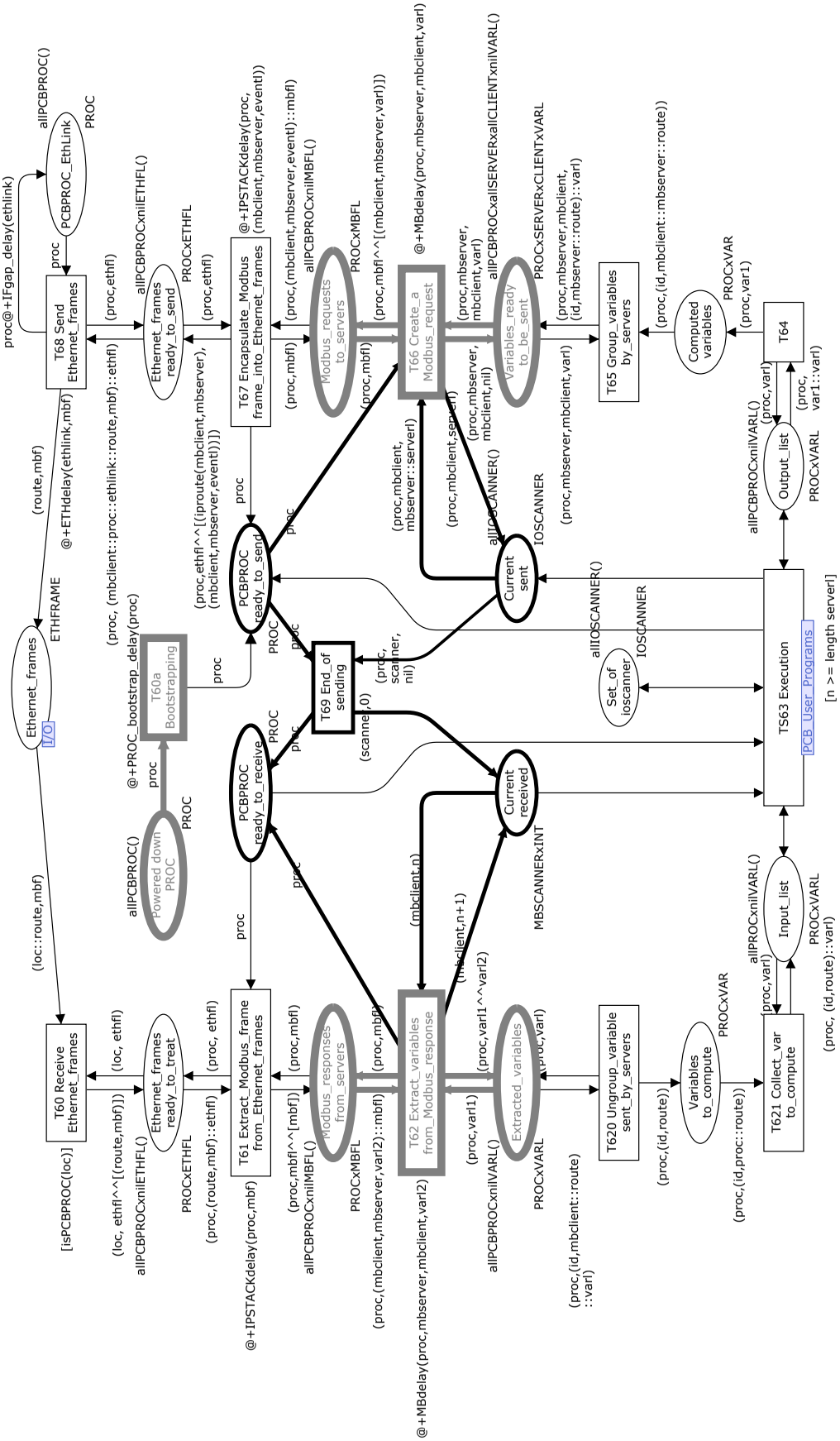


Figure 4.4: Generic HCTPN model of PC-based controller

This model covers more functions than the Modbus client of the Ethernet module. Indeed, the PC-based controller includes the user program and the IO scanning in only one cycle, as presented on figure 4.5.

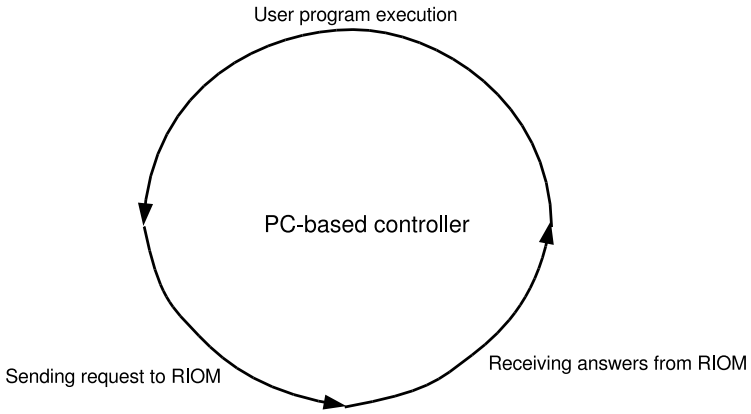


Figure 4.5: PC-based controller

The thick places and transitions (grey and black) correspond to the subset of the Ethernet module studied in the previous subsection. The thick grey ones are exactly the same places and transitions with same name and functions. The thick black ones are different due to the different behaviour of this client which is synchronised with the user program execution, modelled by the transition TS63 *Execution*. The outputs updating after the user program execution is obtained by sending the Modbus requests (transition T66) and the reading of input values before the user program is obtained in the response to the previous request (transition T62). The communication task and the control task are not parallel anymore but serial.

As a consequence, the execution is authorised only when all responses are arrived. This is modelled by the arc between the place *Current received* and the transition TS63. The sending of requests to the RIOMs is authorised only when execution is finished. This is modelled by the arc between the transition TS63 and the place *Current sent*.
It is to notice that the model of the processor resource is split into two parts, modelled by the places *PCBPROC ready to send* and *PCBPROC ready to received*. This is to ensure that all requests are sent to RIOMs before treating the responses, and so to ensure that the outputs updating phase is done before the inputs reading phase. This choice enables to keep the same behaviour for processor modules of modular controllers and of PC-based controllers. The transition T69 *End of sending* ensures the passage from sending phase to receiving phase.

Let us finish on a few words about Ethernet, TCP and IP modelling. The treatments related to these three protocols are merged in a single one in our model. This

is possible due to the fact that we abstract protocols by constant treatment delays. TCP protocol is not modelled in detail because in Modbus/TCP the acknowledgement messages are sent only during non-modelled phases: establishment and closing of connections. Concerning the congestion, the low level of traffic in the network enables to neglect it.

The encapsulation of Modbus frame in an Ethernet/TCP/IP frame is proceeded when firing transition T67. This action requires the processor resource of the controller, as well as the action of extracting Modbus frame from Ethernet frame (T61). This is not the case for sending the frame on the network because this action is assumed to be done by an ASIC (Application Specific Integrated Circuit). The place *PCBPROC Ethlink* models the link resource by which one message can be send every inter-frame gap delay. This delay is given in the model by the function IF-gap(ethlink). When a frame arrives from the network, it is immediately stocked in a buffer modelled by a token in place *Ethernet frames ready to treat*.

As for the Ethernet module of the PLC-based controller, the scanning of RIOMs is time driven. *Therefore, as in the Ethernet Module Modbus client, traffic load on the network depends only on the number of RIOMs to scan and of the period of RIOMs scanning, and then is constant for a particular architecture. It does not depend on the values of events coming from the plant.*

4.2.3 Event source

To evaluate a response time between a cause event and a consequence event on the plant, there is a need for an event source. The event must contain an identifier, to make easier the analysis, and a route which determine the performance to evaluate by giving the causal way in the architecture (section 3.4 page 60).
The model for event generation, called the plant model, is shown on the figure 4.6. As shown on the globel model (figure 3.9, page 55), this models is connected to the RIOM model by the place *Events*.

When firing the transition TO *Event generator*, the token in place *Next event* is removed to produce one token in the place *Plant events* and a new token in the place *Next event*. The first one constitutes the first step to propagate the event in the architecture. The first element of the route, loc, is removed because it represents the plant element where the event occurs which is not needed anymore. The token produced in place *Next event* is the same than the one removed, with an incremented identifier and delayed of a given time value.

The transition T1 models the consumption of the event coming from controller which is the consequence of the previous event that occurred on the plant. The event

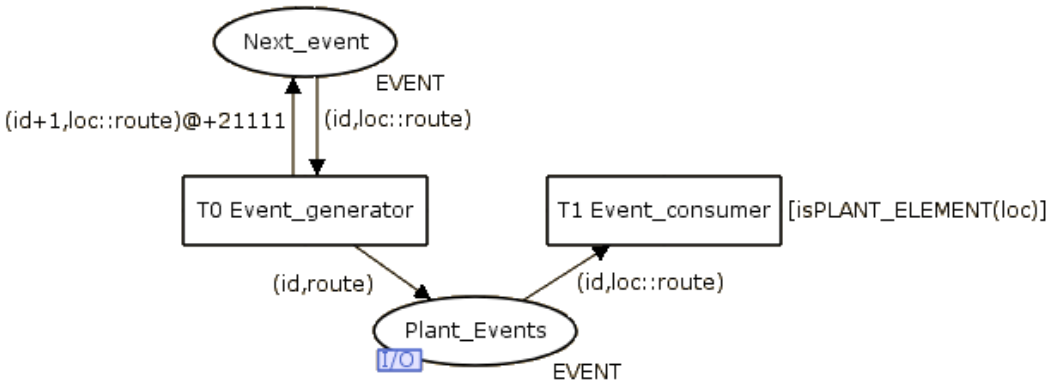


Figure 4.6: Generic HCTPN model of event generation: the plant model

arriving in T1 has the same identifier than its cause event that enables to follow the event from T0 to T1.

4.2.4 Switch

The kind of switch modelled are store and forward which store the entire frame, check its validity before to forward it on the right port. The modelling of all these steps are abstracted by a delay (noted SWdelay) that depends on the switch and on the Modbus frame size. We assume that all frames are valid, as a consequence all communication linked with the re-emission policy are not modelled. The scheduling policy modelled is First In First Out (FIFO) without priority.

The figure 4.7 present the generic HCTPN model of Ethernet switches. The place *Ethernet frames* contains the tokens representing frames that are arriving or leaving switches. If frames have to cross a switch, i.e. if the next element of their Ethernet route is the identifier of a switch, the transition T20 is fired. The tokens from place *Ethernet frames* are then stored in one list per switch in the place *FIFO matrix lists*; each list models the FIFO queue at the input of the switch. Here, all port buffering queues are represented by one queue per switch because we consider that the switch treats only one frame at one time with its CPU resource represented by an available token in the place *SWITCH CPU*.

When the transition T21 is fired, the first element of the frame list is removed from place *FIFO matrix lists* to be put in the list of the place *FIFO port lists* of the corresponding output port. The time associated to this transition is the time for processing data in the switch. As written below, it depends on the switch (variable named ”switch”) and on the frame size (known from the variable ”mbf” for Modbus frame). In this model, there is one queue per port of each switch because it does not need the CPU resource of switch. As a consequence, if several links are free at

the same time, frames can be sent on these links at the same time.

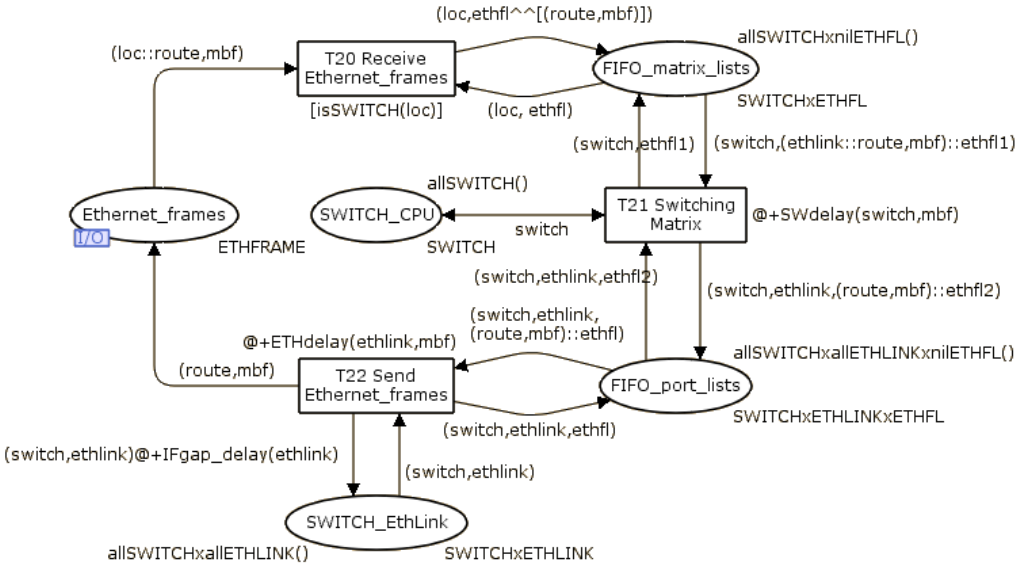


Figure 4.7: Generic HCTPN model of switch

4.3 Instantiation process

The aim of the instantiation process is to obtain models of particular architectures from the generic model. To reach this goal, parameters of the generic model, appearing in initial marking and functions, are to be instantiated with particular values.

4.3.1 Parameters

There are three different types of parameters that must be instantiated:

- parameters depending on physical architecture,
- parameters depending on component configuration and
- parameters depending on time performance to evaluate.

The parameters of physical architecture are the number of components and their connections. The number of components is defined by initial marking of the places containing tokens that model resources, such as the place *SWITCH CPU* in the switch model. The connections between the physical components of a particular architecture are defined by the routes.

The components configuration consists in time features. It must be defined for each component in guard functions. For instance, when firing a transition "create Modbus request", the time consumption depends on the component identifier.
If the time performance to evaluate is the network cycle time, there is no need to other parameters. However, if the time performance to evaluate is the response time, the event route from input to output must be parametrised.
Concerning the parameters of physical architecture and the ones of performance to evaluate, they directly come from the definition of the architecture to study. Concerning components configuration, that is not so direct, in particular for delays taken into account concern the communication.

4.3.2 Obtaining elementary delays values

4.3.2.1 Notations

The delays concerning components configuration that are needed, are:

- the delay to build frames or to extract data from frames, that are supposed to be equals, noted D_{Modbus},
- the delay to transmit frames on cables noted D_{Trans},
- the delay in switches noted D_{Sw} and,
- if necessary, the delay of RIOM to answer a request (containing delays due to all communication layers), noted D_{RIOM}.

Some of them are calculable but some are dependent on CPU features that are not well-documented, and consequently, difficult to set up.

To the best of our knowledge, the values of delay needed for our models are not existing in technical documentation and are very rare in scientific literature. Works are currently done by three laboratories (see chapter 2 page 39). In particular, Cena et al. [2006], gives an order of magnitude of common Ethernet boards to build or read Ethernet frames.
The delay for a switch to forward a frame can be found only for office computation. These last switches have not the same features than industrial ones. Indeed, office switches must transfer big data amount in a large time (about 100ms) while industrial switches must transfer small data amount quickly. To obtain this value, a measurement of forwarding time of industrial switches has been carried out at LURPA, and are going to be explained below.
The delay of a RIOM to answer is available neither in technical documentation nor in scientific literature, consequently the only way to obtain it is also to carry out measurement.

4.3.2.2 Obtaining delays by measurement

The measurement of:

- the delay for an industrial switch to forward a frame and
- the delay of a RIOM to answer a request

that have been carried out at LURPA are now explained.

The problem encountered is to have an accurate time measure on Ethernet frames. Indeed, all tools developed are oriented to measure throughput or bandwidth. However, it is difficult to derive time delays, that interest us, from throughput or bandwidth. Moreover, in these tools, time measurement is generally as accurate as the Operating System (OS). In common OS, there are multiple tasks running that interrupt one each other. In this case, measurement task can be interrupted and so the measurement will not be accurate. In Real-Time OS, this accuracy could certainly be better. However, it is always difficult to know the time accuracy of any OS.
As a consequence, the measurements carried out at LURPA have used a logical analyser which has a sampling period of 20 ns ($2 \cdot 10^{-5}$ ms). However, this device is dedicated to measure logic signals and not Ethernet frames, so it has been plugged on wires inside Ethernet cables to observe the logic signal of a frame. This is possible on 10MB Ethernet because the frame is coded on two levels. Knowing the code used, it is then possible to determine the beginning of a frame and its end.

To measure the latency of a device, the logical analyser is plugged on one side on the wire on which the frame enters, and on the other side on the wire on which goes out the frame, as shown figure 4.8. Then the logical analyser can give us the latency, which is delay between the date when the last bit of the frame has entered the device and the date when the last bit of the frame has gone out of the device.

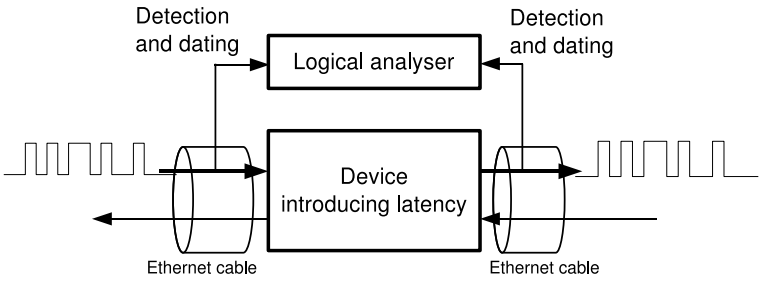


Figure 4.8: Measurement of device latency

This measure in two points implies to detect and date two signals which will be dated with a precision of 20 ns each. As a consequence, the total precision of the

measure is 40 ns, $4 \cdot 10^{-5}$ ms.
This experiment is made to measure the latency of a switch and of a RIOM to answer a request. The conditions of measurements ensure that the measured latencies correspond to the delay to forward one frame in switch and to the delay to answer one request in RIOM. The delay measured for the switch latency is minimum 0.01008 ms and maximum 0.01069 ms with a mean value of 0.010355ms. In the following, we consider only the mean value truncated to 0.01 ms. The delay measured for the RIOM to answer a request is minimum 0.5428 ms and maximum 0.5471 ms with a mean value of 0.543677ms. In the following, we consider only the mean value truncated to 0.54 ms.
This measurement have also permitted to measure other parameters as the lenght of Modbus frames which result is 75 bytes.

The table 4.1 sums up the different communication delays, their values and how they can be obtained.

Table 4.1: Delays taken into account for evaluating Network Cycle Time

Delay	Notation	Value (ms)	obtained by
Build frames or Extract data	D_{Modbus}	0.10	bibliography Cena et al. [2006]
Transmit frame	D_{Trans}	0.06	lenght of frame (75 bytes) divided by the throughput (10Mb/s)
Forwarding one frame in switch	D_{Sw}	0.01	measurement
Answering one request in RIOM	D_{RIOM}	0.54	measurement

4.4 Extract of one particular model

The dynamic behaviour of the part of the model of Ethernet Modbus client (figure 4.2) is explained in details on the example given figure 4.9. The chosen example is very light to enable a clear and detailed explanation.

The initial marking for this architecture, from top left to bottom right, is summed up in the table 4.2.

From this initial marking, a dynamic evolution of the Ethernet Modbus client model is presented in tables of figure 4.10 in the form of the new marking produced

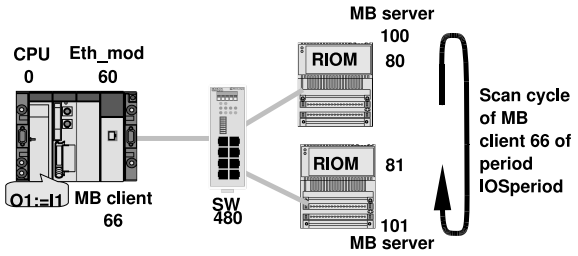


Figure 4.9: Example of a very simple architecture with one controller, one switch and two RIOMS

Table 4.2: Initial marking of Modbus client model on the example of figure 4.9

Place name	Id	Initial marking function	Initial marking value for the example
Modbus responses from servers	1	allETHMxnilMBFL()	1‘(60,[])@0
Extracted variables	2	allETHMxnilVARL()	1‘(60,[])@0
Collected responses	3	allPLCSCANNER responses()	1‘(66,2)@0
Next scan to perform	4	allIOSCANNER()	1‘(60,66,[100,101])@0
Powered down ETHM	5	allETHM()	1‘60@0
EthM CPU	6	allETHM()	1‘60@0
Current Scan	7		
Scan free	8		
Modbus requests to servers	9	allETHMxnilMBFL()	1‘(60,[])@0
Variables ready to be sent	10	allETHMxallSERVERx allCLIENTxnilVARL()	1‘(60,100,66,[])@0 ++ 1‘(60,101,66,[])@0

in each place when a transition is fired.

As this model is a sub-part of the complete model, some evolutions depend on other models, this is notify by the mention *In another model*. It is used for markings of the four interface places and also of the resource place *EthM CPU*. The time, from dO to d7, is increasing. At time dO, the model is in initial configuration. The first action is the boot of the Ethernet module by firing the transition T312 *Bootstrapping*. A token is then produced in the place 8 *Scan free* which is available after a time given by the function ETHM_Bootstrap_delay(60). This function returns randomly an integer between 0 and the scanning cycle time value. All the hardware devices have such a random bootstrap delay, to avoid non-desired synchronisations. After this bootstrap delay, at time d2, the scan cycles can begin immediately. So the transition T313 *Start new scan cycle* is fired to produce a token in the place 7 *Current scan* containing for each Ethernet module and each client, the remaining list of server to scan. The token from place 4 *Next scan to perform* used to fire this transition is available after the time IOSperiod(scanner), notify by *@+IOSperiod(scanner)* which means "adding IOSperiod(scanner) value to the current time value". In this case, the concerned scanner, or client, is 66 and the function IOSperiod(client) gives for each client a period value. The token in the place 7 is immediately available, and so at time d1, the building of Modbus frame begins, with the first firing of the transition T311 *Create a Modbus request*. This operation is proceeded one time for each pair (client, server), here (66,100). As it is explained previously, when firing this transition, the first variable in the list of the token for the corresponding client and server in place 10 *Variables ready to be sent* is taken.
At the same time, an element is added in the list of place 9 *Modbus requests to servers* for the corresponding Ethernet module. This list contains the model of Modbus frames, i.e. a set of client, server and a list of variables. All tokens created by this transition are affected by a delay of MBdelay(60), which models the delay to produce a Modbus frame. It is to notice that by firing transition T311, one token is removed from place 6 *EthM CPU* to model the resource allocation, but it does not produce a new one. Indeed, the resource is allocated from this creation of a Modbus request to its encapsulation into an Ethernet frame which is realised in an other Petri Net, *EThMod_Eth-TCP-IP_layers*. That is why, to perform the next creating of Modbus frame, we must wait for evolution of this other model, which released the resource and leave a token in the place 6 *EthM CPU*. Then, the transition T311 is fired once again to create a Modbus frame from the client 66 to the server 101. The evolution of marking in places is similar to the first firing. After the time MBdelay(60), the token available in the place *Current scan*, (60,66,[]), contains an empty server list, so this is the transition T314 *End of scan cycle* which is fired at the time d2+MBdelay(60). Thus, the token in place *Current scan* is removed to produce one in place 8 *Scan free*.

In the two following lines, at dates d3 and d4, marking of the place 6 evolves

transition fired	time	1	2
	d0 = 0	1`(60,[])	1`(60,[])
T312	d0 = 0	1`(60,[])	1`(60,[])
T313	d1 = ETHM_Bootstrap_delay(60)	1`(60,[])	1`(60,[])
T311	d1 = ETHM_Bootstrap_delay(60)	1`(60,[])	1`(60,[])
In another model	d2	1`(60,[])	1`(60,[])
T311	d2	1`(60,[])	1`(60,[])
T314	d2 + Mbdelay(60)	1`(60,[])	1`(60,[])
In another model	d3	1`(60,[])	1`(60,[])
In another model	d4	1`(60,[(66,100,[(1,[60,0,60,66,101,81])])])	1`(60,[])
T310	d4	1`(60,[])@+MBdelay(60)	1`(60,[(1,[0,60,66,101,81])])@+MBdelay(60)
In another model	d5	1`(60,[(66,101,[])])	1`(60,[(1,[0,60,66,101,81])])
T310	d5	1`(60,[])@+MBdelay(60)	1`(60,[(1,[0,60,66,101,81])])@+MBdelay(60)
T313	d6 = Max{d4+MBdelay(60) , d1+IOSperiod(66)}	1`(60,[])	1`(60,[(1,[0,60,66,101,81])])
computation of variables	...		
	d7	1`(60,[])	1`(60,[])
T311	d7	1`(60,[])	1`(60,[])

transition fired	time	3	4
	d0 = 0	1`(66,2)	1`(60,66,[100,101])
T312	d0 = 0	1`(66,2)	1`(60,66,[100,101])
T313	d1 = ETHM_Bootstrap_delay(60)	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
T311	d1 = ETHM_Bootstrap_delay(60)	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
In another model	d2	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
T311	d2	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
T314	d2 + Mbdelay(60)	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
In another model	d3	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
In another model	d4	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
T310	d4	1`(66,1)@+MBdelay(60)	2`(60,66,[100,101])@+IOSperiod(66)
In another model	d5	1`(66,1)	2`(60,66,[100,101])@+IOSperiod(66)
T310	d5	1`(66,2)@+MBdelay(60)	2`(60,66,[100,101])@+IOSperiod(66)
T313	d6 = Max{d4+MBdelay(60) , d1+IOSperiod(66)}	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)
computation of variables	...		
	d7	1`(66,0)	1`(60,66,[100,101])@+IOSperiod(66)
T311	d7	1`(66,0)	2`(60,66,[100,101])@+IOSperiod(66)

transition fired	time	5	6
	d0 = 0	1`60	1`60
T312	d0 = 0	ϕ	1`60
T313	d1 = ETHM_Bootstrap_delay(60)	ϕ	1`60
T311	d1 = ETHM_Bootstrap_delay(60)	ϕ	ϕ
In another model	d2	ϕ	1`60
T311	d2	ϕ	ϕ
T314	d2 + Mbdelay(60)	ϕ	ϕ
In another model	d3	ϕ	1`60@+MBdelay(60)
In another model	d4	ϕ	ϕ
T310	d4	ϕ	1`60@+MBdelay(60)
In another model	d5	ϕ	ϕ
T310	d5	ϕ	1`60@+MBdelay(60)
T313	d6 = Max{d4+MBdelay(60) , d1+IOSperiod(66)}	ϕ	1`60
computation of variables	...		
	d7	ϕ	1`60
T311	d7	ϕ	ϕ

transition fired	time	7	8
	d0 = 0	ϕ	ϕ
T312	d0 = 0	ϕ	1`60@+ETHM_Bootstrap_delay(60)
T313	d1 = ETHM_Bootstrap_delay(60)	1`(60,66,[100,101])	ϕ
T311	d1 = ETHM_Bootstrap_delay(60)	1`(60,66,[101])@+MBdelay(60)	ϕ
In another model	d2	1`(60,66,[101])	ϕ
T311	d2	1`(60,66,[])@+MBdelay(60)	ϕ
T314	d2 + Mbdelay(60)	ϕ	1`(60,66,[])
In another model	d3	ϕ	1`(60,66,[])
In another model	d4	ϕ	1`(60,66,[])
T310	d4	ϕ	1`(60,66,[])
In another model	d5	ϕ	1`(60,66,[])
T310	d5	ϕ	1`(60,66,[])
T313	d6 = Max{d4+MBdelay(60) , d1+IOSperiod(66)}	1`(60,66,[100,101])	ϕ
computation of variables	...		
	d7	1`(60,66,[101])	ϕ
T311	d7	1`(60,66,[])@+MBdelay(60)	ϕ

transition fired	time	9	10
	d0 = 0	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T312	d0 = 0	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T313	d1 = ETHM_Bootstrap_delay(60)	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T311	d1 = ETHM_Bootstrap_delay(60)	1`(60,[(66,100,[])])@+MBdelay(60)	1`(60,100,66,[])@+MBdelay(60)++ 1`(60,101,66,[])
In another model	d2	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T311	d2	1`(60,[(66,101,[])])@+MBdelay(60)	1`(60,100,66,[])++ 1`(60,101,66,[])@+MBdelay(60)
T314	d2 + Mbdelay(60)	1`(60,[(66,101,[])])@+MBdelay(60)	1`(60,100,66,[])++ 1`(60,101,66,[])
In another model	d3	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
In another model	d4	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T310	d4	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
In another model	d5	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T310	d5	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
T313	d6 = Max{d4+MBdelay(60) , d1+IOSperiod(66)}	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[])
computation of variables	...		
	d7	1`(60,[])	1`(60,100,66,[])++ 1`(60,101,66,[(1,[101,81])])
T311	d7	1`(60,[(66,101,[(1,[101,81])])])@+MBdelay(60)	1`(60,100,66,[])++ 1`(60,101,66,[])@+MBdelay(60)

Figure 4.10: Partial marking evolution of places 1 to 10

due to the model *EThMod_Eth-TCP-IP_layers*. It models first, the release of the resource after having sent the second Ethernet frame and, second, the arrival of a response from RIOM. So, it produces a token in the place 1 *Modbus responses from servers*. Let us note that this token contains an event (1,[60,0,60,66,101,81]) which is identified by number one. It comes from server 100 and is destined to server 101 on RIOM 81 as indicated in the route of event. This event is treated inside the controller as a variable. All the markings concerning this variable are in grey cells. After, at time d5, the transition T310 *Extract variables from Modbus responses* is fired. It takes the first element of variable list from place 1 *Modbus responses from servers* to insert it at the end of the variable list in place 2 *Extracted variables*. It also takes a token in the place 3 *Collected responses* to produce a new one with a value "n" incremented. This is used to count the number of responses of current scan arrived. The produced tokens are going to be available after a delay given by the function MBdelay(60). After this delay, the token in the place 1 can receive a new response. It occurs at time d5, with a response coming from server 101 that does not contain any event. The transition T310 is once again fired, adding a new empty variable list, to the one of place 2. It also increments the number of response received in the place 3. As the client 66 has received all the response, the transition T313 *Start new scan cycle* is enabled. It is immediately fired if the corresponding token in the place 4 *Next scan to perform* is available, i.e. if the current time is upper than the time-stamp of the token which is d1+IOSperiod(66). In the other case, the firing of transition is delayed until this time.

The next evolution of interest is the sending of the Modbus request containing the variable to be sent to the server 101. So, we skip the evolution until the model is at time d7. The situation is a resource available in place 6 *EthM CPU*, the server 101 to scan (token in place 7), and a variable to send for this server ((1,[101,81]) in token of place 10). In this condition the transition T311 *Create a Modbus request* is fired immediately. Thus a token is produced in place 9 *Modbus requests to servers* which contains the variable to send to server 101.

Conclusion

Four extracts of the generic model have been detailed: the IO scanner (Modbus client) of the PLC-based controller, the PC-based controller, the event source and the switch. Then the instantiation process has been presented, before to detail the IO scanner model of a particular architecture.
The next chapter details how to obtain time performances by simulating an instantiation of the generic model.

Chapter 5

Obtaining time performances from a particular model

5.1 Method overview

The previous chapter has detailed the generic model of Ethernet-based automation systems and its instantiation. To obtain time performances by simulation, from a particular model, four steps are usually performed:

1. excitation scenario definition
2. set up of the model
3. simulation itself
4. useful data extraction from the raw data that yielded the simulation and post-treatment of these useful data to obtain time performances

Figure 5.1 depicts the steps of our approach. First, the generic model is instantiated so as to obtain a particular model describing the particular architecture whose time performance are to be assessed. Then this particular model is set up in order to carry out realistic simulation and consequently to obtain meaningful simulation results. Once the particular model set up, simulation is launched provides a log file where all evolutions stored. Finally, a post-treatment is necessary to obtain the required values in an ergonomic form.

It matters to emphasise that no excitation scenario is to be defined in our approach. Indeed, any particular model obtained from our generic model is autonomous and hence can be simulated using merely a time-driven simulation strategy, i.e. a strategy in which the evolutions of the model are computed each time unit, without need of an external model that constraints the evolution.

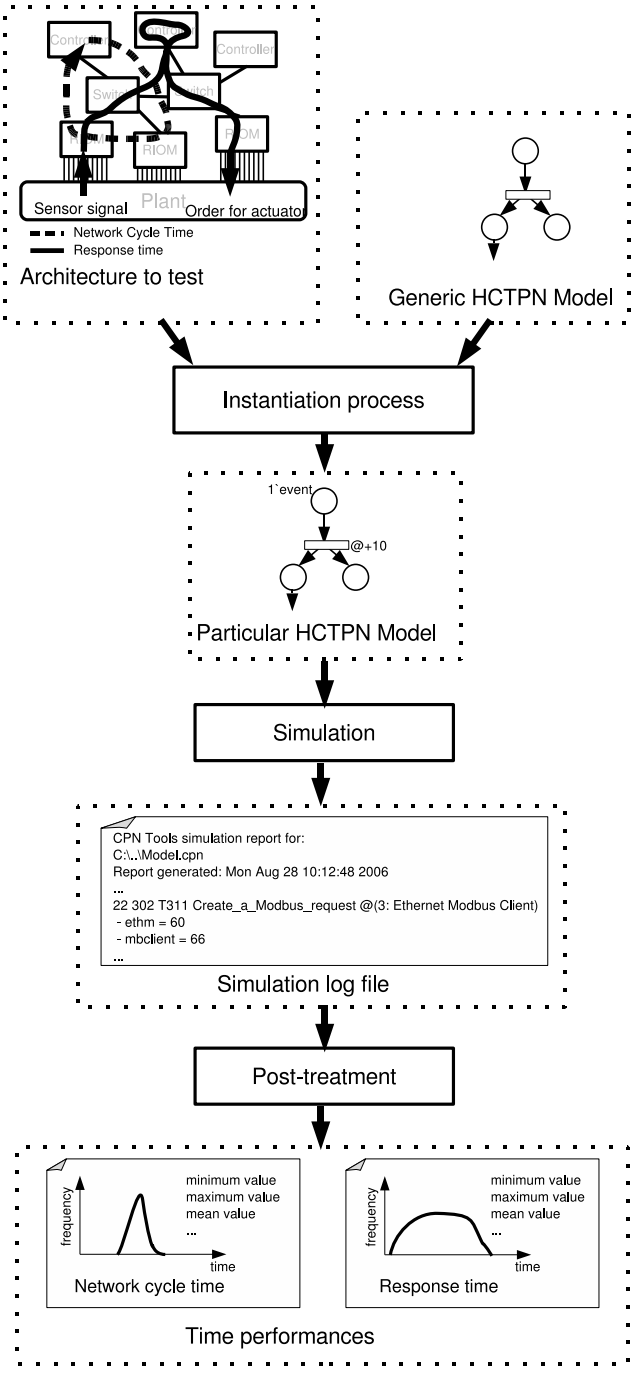


Figure 5.1: Method to evaluate time performances

This choice of a time-driven simulation strategy is motivated by our concern of *simulating a loaded network*. Simulation is run on a particular model of a system including a network whose traffic load is independent of the occurrences of events issued from the plant and depends only on the number of components (RIOMs, switches and controllers), on their connections and on their features (processing time, number of scanned RIOMs). Hence simulation will provide results for a network at constant load.

The next two sections present the details of, first, the set up step and, second, simulation and post-treatment steps.

5.2 Set up of one particular model

Once a particular model obtained from a generic one is built, it must be set up so as to give rise to realistic simulation results. Set up of a particular model consists in:

- introducing random data processing times dispersion to avoid unrealistic synchronisations
- defining the event generation scheduling for response time evaluation

5.2.1 Introducing partially random execution times

It has been mentioned, in section 3.2.3 page 48, that each data treatment is modelled by a transition to which a constant delay is associated. If such a model, including constant firing times, is simulated, perfect synchronisation between different processes may occur. This is not realistic and hence must be avoided because simulation results would not be meaningful in that case. Real processes include always small time deviations. Therefore, to obtain more realistic results, the model must be set up prior to be simulated, by adding:

- Random bootstrap delays for each controllers to initiate their cycles at different dates and
- Dispersions (about some per thousands of the value) on treatment times.

The example below shows how this is performed for the execution times of the processors of controllers:

$$
\begin{aligned}
fun\ allPLCPROC() &= 1`0 ++ 1`1 ++ 1`2;\\
fun\ allPCBPROC() &= 1`3 ++ 1`4 ++ 1`5;\\
fun\ allPROC() &= allPLCPROC() ++ allPCBPROC();\\
fun\ PROCEdelay(proc) &= withdispersion(case\ proc\ of\\
&\quad 0 \Rightarrow 2000\\
&\quad |1 \Rightarrow 18000\\
&\quad |2 \Rightarrow 50000\\
&\quad |3 \Rightarrow 2000\\
&\quad |4 \Rightarrow 18000\\
&\quad |5 \Rightarrow 50000\\
&\quad |\_ \Rightarrow 0\\
&\quad , general\_dispersion\_range\_in\_perthousand);
\end{aligned}
$$

with the function *withdispersion*:

$$
\begin{aligned}
fun\quad & withdispersion(delay, dispersion\_range\_in\_perthousand) = delay +\\
& discrete(\tilde{}\ delay * dispersion\_range\_in\_perthousand\ div\ 1000,\\
& delay * dispersion\_range\_in\_perthousand\ div\ 1000)
\end{aligned}
$$

and the value *general_dispersion_range_in_perthousand*:

$$val\ general\_dispersion\_range\_in\_perthousand = 5$$

In these declarations, three processor modules are declared (0, 1 and 2) as well as three PC-based controller (3, 4 and 5). Their set defines all controllers resources (allPROC()) which have an attribute PROCEdelay(proc) defining the execution time of user program. This time is defined with the function *withdispersion*, used to add dispersion of 1/100 around the value in this case. The function *discrete*s is a CP-NML predefined function that returns a random value between the two values given in brackets, here it is between $-5/1000$ and $5/1000$.

Similar adaptations are made for treatment times of Ethernet modules, RIOMs and switches.

5.2.2 Event generation scheduling for response time evaluation

We remind the reader to not confuse this event generator with a simulation scheduler defining a scenario. Actually, the cause event source must be modelled to evaluate

a response time between a cause event and a consequence event on the plant. This model has been presented in the previous chapter, section 4.2.3. Depending on the function associated to this event, it could occur erratically, for instance for an alarm, or cyclically, for instance for a position sensor on a transfer line. In order to cover all possible cases, the choice made for this model is to have an event occurrence periodically in a way to sweep the IO scanning period.
Thus, the events are generated periodically but to have a well distributed event generation, as mentioned previously, we have set this period to sweep the IO scanning period. This is clear on the figure 5.2 where is pictured the time each event waits for a request in the RIOM, in system with modular controllers (on the top) and in another one with PC-based controllers (on the bottom). This time is well distributed between zero and one IO scanning cycle time.

5.3 Simulation and post-treatment

Simulation of the model is proceeded in the software CPNTools. During simulation, the default data log file stores all fired transitions with the current time and the list of set variables. These variables are all the ones contains in tokens concerned by the current firing transition. From this text file, we have to extract the interesting dates to calculate time performances.

The simulation duration, for the same number of transition firing, depends on the number of tokens in the model. As a consequence, the larger is the architecture, the longer is the simulation. For instance, on the architecture given figure 3.10, to simulate 10^7 steps, i.e. fired transitions, takes around 50 minutes of CPU time on a PC Pentium 4 2 GHz. This corresponds to about 7000 evaluated response time and also to a log file of 1,5 GByte. Then, it remains to extract the interesting data from this log file and calculate the wanted performances. To parse the log file, a prototype was running in Python. Using this interpreted language leads to really long calculus (several hours) hence to reduce this time we used regular expressions that call compiled C code. With this new version, in around 30 minutes of CPU time the interesting data are extracted and saved in new files. Finally, these last data are treated in Matlab to calculate in less than five minutes all the needed performances and to draw graphic representations. Hence to compute response time of the architecture presented figure 3.10, less than 1 hour and 30 minutes is needed.

Given this value, it is reasonable to consider that in one day of work, the proposed approach enables us to compare four to five different architectures, if nothing is done in parallel. However, considering the size of the log file, a problem of storing this file could arise to simulate biggest architectures. A solution could be to stop using the default log file and to save in a adapted log file only the required data.

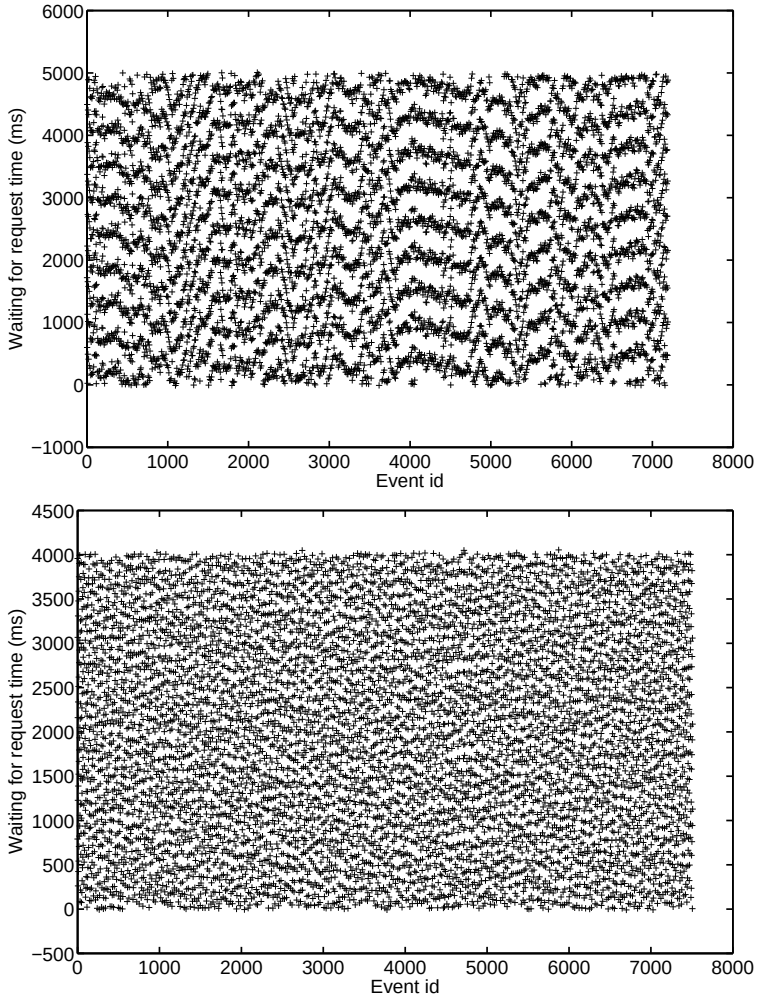


Figure 5.2: Repartition of the time that each event has waited a request in RIOM: on the top, modular controllers are used with a IO scanning cycle of 5 ms and on the bottom PC-based controllers are used with a user program of 2 ms and a network cycle around 2 ms.

Conclusion

To obtain the two time performances that we consider (network cycle time and response time) from a particular model, no excitation scenario is required, the particular models being autonomous by construction. Set up of a particular model before simulation consists in introducing:

- introducing random data processing times dispersion to avoid unrealistic synchronisations
- defining the event generation scheduling for response time evaluation

Simulation results are obtained in a reasonable time (around 50 minutes) for architectures including several controllers, switches and RIOMs, that allows comparison of architectures in times that comply with industrial design constraints.

Hence, it is now possible to simulate easily Ethernet-based Automation Systems using client/server cooperation model and, from simulation results, to compare this cooperation model with the two other ones presented in chapter 1. Moreover, it is also possible to assess the contributions of the different element of these systems (controllers, RIOMs, switches) and of the three time consumption mechanisms (data processing time, waiting for synchronisation time, waiting for resource time) to their time performances. These are the objectives of the next chapter.

Chapter 6

Evaluation of time performances

In this chapter, two major time performances are studied: the network cycle time and the response time.

In chapter 1, three cooperation models have been presented: master/slave, producer/consumer and client/server. They are compared in section 6.2 on the basis of their Network Cycle Time. First, this enables us to compare our results to those obtained previously by other researchers (Tovar and Vasques [1999], Vitturi [2000, 2001]) who evaluated Network Cycle Time for master/slave and producer/consumer models. Moreover, Network Cycle Time is an important characteristic widely used by fieldbus users to choose among various solutions. this comparison will allow us to determine whether client/server is a suitable solution for automation systems, compared to master/slave and producer/consumer.
The simulation model presented chapter 5 is then used to evaluate response time of Ethernet-based Automation Systems and to characterise the major time consumption mechanisms in both PC-based and PLC-based architectures. This analysis is carried out through the decomposition of response time into three elementary delays: time for processing data, waiting time for synchronisation and waiting time for availability of resources. The evaluation is done on three case studies for PC-based and PLC-based architectures, with different level of shared resources.

6.1 Presentation of the case studies

The studied cases are based on three different configurations which are simple enough to be instantiated and simulated fast. For each one of these configurations, PLC-based (figure 6.1) and PC-based architectures (figure 6.2) will be studied, so as to compare these two kinds of architectures.

The parameters set up for all three configurations are summed up in table 6.1.

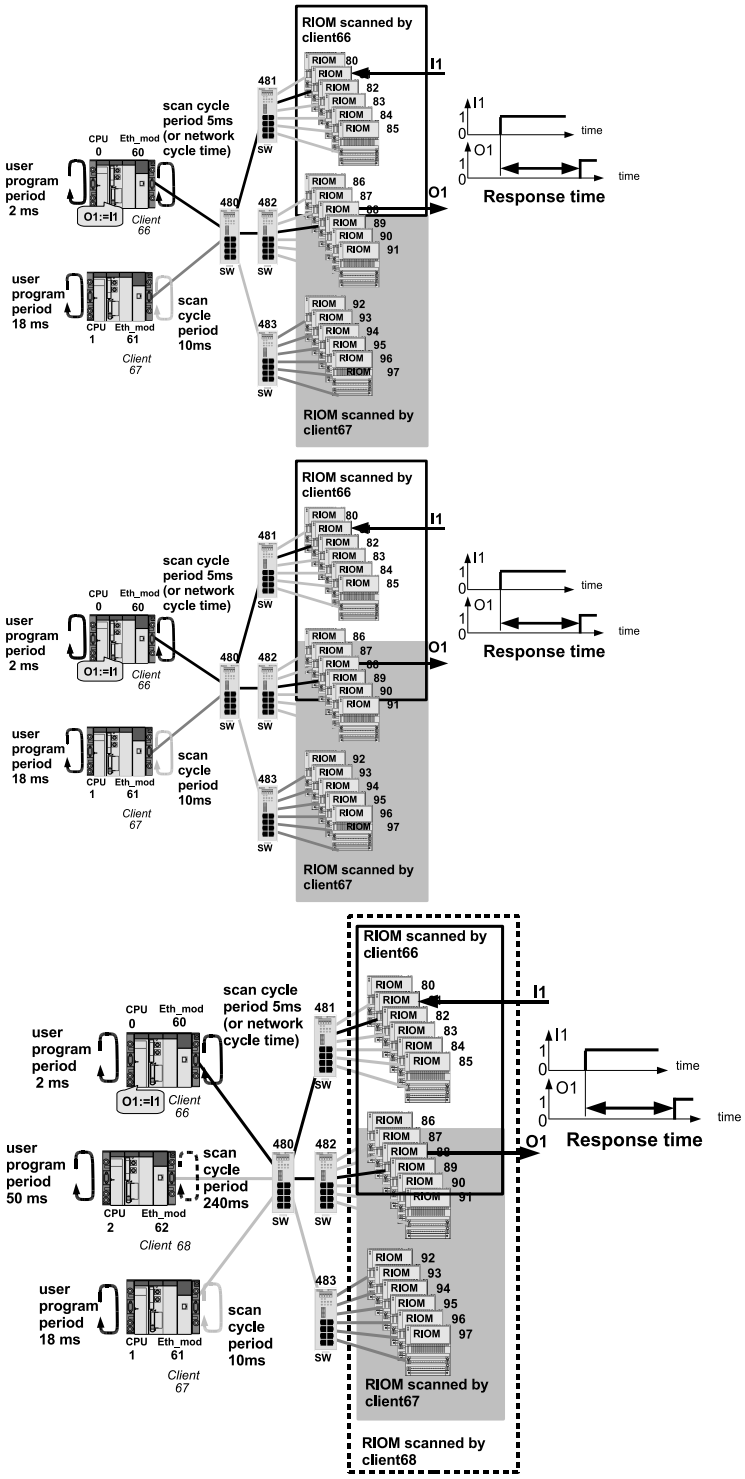


Figure 6.1: Configurations 1 (top), 2 (middle) and 3 (bottom) studied with PLC-based architectures

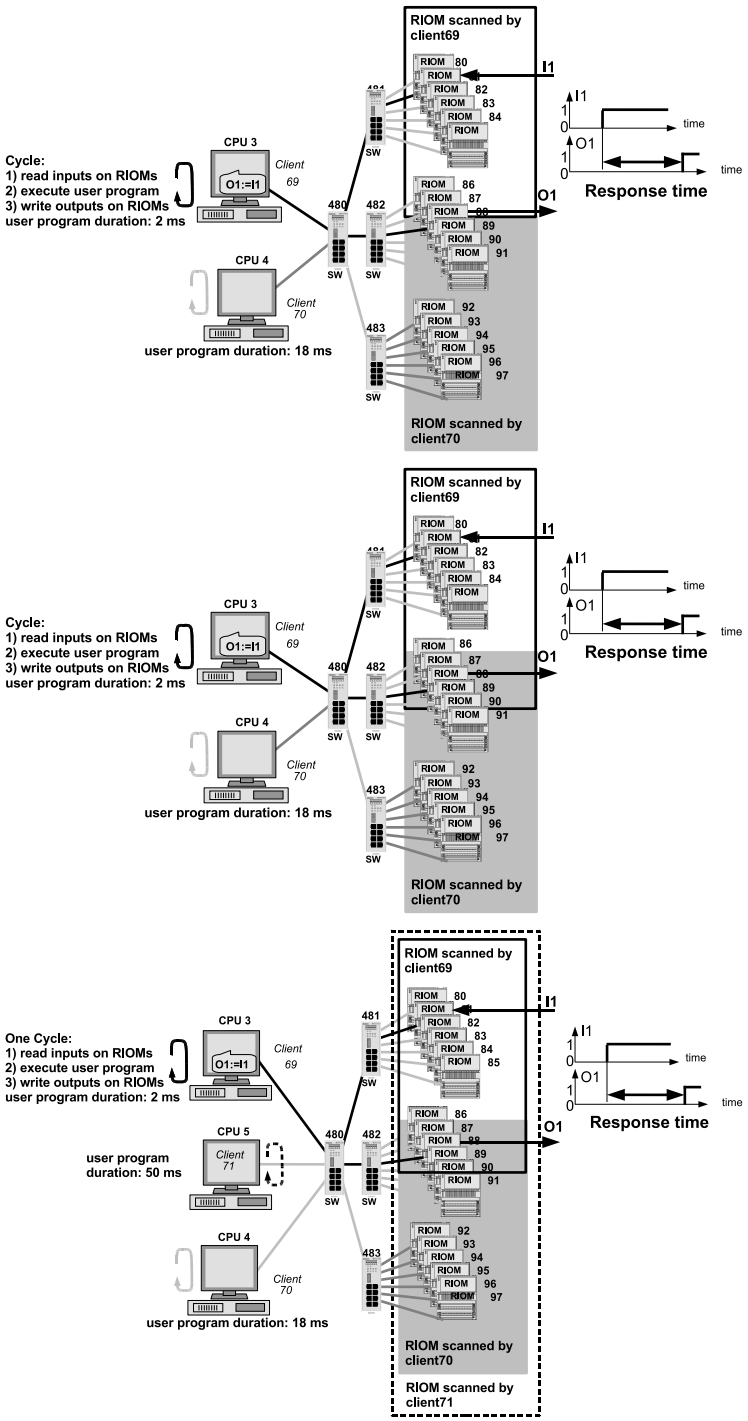


Figure 6.2: Configurations 1 (top), 2 (middle) and 3 (bottom) studied with PC-based architectures

Table 6.1: Characteristics of modular controllers and PC-based controllers architectures for the three configurations (time in ms)

Configuration		1	2	3
User program period on CPU 0		2	2	2
User program period on CPU 1		18	18	18
User program period on CPU 2				50
EthMod 60	IO Scanning Cycle Time	5	5	5
	RIOMs scanned	80 to 88	80 to 90	80 to 90
EthMod 61	IO Scanning Cycle Time	10	10	10
	RIOMs scanned	89 to 97	87 to 97	87 to 97
EthMod 62	IO Scanning Cycle Time			240
	RIOMs scanned			80 to 97
CPU 3	User program duration	2	2	2
	RIOMs scanned	80 to 88	80 to 90	80 to 90
CPU 4	User program duration	18	18	18
	RIOMs scanned	89 to 97	87 to 97	87 to 97
CPU 5	User program duration			50
	RIOMs scanned			80 to 97

In the first configuration, no Ethernet RIO module is shared. In this case all components are not strongly solicited. This configuration is very reactive, with user program cycle times of 2 ms and 18 ms, and IO scanning cycle times, i.e. the time set up for one controller to scan all the RIOMs needed, of 5 ms and 10 ms.

In the second configuration, the two controllers share four Ethernet RIO modules (number 87 to 90). More resources are shared and the cycle times in controllers are the same, so both controllers and shared RIOM are more solicited than in configuration 1.

In the third configuration, a third controller that scans all RIOMs is added to the previous configuration. This third scan has a slow period (50 ms) and can be understood as associated to a supervision function. Here, all the resources are highly solicited.

We would want to remind also that in PLC-based controllers two processes (user program execution and IO scanning) are running in parallel while in a PC-based controller, these two processes are performed sequentially. In this later case, the duration of the global CPU cycle (user program execution and IO scanning) is not constant because the duration of IO scanning can vary. Therefore to obtain comparable results, the duration of the user program executions in PC-based architectures

are set up equals to the user program period in the corresponding PLC-based architectures.

Only the PLC-based architectures are considered for comparison of the three cooperation models on the basis of their Network Cycle Time. Controllers using master/slave or producer/consumer models must have a dedicated communication module. This explains why only modular controllers are retained in section 6.2.

Conversely, the evaluation of response time of Ethernet-based Automation Systems will address comparison of time performances of PLC-based and PC-based architectures.

6.2 Comparison of master/slave, producer/consumer and client/server cooperation models

6.2.1 Evaluation methods of Network Cycle Time

We remind here that the network cycle time is defined as the time between two consecutive sendings of message from one master, or producer or client to one slave, or consumer, or server.

Concerning master/slave and producer/consumer models, their Network Cycle Time can be determined exactly by analytic calculus given the feature of the network. Concerning client/server model, on the contrary, its Network Cycle Time cannot be determined exactly by analytic calculus from the feature of the network, because the delays due to shared resources are variable. So for this last model, the simulation model is used to evaluate the distribution of its Network Cycle Time.

Master/slave model

The calculus is proceeded considering an Ethernet-based Automation System using TCP and IP protocols.

In master/slave model (figure 1.6 page 14), a master sends a request to a slave, waits for the answer then it can proceed for the next slave. In multi-master configuration, one master scans all its slaves then next master can proceed. First, only one pair of one master M_i and one slave S_j is considered. For this pair, we can calculate the time of an exchange, $T_{M_iS_j}$, taking into account the number of switches, N_{Sw}, between the two devices.

$$\begin{aligned} T_{M_i S_j} &= D_{Modbus} + N_{Sw} \cdot D_{Sw} + (N_{Sw} + 1) \cdot D_{Trans} + \\ & \quad D_{RIOM} + N_{Sw} \cdot D_{Sw} + (N_{Sw} + 1) \cdot D_{Trans} + D_{Modbus} \end{aligned}$$

The delays that are taken into account in this formula are:

- the delay to build one Modbus frame in the controller, D_{Modbus},
- the delay introduced by one switch to forward the request, D_{Sw}, multiplied by the maximum number of switch between controller and RIOMs, N_{Sw},
- the delay to transmit the request, D_{Trans}, multiplied by the maximum number of links between controller and RIOMs, $N_{Sw} + 1$,
- the delay to answer a request from the controller in the RIOM, D_{RIOM},
- the delay introduced by one switch to forward the answer, D_{Sw}, multiplied by the maximum number of switch between controller and RIOMs, N_{Sw},
- the delay to transmit the answer, D_{Trans}, multiplied by the maximum number of links between controller and RIOMs, $N_{Sw} + 1$,
- the delay to extract data from Modbus answer in the controller, D_{Modbus}.

The assumptions made on the system are the following:

- all devices of one type (switch, controller, RIOM) have the same features,
- all cables have the same throughput,
- all frames have the same size,
- all requests from controllers are of the same type and size and as a consequence the answers from RIOMs also,
- requests and answers have the same size.

The consequences on the elementary delays are:

- all the switches have the same forwarding time,
- all the RIOMs have the same answering time,
- all Ethernet modules have the same processing time,
- the time for building a Modbus frame is the same that the time for extracting data from a Modbus frame.

To coordinate the masters, a message is sent by one master, when it has finished is scan, to the next master that should proceed. The message is supposed to be in Modbus and to have the same length that requests and answers and so to take the same time to be built, to be read and to be transmitted. The time to coordinate masters is noted $T_{M_iM_{i+1}}$:

$$T_{M_iM_{i+1}} = D_{Modbus} + N_{Sw} \cdot D_{Sw} + (N_{Sw} + 1) \cdot D_{Trans} + D_{Modbus}$$

Then to obtain the Network Cycle Time, first, one sums for each master the times for all exchanges with all slaves which is $\sum_j T_{M_iS_j}$, with j in the number of slaves scanned by the master M_i. Finally, it is to add all this times for all masters. This gives the following equations:

$$T_{NetworkCycle} = \sum_{i=1}^{Nb\ Master} (\sum_j T_{M_iS_j} + T_{M_iM_{i+1}})$$

Among the references given in chapter 2, two authors have proposed expressions of the Network Cycle Time.

Vitturi [2001] focuses on Network Cycle Time for master/slave model and proposes a formula taking into account variable delays and based on exchange delays and not on component latencies. Hence this formula cannot be directly compared to our.

Tovar and Vasques [1999] focuses on Network Cycle Time of Profibus, which also uses a master/slave model. The authors take into account two priority levels for messages, that is not the case in our approach. However, they consider the same elementary delays.

Producer/ consumer model

The calculus is proceeded considering an Ethernet-based Automation System using UDP and IP protocols. In this case, the use of TCP is not possible because it does not enable to use broadcast mechanism. Given that UDP is similar to TCP, we consider for both protocols the same delays for building or treating frames.

In producer/consumer model (figure 1.7 page 16), a producer sends a frame to one or several consumers. When the frame is consumed then another producer can use the network. In the case of automation systems we focus on, a producer (or a consumer) can be either an Ethernet module or a RIOM. When a message is sent from a producer to a consumer, four delays occur:

- the delay to build one Modbus frame or extract data from one Modbus frame in the controller, D_{Modbus},

- the delay introduced by one switch to forward the request, D_{Sw}, multiplied by the maximum number of switch between controller and RIOMs, N_{Sw},
- the delay to transmit the request, D_{Trans}, multiplied by the maximum number of links between controller and RIOMs, $N_{Sw} + 1$,
- the delay to read the request from the controller or to build the Modbus answer in the RIOM, $D_{RIOM}/2$,

Whatever the direction of exchange (from Ethernet module to RIOM or from RIOM to Ethernet module), the delay for one producer is:

$$T_{P_iC} = D_{Modbus} + N_{Sw} \cdot D_{Sw} + (N_{Sw} + 1) \cdot D_{Trans} + D_{RIOM}/2$$

The assumptions made on the system are the same than for master/slave. Moreover, we assume that the time for RIOM to answer a request is split in two equal parts ($D_{RIOM}/2$). One part is for consuming the frame from controller, and the other one is for producing one frame for controllers.
We assume that the producers are coordinated off-line, i.e. each producer knows when it should produce, and there is neither bus arbiter nor token passing system. As a consequence, there is no delay to consider for coordinating producers.
Then to obtain the Network Cycle Time, one should add all this times for all producers. This gives the following equations:

$$T_{NetworkCycle} = \sum_{i=1}^{Nb\ Producer} T_{P_iC}$$

An other proposal of an analytic formula of Network Cycle Time of producer/consumer model can be found in Vitturi [2001]. Unfortunately, like for master/slave, this analysis does not use the same features than ours and then the formula proposed in this reference cannot be compared to that we obtain.

Client/server model

The simulation is proceeded considering an Ethernet-based Automation System using TCP and IP protocols.

In client/server model (figure 1.11 page 21), a client send requests one after the other to all servers it has to scan, then it waits for the answers before to proceed to the next cycle. In multi-client configuration, the clients scan their servers in parallel independently one of each other. Therefore there are shared resources: switches, RIOMs and controllers themselves. That is why the Network Cycle Time cannot be calculated with a formula. As a consequence, it will be evaluated by simulation of the particular models of the studied architectures.

6.2.2 Numerical values of Network Cycle Times for the studied architectures

For master/slave and producer/consumer Network Cycle Time, one has to apply the formulas given in the previous section. For the three architectures we study, delays are replaced by the numerical values of table 4.1 and the maximum number of switches, N_{Sw}, is equal to two. For client/server Network Cycle Time, simulation enables us to obtain its distribution. The results for the three models are given table 6.2 with only minimum and maximum delays for client/server model.

Table 6.2: Network Cycle Time in milliseconds

Conf.	master/slave	producer/consumer	client/server	
			Min	Max
1	21.18	11.40	2.02	2.04
2	25.74	11.40	2.39	3.29
3	46.59	11.97	2.39	3.84

For the master/slave model, our results are close to the delays calculated in Tovar and Vasques [1999] which found for Profibus with three masters a cycle time of 41 ms. The difference between their value and our can be explained by the difference of configuration and of latencies between Ethernet and Profibus components.

6.2.3 Discussion on Network Cycle Times obtained

6.2.3.1 Comparison of the three cooperation models

Compared to master/slave model, the producer/consumer model is more and more efficient when the number of components increases. The use of broadcast mechanism enables to have the same delay in the first two configurations because there is the same number of producers even if there are more consumers for some data. It also enables to increase just a little bit the cycle time for the third configuration, where only one producer is added.

Indeed, in the three configurations, master/slave is the slowest while client/server is the fastest. Even if the Network Cycle Time is not constant for client/server, for the tested architectures, it stays approximately ten times faster than master/slave and five times faster than producer/consumer. This first result highlights the high capability of client/server model, even with shared resources. It shows that this model enables so fast exchanges that it is possible to use it in real-time systems as

claimed in Stankovic [1992], Jasperneite and Neumann [2004].

6.2.3.2 Detailed analysis of Network Cycle Time of client/sever model

Before to skip to response time analysis, let us take a deeper look at the three distributions obtained for client/server model that are showed figures 6.3, 6.4 and 6.5.

In the first configuration, where only switches are shared, the Network Cycle Time is quasi-constant because the switches are so fast ($D_{Sw} = 0.01ms$) that they have a very small impact on the cycle time. This distribution of Network Cycle Time is then very narrow around a peak at 2.03 ms.

For the second configuration, the Network Cycle Time distribution covers a wider range. The mean and median values of this distribution are respectively 2.46 and 2.4 ms. The main peak corresponds to Network Cycle Time values between 2.39 and 2.43 ms; 68% of the distribution stands in this main peak. Then compared to the first configuration, this second configuration leads to longer Network Cycle Times and to increase of the difference between the minimum and maximum values of Network Cycle Times. This comes from the increasing number of shared resources: the two controllers share two RIOMs in configuration 2.

In the peak is also present but it is translated on the right, i.e. the minimum Network Cycle Time is longer than in configuration 1. The translation of this peak corresponds to the two new RIOMs to scan for each controller. After the peak, lines appears regularly until the maximum, i.e. the value of the Network Cycle Time is varying. The percentage of values inside the peak is 68%, between 2.39 and 2.43 ms. The mean value is 2.46 ms and the median one is 2.40 ms. So, this variation concerns few cases but the maximum is now 38% higher than the minimum.

The peak of the distribution of configuration 3 is not translated. It remains between 2.39 and 2.43 ms and includes approximately the same percentage (65%) of the distribution. The mean and median values are now 2.49 and 2.40 ms. Introducing a third controller that scans all RIOMs does not impact strongly the Network Cycle Time distribution mean and median values but leads to an increased maximum value of the Network Cycle Time. Indeed, the maximum value is now 61% higher than the minimum value while it was 38% higher in the configuration 2.

6.2.3.3 Synthesis

Comparison of the three cooperation models shows clearly that the client/server model permits faster cycles than the two other models and therefore suits Networked Automation Systems. However, the time performances of architectures using this

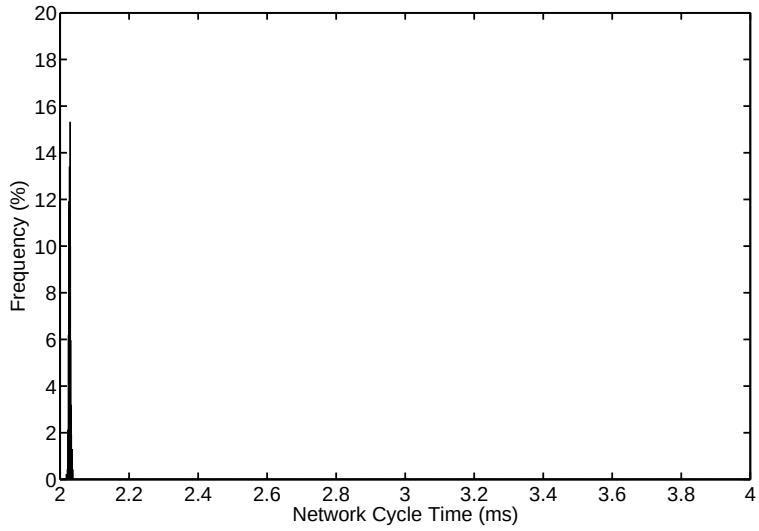


Figure 6.3: Client/server distribution of Network Cycle Time in configuration 1 with in abscissa the time in microseconds and in ordinate the frequency

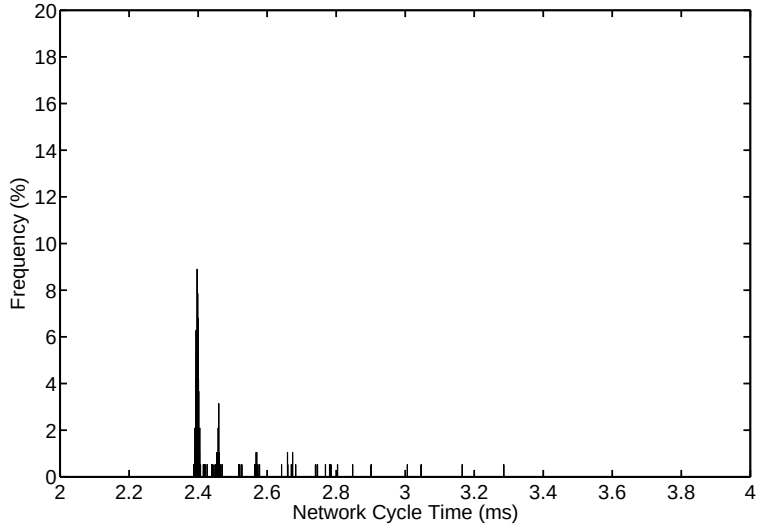


Figure 6.4: Client/server distribution of Network Cycle Time in configuration 2

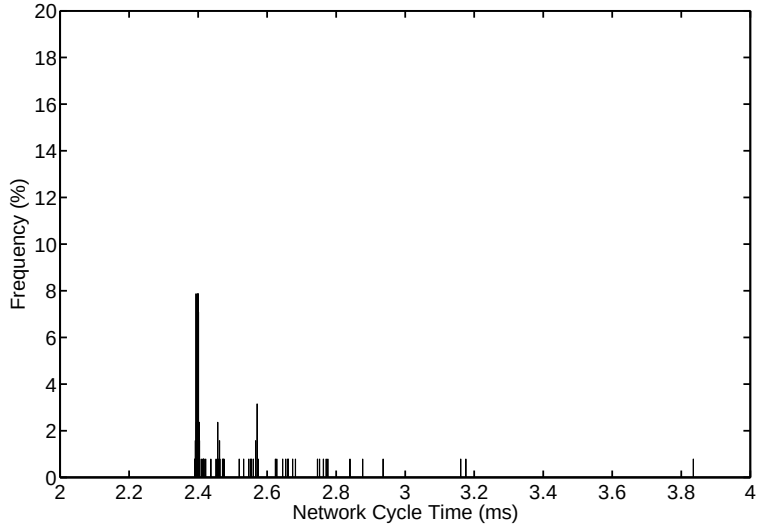


Figure 6.5: Client/server distribution of Network Cycle Time in configuration 3

cooperation model are not constant but given by a distribution that depends on the amount of shared resources. Hence the distribution of the response time of an Ethernet-based Automation Systems, the major time performance with the end user point of view, must be evaluated before operating this system so as to determine whether it fits the users requirements. Evaluation of the response time of the Ethernet-based Automation Systems is therefore the issue addressed in the next section.

6.3 Evaluation of response time of Ethernet-based Automation Systems

In this section, we focus on the response time of Ethernet-based Automation Systems using a client/server cooperation model. First, the whole response time distribution is analysed before to evaluate the influence of the different time consumption mechanisms and of switches.

6.3.1 Response time distribution

The distribution of response time is the characteristic on which focus is put in this subsection. It has been explained, in the chapter 1 page 7, that the response time of the automation architecture influences the performance of the plant (for instance, accuracy on the stop position of a trolley or on the final level of a liquid in a tank). Then the distribution of response time impacts the distribution of physical features of the plant, that explains why its knowledge is of great interest.

The distributions obtained for the three studied configurations are presented on the figures 6.6, 6.7 and 6.8, and the minimum and maximum values of these distributions are summed up in table 6.3.

The distribution obtained in configuration 1 for PLC-based architecture can be roughly approximated by a uniform distribution, between the minimum and maximum values. When the amount of shared resources increases (from figure 6.6 to 6.7), focusing always on PLC-based architectures, three comments can be made:

- the ”rectangular model” is no more valid, some values that corresponds to the highest response times, being clearly outside the main packet
- the range of response time increases
- this increased range comes only from the increase of the maximum value, the minimum value being not changed (the 0.01 ms difference is not significant

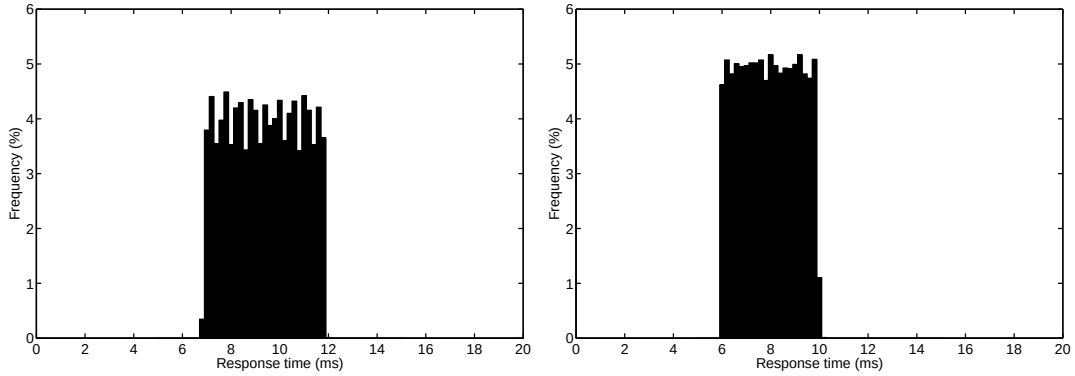


Figure 6.6: Histograms of response time in PLC-based architecture (left) and PC-based architecture (right) in configuration 1

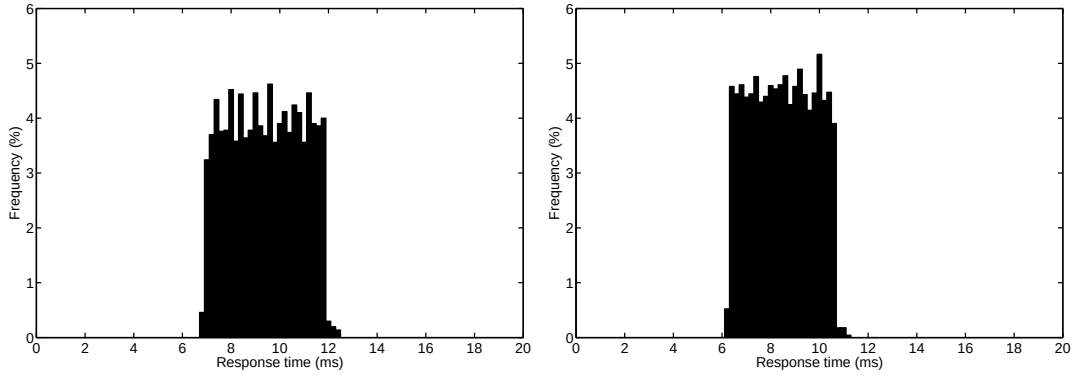


Figure 6.7: Histograms of response time in PLC-based architecture (left) and PC-based architecture (right) in configuration 2

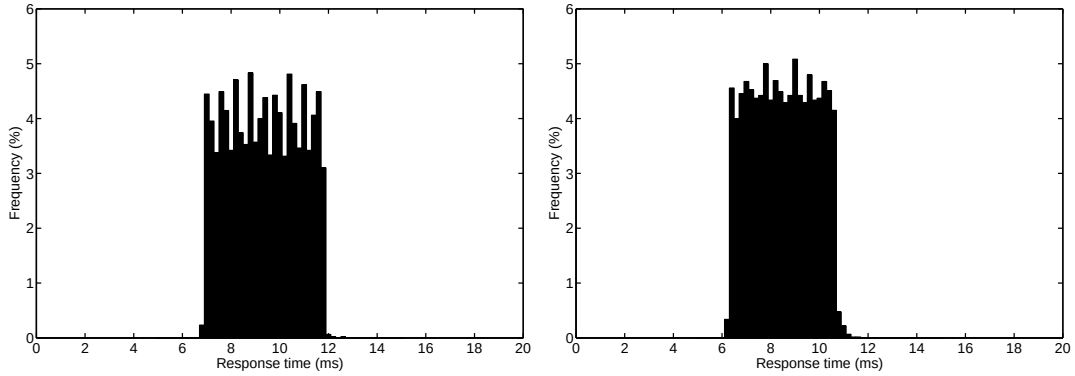


Figure 6.8: Histograms of response time in PLC-based architecture (left) and PC-based architecture (right) in configuration 3

and can be explained easily by the dispersions added to treatment times and by simulations uncertainties)

PC-based architectures yield distributions with shorter minimum and maximum response times and narrower ranges. Similar comments can be made for these architectures, when the amount of shared resources grows up, except that:

- the minimum values of the distributions in configurations 2 and 3 are higher than the minimum value of configuration 1
- the range increase is more important than in PLC-based architecture

hence, generally speaking, the response time of these architectures is better than that of PLC-based architectures but they are more sensitive to resource sharing increase.

Nevertheless, the influence of the increase of the amount of shared resources on response time is not so important that it could be planned. This can be explained, in an intuitive fashion, by an assumption on the relative importance of the delay caused by resource sharing mechanisms compared to the other delay causes, data processing and synchronisation of parallel processes. We recall here that response time can be decomposed in three kinds of delays:

- time for processing data within components,
- waiting time for synchronisation between parallel processes and
- waiting time for availability of resources.

Only the later two delays are variable and explain the distribution. However, if the overall delay due to resource sharing mechanism is small compared to the other ones, its variation will not impact strongly the distribution of response time.

This assumption is consolidated when comparing response time distributions of PLC- and PC-based architectures. PLC-based architectures are less sensitive to the increase of the number of shared resources. These architectures include more synchronisation between processes than PC-based ones, where each controller has only one processor that performs both user program execution and RIOM scanning, and therefore do not include synchronisation between two processors executing these two functions.

However, this assumption must be confirmed by analysing in details the simulation results and in particular by comparing the values of the delays caused by resource sharing and by synchronisation between processes. This analysis is carried out in the next subsection.

Table 6.3: Minimum and maximum response time for the studied architectures

Response time (in ms)		architecture with PLC-based controllers	architecture with PC-based controllers
Conf 1	min	6.87	5.91
	max	11.90	10.10
Conf 2	min	6.88	6.28
	max	12.45	11.26
Conf 3	min	6.87	6.28
	max	12.52	11.55

6.3.2 Influence of resource sharing and of synchronisation between processes

The post-treatment of simulation log files yield also the contribution of the different time consumption mechanisms to the overall response time and enables us to draw the pie charts pictured in figures 6.9, 6.10 and 6.11.
Figure 6.9 presents four pie charts for configuration 1, the two at the left concern PLC-based architecture while the two of the right concern PC-based architectures. For each architecture, the pie chart at the top is for the minimum response time, while the pie chart at the bottom is for the maximum response time. Figures 6.10 and 6.11 are similar for configurations 2 and 3.

In these pie charts :

- *Treat* represents the sum of processing time in the RIOMs and the controllers that are involved in the data exchange whose response time is to evaluate, i.e.:
  - Electronic filtering of signals in RIOM that receive the input event
  - Frame treatment in RIOM in the RIOM that receive the input event (construction of a frame that will be sent to controller) and in the RIOM that sends the output event to the plant (extraction of an event from a frame)
  - Transmission delay of the frame (D_{Trans}) from RIOM to controller
  - Frame treatment in controller, in the Ethernet module for PLC-based controllers or by the CPU in PC-based controllers,
  - User program execution in controller
  - Transmission delay of the frame (D_{Trans}) from controller to RIOM
- *Synchr* represents the sum of all delays coming from synchronisation between asynchronous processes in the RIOMs and controller:

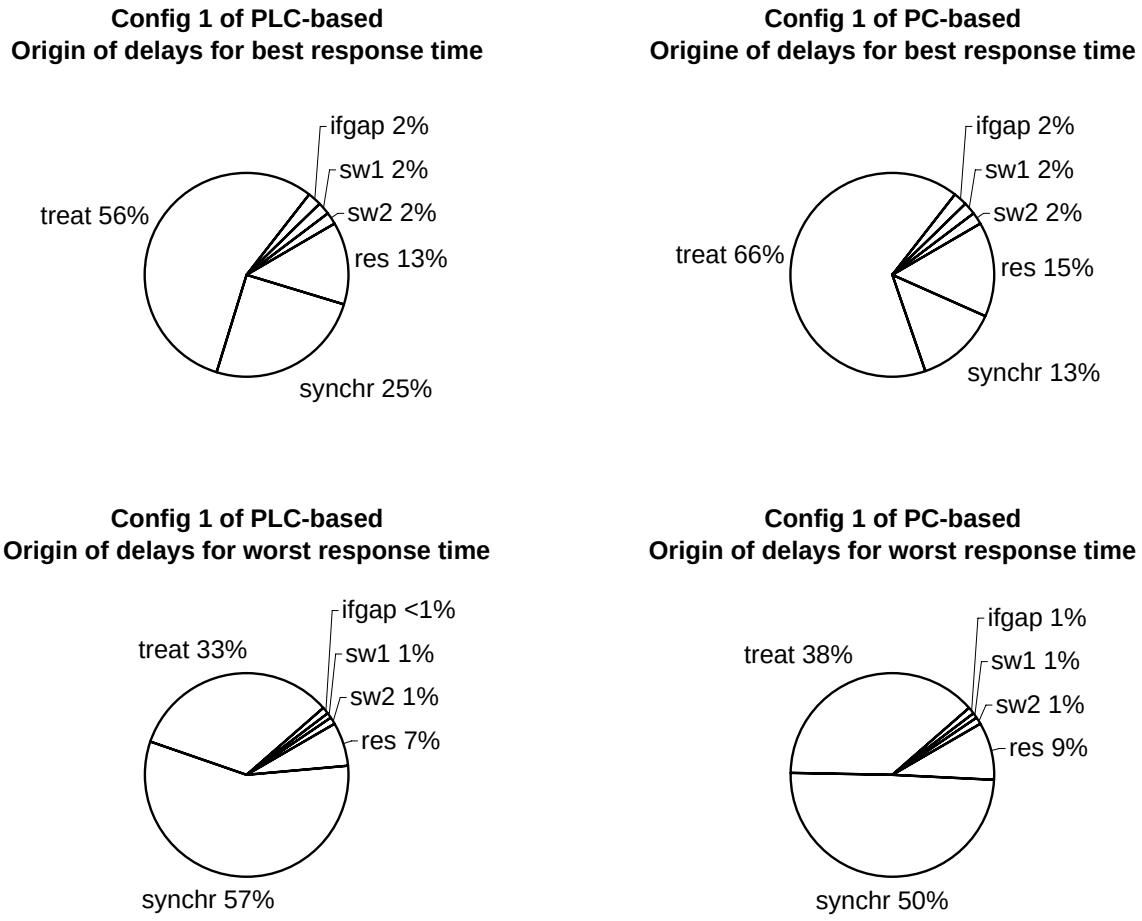


Figure 6.9: Pie charts of response time in PLC-based architecture and PC-based architecture in configuration 1

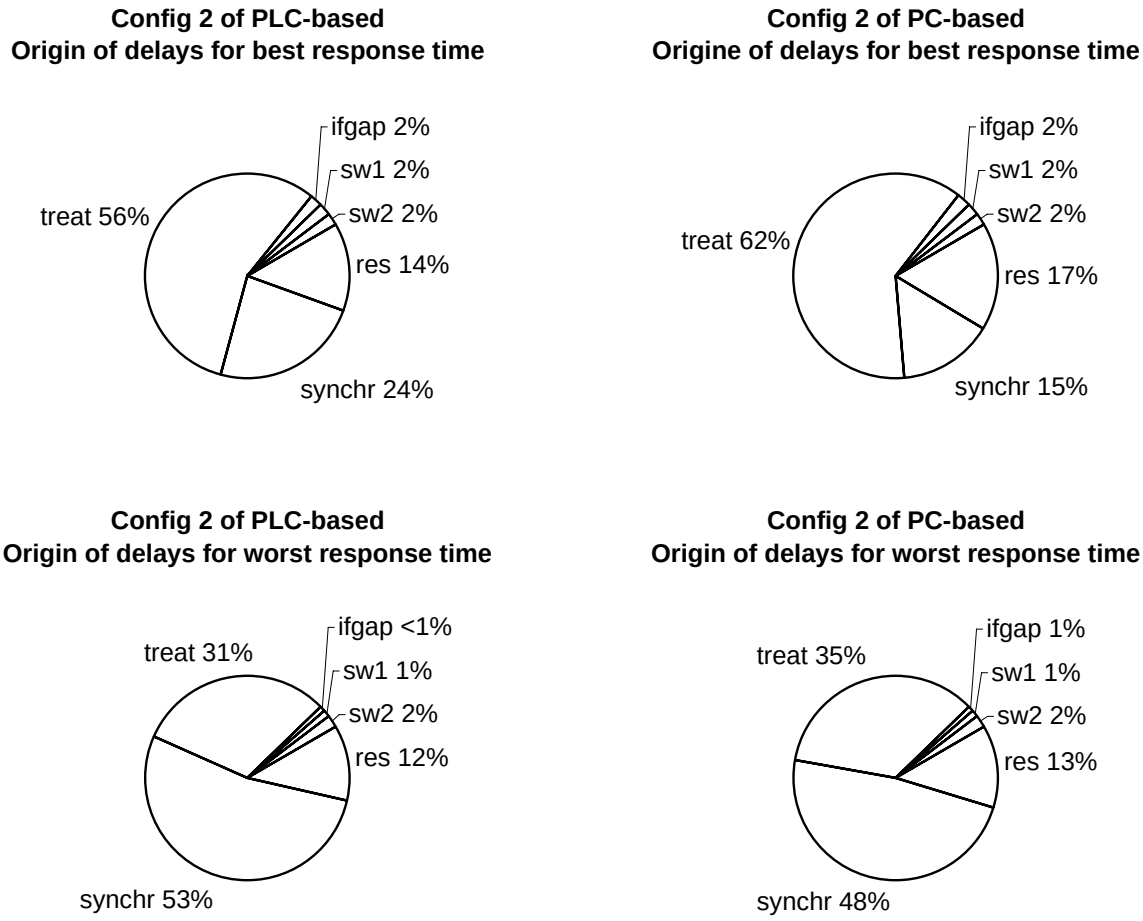


Figure 6.10: Pie charts of response time in PLC-based architecture and PC-based architecture in configuration 2

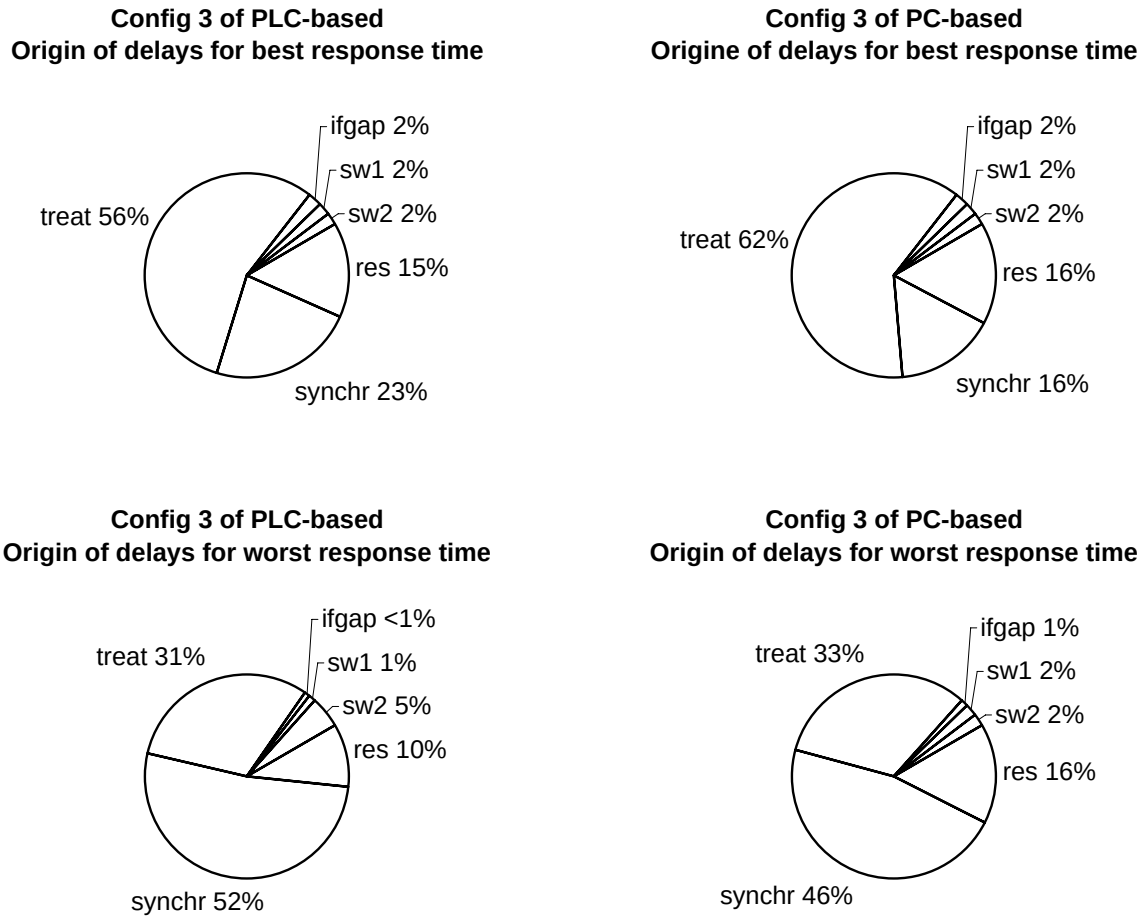


Figure 6.11: Pie charts of response time in PLC-based architecture and PC-based architecture in configuration 3

  - Waiting time in RIOM from the end of the filtering to the arrival of the next request from controller
  - Waiting time in controller from the end of the considered frame treatment to the beginning of the next user program
  - Waiting time in controller from the end of the user program execution to the beginning of the frame treatment (remark: this frame will be then sent to the RIOM which generates the consequence event)

- *Res* represents the sum of delays due to waiting for resource availability in the RIOMs and the controllers:
  - Waiting time in RIOM from the date a request arrives to the date this request is actually treated
  - Waiting time in controller from the date a request arrives to the date this request is actually treated

- *Sw1* represents the overall delay due to switches from RIOM to controller for the frame containing the considered event. We recall that the delay in switches is the sum of a processing time and a waiting time for resource availability.

- *Sw2* represent the overall delay due to switches from controller to RIOM for the frame containing the considered event.

- *Ifgap* represents the sum of all delays due to the Ethernet inter-frame gap.

Apart from data processing delays which are constant and non zero, all other delays are variables and can be zero.

It shall be noted that, for switches, we merged in sw1 and sw2 two kinds of delays whereas the three kinds of delays are separated for the other components. Indeed, switches are typical components of Ethernet-based Automation Systems and we would wish to examine the relative contribution of these components to the overall response time so as to pinpoint their impact on automation architectures. A detailed analysis of delays introduced by switches is carried out in subsection 6.3.3. At the moment, we will only note that the sum of the three delays sw1, sw2 and ifgap is always very small, lower than 7% in all cases, and therefore that the two switches do not contribute very much to the overall response time.

6.3.2.1 Influence of resource sharing

For all cases, waiting for availability of resources (res) stands between ten and seventeen percent of the response time (minimum value 0.875 ms obtained in configuration 1, and minimum value 1.807 ms obtained in configuration 3 for PC-based

architectures). Hence, resource sharing mechanism influences significantly this time, even if *they do not cause the most important delay whatever the configuration is.*

It matters at this point of the analysis to remind the reader that, in all the configuration we studied, the duration of the user program (2 ms) is rather small; these configurations aim to control very reactive systems. For configuration with higher user program duration, the ratio of response time due to resource sharing is lessened; simulation with user program duration equal to 30 ms for instance shows that this ratio is only six percent.

The strength of pie charts is to provide an easy way for comparing different delays, but this strength is a weakness when focus is put on the evolution of one of this delay from a configuration to another one. In particular, our pie-chart do not show how the waiting time for resource availability changes from configuration 1 to configuration 2 or 3. This analysis is the subject of the next paragraph.

The figure 6.12 presents two histograms of the evaluated values for delay due to waiting for resource availability in PLC-based architecture in configuration 1, on the left, and in configuration 2, on the right. The distribution of this time is very narrow for configuration 1; the difference between the maximum and minimum value is equal to 50μs. For this configuration in which few resources are shared, the delay due to resource sharing could be considered in a first approximation as a constant value, equal for instance to the average value of this distribution. This is not the case in the configuration 2, where the distribution lies between 0.92 ms to 1.52 ms, a wider range than that of the previous distribution. Configurations with several shared resources give rise to very variable delays due to resource sharing. The distribution of these delays can obtained only by simulation.

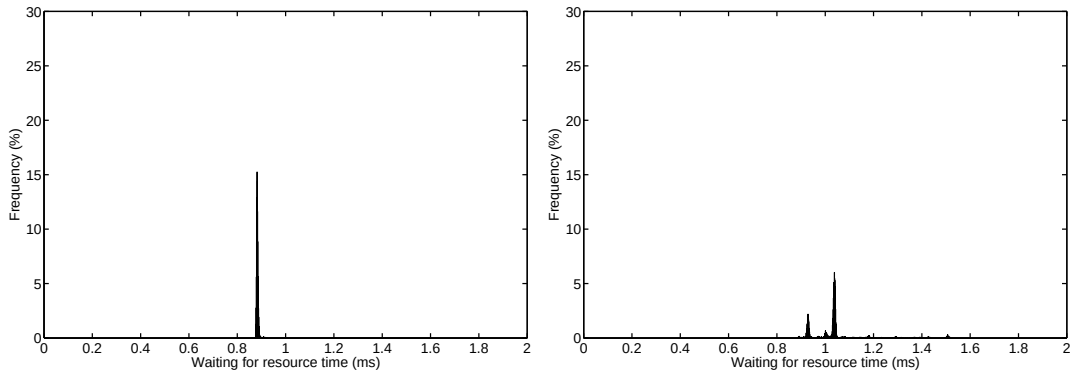


Figure 6.12: Histograms of delay due to waiting for resource availability in PLC-based architecture in configuration 1 (left) and in configuration 2 (right)

6.3.2.2 Influence of synchronisation between asynchronous processes

For both PLC-based and PC-based architectures, the major cause of gap between minimum and maximum response time is the delay due to waiting for synchronisation of asynchronous processes (synchr), whatever is the configuration. as mentioned above, this time includes:

- Waiting time in RIOM from the end of the filtering to the arrival of the next request from controller
- Waiting time in controller from the end of the considered frame treatment to the beginning of the next user program
- Waiting time in controller from the end of the user program execution to the beginning of the frame treatment

The first time is not really typical of Ethernet-based Automation Systems. It exists also in architectures using master/slave or producer/consumer cooperation model because it comes from synchronisation between the two following asynchronous processes: the plant and the data acquisition process (called IO scanning in Ethernet-based Automation Systems). This time is equally distributed between zero and one IO scanning cycle time in real system when the plant is not synchronised with the control process. This is also the case in our simulation (chapter 5 page 82).

For the discussion of the two other delays, we must consider separately PLC-based and PC-based architectures. Concerning PLC-based architectures, these two delays do not depend on the size of the architecture and on the number and type of shared resources but of the duration of the cycles of the controller, user program cycle time and IO scanning cycle time. Therefore, the bounds of the sum of these two delays is easy to compute and do not depend on the configuration. For instance, the waiting time in controller from the end of the considered frame treatment to the beginning of the next user program, with a user program cycle time of 2 ms, is between 0 and 2 ms; as well as, the waiting time in controller from the end of the user program execution to the beginning of the frame treatment, with a IO scanning cycle time of 5 ms, is between 0 and 5 ms.
Concerning PC-based architectures, there is only one cycle in each controller, including waiting time for RIOMs responses, user program execution and requests sending to RIOMs. Hence the waiting for synchronisation does not depend on cycle times but on waiting time for all RIOMs responses to arrive and is impacted by the number of shared resources. The maximum value of this delay grows up from configuration 1 to 3; PC-based architectures are, regarding this delay, more sensitive to the increase of the number of shared resources. The distribution of synchronisation delay for PC-based architectures can be obtained only by simulation.

6.3.2.3 Synthesis

Analysis of the relative importance of delays caused by waiting for availability and for synchronisation of asynchronous processes confirms the asumption we made at the end of the previous section, the sum of waiting times for resource availability represents less than one fifth of the overall response time. Synchronisation mechanism causes delays whose do not depend on the configuration for PLC-based architectures while for PC-based architectures, the maximum value of the delays caused by this mechanism grows up with the number of shared resources.

Even if their contribution is not significant, delays due to the use of a switched Ethernet fieldbus, switch latencies and inter-frame gap, are evaluated and discussed in the next section because they are new components in automation systems.

6.3.3 Delays caused by switches

It is often claimed that the main contribution to response time is the switches latency. We saw in the previous subsection that this is not the case for the industrial switches that we used (Telemecanique ConneXium reference 499 NES 181 00). However an analysis of the variation of the delays provoked by these components deserves to be performed. This is the goal of this part.

Figures 6.13 and 6.14 picture, for each event, the delay caused by switches for respectively PLC-based and PC-based architectures in configuration 2. "Switch delay 1" represents the sum of the delays caused by the two switches when the frame goes from the input RIOM to the controller (response to a request) and "switch delay 2" represents the sum of the delays caused by the two switches when the frame goes from the controller to the output RIOM (the frame embeds a request). Each switch introduces a delay which is the sum of its forwarding time and of its transmission time.

Only the plots obtained for configuration 2 are represented because those obtained for the other configurations have the same form and the same minimum and maximum values. This shows clearly that, in our study, the delays caused by switches are not affected by the variation of the number of shared resources, due to the short frame forwarding time of switches.

Comparison of the top and bottom plots of each one of these figures shows clearly that the maximum of delay 2 is higher than that of delay 1. This can be easily explained by considering the mechanisms of the communications between the controller and the RIOMs. Indeed, delay 2 corresponds to the sending of the request from the controller; as, in the client/server cooperation model, several controllers can send

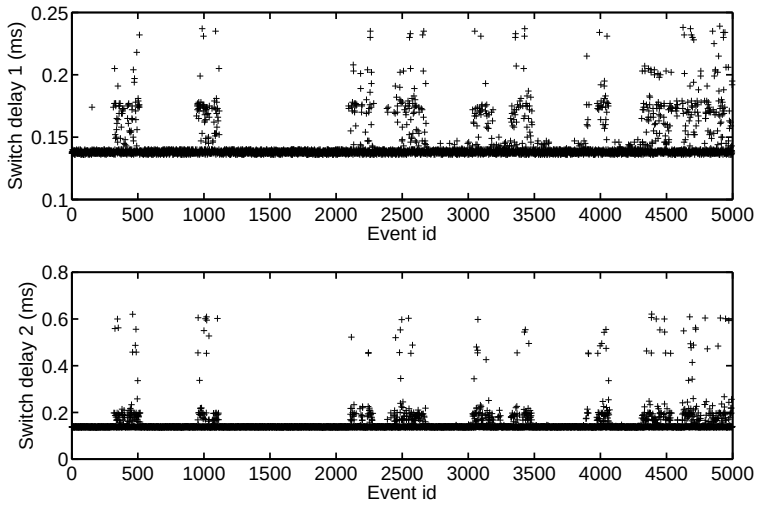


Figure 6.13: Switch delay in PLC-based architecture in configuration 2

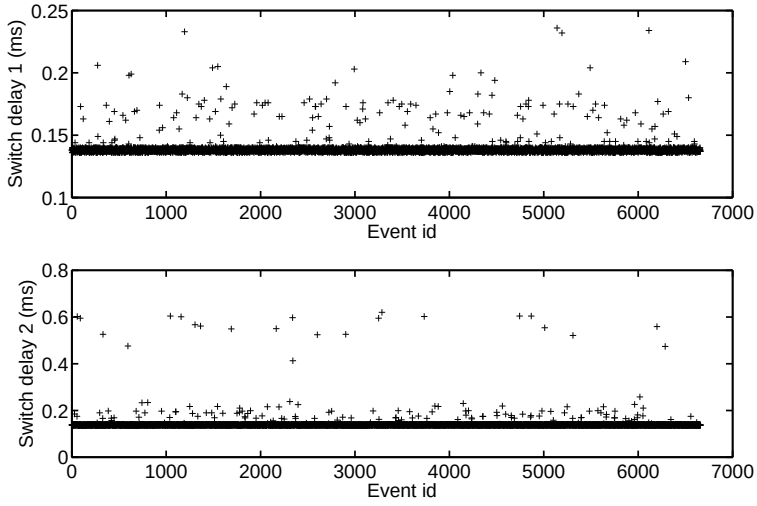


Figure 6.14: Switch delay in PC-based architecture in configuration 2

simultaneously requests to the RIOMs they scan, the work load of the first switch, connected to the controllers, can be very high during this request sending phase. On the contrary, delay 1 corresponds to the reception of the response sent by the input RIOM; as the simultaneous requests of controllers have been regulated (sent sequentially to the RIOMs) by the first switch in the previous phase, the likelihood of simultaneous responses is far smaller. Then the difference between the maximal values of the two plots comes only from the potential number of processes which attempt to access the switch CPU resource in both cases.

Comparison of figures 6.13 and 6.14 leads to note that the minimum and maximum values of the two delays are the same in both PLC-based and PC-based architectures, but that the distributions of values higher than 0.140 ms along time are not the same for these two architectures. The plots of figure 6.13 contain indeed vertical 'blobs' which point out that the highest values of delays are obtained cyclically for PLC-based architectures, that is not the case for PC-based ones. This difference can be explained by considering the IO scanning methods of the two architectures. Indeed, modular PLCs scan their IOs periodically whereas the time between two consecutive IO scannings of a PC controller is not constant (these controllers scan their IOs 'as fast as possible', i.e. without waiting time after the user program execution). Therefore the IO scannings of PLC controllers can be synchronised from time to time, depending on the values of their periods and of the deviations on these values. Synchronisation of IO scannings of PC controllers is far less likely and, anyhow, does not give rise to cyclic phenomenon.

The histogram of delay 1 for PLC-based architectures in configuration 2 is given figure 6.15; the histograms of the other delays are similar. We can note that the upper bound of this histogram is 0.240 ms and that the major part of the values is concentrated around 0.138 ms. Focusing only on this part, it matters to say that this distribution comes only from the choice we made for the simulation strategy and in particular for the dispersion on processing times values. In this simulation indeed, we chose $\pm 5\%$ dispersion. This dispersion results in a forwarding time equal to $0.01 \pm 0.00005ms$ and a transmission time equal to $0.06 \pm 0.0003ms$. However, the time unit of the model is $10^{-3}ms$, so the software rounded both previous dispersions to $[-0.001, 0]ms$. Given this computational dispersion and the average values of the two times, the lower and upper bounds of the crossing time of two switches are 0.136 ms and 0.140 ms, which are exactly the values that yields simulation. Hence, the width of the main peak of this histogram is not meaningful; the overall delay "delay 1" is most of time constant and equal to the processing time of the switches, even if from time to time it grows up because of resource sharing mechanisms.

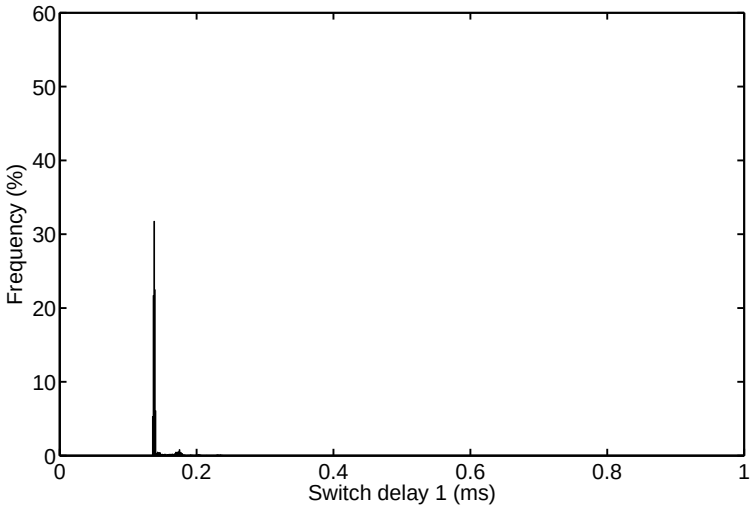


Figure 6.15: Histogram of switch delay 1 in PLC-based architecture in configuration 2

Conclusion

Comparison of the three cooperation models on the basis of their Network Cycle Time shows clearly that client/server is a suitable solution for switched networked automation systems. The Network Cycle Time is always shorter (from five to ten times) than that of architectures where communications are controlled thanks to master/slave or producer/consumer models.

Simulation of the particular models of Ethernet-based Automation Systems, that rely on client/server cooperation model, yields the distributions of their response time. The analysis of these distributions pinpoints that:

- when the number of shared resources grows up, the distribution is widened and its maximum value is increased;
- delays caused by switches are only a small part of the overall response time (less than 7%);
- waiting times for resource availability are not the only cause of response time variability; waiting times for synchronisation of asynchronous processes impact also strongly the response time;
- PC-based architecture give rise to response time distributions whose mean value is smaller than that of the corresponding PLC-based architecture; however PC-based architectures are more sensitive to the increase of the number of shared resources.

Conclusion

Industrial Ethernet, which brings interesting prospects for both interoperability and flexibility, is often presented as the future standard for fieldbuses. However this label is used for networks which own very different features. This research focused on networked automation systems based on switched Ethernet and on the client/server cooperation model. More precisely, the objectives of this PhD work were:

- To propose a method for evaluation of time performances of these systems;
- Once this method developed, to compare the performances of the client/server model to those of the master/slave and producer/consumer models;
- If this comparison showed that client/server is a suitable model for automation systems, to analyze in details the performances of systems based on this cooperation model.

Two relevant time performances for networked automation systems were first defined:

- the network cycle time that enables us to compare the cooperation models;
- the response time that permits to decide whether the designed automation system complies with the application requirements.

Then, a bibliographic survey of scientific works that addressed time performances evaluation of networked automation systems was carried out. Two kinds of evaluation methods have been identified: 'a priori' methods which assess the time performances prior to system design and implementation, by using a model of the system, and 'a posteriori' (or experimental) methods which obtain the time performances from measures on the real system. The first category includes itself three approaches: analytic calculus, formal verification and simulation. The first approach is based on analytic formulae which contain invariant features of network components; it has been widely used for computing upper bounds of time performances, for instance in the frame of the network calculus theory. Formal verification approaches based on model-checking techniques are very promising but unfortunately lead very easily to state-space explosion. At last, simulation approaches can be split themselves in two sub-categories: use of a tool for network analysis, like OMNET, and simulation

of a tailored model of the networked automation system that is designed in a formalism issued from Discrete Event System theory, such as state automata or Petri nets. Given the following constraints which come from our objectives and from the characteristics of the considered systems:

- a priori evaluation of the time performances,
- evaluation of the distributions of time performances, and not only of upper bounds,
- ability to analyse non trivial systems,
- simulation of all components of the system, and not only of network components,
- possibility of detailed analysis of all causes of delay,
- we selected an approach based on simulation of a Petri net model of the system.

The class of Petri nets we selected is Hierarchical Coloured Timed Petri Nets because:

- Petri nets are quite appropriate for modelling synchronisation of asynchronous processes and resource sharing, two time consumption mechanisms encountered in switched Ethernet-based automation systems;
- Use of a Timed model is mandatory, given our objectives;
- Coloured Petri Nets permit us to design a generic model which can then give rise to particular models by instantiation;
- Hierarchy is helpful to develop a modular model whose structure is issued from that of the automation system.

Then, a generic model of switched Ethernet-based automation systems has been developed. This model includes connected sub-models that describe in details the behaviour of generic hardware and software components, such as controller (either PLC-based or PC-based), Remote IO Module, Ethernet switch, client, ... An instantiation process has also been proposed. During this process, the generic values of the initial marking and of parameters associated to transitions (guards and durations) are replaced by the features of the particular system which must be analysed.

Simulation of a particular model can yield, after post-treatment, the network cycle time or the response time of the Ethernet-based automation system that is modelled. This enabled us, first, to compare, on the basis of the network cycle times of several case studies including a variable number of shared resources, the client/server cooperation model to the common master/slave and producer/consumer models.

The network cycle times of architectures based on these latter two models have been obtained by analytic calculus.

The comparison of these different network cycle times showed clearly that the client/server model permits faster cycles than the two other models, even for architectures with a significant number of shared resources. This can be explained by the possibilities of parallel processes authorised by client/server. The overall conclusion of this comparative study of cooperation models is that client/server is quite suitable for networked automation systems and can be even more efficient than the master/slave and producer/consumer models.

Simulation was also used to obtain the distributions of the response times of several switched Ethernet-based automation systems that included a variable number of shared resources and in which controllers were either PLCs or PCs. The analysis of these distributions pinpointed that:

- when the number of shared resources grows up, the distribution of response time is widened and its maximum value is increased;
- the main causes of the response time are the waiting times for synchronisation of asynchronous processes and for resources availability, and not the delays caused by the switches we use;
- PC-based architectures give rise to response time distributions whose mean values are lower than those of the corresponding PLC-based architectures; however PC-based architectures are more sensitive (the response time distribution is more deeply modified) to the increase of the number of shared resources.

To sum up, the results of this research is a complete frame for modelling, simulation and analysis of switched Ethernet-based automation systems that is aimed at helping the designers of such systems.

From the results that we obtained, several development and research prospects can be proposed. The first two ones of the list below may provide results in a near future, whereas the results of the other ones are rather middle- or long-term expectations.

- First, automatic instantiation of the generic model must be carried out. Currently each particular model is obtained indeed from the generic model 'by hand', that is time-consuming and source of errors. As the parameters that must be instantiated are written in XML format, it is possible to develop a code to instantiate these parameters from a textual description of the architecture.
- Second, the results of this research are to be transferred in an industrial software tool for automation systems design. The beginning of this work is planned for the next year in the frame of a cooperative research project with a French company of the domain.

- Third, the generic model of Ethernet-based automation system should be extended so as to be able to model more complex systems, including for instance routers, switches with QoS, and in which controllers could act as clients or servers. This will enable us to model communication between controllers as well as communication between different cells or lines connected by routers.

- A more detailed model of plant could also be helpful to investigate more finely the behaviour of the closed loop system (plant plus automation system). In the simulations we performed for response time evaluation, input events were indeed not correlated to the controllers cycles. This is not always the case in real systems where an input event, e.g. information delivered by a sensor, is often the consequence of an output event, e.g. an order sent to an actuator, which is itself the consequence of a request from a controller.

- At last, a promising while challenging prospect consists in the use of the results of this study for constructing abstracts models which can be analysed by model-checking tools. We mentioned indeed that model-checking is a promising approach for obtaining bounds of time performances, but that it can be applied only to very abstract models. The analysis of simulation results showed that some time consumption mechanisms are more important than other ones. Then it could be possible to derive from the generic model we built more abstract models, containing only the significant elements regarding time consumption and which could be analysed by model-checkers.

Bibliography

Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

Belgacem Ben Hédia, Fabrice Jumel, and Jean-Philippe. Babau. Formal evaluation of quality of service for data acquisition systems. In *FDL'05*, Lausanne, 2005.

Melha Bitam and Hassane Alla. Modeling of a communication network under TCP/IP protocols using hybrid Petri nets. In *IFAC Conference on Analysis and Design of Hybrid Systems, ADHS03*, Saint Malo, France, June 2003.

Melha Bitam and Hassane Alla. Performance evaluation of a TCP/IP transmission using hybrid Petri nets. In *Proc. of IEEE International Conference on Computer Systems and Applications, ICCSA*, July 2005.

Belinda Brahimi, Christopha Aubrun, and Eric Rondeau. Modelling and Simulation of Scheduling Policies Implemented in Ethernet Switch by Using Coloured Petri Nets. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September 2006.

Gianluca Cena, Ivan Cibrario Bertolotti, and Adriano Valenzano. Experimental analysis of latencies in Ethernet. In *Proc. WFCS 2006*, June 2006.

Antoine Colin and Isabelle Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems, Special issue on worst-case execution time analysis*, 18(2):249–274, April 2000.

Rene L. Cruz. A calculus for network delay. *IEEE Transactions on Information Theory*, 37(1), January 1991.

René David and Hassane Alla. Petri nets for modeling of dynamic systems : A survey. *Automatica*, 30(2):175–202, 1994.

Jorge de Figueiredo and Lars Kristensen. Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols. In *Second Workshop and Tutorial on Practical Use of Coloured Petri Nets and Design/CPN*, Aarhus, Denmark, October 1999.

Alessandro Depari, Paolo Ferrari, Alessandra Flammini, Daniele Marioli, and Andrea Taroni. Multi-probe measurements instrument for real-time Ethernet networks. In *Proc. WFCS 2006*, June 2006.

Max Felser. The fieldbus standard: History and structure. In *Technology Leadership Day 2002*, Luzern, Swiss, October 2002.

Paolo Ferrari, Alessandra Flammini, Daniele Marioli, and Andrea Taroni. Experimental evaluation of PROFINET performance. In *Proc. WFCS 2004*, September 2004.

Paolo Ferrari, Alessandra Flammini, and Stefano Vitturi. Performance of PROFINET networks. *Computer standards & interfaces*, 28(4):369–494, 2006.

Jean-Philippe Georges, Eric Rondeau, and Thierry Divoux. How to be sure that Ethernet networks will satisfy the real-time requirements ? In *Proc. ISIE'2002, IEEE International Symposium on Industrial Electronics*, July 2002.

Jurgen Greifeneder and Georg Frey. Optimizing Quality of Control in Networked Automation Systems using Probabilistic Models. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September 2006.

IEEE 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE standards 802.11, IEEE computer society, 1999.

IEEE 802.15. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs). IEEE standards 802.15, IEEE computer society, June 2005.

IEEE 802.3. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. IEEE standards 802.3, IEEE computer society, March 2002.

Intelligent Control Engineering Laboratory. University of maryland, october 1998. URL `http://www.enme.umd.edu/ice_lab/ncs/ncs.html`.

ISO Standards. High-level Petri nets – part 1: Concepts, definitions and graphical notation. ISO/IEC 15909-1, December 2004.

Jurgen Jasperneite and Peter Neumann. Switched Ethernet for factory communication. In *Proc. ETFA 2001*, pages 205–212, October 2001.

Jurgen Jasperneite and Peter Neumann. How to guarantee realtime behavior using Ethernet. In *Proc. 11th IFAC Symposium on Information Control Problems in Manufacturing*, April 2004.

Jurgen Jasperneite, Peter Neumann, Michael Theis, and Kim Watson. Deterministic real-time communication with switched Ethernet. In *Proc. WFCS 2002*, August 2002.

Kurt Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol. 1: Basic Concepts*. Springer-Verlag, eatcs monographs on theoretical computer science edition, 1992.

Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1, Basic Concepts. Springer-Verlag, monographs in theoretical computer science edition, 1997.

Guy Juanole. Quality of Service of communication networks and distributed automation: models and performances. In *Proc. of 15th Triennial IFAC World Congress*, July 2002.

H. Kopetz. Time-triggered real-time computing. *Annual Reviews in Control*, 27: 3–13, 2003.

Stefan Kowalewski, Nanette Bauer, Jorg Preussig, Olaf Stursberg, and Heinz Treseler. An open tool architecture for the formal verification of logic controllers in processing systems. In *14th IFAC World Congress*, Beijing, China, July 1999a.

Stefan Kowalewski, Sebastian Engell, Jorg Preussig, and Olaf Stursberg. Verification of Logic Controllers for Continuous Plants Using Timed Condition/Event-System Models. *Automatica*, 35(3):505–518, 1999b.

Jan Krakora and Zdenek Hanzalek. Timed Automata Approach for CAN Verification. In *11th IFAC Symposium on Information Control Problems in Manufacturing, INCOM*, Salvador, Brasil, April 2004.

Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*. Springer Verlag LNCS 2050, 2001.

Kyung Chang Lee and Suk Lee. Performance evaluation of switched Ethernet for real-time industrial communications. *Computer standards & interfaces*, (24):411–423, 2002.

Gaelle Marsal, Bruno Denis, and Jean-Marc Faure. Evaluation des délais de réactivité des architectures de commande distribuées sur réseau Ethernet. In *Conférence Internationale Francophone d'Automatique, CIFA 2006*, Bordeaux, France, May 2006a.

Gaelle Marsal, Bruno Denis, Jean-Marc Faure, and Georg Frey. Evaluation of Response Time in Ethernet-based Automation Systems. In *6th IEEE International Workshop on Factory Communication Systems, WFCS'2006*, pages 95–98, Turin, Italy, June 2006b.

Gaelle Marsal, Bruno Denis, Jean-Marc Faure, and Georg Frey. Evaluation of Response Time in Ethernet-based Automation Systems. In *11th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA06*, Prague, Czech Republic, September 2006c.

Thomas Mertke and Georg Frey. Formal Verification of PLC-programs generated from Signal Interpreted Petri Net. In *IEEE Systems, Man, and Cybernetics Conference*, pages 2700–2705, Tucson, USA, October 2001.

Pascal Meunier. *Validation formelle de programmes Ladder Diagram pour Automates Programmables Industriels*. Thèse de doctorat, Ecole Normale Supérieure de Cachan, 2006.

Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML - Revised*. May 1997.

Daniele Miorandi and Stefano Vitturi. Performance analysis of Producer/Consumer protocols over IEEE802.11 wireless protocols. In *Proc. WFCS 2004*, September 2004.

Modbus-IDA. *Modbus messaging on TCP/IP implementation guide v1.0a*, 2004. URL `http://www.modbus.org/docs/Modbus\_Messaging\_Implementation\_Guide\_V1\_0a.pdf`.

Modbus-IDA. *Modbus Serial Line Implementation Guide V1.0*, November 2002. URL `http://www.modbus.org/docs/Modbus_over_serial_line_V1.pdf`.

J. T. Parrott, J. R. Moyne, and D. M. Tilbury. Experimental Determination of Network Quality of Service in Ethernet: UDP, OPC, and VPN. In *2006 American Control Conference, ACC'06*, Minneapolis, USA, June 2006.

Nuno Pereira, Eduardo Tovar, and Luis Miguel Pinho. Timeliness in COTS factory-floor distributed systems: what role for simulation? In *Proc. WFCS 2004*, September 2004.

Gaelle Poulard, Bruno Denis, and Jean-Marc Faure. Modélisation par réseau de Petri coloré des architectures de commande distribuées sur réseau de terrain Ethernet et TCP/IP. In *5ème Conférence francophone de MOdélisation et SIMulation, MOSIM 2004*, pages 405–412, Nantes, France, September 2004.

Gaelle Poulard-Marsal, Daniel Witsch, Bruno Denis, Jean-Marc Faure, and Georg Frey. Evaluation of Real-Time Capabilities of Ethernet-based Automation Systems using Formal Verification and Simulation. In *1ère Rencontres des Jeunes Chercheurs en Informatique Temps Réel 2005, RJCITR'05*, pages 27–30, Nancy, France, September 2005.

PROFIBUS Nutzerorganisation e.V. Profibus web site, July 2006. URL `http://www.profibus.com/pb`.

Peter Puschner and Christian Koza. Calculating the maximum execution time of real–time programs. *Real-Time Systems*, 1(2):159–176, September 1989.

Schneider Electric. Transparent Ready, January 2006. URL `http://www.transparentfactory.com/en/index.htm`.

Alexandre Seuret, Miriea Termens-Ballester, Armand Toguyeni, Samir El Khattabi, and Jean-Pierre Richard. Implementation of an Internet-Controlled System Under Variable Delays. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September 2006.

Ye Qong Song. Time constrained communication over switched Ethernet. In *4th IFAC International Conference on Fieldbus Systems and their Applications (FeT'2001)*, Nancy, 2001.

John Stankovic. Real-Time Computing. In *BYTE invited paper*, pages 155–160, August 1992.

Jean-Pierre Thomesse. Fieldbuses and interoperability. *Control Engineering Practice*, (7):81–94, 1999.

Ken Tindell. Holistic schedulability analysis for distributed hard real-time systems. Technical Report YCS-93-197, University of York, Dept. of Computer Scienc, Englande, 1993. Available online at ftp://ftp.cs.york.ac.uk/reports/YCS-93-197.ps.Z.

Eduardo Tovar and Francisco Vasques. Distributed computing for the factory-floor: a real-time approach using WorldFIP networks. *Computer In Industry*, 44:11–31, January 2001.

Eduardo Tovar and Francisco Vasques. Cycle Time Properties of the PROFIBUS Timed Token Protocol. *Computer Communications*, 22:1206–16, 1999.

Jeffrey D. Ullman. *Elements of ML Programming, 2nd Edition (ML97)*. 1998.

University of Aarhus. CPN Tools, March 2006a. URL `http://wiki.daimi.au.dk/cpntools/_home.wiki`.

University of Aarhus. Standard ML in CPN Tools, March 2006b. URL `http://wiki.daimi.au.dk:8000/cpntools/standard_ml.wiki`.

University of Aarhus. Design/CPN, January 2006c. URL `http://www.daimi.au.dk/designCPN/`.

University of Southern California. Internet protocol, DARPA internet program protocol specification. RFC 791, Information Sciences Institute and IETF, September 1981a.

University of Southern California. Transmission Control Protocol, DARPA Internet program protocol specification. RFC 793, Information Sciences Institute and IETF, September 1981b.

Andràs Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, Prague, Czech Republic, June 2001.

Nikolai Vatanski, Jean-Philippe Georges, Christophe Aubrun, Eric Rondeau, and Sirkka-Liisa Jamsa Jounela. Control compensation based on upper bound delay in networked control systems. In *17th International Symposium on Mathematical Theory of Networks and Systems (MTNS)*, Kyoto, Japan, July 2006.

Stefano Vitturi. Some features of two fieldbuses of the IEC 61158 standard. *Computer Standards & Interfaces*, 22:203–15, 2000.

Stefano Vitturi. On the use of Ethernet at low level of factory communication systems. *Computer Standards & Interfaces*, 23:267–77, 2001.

Daniel Witsch, Birgit Vogel-Heuser, Jean-Marc Faure, and Gaelle Poulard-Marsal. Performance analysis of industrial Ehternet networks by means of timed model-checking. In *12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006*, pages 101–106, Saint-Etienne, France, May 2006.

WorldFIP organisation. WorldFIP web site, June 2003. URL `http://www.worldfip.org/`.

Dmitry A. Zaitsev. Switched LAN simulation by colored Petri nets. *Mathematics and Computers in Simulation*, 65(3):245–249, 2004a.

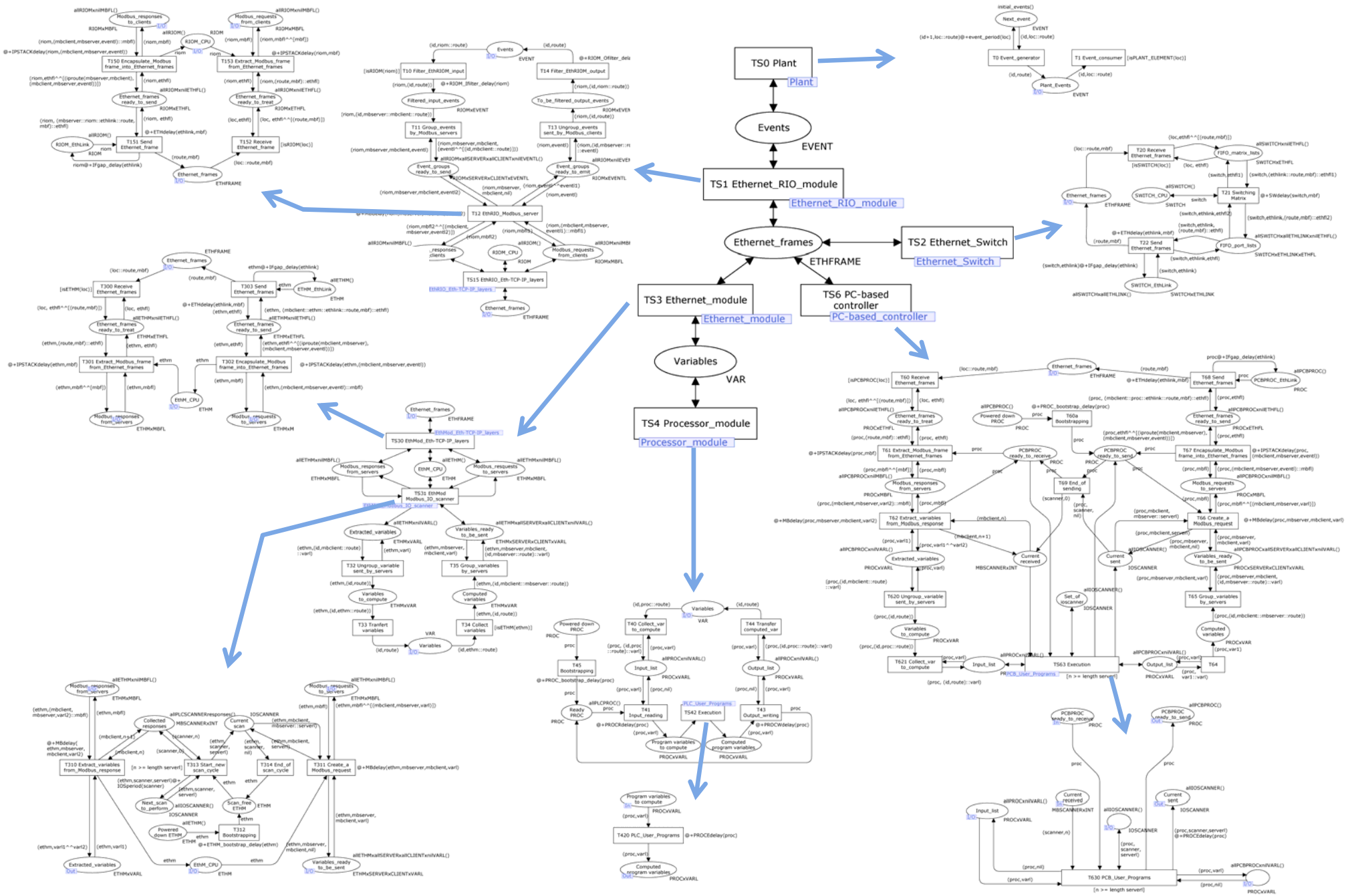
Dmitry A. Zaitsev. An Evaluation of Network Response Time using a Coloured Petri Net Model of Switched LAN. In *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 157–167, Aarhus, Denmark, October 2004b.

Hubert Zimmermann. OSI reference model – the iso model of architecture for Open System Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

Appendix A

Generic model of Ethernet-Based Automation System

Generic Model overview



Declarations of generic model

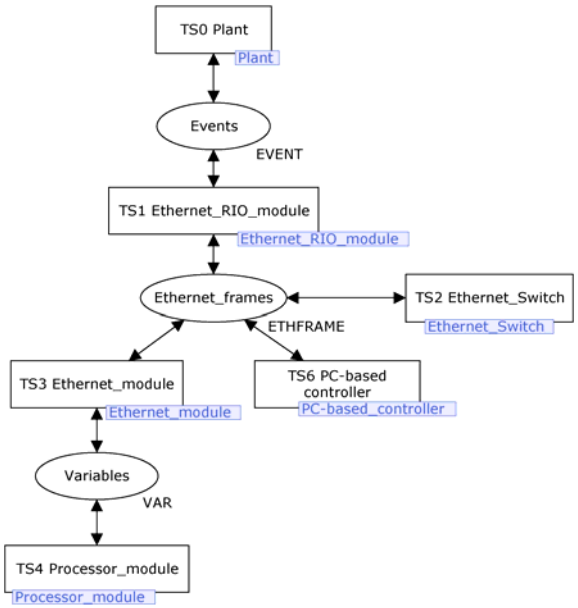
```
(* Standard declarations *)
    colset E = with e timed;
    colset INT = int;
    colset BOOL = bool;
    colset STRING = string;
    var n: INT;
    fun withdispersion(delay, dispersion_range_in_perthousand) =
         delay +
         discrete(~ delay * dispersion_range_in_perthousand div 1000,
                    delay * dispersion_range_in_perthousand div 1000);
    val general_dispersion_range_in_perthousand = 5;
    val multiset_to_be_instantiated = empty;
    val delay_to_be_instantiated = 0;
    val guard_to_be_instantiated = true;


(* Locations declarations *)

    (* Colors *)
        colset LOCATION = INT timed;
        colset DEST = LOCATION;
        colset SRC = LOCATION;
        colset SERVER = LOCATION timed;
        colset CLIENT = LOCATION timed;
        colset PROC = LOCATION timed;
        colset IOM = LOCATION timed;
        colset ETHM = LOCATION timed;
        colset RIOM = LOCATION timed;
        colset SWITCH = LOCATION timed;
        colset ETHLINK = LOCATION timed;
        colset PLANT_ELEMENT = LOCATION timed;

    (* Variables *)
        var loc: LOCATION;
        var dest: DEST;
        var src: SRC;
        var mbserver : SERVER;
        var mbclient, scanner : CLIENT;
        var proc: PROC;
        var ethm : ETHM;
        var riom: RIOM;
        var switch: SWITCH;
        var ethlink: ETHLINK;
        val IFgap_10Mbs = 10;
        val IFgap_100Mbs = 1;

    (* Functions *)
        fun allPLCPROC() = multiset_to_be_instantiated;
        fun allPCBPROC() = multiset_to_be_instantiated;
        fun allPROC() = allPLCPROC() ++ allPCBPROC();
        fun PROCEdelay(proc) = delay_to_be_instantiated;
        fun PROCRdelay(proc) = delay_to_be_instantiated;
        fun PROCWdelay(proc) = delay_to_be_instantiated;
        fun PROC_bootstrap_delay(proc) = delay_to_be_instantiated;
        fun allETHM() = multiset_to_be_instantiated;
        fun isETHM(loc) = guard_to_be_instantiated;
        fun ETHM_bootstrap_delay(ethm) = delay_to_be_instantiated;
        fun isPCBPROC(loc) = guard_to_be_instantiated;
        fun allRIOM() = multiset_to_be_instantiated;
        fun isRIOM(loc) = guard_to_be_instantiated;
        fun RIOM_Ifilter_delay(riom) = delay_to_be_instantiated;
        fun RIOM_Ofilter_delay(riom) =  delay_to_be_instantiated;
        fun allSWITCH() = multiset_to_be_instantiated;
        fun isSWITCH(loc) = guard_to_be_instantiated;
        fun IFgap_delay(ethlink) = delay_to_be_instantiated;
        fun isPLANT_ELEMENT(loc) = guard_to_be_instantiated;
        fun initial_events() = multiset_to_be_instantiated;
        fun event_period(loc) = delay_to_be_instantiatec;


(* Exchanges declarations *)

    (* Colors *)
        colset ID = INT timed;
        colset ROUTE = list LOCATION timed;
        colset EVENT = product ID * ROUTE timed;
        colset EVENTL = list EVENT timed;
        colset VAR = product ID * ROUTE timed;
        colset VARL = list VAR timed;
        colset MBFRAME = product SRC * DEST * EVENTL timed;
        colset MBFL = list MBFRAME timed;
        colset ETHFRAME = product ROUTE * MBFRAME timed;
        colset ETHFL = list ETHFRAME timed;

    (* Variables *)
        var id: ID;
        var route: ROUTE;
        var eventl, eventl1, eventl2: EVENTL;
        var var1: VAR;
        var varl, varl1, varl2 : VARL;
        var mbf: MBFRAME;
        var mbfl, mbfl1, mbfl2: MBFL;
        var ethframe: ETHFRAME;
        var ethfl, ethfl1, ethfl2 : ETHFL;
```

```
(* Complex declarations *)

    (* TCP/IP Ethernet protocol *)
        fun iproute(src,dest) = multiset_to_be_instantiated;
        fun IPSTACKdelay(loc,(ssrc,sdest,evl)) = delay_to_be_instantiated;
        fun ETHdelay(wire,(ssrc,sdest,evl)) = delay_to_be_instantiated;

    (* Modbus protocol *)
        colset SERVERL = list SERVER timed;
        colset IOSCANNER = product LOCATION * CLIENT * SERVERL timed;
        colset MBSCANNERxINT = product CLIENT * INT timed;
        var serverl : SERVERL;
        fun allSERVER() = multiset_to_be_instantiated;
        fun allCLIENT() = multiset_to_be_instantiated;
        fun allPLCSCANNERresponses() = multiset_to_be_instantiated;
        fun allPCBSCANNERresponses() = multiset_to_be_instantiated;
        fun allIOSCANNER() = multiset_to_be_instantiated;
        fun IOSperiod(scanner) = delay_to_be_instantiated;
        fun MBdelay(loc,ssrc,sdest,eventl) = delay_to_be_instantiated;

    (* Processor module *)
        colset PROCxETHFL = product PROC * ETHFL timed;
        colset PROCxVAR = product PROC * VAR timed;
        colset PROCxVARL = product PROC * VARL timed;
        colset PROCxMBFL = product PROC * MBFL timed;
        colset PROCxSERVERxCLIENTxVARL = product PROC * SERVER * CLIENT * VARL timed;
        fun allPROCxnilVARL() = PROCxVARL.mult(allPROC(), 1`nil);
        fun allPCBPROCxnilVARL() = PROCxVARL.mult(allPCBPROC(), 1`nil);
        fun allPCBPROCxnilETHFL() = PROCxETHFL.mult(allPCBPROC(), 1`nil)
        fun allPCBPROCxnilMBFL()= PROCxMBFL.mult(allPCBPROC(), 1`nil);
        fun allPCBPROCxallSERVERxallCLIENTxnilVARL() =
            PROCxSERVERxCLIENTxVARL.mult(allPCBPROC(), allSERVER(), allCLIENT(), 1`nil);

    (* Ethernet modules *)
        colset ETHMxETHFL = product ETHM * ETHFL timed;
        colset ETHMxVAR = product ETHM * VAR timed;
        colset ETHMxVARL = product ETHM * VARL timed;
        colset ETHMxMBFL = product ETHM * MBFL timed;
        colset ETHMxSERVERxCLIENTxVARL = product ETHM * SERVER * CLIENT * VARL timed;
        fun allETHMxnilETHFL() = ETHMxETHFL.mult(allETHM(), 1`nil);
        fun allETHMxnilVARL() = ETHMxVARL.mult(allETHM(), 1`nil);
        fun allETHMxnilMBFL()= ETHMxMBFL.mult(allETHM(), 1`nil);
        fun allETHMxallSERVERxallCLIENTxnilVARL() =
            ETHMxSERVERxCLIENTxVARL.mult(allETHM(), allSERVER(), allCLIENT(), 1`nil);

(* Ethernet RIO Modules *)
    colset RIOMxEVENT = product RIOM * EVENT timed;
    colset RIOMxETHFL = product RIOM * ETHFL timed;
    colset RIOMxEVENTL = product RIOM * EVENTL timed;
    colset RIOMxMBFL = product RIOM * MBFL timed;
    colset RIOMxSERVERxCLIENTxEVENTL = product RIOM * SERVER * CLIENT * EVENTL timed;
    fun allRIOMxnilMBFL() = RIOMxMBFL.mult(allRIOM(),1`nil)
    fun allRIOMxnilETHFL()= RIOMxETHFL.mult(allRIOM(), 1`nil);
    fun allRIOMxnilEVENTL()= RIOMxEVENTL.mult(allRIOM(), 1`nil);
    fun allRIOMxallSERVERxallCLIENTxnilEVENTL() =
    RIOMxSERVERxCLIENTxEVENTL.mult(allRIOM(), allSERVER(), allCLIENT(), 1`nil);

(* Switches *)
    colset SWITCHxETHFL = product SWITCH * ETHFL timed;
    colset SWITCHxETHLINK = product SWITCH * ETHLINK timed;
    colset SWITCHxETHLINKxETHFL = product SWITCH * ETHLINK * ETHFL timed;
    fun allSWITCHxnilETHFL() = SWITCHxETHFL.mult(allSWITCH(), 1`nil);
    fun allSWITCHxallETHLINK() = multiset_to_be_instantiated
    fun allSWITCHxallETHLINKxnilETHFL() = multiset_to_be_instantiated;
    fun SWdelay(switch,mbf) = delay_to_be_instantiated;
```

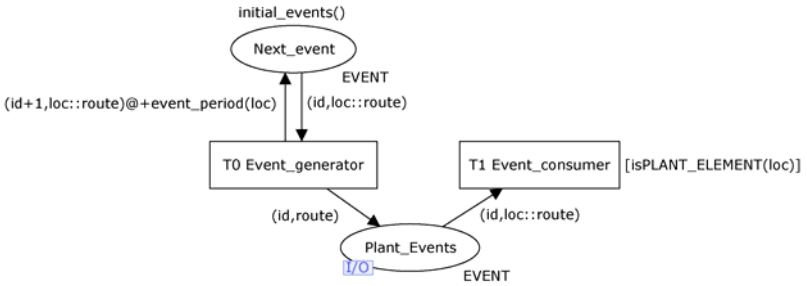
Model hierarchy

- Global
  - Ethernet_RIO_module
    - EthRIO_Eth-TCP-IP_layers
    - Plant
    - Ethernet_Switch
  - Ethernet_module
    - EthMod_Modbus_IO_scanner
    - EthMod_Eth-TCP-IP_layers
  - Processor_module
    - PLC_User_Programs
  - PC-based_controller
    - PCB_User_Programs

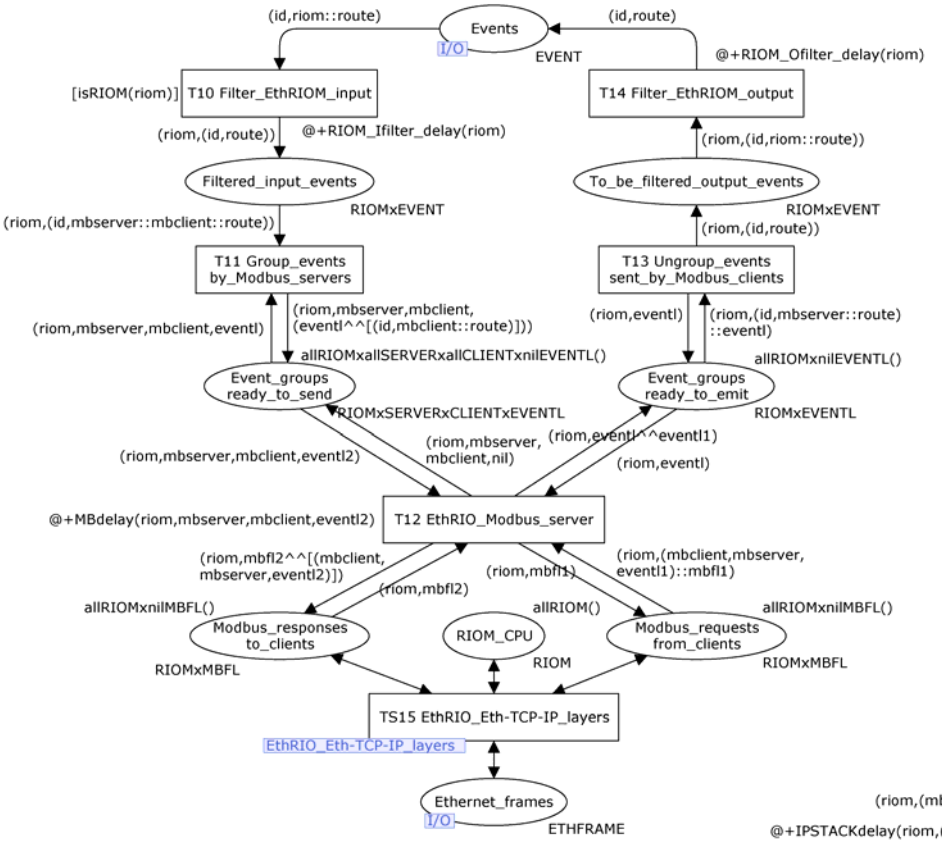
Global (Top level)



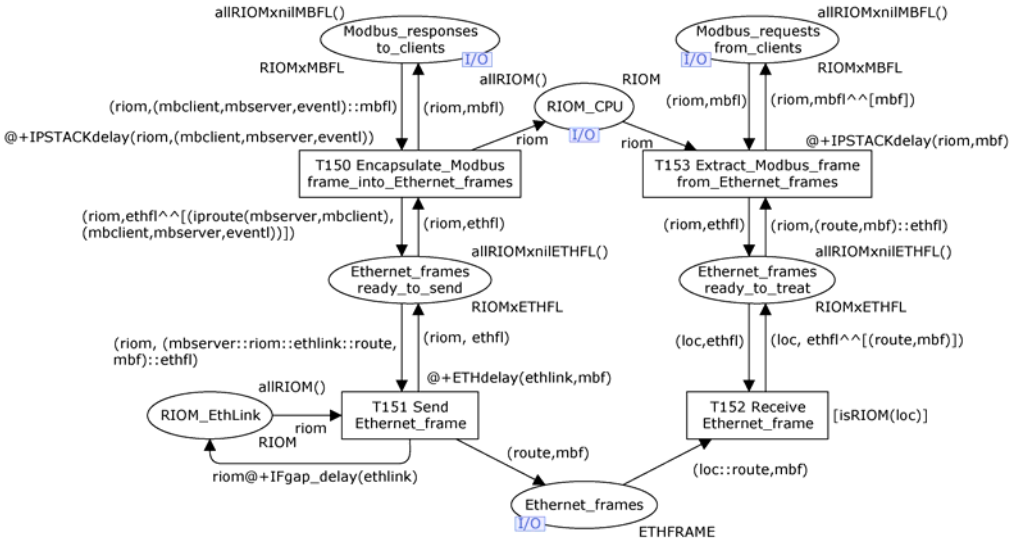
Plant



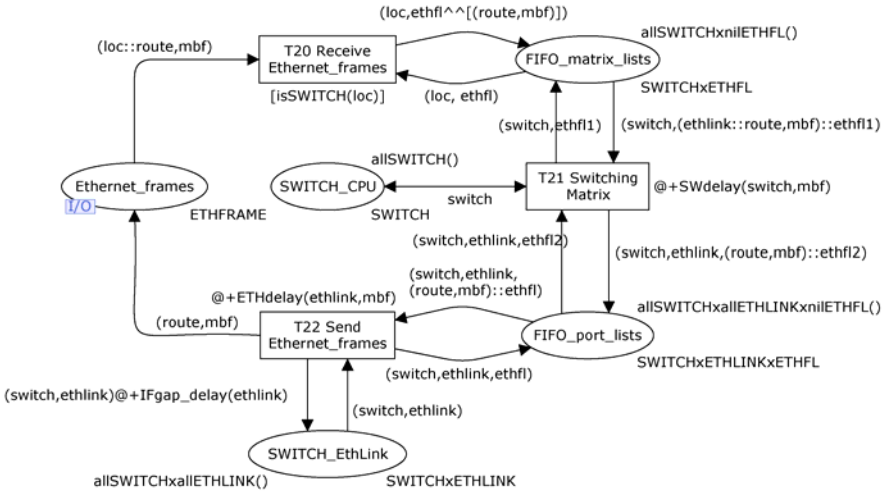
Ethernet_RIO_module



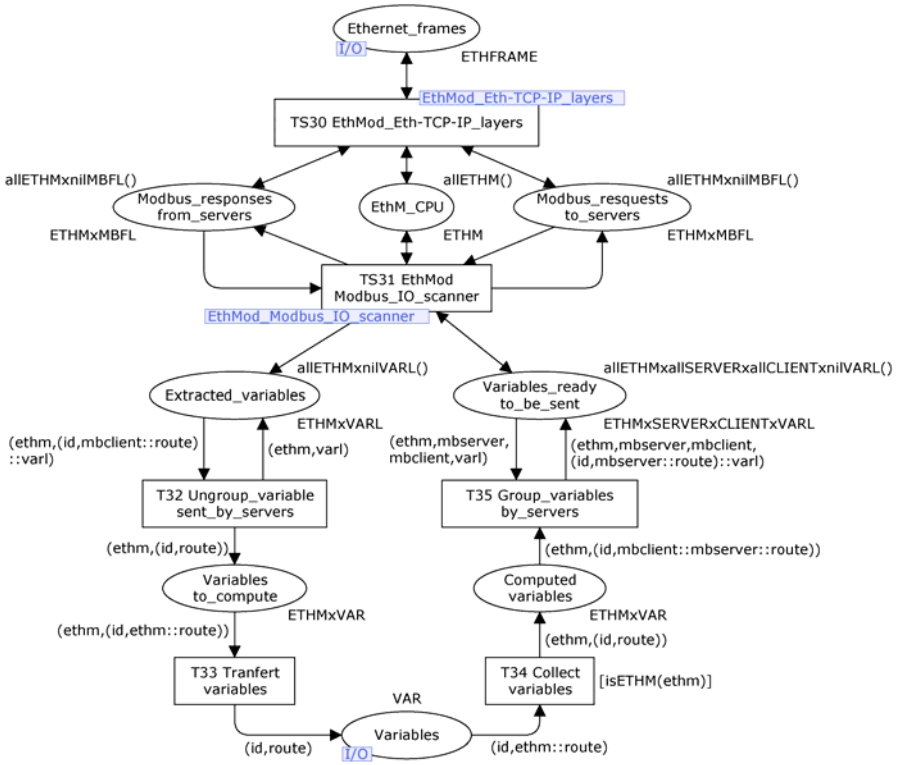
EthRIO_Eth-TCP-IP_layers



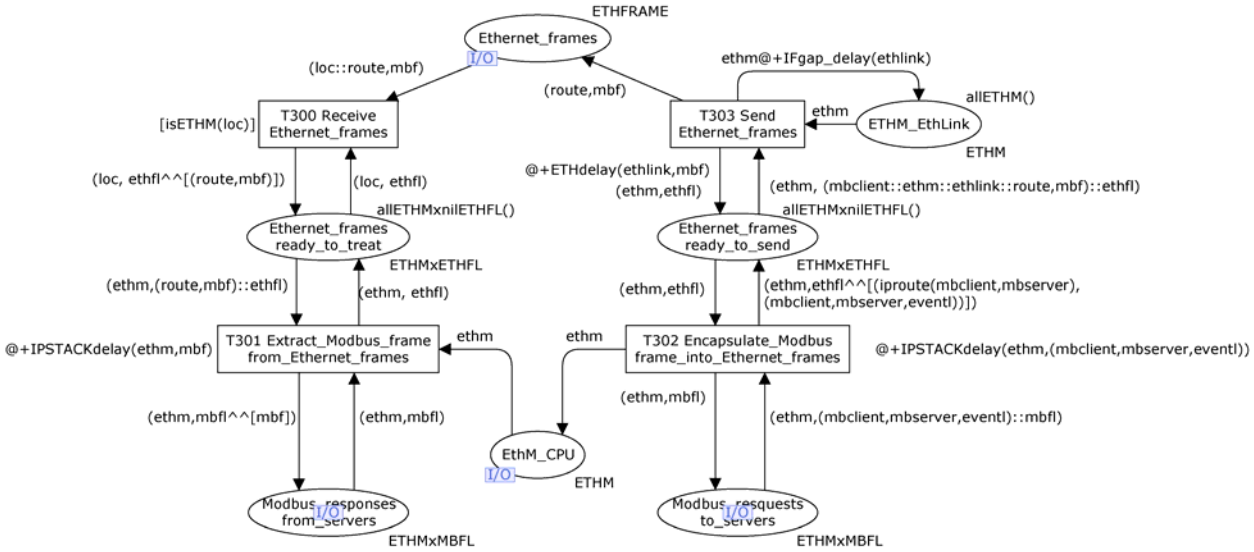
Ethernet_Switch



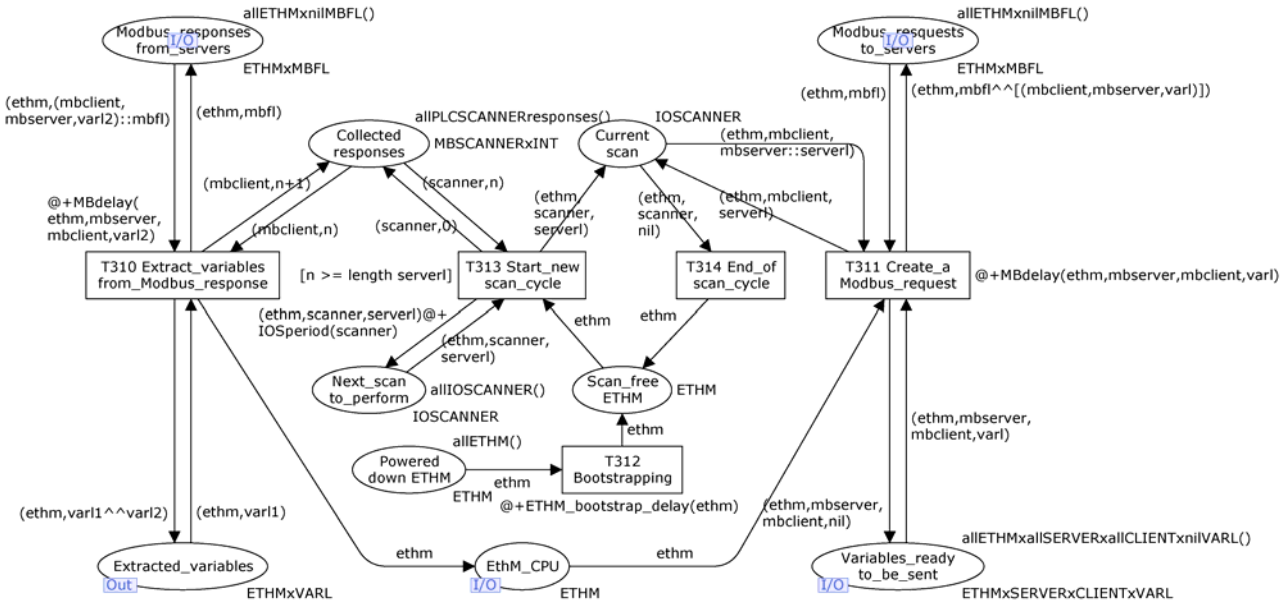
Ethernet_module



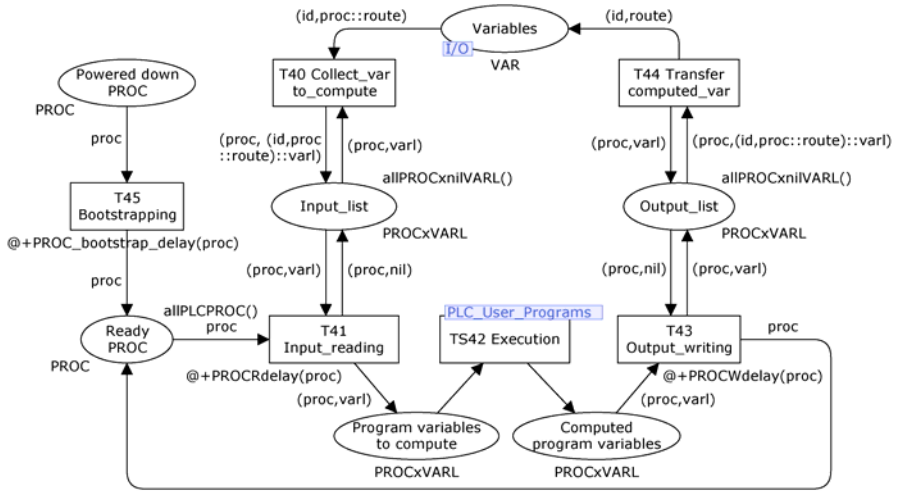
EthMod_Eth-TCP-IP layers



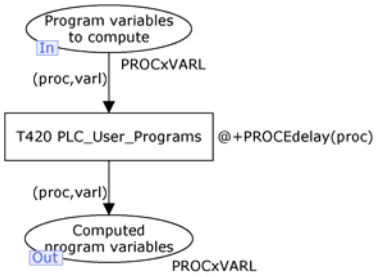
EthMod_Modbus_IO_scanner



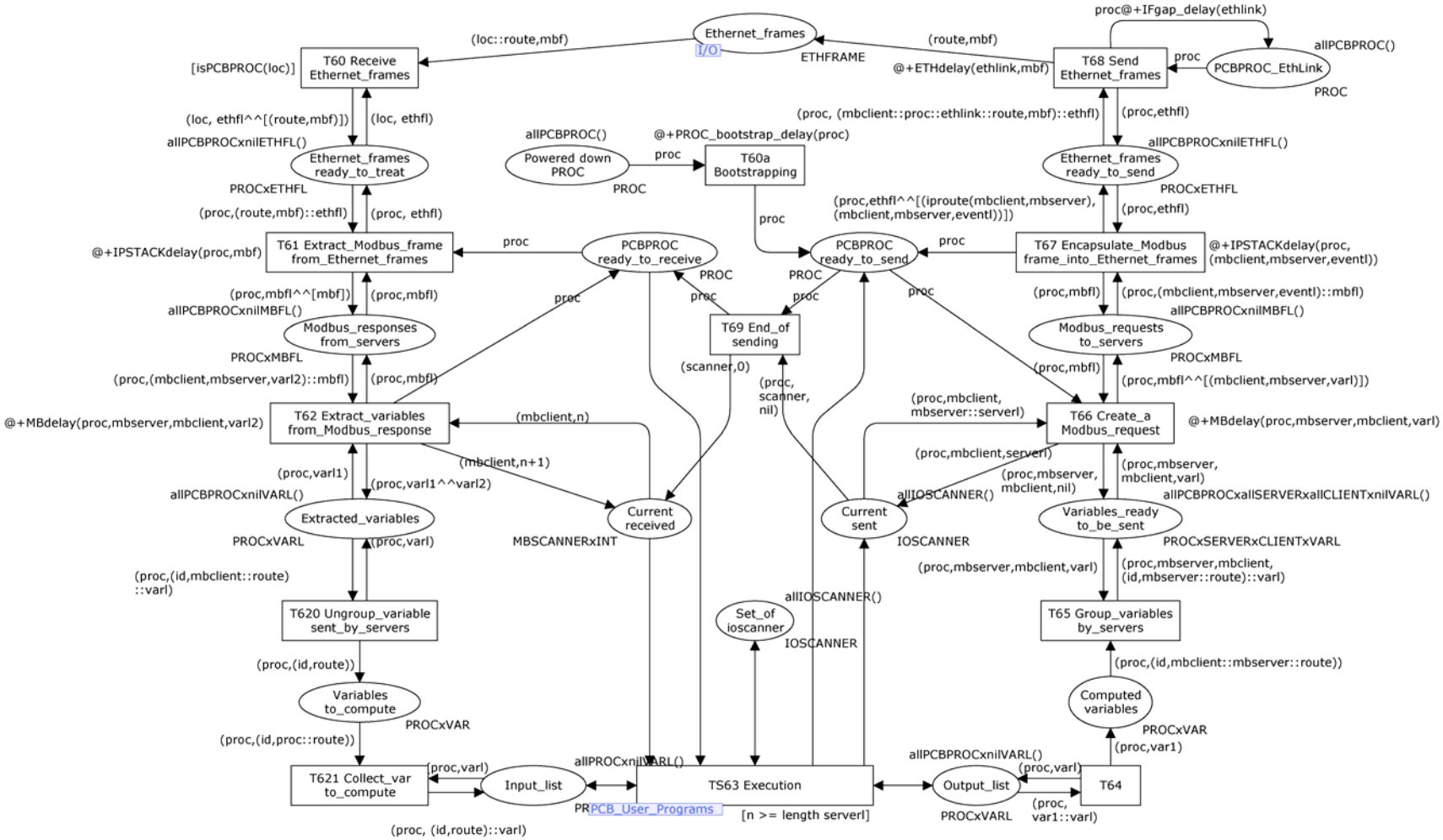
Processor_module



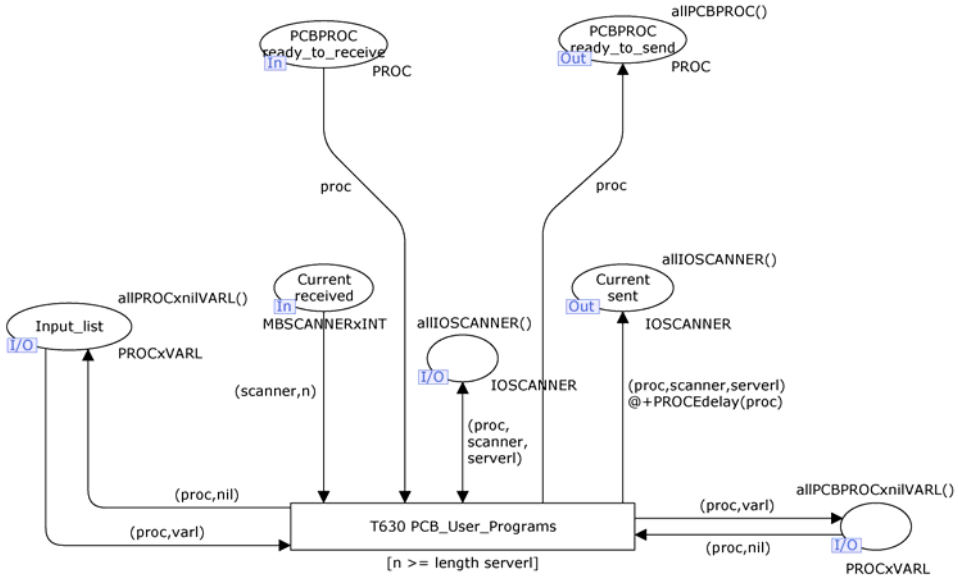
PLC_User_Programs



PC-based_controller



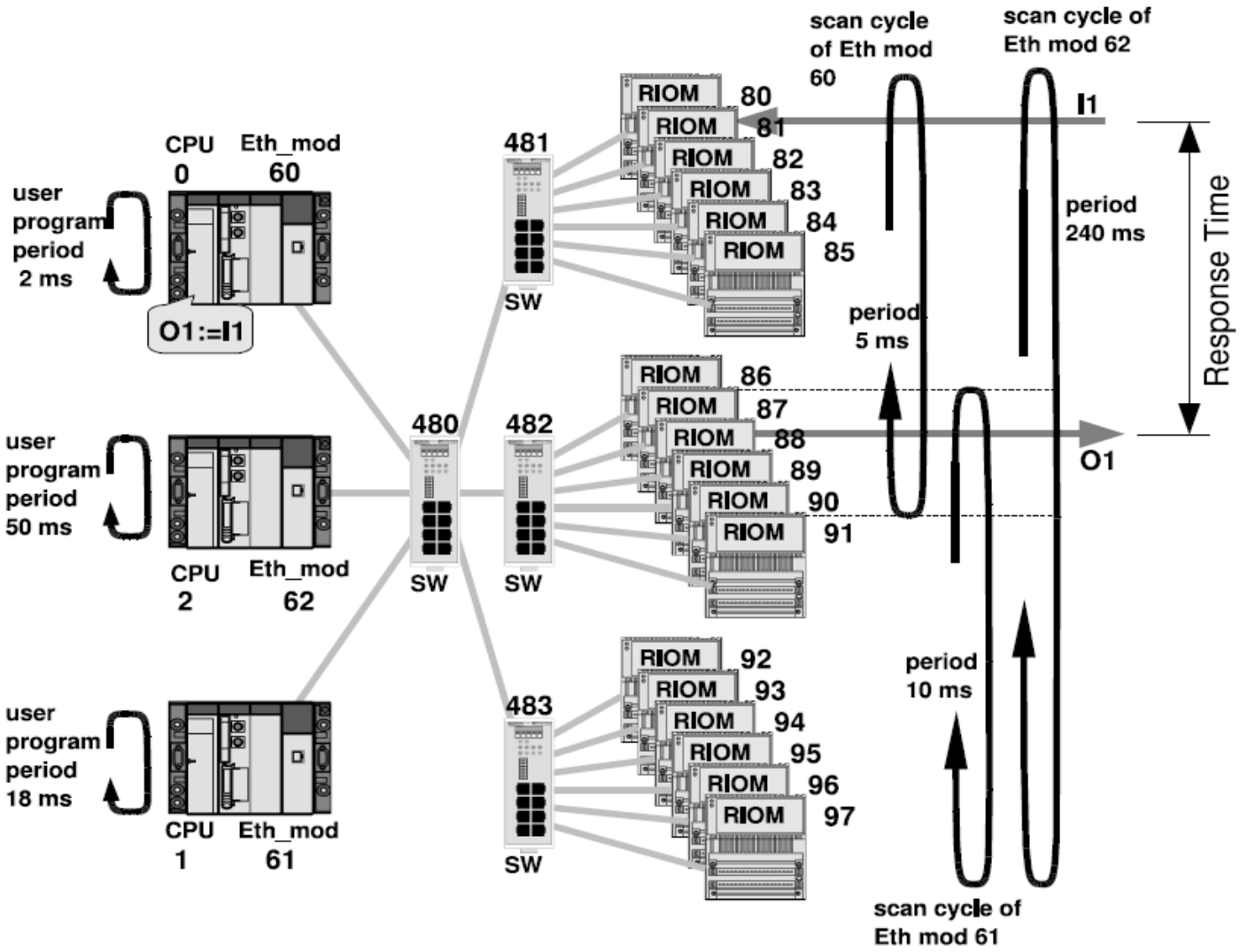
PLC_User_program



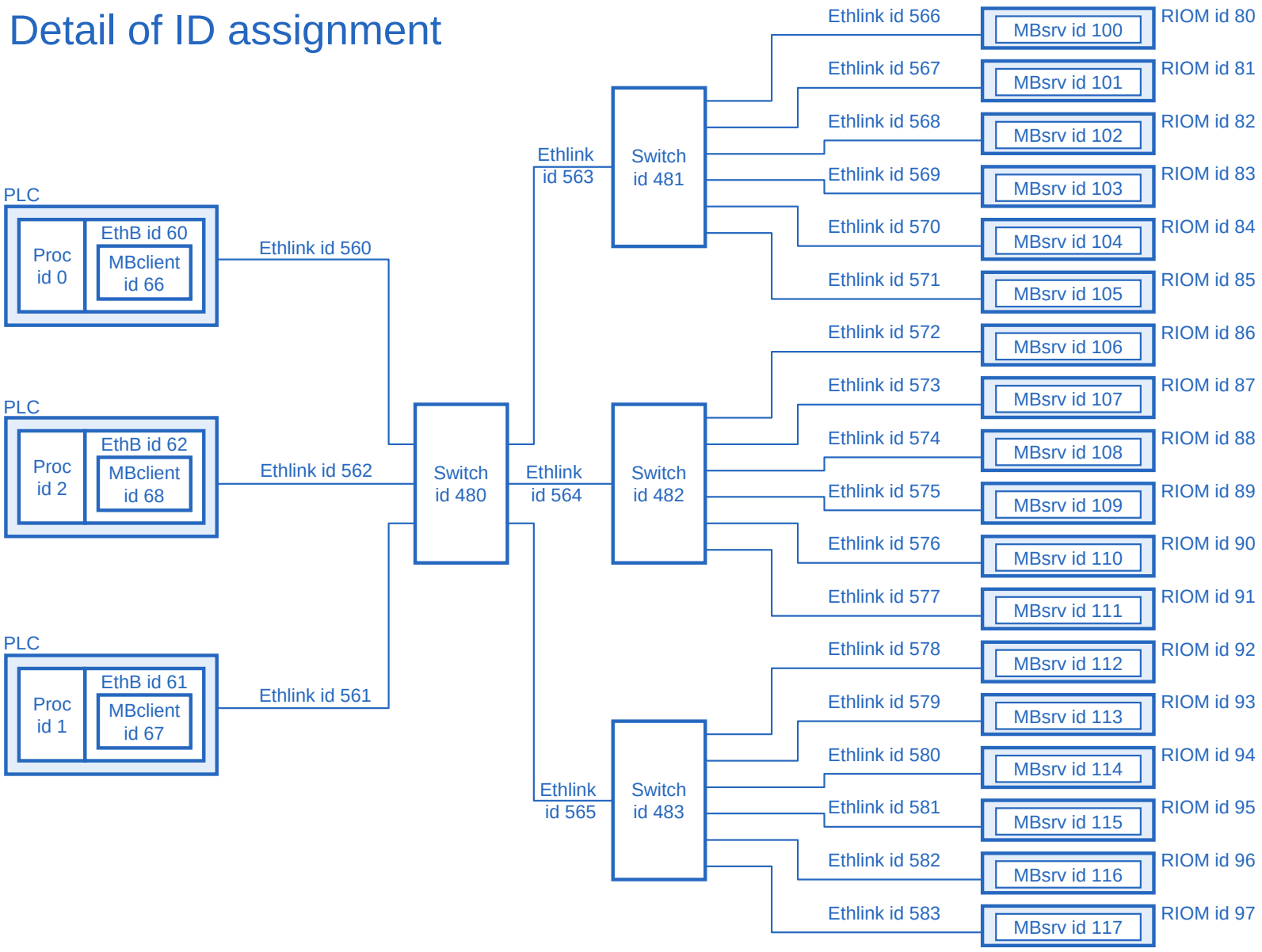
Appendix B

Example of a particular model of Ethernet-Based Automation System

Overview of PLC-based architecture (configuration 3)



Detail of ID assignment



Instantiated declarations of the studied PLC-based architecture (configuration 3)

```
(* Standard declarations *)
    colset E = with e timed;
    colset INT = int;
    colset BOOL = bool;
    colset STRING = string;
    var n: INT;
    fun withdispersion(delay, dispersion_range_in_perthousand) =
    delay +
    discrete(~ delay * dispersion_range_in_perthousand div 1000,
                delay * dispersion_range_in_perthousand div 1000)
    val general_dispersion_range_in_perthousand = 5
    val multiset_to_be_instantiated = empty;
    val delay_to_be_instantiated = 0;
    val guard_to_be_instantiated = true;


(* Locations declarations *)

    (* Colors *)
        colset LOCATION = INT timed;
        colset DEST = LOCATION;
        colset SRC = LOCATION;
        colset SERVER = LOCATION timed;
        colset CLIENT = LOCATION timed;
        colset PROC = LOCATION timed;
        colset IOM = LOCATION timed;
        colset ETHM = LOCATION timed;
        colset RIOM = LOCATION timed;
        colset SWITCH = LOCATION timed;
        colset ETHLINK = LOCATION timed;
        colset PLANT_ELEMENT = LOCATION timed;

    (* Variables *)
        var loc: LOCATION;
        var dest: DEST;
        var src: SRC;
        var mbserver : SERVER;
        var mbclient, scanner : CLIENT;
        var proc: PROC;
        var ethm : ETHM;
        var riom: RIOM;
        var switch: SWITCH;
        var ethlink: ETHLINK;
        val IFgap_10Mbs = 10;
        val IFgap_100Mbs = 1;

(* Functions *)
    fun allPLCPROC() = 1`0++1`1++1`2;
    fun allPCBPROC() = empty;
    fun allPROC() = allPLCPROC() ++ allPCBPROC();
    fun PROCEdelay(proc) = withdispersion (
    case proc of
      0 => 2000
    | 1 => 18000
    | 2 => 50000
    |_ => 0
    , general_dispersion_range_in_perthousand);
    fun PROCRdelay(proc) = withdispersion (
    case proc of
      0 => 0
    | 1 => 0
    | 2 => 0
    |_ => 0
    , general_dispersion_range_in_perthousand);
    fun PROCWdelay(proc) = withdispersion (
    case proc of
      0 => 0
    | 1 => 0
    | 2 => 0
    | _ => 0
    , general_dispersion_range_in_perthousand);
    fun PROC_bootstrap_delay(proc) = discrete(0, PROCEdelay(proc))
    fun allETHM() = 1`60++1`61++1`62;
    fun isETHM(loc) = case loc of
        60 => true
      | 61 => true
      | 62 => true
      | _ => false;
    fun ETHM_bootstrap_delay(ethm) = discrete(0, 10000)
    fun isPCBPROC(loc) = case loc of _ => false;
    fun allRIOM() = 1`80++1`81++1`82++1`83++1`84++1`85++1`86++1`87++1`88++
    1`89++1`90++1`91++1`92++1`93++1`94++1`95++1`965++1`97;
    fun isRIOM(loc) = case loc of
        80 => true
      | 81 => true
      | 82 => true
      | 83 => true
      | 84 => true
      | 85 => true
      | 86 => true
      | 87 => true
      | 88 => true
      | 89 => true
      | 90 => true
      | 91 => true
```

```
    | 92 => true
    | 93 => true
    | 94 => true
    | 95 => true
    | 96 => true
    | 97 => true
    | _ => false;
  fun RIOM_IfiIter_delay(riom) = withdispersion(
     case riom of
       80 => 0 | 81 => 0 | 82 => 0
     | 83 => 0 | 84 => 0 |_ => 0
   , general_dispersion_range_in_perthousand);
  fun RIOM_OfiIter_delay(riom) =  withdispersion(
     600
   , general_dispersion_range_in_perthousand);
  fun aIISWITCH() = 1`480++1`481++1`482++1`483;
  fun isSWITCH(loc) = case loc of
      480 => true
    | 481 => true
    | 482 => true
    | 483 => true
    | _ => false;
  fun lFgap_delay(ethlink) = withdispersion (
  case ethlink of
    560 =>  IFgap_100Mbs
  | 561 =>  IFgap_100Mbs
  | 562 =>  IFgap_100Mbs
  |     _ =>  IFgap_10Mbs
  , general_dispersion_range_in_perthousand);
  fun isPLANT_ELEMENT(loc) = case loc of
    2000 => true
  | 2001 => true
  |       _ =>false;
  fun initial_events() = 1`(0,[2000,81,101,66,60,0,60,66,108,88,2001])@+7500;
  fun event_period(loc) = case loc of
    2000 => 21111
  | _ => 0;



(* Exchanges declarations *)

  (* Colors *)
    colset ID = INT timed;
    colset ROUTE = list LOCATION timed;
    colset EVENT = product ID * ROUTE timed;
    colset EVENTL = list EVENT timed;
    colset VAR = product ID * ROUTE timed;
    colset VARL = list VAR timed;
    colset MBFRAME = product SRC * DEST * EVENTL timed;
    colset MBFL = list MBFRAME timed;
    colset ETHFRAME = product ROUTE * MBFRAME timed;
    colset ETHFL = list ETHFRAME timed;


  (* Variables *)
    var id: ID;
    var route: ROUTE;
    var eventl, eventl1, eventl2: EVENTL;
    var var1: VAR;
    var varl, varl1, varl2 : VARL;
    var mbf: MBFRAME;
    var mbfl, mbfl1, mbfl2: MBFL;
    var ethframe: ETHFRAME;
    var ethfl, ethfl1, ethfl2 : ETHFL;



(* Complex declarations *)

  (* TCP/IP Ethernet protocol *)
    fun iproute(src,dest) = case src of
     66 => (case dest of
        100=> [66,60,560,480,563,481,566,80,100]
       |101=> [66,60,560,480,563,481,567,81,101]
       |102=> [66,60,560,480,563,481,568,82,102]
       |103=> [66,60,560,480,563,481,569,83,103]
       |104=> [66,60,560,480,563,481,570,84,104]
       |105=> [66,60,560,480,563,481,571,85,105]
       |106=> [66,60,560,480,564,482,572,86,106]
       |107=> [66,60,560,480,564,482,573,87,107]
       |108=> [66,60,560,480,564,482,574,88,108]
       |109=> [66,60,560,480,564,482,575,89,109]
       |110=> [66,60,560,480,564,482,576,90,110]
       |_ => [] )
     |67 => (case dest of
        107=> [67,61,561,480,564,482,573,87,107]
       |108=> [67,61,561,480,564,482,574,88,108]
       |109=> [67,61,561,480,564,482,575,89,109]
       |110=> [67,61,561,480,564,482,576,90,110]
       |111=> [67,61,561,480,564,482,577,91,111]
       |112=> [67,61,561,480,565,483,578,92,112]
       |113=> [67,61,561,480,565,483,579,93,113]
       |114=> [67,61,561,480,565,483,580,94,114]
       |115=> [67,61,561,480,565,483,581,95,115]
       |116=> [67,61,561,480,565,483,582,96,116]
       |117=> [67,61,561,480,565,483,583,97,117]
       |_ => [] )
```

```
|68 => (case dest of
    100=> [68,62,562,480,563,481,566,80,100]
   |101=> [68,62,562,480,563,481,567,81,101]
   |102=> [68,62,562,480,563,481,568,82,102]
   |103=> [68,62,562,480,563,481,569,83,103]
   |104=> [68,62,562,480,563,481,570,84,104]
   |105=> [68,62,562,480,563,481,571,85,105]
   |106=> [68,62,562,480,564,482,572,86,106]
   |107=> [68,62,562,480,564,482,573,87,107]
   |108=> [68,62,562,480,564,482,574,88,108]
   |109=> [68,62,562,480,564,482,575,89,109]
   |110=> [68,62,562,480,564,482,576,90,110]
   |111=> [68,62,562,480,564,482,577,91,111]
   |112=> [68,62,562,480,565,483,578,92,112]
   |113=> [68,62,562,480,565,483,579,93,113]
   |114=> [68,62,562,480,565,483,580,94,114]
   |115=> [68,62,562,480,565,483,581,95,115]
   |116=> [68,62,562,480,565,483,582,96,116]
   |117=> [68,62,562,480,565,483,583,97,117]
   |_ => [] )
|100 => (case dest of
    66=> [100,80,566,481,563,480,560,60,66]
   |68=> [100,80,566,481,563,480,562,62,68]
   |_ => [] )
|101 => (case dest of
    66=> [101,81,567,481,563,480,560,60,66]
   |68=> [101,81,567,481,563,480,562,62,68]
   |_ => [] )
|102 => (case dest of
    66=> [102,82,568,481,563,480,560,60,66]
   |68=> [102,82,568,481,563,480,562,62,68]
   |_ => [] )
|103=> (case dest of
    66=> [103,83,569,481,563,480,560,60,66]
   |68=> [103,83,569,481,563,480,562,62,68]
   |_ => [] )
|104=> (case dest of
    66=> [104,84,570,481,563,480,560,60,66]
   |68=> [104,84,570,481,563,480,562,62,68]
   |_ => [] )
|105 => (case dest of
    66=> [105,85,571,481,563,480,560,60,66]
   |68=> [105,85,571,481,563,480,562,62,68]
   |_ => [] )
|106 => (case dest of
    66=> [106,86,572,482,564,480,560,60,66]
   |68=> [106,86,572,482,564,480,562,62,68]
   |_ => [] )
|107=> (case dest of
    66=> [107,87,573,482,564,480,560,60,66]
   |67=> [107,87,573,482,564,480,561,61,67]
   |68=> [107,87,573,482,564,480,562,62,68]
   |_ => [] )
|108 => (case dest of
    66=> [108,88,574,482,564,480,560,60,66]
   |67=> [108,88,574,482,564,480,561,61,67]
   |68=> [108,88,574,482,564,480,562,62,68]
   |_ => [] )
|109 => (case dest of
    66=> [109,89,575,482,564,480,560,60,66]
   |67=> [109,89,575,482,564,480,561,61,67]
   |68=> [109,89,575,482,564,480,562,62,68]
   |_ => [] )
|110 => (case dest of
    66=> [110,90,576,482,564,480,560,60,66]
   |67=> [110,90,576,482,564,480,561,61,67]
   |68=> [110,90,576,482,564,480,562,62,68]
   |_ => [] )
|111=> (case dest of
    67=> [111,91,577,482,564,480,561,61,67]
   |68=> [111,91,577,482,564,480,562,62,68]
   |_ => [] )
|112 => (case dest of
    67=> [112,92,578,483,565,480,561,61,67]
   |68=> [112,92,578,483,565,480,562,62,68]
   |_ => [] )
|113 => (case dest of
    67=> [113,93,579,483,565,480,561,61,67]
   |68=> [113,93,579,483,565,480,562,62,68]
   |_ => [] )
|114 => (case dest of
    67=> [114,94,580,483,565,480,561,61,67]
   |68=> [114,94,580,483,565,480,562,62,68]
   |_ => [] )
|115 => (case dest of
    67=> [115,95,581,483,565,480,561,61,67]
   |68=> [115,95,581,483,565,480,562,62,68]
   |_ => [] )
|116 => (case dest of
    67=> [116,96,582,483,565,480,561,61,67]
   |68=> [116,96,582,483,565,480,562,62,68]
   |_ => [] )
|117 => (case dest of
    67=> [117,97,583,483,565,480,561,61,67]
   |68=> [117,97,583,483,565,480,562,62,68]
   |_ => [] )
|_ => [];
```

```
fun IPSTACKdelay(loc,(ssrc,sdest,evl)) =
withdispersion (10, general_dispersion_range_in_perthousand);
fun ETHdelay(wire,(ssrc,sdest,evl)) =
withdispersion (60, general_dispersion_range_in_perthousand);

(* Modbus protocol *)
colset SERVERL = list SERVER timed;
colset IOSCANNER = product LOCATION * CLIENT * SERVERL timed;
colset MBSCANNERxINT = product CLIENT * INT timed;
var serverl : SERVERL;
fun allSERVER() = 1`100++1`101++1`102++1`103++1`104++1`105++1`106++1`107++1`108++
                  1`109++1`110++1`111++1`112++1`113++1`114++1`115++1`116++1`117
fun allCLIENT() = 1`66++1`67++1`68++1`69++1`70++1`71
fun allPLCSCANNERresponses() = 1`(66,11)++1`(67,11)++1`(68,18)
fun allPCBSCANNERresponses() = 1`(69,9)++1`(70,9)++1`(71,0)
fun allIOSCANNER() = 1`(60, 66, [100,101,102,103,104,105,106,107,108,109,110]) ++
          1`(61, 67, [107,108,109,110,111,112,113,114,115,116,117])++
          1`(62, 68, [100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117]);
fun IOSperiod(scanner) = withdispersion (
case scanner of
  66 => 5000
| 67 => 10000
| 68 => 240000
|  _ => 0
, general_dispersion_range_in_perthousand);
fun MBdelay(loc,ssrc,sdest,eventl) = withdispersion (
case loc of
  60 => 100
| 61 => 100
| 62 => 100
| 3 => 100
| 4 => 100
| 5 => 100
| 80=>520
| 81 => 520
| 82 => 520
| 83 => 520
| 84 => 500
| 85 => 520
| 86 => 520
| 87 => 520
| 88 => 520
| 89 => 520
| 90 => 520
| 91 => 520
| 92 => 520
| 93 => 520
| 94 => 520
| 95 => 520
| 96 => 520
| 97 => 520
|  _ => 0
, general_dispersion_range_in_perthousand);

(* Processor module *)
colset PROCxETHFL = product PROC * ETHFL timed;
colset PROCxVAR = product PROC * VAR timed;
colset PROCxVARL = product PROC * VARL timed;
colset PROCxMBFL = product PROC * MBFL timed;
colset PROCxSERVERxCLIENTxVARL = product PROC * SERVER * CLIENT * VARL timed;
fun allPROCxnilVARL() = PROCxVARL.mult(allPROC(), 1`nil);
fun allPCBPROCxnilVARL() = PROCxVARL.mult(allPCBPROC(), 1`nil);
fun allPCBPROCxnilETHFL() = PROCxETHFL.mult(allPCBPROC(), 1`nil)
fun allPCBPROCxnilMBFL()= PROCxMBFL.mult(allPCBPROC(), 1`nil);
fun allPCBPROCxallSERVERxallCLIENTxnilVARL() =
    PROCxSERVERxCLIENTxVARL.mult(allPCBPROC(), allSERVER(), allCLIENT(), 1`nil);

(* Ethernet modules *)
colset ETHMxETHFL = product ETHM * ETHFL timed;
colset ETHMxVAR = product ETHM * VAR timed;
colset ETHMxVARL = product ETHM * VARL timed;
colset ETHMxMBFL = product ETHM * MBFL timed;
colset ETHMxSERVERxCLIENTxVARL = product ETHM * SERVER * CLIENT * VARL timed;
fun allETHMxnilETHFL() = ETHMxETHFL.mult(allETHM(), 1`nil);
fun allETHMxnilVARL() = ETHMxVARL.mult(allETHM(), 1`nil);
fun allETHMxnilMBFL()= ETHMxMBFL.mult(allETHM(), 1`nil);
fun allETHMxallSERVERxallCLIENTxnilVARL() =
    ETHMxSERVERxCLIENTxVARL.mult(allETHM(), allSERVER(), allCLIENT(), 1`nil);

(* Ethernet RIO Modules *)
colset RIOMxEVENT = product RIOM * EVENT timed;
colset RIOMxETHFL = product RIOM * ETHFL timed;
colset RIOMxEVENTL = product RIOM * EVENTL timed;
colset RIOMxMBFL = product RIOM * MBFL timed;
colset RIOMxSERVERxCLIENTxEVENTL = product RIOM * SERVER * CLIENT * EVENTL timed;
fun allRIOMxnilMBFL() = RIOMxMBFL.mult(allRIOM(),1`nil)
fun allRIOMxnilETHFL()= RIOMxETHFL.mult(allRIOM(), 1`nil);
fun allRIOMxnilEVENTL()= RIOMxEVENTL.mult(allRIOM(), 1`nil);
fun allRIOMxallSERVERxallCLIENTxnilEVENTL() =
    RIOMxSERVERxCLIENTxEVENTL.mult(allRIOM(), allSERVER(), allCLIENT(), 1`nil);

(* Switches *)
colset SWITCHxETHFL = product SWITCH * ETHFL timed;
colset SWITCHxETHLINK = product SWITCH * ETHLINK timed;
colset SWITCHxETHLINKxETHFL = product SWITCH * ETHLINK * ETHFL timed;
fun allSWITCHxnilETHFL() = SWITCHxETHFL.mult(allSWITCH(), 1`nil);
fun allSWITCHxallETHLINK() =
    1`(480,560)++1`(480,561)++1`(480,562)++
    1`(481,563)++1`(481,566)++1`(481,567)++1`(481,568)++
    1`(481,569)++1`(481,570)++1`(481,571)++
```

```
      1`(482,564)++1`(482,572)++1`(482,573)++1`(482,574)++
      1`(482,575)++1`(482,576)++1`(482,577)++
      1`(483,565)++1`(483,578)++1`(483,579)++1`(483,580)++
      1`(483,581)++1`(483,582)++1`(483,583);
fun allSWITCHxallETHLINKxnilETHFL() =
      1`(480,560,nil)++1`(480,561,nil)++1`(480,562,nil)++
      1`(481,563,nil)++1`(481,566,nil)++1`(481,567,nil)++1`(481,568,nil)++
      1`(481,569,nil)++1`(481,570,nil)++1`(481,571,nil)++
      1`(482,564,nil)++1`(482,572,nil)++1`(482,573,nil)++1`(482,574,nil)++
      1`(482,575,nil)++1`(482,576,nil)++1`(482,577,nil)++
      1`(483,565,nil)++1`(483,578,nil)++1`(483,579,nil)++1`(483,580,nil)++
      1`(483,581,nil)++1`(483,582,nil)++1`(483,583,nil);
fun SWdelay(switch,mbf) = withdispersion (
case switch of
     480 => 10
   | 481 => 10
   | 482 => 10
   | 483 => 10
   |     _ => 0
, general_dispersion_range_in_perthousand);general_dispersion_range_in_perthousand);
```

Résumé • Nous évaluons dans cette thèse deux performances temporelles des architectures d'automatisation distribuées sur Ethernet commuté et utilisant un modèle de coopération client/serveur :

- Le temps de réponse entre une occurrence d'un événement d'entrée et l'occurrence de l'événement de sortie correspondant;
- Le temps de cycle réseau pour la scrutation par un contrôleur de l'ensemble de ses modules d'entrées/sorties déportées.

La conjonction de trois mécanismes de consommation de temps rend ces deux performances variables et difficiles à déterminer de manière analytique. Par conséquent, la méthode proposée se base sur la simulation d'un modèle en réseau de Petri temporisé et coloré du comportement dynamique de l'architecture complète. Les résultats obtenus sur six architectures test permettent de :

- Montrer que les architectures multi-contrôleurs utilisant le modèle de coopération client/serveur donnent des temps de cycle réseau plus rapide que celles basées sur les modèles maître/esclave et producteur/consommateur ;
- Quantifier l'influence du réseau et des mécanismes de consommation du temps.

Mots Clés • Architectures d'automatisation, temps de réponse, temps de cycle réseau, Ethernet commuté, client/serveur, simulation, réseau de Petri

Abstract • In this work, two time performances of switched Ethernet automation systems that use a client/server cooperation model are evaluated:

- The response time from an occurrence of an input event to the occurrence of the corresponding output event;
- The network cycle time for the scanning by a controller of the whole set of its remote inputs/outputs modules.

The conjunction of three time consumption mechanisms makes both time performances variable and difficult to compute in an analytic fashion. Thus, the proposed method is based on simulation of a timed and coloured Petri net model of the dynamic behaviour of the whole automation architecture. The results which have been obtained on six benchmark architectures enabled us:

- To show that multi-controllers architectures using a client/server cooperation model provide faster network cycle times than those based on master/slave and producer/consumer models;
- To quantify the influence of the time consumption mechanisms on these performances.

Keywords • Automation architectures, response time, network cycle time, switched Ethernet, client/server, simulation, Petri net