NTNU – Trondheim
Norwegian University of
Science and Technology

Practical relevance of the Petri nets model language for a real-time multi-processor system

Pelayo Medrano Garcia

Master of Science in Cybernetics and Robotics
Submission date: December 2014
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Contents

Abstract 4
Keywords 4
Acknowledgements 4
1. INTRODUCTION 6
1.1 Area of research 6
1.2 Summary 6
1.3 Importance of the research 6
1.4 Relevant previous works 7
1.5 Problem Statement 7
1.6 Personal motivation 7
1.7 Methodology 7
1.8 Structure of the thesis 8
2 THEORETICAL FOUNDATION 9
2.1 Introduction Petri Nets 9
2.2 Theory: Petri Nets 10
2.3 Theoretical definition 10
2.4Firing a Transition in Petri Nets 11
2.5 Petri net structures which may appear 12
2.5.1 Sequence 12
2.5.2 Conflict 12
2.5.3 Confusion 13
2.5.4 Concurrency 13
2.5.5 Synchronization 14
2.5.6 Merging 14
2.6 Properties of Petri nets 15
2.6.1 Reachability 15
2.6.2 Boundedness 15
2.6.3 Liveness 15
2.6.4 Reversibility 15
2.6.5 Coverability 16
2.6.6 Persistence 16

2.7 Extensions of the Petri nets 16
2.7.1 Multiple Arcs 16
2.7.2 Inhibited arcs 17
2.7.3 High-level Petri net 17
2.7.3.1 Timed Petri Nets 17
2.7.3.1.1 Example 18
2.7.3.2 Stochastic Petri Net [1] [4] 19
2.7.3.3 Coloured Petri net: 19
2.8 Applications of the Petri nets 20
2.8.1 Finite state machines 20
2.8.2 Data flow 20
2.8.3 Prioritized producer-consumer system 20
2.8.4 Protocols of the communications 20
2.8.5 Multi-core systems 21
2.9 CONCURRENT SYSTEMS 21
2.9.1 Concurrency in Petri nets 21
2.10 QUEUES 21
2.10.1 Introduction 21
2.10.2 Characteristics of a queue 22
2.11 Analysis of a communication protocol system 24
2.12 Where have Petri nets been implement with success? 24
2.13 Monitors 25
3 RESEARCH DESIGN AND METHODOLOGY 26
3.1 Suppositions and relevant information regarding the models. 26
3.2 Model 1 27
3.2.1 Suppositions model 1 27
3.2.2 Hardware 29
3.2.3 Random creation 30
3.2.4 Auxiliary information 31
3.2.5 Main System 32
3.2.6 Alarm 36
3.2.7 Lines 40
3.2.8 Probability 42
3.2.9 Checking the bits sent 43
3.2.10 Modification 44
3.3 Model 2 46

3.3.1 Suppositions 46
3.3.2 Modification to the former model if more are to be used for the analysis of the information can be held. 47
3.3.3 Main system 48
3.3.4 Modification to the second model 51
3.4 Model 3 54
3.4.1 Suppositions 54
3.4.2 Third model 55
3.4.3 Main system 56
3.4.4 Message flow: 57
3.4.5 Sending cluster 61
3.4.6 Processing cluster 64
3.4.6 Modifications 65
3.4.6.1 Lines down 65
3.4.6.2 Alarm modification 66
3.4.7 Time when using more than one variable to select the specific time 69
4. DISCUSSION AND IMPLICATION OF THE RESULTS 71
Overflow 71
4.1 Model 1 71
4.1.1 Simulation 1 model 1 71
4.1.2 Simulation 2 model 1 73
4.1.3 Simulation 3 model 1 74
4.2 Model 2 76
4.2.1 Simulation 1 model 2 FIFO 76
4.2.2 Simulation 2 model 2 LIFO 78
4.2.3 Simulation 3 model 2 LIFO modified 79
4.3 Model 3 82
4.3.1 Model 3 simulation 1 FIFO 82
4.3.2 Model 3 simulation 2 LIFO 84
4.3.3 Model 3 simulation 3FIFO with modifications 86
5 DISCUSSION AND CONCLUSION 89
5.1 Introduction 89
5.2Goals conclusion 89
5.3 Implications 89
5.4 Limitations 90
5.5 Recommendations and Further investigation 91

References 92
ANEXES 94

Abstract

This is a master thesis that studies the practical applications of the Petri nets –a graphical and mathematical modelling tool [1] - for concurrent, real-time systems. Petri nets are a powerful tool for creating and analysing models and its behaviour in order to increase their performance. The systems that might be studied include concurrent, stochastic, nondeterministic, parallel and/or asynchronous systems.

The master thesis starts defining briefly the theory used in the developing of tools that simulates a real-time, concurrent, stochastic system. These tools or models will be describe in the next section and, while characterizing the models, another development options are offered so that other systems can be modelled from the ones presented here. It continues showing different application of the created models and discussing its simulation reports. In addition, it is provided in the conclusion further discussion about the suitability of the Petri net and the software used for the study of the concurrent, stochastic, nondeterministic, parallel and/or asynchronous systems. It also shows the appropriateness of a combination of the Petri nets and a programming language for studying this kind of systems.

Suggestions are provided for further investigations and extensions of this topic.

Keywords

Petri nets, real time system, optimization of share resources, concurrent system.

Acknowledgements

There are several people who have been of help and inspiration to me in my work and I would like to thank them for their assistance.

First, I would like to thank my advisor Tor E. Onshus for his guidance and support. I would like to thank my flatmate Alexander Donath who has been a great and supportive friend during all my period in Norway.

I would like to especially thank my parents, Jose Manuel Medrano Valle and Covadonga Garcia Rojo, because if I have been able to carry out this project and all my studies in my life was because of the way they raised me.

I would like to thank my friends Pablo Torres Garcia and Francisco J. Garcia Arlandis for their support and their friendship during the good and the bad moments in my studies.

I do not want to forget to thank the Erasmus Programme for the opportunity given to me of studying abroad and the European Union for making it possible.

I want to thank both universities, the NTNU and the UPV, for all the resources and information offered, and specially the NTNU international office for its help.

Finally I would like to thank the Ministerio de Educación de España, and the Gobierno de España, for their confidence in me and all the scholarships that I have been granted with.

I am really thankful to all of them and to everyone who has ever helped me, specially the biggest help that I have received.

1. **INTRODUCTION**

1.1 Area of research

Petri nets and real-time systems embedded computing systems.

1.2 Summary

Three models of simulating real-time systems are developed the present work, these models or simulations can be modified –and the modification are provided in the description of each model- in order to use them in a practical real-time system that might be applied for the user.

The main goals are to model a concurrent, stochastic, real-time system that could represent the behavior of a multi-processor and multi-sensor communication protocol, study and improve its performance and the used of the shared resources. Furthermore, it was also intended to study the suitability of the Petri nets graphical system, combined with the programming language, for the study of real-time systems.

The results are satisfactory with certain limitations: it is possible to represent any real-time system, as big as needed, with the combination of programming language and graphical Petri net language in an easy and visual approach. In case that the software CPN tools is used, the user should have good command on ML language –which is not a common programming language taught at universities- and, in case of more than one combination of variables is needed to select the specific amount of time that will last a task, more programming language code is required and it will gradually lose the visual easiness of understanding.

1.3 Importance of the research

This study is relevant in case that practical applications of the Petri nets are desired and a deeper understanding for a real-time system in both the suitability of the software CPN tools and the appropriateness of a combination of programming language system and Petri nets.

1.4 Relevant previous works

For this project the works of Claudine Chaouiya [10], Vlatka and Vesna [8] and Mor Peleg [7] are relevant previous works that show how to apply petri nets to complex system with deep understanding in such different fields. But the one that that is applied to solve a huge complex real-time system is the work of A. Spiteri Staines [3]: it is described and implemented a model of the metro in Paris, capable of analyzing and improving the one of the biggest metros in the world.

This works aims to follow these previous works.

1.5 Problem Statement

Nowadays, real-time, concurrent systems are part of our daily life in the digital age; it is intended to proportionate models capable of easy and fast understanding and high capability.

1.6 Personal motivation

Real-time systems are the requirement that is needed in the development of the computers, the machine and the tool for the humanity in the XXI century: they are leading the future of all knowledge, business and daily life. Furthermore, real-time systems can be applied to numberless process in the real world. My personal motivation was to study a visual, easy-understanding tool that could favor this denouement.

1.7 Methodology

Three models are created in order to provide different options while simulating and reading the reports and the results. In order to create the latter models it was chosen the use of the Petri nets graphical language and the software CPN tools for.

Petri nets was chosen for its easiness of understanding the programmes due to its graphical appearance and secondly because it is able to run the most complex real-time systems, theoretically.

There are several reasons why CPN tool was chosen instead of other programmes, but the main ones are the following motives:

a. It is in process of becoming an open-source programme. [18]

b. It is free software [18]

c. A CPN tool is a programme associated with a university, the Eindhoven University of Technology, in the Nederlands, and it provides support to its users.[18]

d. The web page that describes the programme explains how to create your simulations both by videos and by accurate narratives of its components.

e. Furthermore, during the programme selection process an email was sent asking some doubts about how a programme should be create and the email was answered the next day.

f. There are numerous studies of scientific literature with CPN tools in a hugely wide range of fields. [5] [2] [12]

1.8 Structure of the thesis

The thesis presents the following structure:

- A brief introduction introduce disclosing the thesis, its structure goals and reasons for been realized.
- Introduction and succinct comments on the theory of Petri nets used in this work.
- Description of the models created.
- Analysis and discussion of the results of the simulations.
- Conclusions and further investigations.

2 THEORETICAL FOUNDATION

2.1 Introduction Petri Nets

Petri nets are a very effective tool for modelling and analyzing concurrent processes [1]. The success of the Petri nets is largely due to the simplicity of model creation- Petri nets involve mainly two elements: transitions and places- and its success is also due to the visual representation of a system that makes for the user really easy to understand by just having a quick look.

However, the representation of large systems is costly. Further improvements have been made to the Petri nets and different programming languages have been added to the Petri nets tools so that these tools can deal better with a real system.

Petri Nets have been used as the basis for the introduction of time extensions that allow an accurate description of complex applications, which provides the possibility of exploiting the structure of the models to develop a proper simulation to the system.

Petri Nets were first described by Carl Adam Petri in 1962, a time where programs and processing were considered to be sequential and deterministic. This allowed the Petri nets to be one of the first models that were applied to the study of concurrent systems.

Petri nets are widely utilized for analyzing Dynamic Systems with Discrete Events. These dynamic systems are increasing its importance in our nowadays society as automation and control systems grow around us in different fields such as robotics, transportation systems, distributed systems, and telecommunications networks, among many others.

In order to optimize the design, improve the performance control and develop a better implementation of the system –this is, creating a fault tolerant system-, it will be needed models that allow us study different implementations along with qualitative and quantitative analysis.

Petri Net models have turned out to be outstanding tools to analyse real-time systems, which are characterized by having a specific number of processes which communicate and coordinate with each other and share common resources. Although in the case of parallel computation are not used due to its complexity, the Petri nets are an excellent tool as an alternative method of design and simulation. [20]

This is the goal that will be pursue in this master thesis, it will be created a tool that will be able to analyse and optimize different concurrent systems: in this case a communication system was chosen, but any concurrent system can be implemented by just changing some characteristics. It will be showed how to create the reports needed to study the evolution of the system and which improvements could be made. Furthermore, the Petri nets and the CPN Tools will be analyse in order to determine how accurate are to develop any kind of tool for these goals.

In showing the theory of Petri nets it is not intended to provide the whole background of Petri nets theory, but the theory that is used and it is desirable to know in the running of the latter models.

2.2 Theory: Petri Nets

A Petri net is an oriented graph with two kinds of nodes: places and transitions [4]. These two are commonly represented as circles and rectangles each. The arches connect the places and the transitions or vice versa, the arcs will show the path that the marks will follow in the model.

Each place can contain either a positive number of marks or zero marks. Each mark will represent information depending on the model that has been created: a mark is called token and a token could represent a state in the process, a unit of time or a resource that can be used, among others. The marking specifies an amount of tokens in each place. [4]

The tokens are used in modelling the transitions – they move from the places towards ingoing arcs to the transition node, and from there via outgoing arcs to the next place. Transition can have several input and output nodes or places which are interpreted as pre - input data, input signals, logic conditions, buffers, resources needed- and post conditions - output data, output signals, logic conclusions, resources released- of the event corresponded to this transition - computation step, processor's signal, logic clause, CPU cycle-. The token held in the place reflects the evaluation of the corresponding condition to the logical true.

Transitions are events or actions which cause the change of state. A transition is enabled if all input nodes have the number of tokens equal to or greater than those needed for firing the transition. This is, if a transition has 2 arcs that aim to it, then it should have at least two tokens, each one of them going through each arc.

2.3 Theoretical definition

A Petri net is formally defined [1] as a 5-tuple N= (P, T, I, O, M_0), where:

I. $P=\{p_1,p_2,\ldots,p_m\}$ is a finite set of places;

II. $T=\{t_1,t_2,\ldots t_n\}$ is a finite set of transitions, $P \cup T \neq \emptyset$, and $P \cap T=\emptyset$

III. $I: P \times T \rightarrow N$ is an input function that defines directed arcs from places to transitions, where N is a set of nonnegative integers.

IV. $O:T \times P \rightarrow N$ is an output function that defines directed arcs from transitions to places; and

V. $M_0: P \rightarrow$ is the initial marking.

The tokens are used in the previous definition in M_0, as they defined the Petri net that was built. But the number and positions of the tokens may change as the model of the Petri net is executed, so in the definition just the initial marking is used. The different position that are reached by the tokens are the different states that the system can reach and will be marked M_1,M_2,M_3…,M_i.

2.4Firing a Transition in Petri Nets

When a transition is enable it means that it can be fired, and sometimes more than two transitions are enable at the same time, as the example showed in the figure. [16]

In the software used to develop the models created for studying different systems every enable transition is represented with a green shadow, meaning that it would allow the firing.

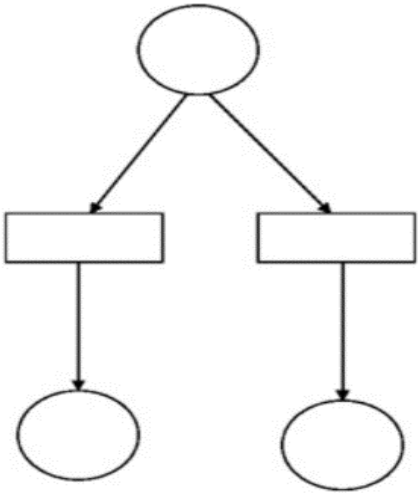

Figure 1: A place with two paths

In a basic Petri nets model the firing will be completely randomly, and in higher level Petri nets it is possible to add conditions and settings to the basic model in order to represent more realistic or more complex systems.

Initially, the basic Petri nets model is a state machine, where each state is a place and the current state is marked by a token. Each state is connected to others states through transitions that will be fired when the conditions for enabling them are reached.
This could be a system like a traffic light: red, green and orange will be the states and the time that that has passed would be the conditions that will be applied in the transitions.

2.5 Petri net structures which may appear

If no extensions of the Petri nets are used, there are some basic configurations that will conform most of the systems that are created. The most common ones will be: [2] [16]

2.5.1 Sequence

Several activities follow each other.

In the picture the initial place have a token, and just one token is needed to fire the first transition- there is just one arc that connects both elements and there are no numbers in the upper-side of the arc indicating how many tokens are needed-. The token will follow the only path possible in this case, this is, a sequence of states and actions.

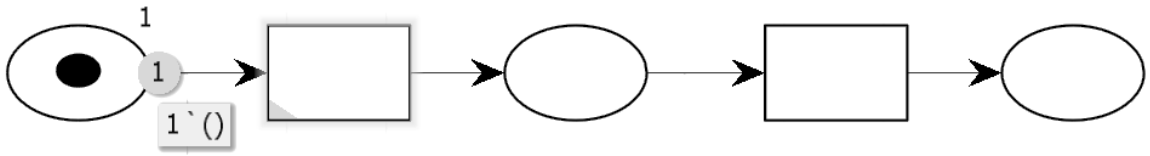

Figure 2: Sequence of transitions and places

2.5.2 Conflict

The token in a place enables more than one transitions, making the system run different states at the same time if there are more than one token available for different transitions.

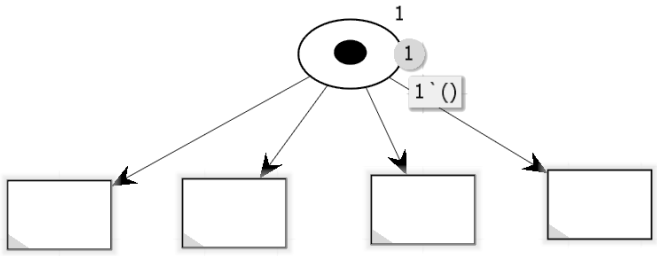

Figure 3: Conflict

In the picture 3 firing one of the transitions disables the others transitions that were also ready to be fired. If there are no further conditions or settings the firing will be done randomly and different paths will be followed. This is known as a conflict and helps us to model synchronization problems.

If in this case there were more than one token in the initial place then more than one transition could be fired and more than one track could be followed. If each transition has a path after it then the system will be executing two or more paths at the same time and this will create a concurrent system.

2.5.3 Confusion

This is the combination of the previous cases concurrency and conflict. Each place enables two transitions, but only one can be fired. In a basic Petri net system this will be fired randomly, and in a higher Petri net system it would be possible to set some conditions to lead the process to the adequate path.

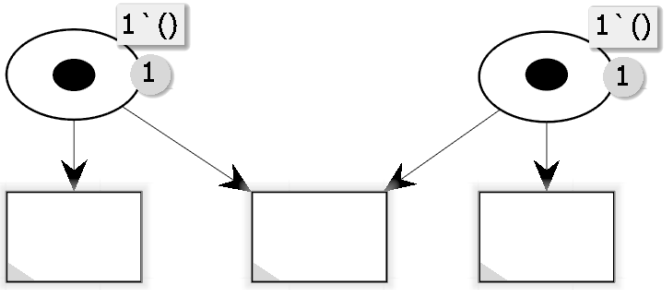

Figure 4: Confusion

2.5.4 Concurrency

Several systems work with concurrent tasks, this mean, with actions that happen at the same time. It can be done the way showed in the picture below or as it was explain in the case “conflict”.

A further explanation is provided in the next section.

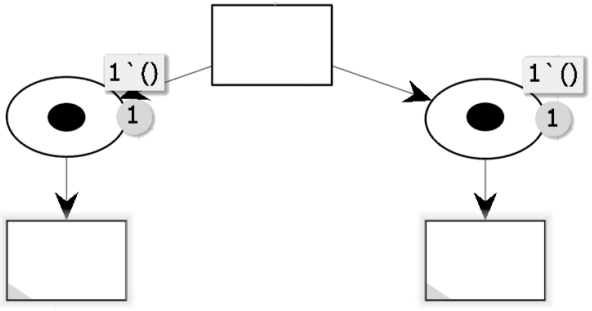

Figure 5: Concurrency

2.5.5 Synchronization

Transition is activated when all the previous states have been reached. This is used when the concurrency is running the system and the process need to be synchronized.

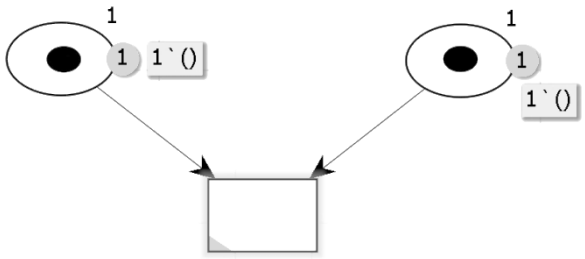

Figure 6: Synchronization

2.5.6 Merging

Merges several parallel processes but not necessarily at the same time. It can be seen in the picture 8 how different processes merge in the same path but it is not needed synchronization.

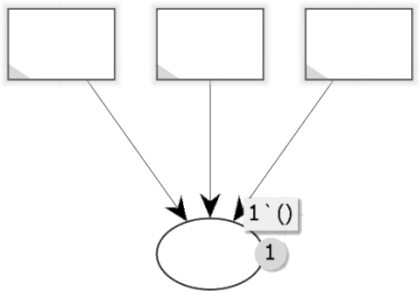

Figure 7: Merging in a place

2.6 Properties of Petri nets

Petri nets are graphical representations of a discrete event system and these representations can have some of the following features [1] [2]:

2.6.1 Reachability

This characteristic describes whether a system can reach a state or not. An analysis of the reachability describes all the reachable states.

This analysis answers to the question whether the marking M_n is reachable from the marking M_0. This is true if there is a sequence of transitions firings that will lead to M_n starting from M_0.

This analysis answers the following question: “Can the model reach one particular state from another?”

2.6.2 Boundedness

A Petri net is called m-bounded if each place contains less or equal than a finite integer number m tokens for any reachable from M_0 marking.

The analysis answers: “Will a storage place overflow?”

2.6.3 Liveness

A Petri net is called live it is possible to execute any transition of the net through a firing sequence.

This property is utilized for checking if the dynamic behaviour contains no deadlocks.

The analysis asks and answers the following question: “Will the system die in a particular state?”

2.6.4 Reversibility

A Petri net is said to be reversible if for any reachable marking M_n, M_{n+i} can be reached from M_n, with $i \in \mathbb{N}$ and $i > 0$, and M_n can be reached from M_{n+i} .This characteristic is utilized for checking the properties of the Petri nets when a cycle is within its nets.

2.6.5 Coverability

A marking M is called coverable if exists reachable marking M1 such that for each place in the net the amount of tokens in this place is more or equal than the amount of tokens in the same place under marking M.

2.6.6 Persistence

A Petri net is called persistent if for two enabled transitions the firing of one will not affect the other.

2.7 Extensions of the Petri nets

The basic model has been extended in order to be able to face more complex models so that more precision is achieved.

Different kinds of systems can be described using Petri nets: asynchronous, parallel, nondeterministic and stochastic, among others. Petri nets can also be applied as a precise graphical representation of such systems. From the Graph theory point of view Petri net is a directed bipartite graph, in which the nodes represent the transitions or events that may occur and places or conditions.

The first modifications made to the basic model were the inhibitors arcs and the possibility of using multiple arcs.

2.7.1 Multiple Arcs

It is said that there are multiple arcs when there is more than one token coming from one place that enables a transition. For example, if it is wished that an action happens three times before another action can be processed: like the numbers of mistakes that can be supported by the system. [14]

The graphical solution is depicted in the picture 9:

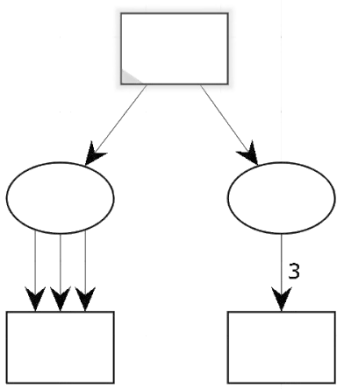

Figure 8: Multiple arcs

2.7.2 Inhibited arcs

An inhibitor arc is another component of extended Petri nets, it goes through one place to a transition and it is represented with a small circle at the end of the arc.

The transition that has inhibitor arcs cannot be fired if the input place contains at least as many tokens as the multiplicity of the inhibitor arc. [14] [16] For example the following figure fire when p1 have a token and p2 do not have tokens.

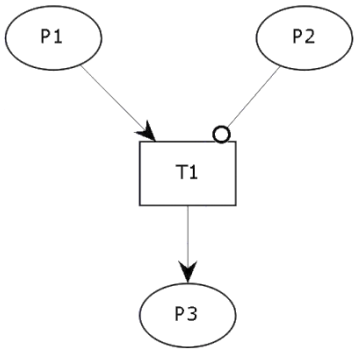

Figure 9: Inhibited arc

2.7.3 High-level Petri net

A Petri net received a name of high-level Petri net when some complex extensions are utilized for generating a model that represents a system. The following extensions showed here are the most common extensions and they are also the ones that run the models that were created to study the multi-processor system.

2.7.3.1 Timed Petri Nets

Petri Nets were originally proposed as a causal model, where it explicitly despised time, but the time was needed to simulate proper real-time models and make the Petri nets more capable.

The concept of time is introduced in the Petri nets in order to model the temporal behavior of dynamic systems. This behavior includes the time that a transition –if the transition simulates an action- needs to be processed, but the behavior can also represent the time that is added to the arcs, simulating the time that the token needs to be moved. Just with these two additions of time every timed process can be simulated. [12] [4]

The completion of an activity may be viewed as a change in the state of system.

Since the changes of status in Petri Nets are determined by fired transitions it is natural and intuitive relate the completion of activities to the firing of the transitions.

The enabling of a transition corresponds to put a server to work in an activity, and the firing of the transition corresponds to the completion of the activity, that has produced results-results that are showed as tokens-.

Every token will have its own time attached to itself and the tokens will be available when the time of the system and the time attached to the token concur.

On the other hand, transitions can be fired only after they are enabled. Thus the system will identify enabled transitions to produce the next step every time that a step is finished.

As this extended model is one of the basic extensions chosen to create the programs to simulate a real-time system, further explanations should be provided: it will be showed with an example.

2.7.3.1.1 Example

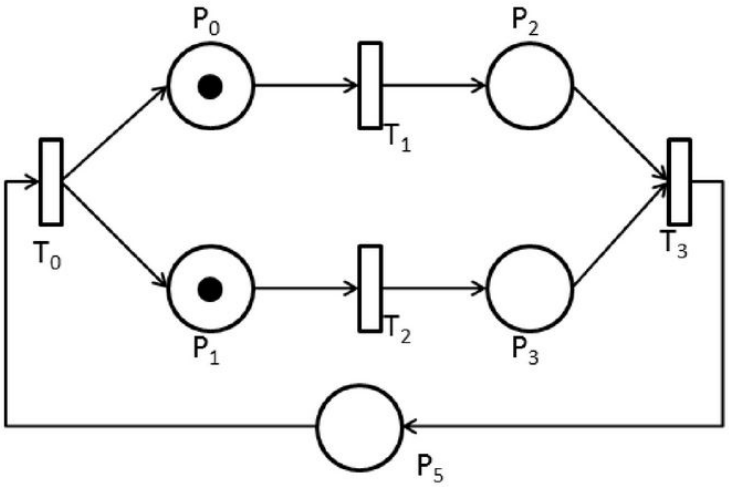

Figure 10: Example

The picture 11 has the following conditions: P0 has a token that is available at the unit of time 3- and the transition T1 needs 6 units of times to be processed- and the token that is contained in the place P1 is available at the time 6- and the transition T2 needs 3 units of time to process the token-.

Then, if the system starts at the initial time, 0, the time of the system will remain in this unit of time until all the transitions that can be fired at this point in time are performed. As in the example of the picture there is no enable transition the system will remain without changing until at least one of the transitions is enable. Therefore the system analyses the model and transitions. In this case the next available transition is T1 because it is enable at the time 3 for the token in P0. Consequently, the system will move towards the time 3, this is, the time of the system will be now 3 and the transition 1 can be fired.

The transition T1 will create a token in the place P2 when it is fired and this token will be enabled at 9 units of time. As the token in the place P1 is enabled at 6 units of time and

the transition that processed it needs 3 units of time, the token that will be created in P3 will be also available at 9 units of time. Thus the transition T3 will reach the conditions needed for firing at the time 9.

But if it is considered another case, like if one of the transitions lasts longer, for example three units of time more, then the token that is available before is not enough to fire the transition T3, so the transition T3 will have to wait 3 more units of time and then the system can go to the next step.

2.7.3.2 Stochastic Petri Net [1] [4]

A definition of the Stochastic Petri nets could be summarized in the next sentence:

"The timed behavior of the Petri net system is specified with exponential random variables".

This extension of the Petri nets is an attempt to represent complex models with stochastic processes.

This makes possible to avoid a detailed description of the deterministic system operations by replacing them by probability assumptions. What this means is that, in order to introduce probability functions in the system, the probability functions are associate with each transition in a Petri net.

For example, it could be a random variable with exponential distribution that expresses the delay until the firing of the transition.

It might be possible that it is not needed all the transitions to have a delay, there might be some immediate transitions.

If this is the case and immediate transitions are going to be used then it is said that Generalized Stochastic Petri Nets (GSPN) are being used.

2.7.3.3 Coloured Petri net:

Coloured networks and systems are an extension of Petri Nets. [11] It is because of this Petri nets extension that elaborate inscriptions -associated with places, transitions and arcs- can write compact models of complex systems.

The new feature introduced is the possibility for the Petri nets to have marks for each token. Each mark may have information whose type depends on where it is located: each place defines what kind of tokens can be hold in it.

The color tokens may represent a distinctive attribute, and in case it is needed to define two or more attributes, the places can have the attributes if they are defined before.

For example, if a producer-consumer system is modelled, it might be possible that more than one kind of consumer is needed and more than one kind of product is required. Thus, every different kind of consumer can be identified with a specific string, such as “consumer1” and “consumer 2” among others, and different products can be created by the producer –“prod1”,”prod2”-.

This extension of the Petri nets will also allow adjudging to each kind of consumer a product: maybe consumer1 just needs prod2.

This extension was also used in the creation of the tool that simulates a real-time system.

2.8 Applications of the Petri nets

2.8.1 Finite state machines

State machine is a behavior model that simulates a system with inputs and outputs, where the outputs depend not only on the current input signals but also on the former input signals.

A finite state machine is a state machine that has a limited number of states.

2.8.2 Data flow

Petri nets can also be utilized for modelling the flow of the data in order to boost its performance. This section could be considered a subdivision of the protocol of communication section, but sometimes just the dataflow is study because there are not communication protocols.

2.8.3 Prioritized producer-consumer system

Classic problem of multi-processes synchronization.

There are producers that create a resource and consumers that use it and the best protocol that allows synchronizing all of them is searched.

2.8.4 Protocols of the communications

A communications protocol is a set of rules and standards that allow two or more entities of a communication system communicate with each other to transmit information. Petri nets can be applied in these protocols to improve the performance of the system.

2.8.5 Multi-core systems

Combines two or more independent microprocessors, the Petri nets can be utilized for determining which combination of characteristics and communication protocols can be implemented to obtain the best possible system.

2.9 CONCURRENT SYSTEMS

Concurrent systems are simultaneous interactive task that are running at the same time. This task can be executed by one or more executors. They are non-deterministic and are present in a wide range of areas like informatics and logistics. [17]

The execution may proceed in many different ways, e.g. depending on:

- The scheduling of the individual processes.
- If the messages can be lost during its transmission.
- The time at which inputs are received in the system.

2.9.1 Concurrency in Petri nets

Two transitions are concurrent if both of them can be fired simultaneously.

It is important to distinguish between concurrent transitions and the concurrence of system activities, which is how the model is interpreted.

In case that the transitions implement system activities or actions, then both the transitions and the system activities must agree: in order to achieve this goal the transitions will simulate as accurately as possible the real actions. [13]

2.10 QUEUES

Queues have been widely used in evaluating performance of the dynamic systems [12] [16], mainly for computer systems, communication networks and systems manufacture.

2.10.1 Introduction

Simple queues are places where the items will remain until a share resource becomes free and it can be used with the items. These places are commonly named buffers, and the resources are called servers.

In computer science they are virtual tools that are reserved for the temporary storage of digital information while it is waiting to be processed.

The items that are to be processed or attended are called clients and they are generated by external arrival processes and will be place in the queues waiting for receiving a service from one of the servers.

When all the actions have been performed, the supposition is made that the customer leaves the system. In order to fully characterize the behavior of the system, it will be needed to define these concepts: the number of servers, the service discipline - this is, the policy followed in the queues, the queue capacity, arrival processes and service processes- .

The number of servers may be one, a finite number greater than one, or infinity, but at least there must be one. The discipline of service or planning rule defines the policy by which the servers are assigned to various customers in the system.

This policy is extremely important in the design and creation of any system: depending on the policy used the system will be able to be more effective, what this implies is that the system can be hugely improved without change any of its components, just by changing the condition of the process of the clients. This is particularly useful if the system already exists and it is difficult or expensive to make change in its components.

The queue capacity is the total number of clients that can be simultaneously waiting in the system. This will be strongly subjected to the characteristics of the system modeled; it is not the same the number of messages that can be store in a RAM memory as the number of clients that can stay in a dentist waiting room.

Arrival processes characterize the way in which customers arrive at system. Mathematical approximation can be utilized for simulating the process, like the Poisson distribution or a Normal probability density function, but either way, the function chosen should be the worst case possible so that the system is design to cope without problems when it is implemented.

2.10.2 Characteristics of a queue

The importance of evaluating the performance of the system is that it provides the information of the behaviour of a discrete event system.

The main performance parameters of interest are: the average response time, the average waiting time, the average queue length, productivity, and server utilization. Furthermore, regarding the average values, it may be of interest to know the amount of variance in the average data, and also the distributions of this data. [12]

For example, it may be important to know the variance of the time that the clients remain waiting, besides knowing its average value. If the variance is really high and in the process is limited the maximum waiting time, then the first priority should be the improvement of the variance of the clients.

In queue representations may be appropriate to distinguish between different types of clients. If this option is chosen, it would be necessary to define performance parameters for all of the clients without depending on the class they are and it would also be necessary to define performance parameters depending on the class. In the models showed in this work just the first option was needed.

Finally, some real systems clients do not arrive one by one, but in groups, and it might be also possible that these clients have to be process at one time each one of them, or all of them at the same time. Again this will depend on the kind of system that it is modelled, and the software used in the master thesis is capable of dealing with both cases, and both cases are implemented in the different models showed.

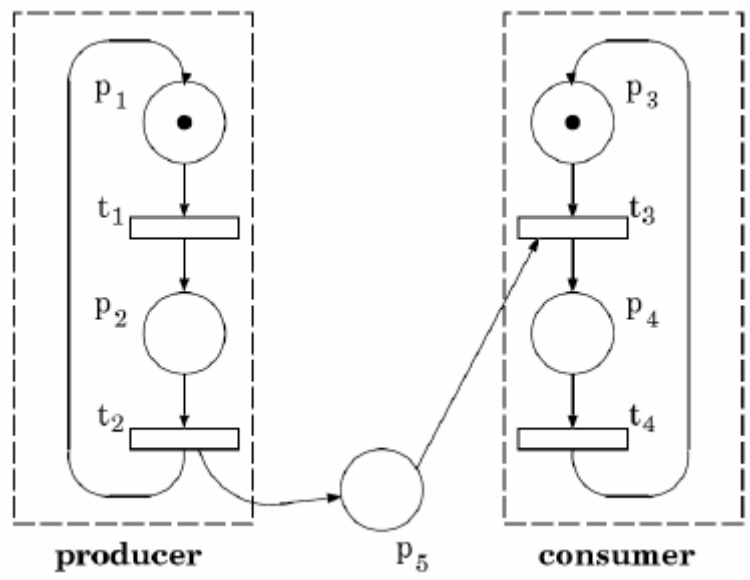

Figure 11: producer and consumer system

It can be seen in the picture a schematic representation of a producer and a consumer in a Petri net system. To this model it might be possible that it is needed to add more than one producer and more than one consumer -that would be created reproducing the schemes that are in the picture-.

It might be also possible that more than one product is needed, and again, if the user wants to operate with the basic Petri net configuration then the schemes should be repeated.

Different queue protocols can be used and in this work mainly two have been used:

- FIFO is an acronym for First In, First Out, a protocol for shaping and operating a data buffer, where the first entry is processed first. [12]
- LIFO is an acronym for last-in, first-out, a protocol for modelling and operating a data buffer, where the last entry is processed first. [12]

In this models the queues will be called lists; a list is an instrument provided by CPN tools that saves different clusters of elements in a place –like the combination message1, processor1 saved with the combination message2, processor3 all in the same place-. These combinations are link together and will remain together until a transition disjoins them.

2.11 Analysis of a communication protocol system

The behaviour analysis of the communication protocol is an effort to quantify the performance of the network and represent it through some significant numerical measures. It can be defined the behaviour analysis as the development and study of mathematical models that predict the performance of the networks in a defined numerical data.

In order to create the best performance analysis possible, there are at least four key elements that should be incorporated into the model:

- The network protocol: it is a set of conventions and standards that govern the processing and data format. For example, how the different devices access to the net or how they identify priorities among others actions.
- The network topology and network configuration: These are the graphic description of the arrangement of the different network components and their interconnections.
- Operational action measures.
- Finally, the characteristics of sending network packets.

2.12 Where have Petri nets been implement with success?

- Software design.
- Workflow Management
- Data Analysis
- Concurrent programming
- Reliability Engineering
- Diagnosis
- Discrete process control
- Simulation
- KPN Modeling

2.13 Monitors

The monitors are a tool provided by CPN tools that analyses the information generated by the simulations.

The monitors used for the models are counters: there are counters of transitions - it count the number of times a transition has been fired -, counter of places - it counts the number of tokens in the place- and counter of list length - it counts the average number of tokens per unit of time in the place where the list is located -. [19]

3 RESEARCH DESIGN AND METHODOLOGY

In order to study the different systems and analyze its behavior three models were created with the software CPN tools, the reasons why this was chosen this software were formerly explained.

Moreover, not only three models are explained in the following section, but also some of its variations are offered in case the user needs to create a similar model.

3.1 Suppositions and relevant information regarding the models.

- Programming language: CPN Tolls uses the Petri nets graphical programming language as a main language of programming, but in its expansions it uses the CPN ML language. This language is an extension of the functional programming language Standard ML. It will be employed in the Coloured expansion.
- The models showed in this work can be modified to be used in another task and for this reason modifications of the models are discussed.
- In the models a processor is attached to a message and, just afterwards, the units of times that the information needs to be sent are added to the. This may be seen as a protocol that assigns a processor to a message necessary for the system to work properly, and after the assignation the message is sent.
- The monitors are tools provided by the software that analyses the information and presents it in reports that deliver plenty of information useful for improving and analysing the suitability the models.
- In the model there are two clusters of transitions and places that are called alarm and lines –that will be described latter- and represent the simulation of random, asynchronous events. From these examples other events might be developed by the user.
- Different coloured extension units are: Boolean, integer, real, strings, time and unit, among others. These colours are utilized to represent different implementations of the petri nets: in these models two examples would be the coloured strings units that represent messages and processors.

3.2 Model 1

The goal of this model is to accurately simulate the behaviour of a communication system, this is the first and most simple models of the three showed here and some suppositions are needed to be considered.

3.2.1 Suppositions model 1

In this simulation the followings suppositions were adopted:

- The information is carried by bits-the bits are Boolean units-.
- There are four clusters of transitions and places that simulate sensors or units that creates information.
- The timed, coloured and stochastic extensions of the Petri nets language are used.
- It is only possible to use one processor.
- Pages hardware 1, 2, 3 and 4 represent the creation and flow of the messages.
- Page probability represents the function probability.
- The main page shows the system and the relationships between pages.
- Each token has a number that shows where each tokens belongs:
    - Number 1: is a token of cluster hardware number 1.
    - Number 2: is a token of cluster hardware number 2.
    - Number 3: is a token of cluster hardware number 3.
    - Number 4: is a token of cluster hardware number 4.
    - Number 5: is a token that represents the processor.
    - Number 9: is a token that represents the flow of the states of the event-simulator called alarm.
    - Number 10: is a token that represent the flow of the states of the event-simulator called lines.
    - Number 11: is a token that enables the probability function.

This model will be useful for simulate a low-level communication protocol, although instead of bits the supposition could be bytes, Kbytes or bigger groups of bits. The model presents the limitation that just one processor can be used, and therefore it will be just useful for medium or small system, like a PLC. Some applications of PLC are showed in the paper [20].

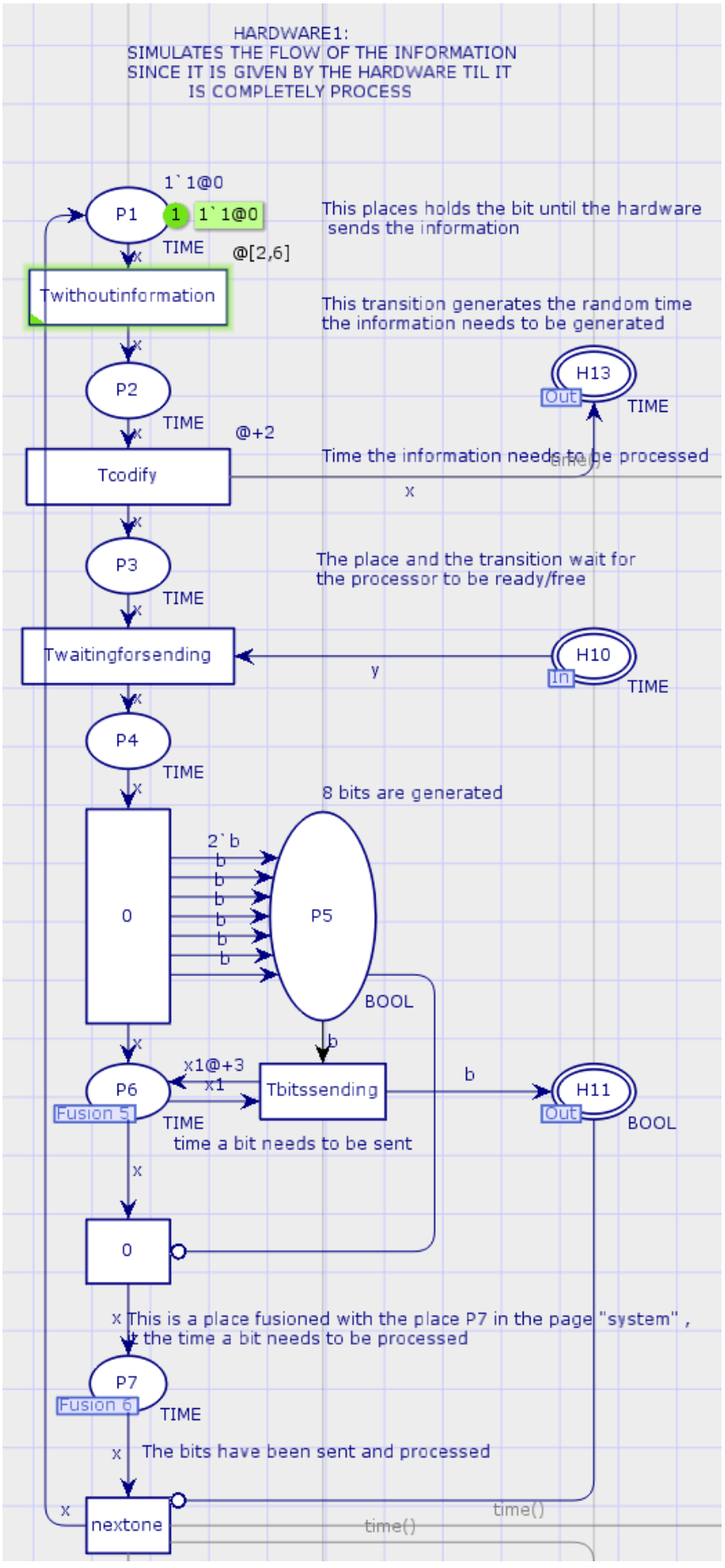

Figure 13: Hardware 1

On the other hand this model is extremely easy to understand as it is highly visual and the use of ML language is reduced. It is also suggested for its use in case that the sensors generate more than one message at the same time.

This model would be recommended either to study a small system or to start programming with CPN tools.

3.2.2 Hardware

The process starts with a token or "information" waiting for being sent, it is set as a random time, but can easily be change into a specified number: instead of @[2,6] it should be write @+j, where j would be the time that periodically the information is created.

Furthermore, in this design there is just one token, but in a real system the system can send different information before the previous message can be processed, so it would be more realistic added the figure 15 to the first place P1.

After the hardware receives the information of the system, it generates either a digital or analog signal and waits for the processor to be free. Thus there is here a bottleneck and shared resources, and, as it is a shared resource, just before it there will be a queue. This will be the main elements in this study of the system performance.

In the second design it will be possible to add as much processors and messages of information as the designer wishes.

Furthermore, after P4 it is found a transition that generates 8 bits, but it could be as much bits as it is wished. It has been represented with 7 lines in order to make more evident how multiple arcs can be created.

Afterwards, P6 will add time to the message token for each bit that it is sent and, as it can be observed in the picture, the bits are sent out of the hardware to the page of the system, which will be described latter.

P5 and the transition next to P6 are connected with a specific arc, called inhibitor arc, which does not allow the performance of the transition until there are no tokens in P5. This is made in order to get all the bits sent before the token gets to the next step.

And the next step is P7, and this is a fusion place. What this means is that it is connected with at least one other place, and they work as one: if one of them receives a token, the other place can use it. These kinds of places are utilized to connect places that are far from each other or that are in different pages, as in this example in picture 14.

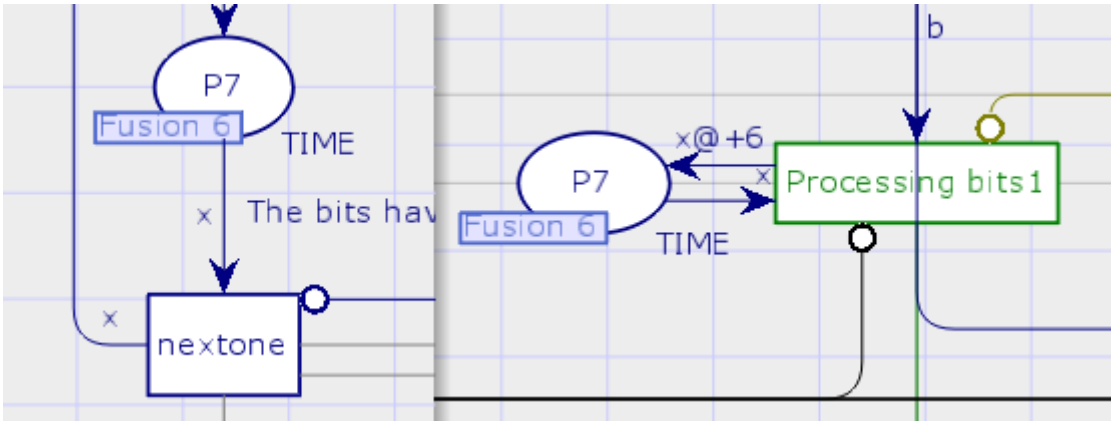

Figure 14: Detail fusion place

And again, this place P7 works as the previous one does, it adds time to the token each time a bit is processed, and does not get to the next step until all the bits have been processed.

This is the last step of all of them, and it goes again at the beginning of the process, and there it waits for the system to send the information to the hardware1 and it starts again.

3.2.3 Random creation

This is an easy way of generate a random number of messages; between 5 and 500 units of time - in this case in the picture 15- a new message will be generate.

Of course there are more examples of how to create a random number of messages, but here it is showed one of the easiest. Obviously, when a message is processed, it should not get back to P1 unless the system specifies it that way, and therefore the arc that joins the transition “next one” and P1 should be deleted.

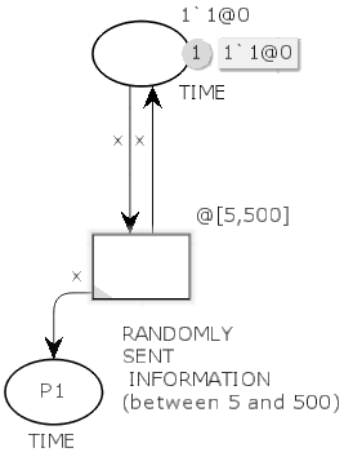

Figure 15: Random tokens creation

3.2.4 Auxiliary information

In order to know the times for each token, auxiliary places and tokens have been created.

The places on the right, from px5 to px7 count the time that a token needed to be processed, counting the time that has been in the queue. - If there is no need for counting this time then the arc with the function time () should be took from the transition “T waiting for sending”- . The arc that goes to px5 comes from the transition “T codify” with the function time ().

Here function time() is utilized to compare two things: the places on the right compare the time since the information is sent by the system to the hardware till the information is completely processed, and the places on the left compare the time that has passed since the last token of this hardware has been sent.

As it can be seen, the place px3 is use to retain a token with the time of the last one that has been sent so as to be able to make the comparison.

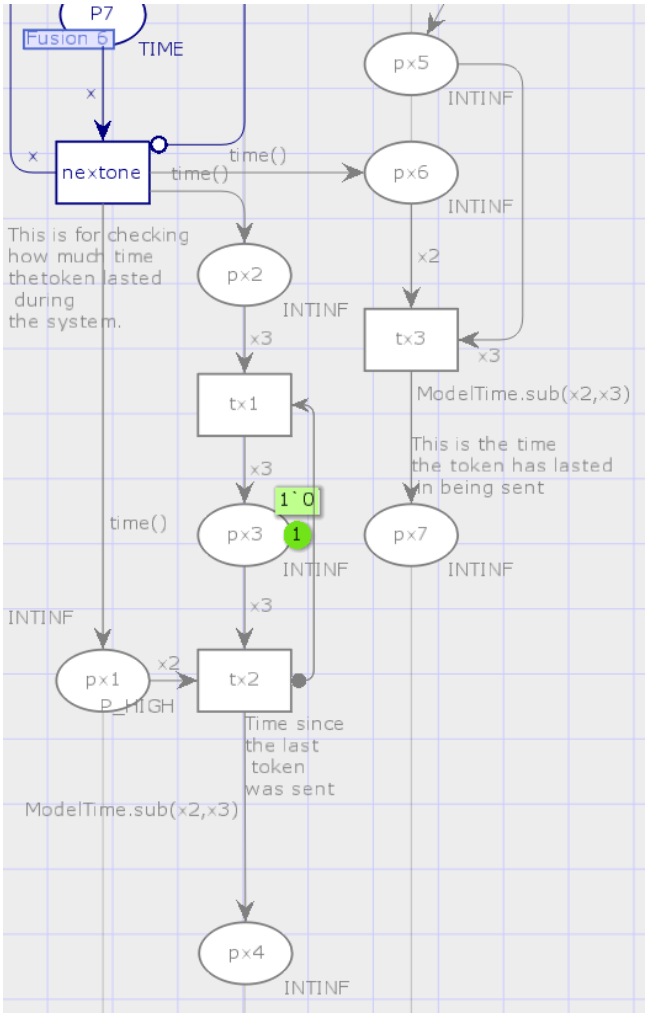

Figure 16: Visual information

3.2.5 Main System

In the next picture 17 it can be observe how the different hardware pages are related. All four of them share one processor, represented by the TIME token number 2, and at the beginning the processor is free: it can be seen here why a fusion places is a useful tool for modelling, in this case the places were fused to make it easier to view, although it was not necessary.

It should be also explain how the different pages can be related to one another: if the program is observed, it can be seen that each hardware and the system are in different pages and the different pages are not related with arcs that join any transition or place between the pages.

The relations between pages is obtained with the tool hierarchy in the program CPN Tools, but, briefly explained, in order not to have all the transitions and places in the same page, the user can make a page inside the main page that contains some of this transitions and places. So, the transition on the main page called “Hardware1” in blue is referred to all the places, arcs and transition showed in the page that was called “Hardware1” and was described before. All these transitions are the transition Hardware1 for the page system.

Each cluster of transitions, arcs and places related to one function such as “hardware1, processor, alarm…” have been given a different color to help the understanding of the user.

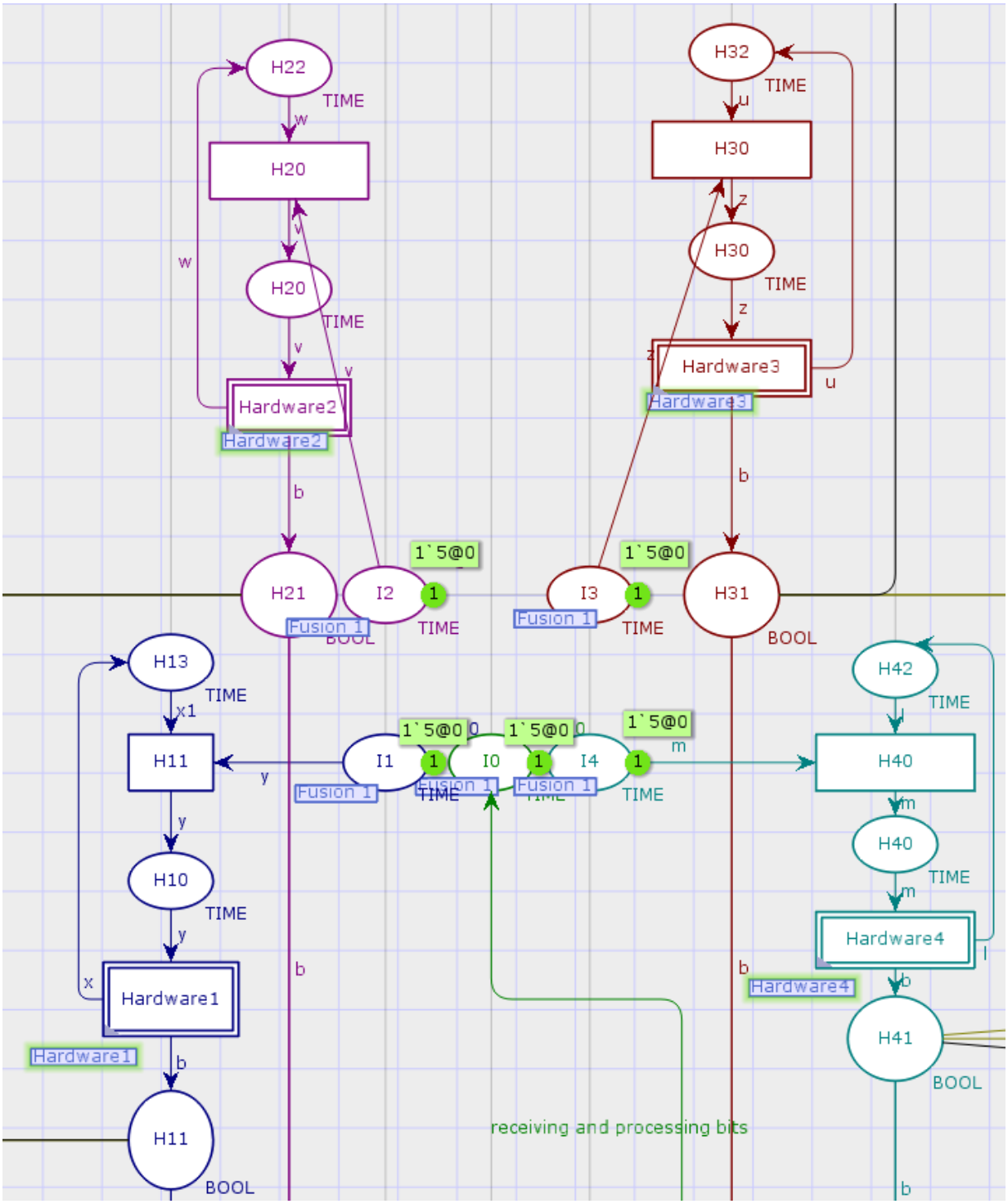

Figure 17: Shared processor

Before the processor can be matched with the information, this is, before the processor can enter the “Hardware1” and keeps the flow of the information; the information needs to be codified. This is why an arc goes out of the main transition –Hardware1, 2, 3 and 4- towards the upward place in each color cluster.

As it can be seen in the figure 13, when the transition “T codify” simulates the creation of the information in readable data, then a token goes out of the page towards the main page to allow the processor to be attached to the information and continue its flow. If

there are no processors available, then the information will remain in the place until one is free.

In case it is needed to add a maximum waiting time it can be implemented the next figure 18, or another similar one:

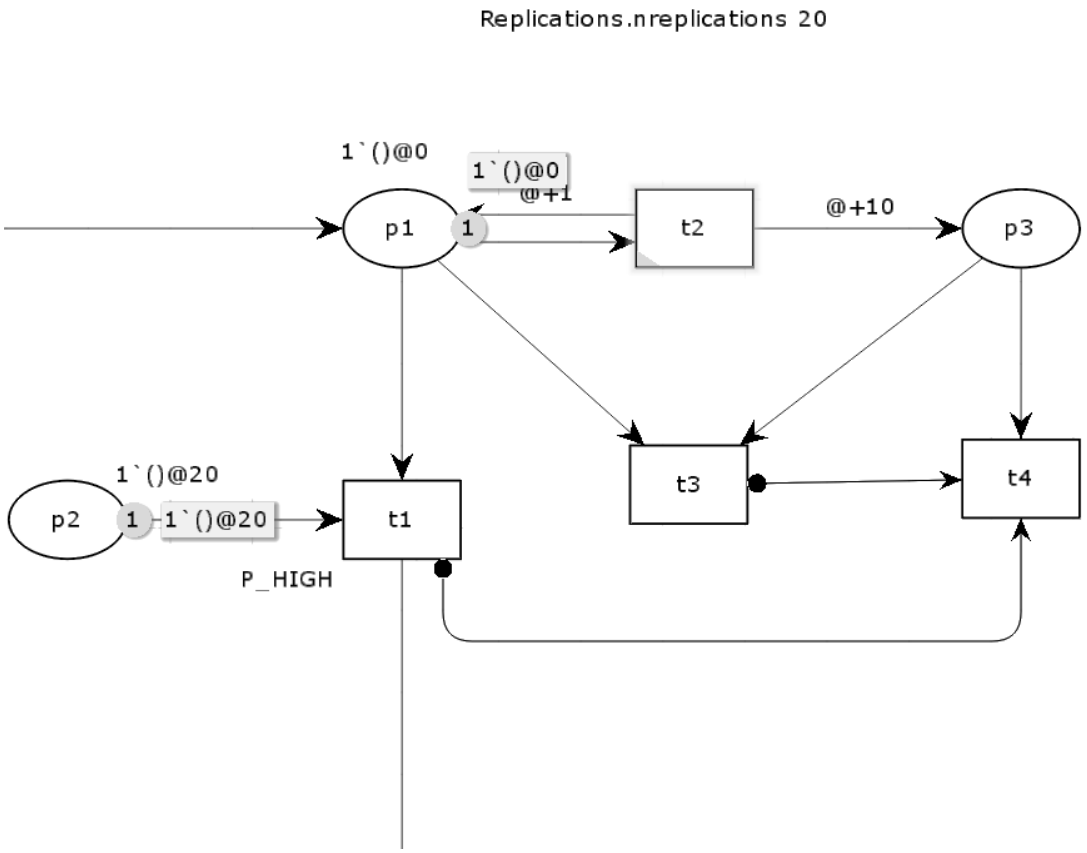

Figure 18: Modification waiting time

P1in the figure 18 is P3 in the figure 13, t1 is the transition “T waiting for sending” and p2 is H10.

What this figure does is to wait a certain amount of time until the processor can be used or information is finally dismissed. As it can be seen in the former picture, the processor will be available at the time 20, the information has arrive at the time 0, and the information will remain in p1 always 10 units of time, as it is state in the arc that connects t2 and p3. This, of course, can always be changed as the user wishes.

Analysing it, when the information is in p1 the transition t2 will be continuously adding one unit of time at the token, until the time comes when either the processor is free and allows the information to keep flowing or the waiting is too long and the t3 is enable and takes away the information. As p3 has one token for each unit of time created by the transition t2 it should be delayed all of them when the waiting has come to an end. This is

why the t4 is only enabled after t1 or t4 have been fired and will delay all the tokens that are in p3.

The arc that are drawn with a ball at the beginning of the arcs are a kind of constraints arcs, and these ones do not allow the transition that is in their end of its arrow to be fired until the transition that is in their beginning has been fired. This protocol has not been drawn in the figures in order not to make the users misperceive the system because of the amount of data that has been showed.

If the green cluster is observed now in the figure 19 it will be seen the next step of the flow of the information. It has been described before how P7 was used.

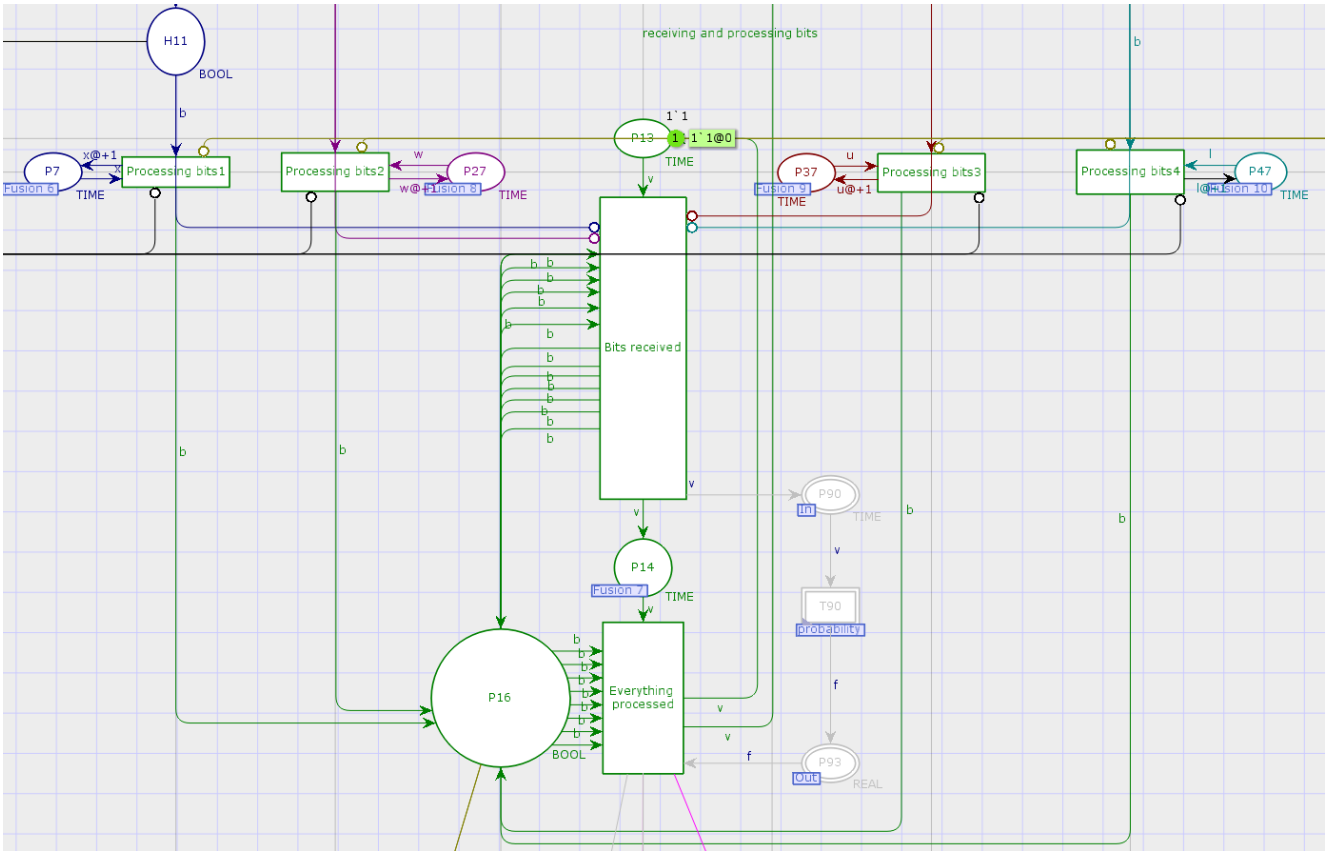

Figure 19: Cluster of transitions and places processing

There you can find the different transitions that process the bits that produces every hardware, one transition for each hardware. This way, if every hardware has different times of process-due to different communication protocol or different instructions to operate for the different information- then it can be easily written down for the user. In this example it has been chosen one unit of time for each bit processed.

When a bit is processed the amount of time that has been written in the arc will be added to the token. Afterwards, when the bits are processed they go to the place

P16, and here they remain until they are 8 tokens, this will allow the transition called “Bits received” to be fired.

This transition is not really necessary for the proper performance of the program, but it has been drawn in order to add to the model a probability function that will be explained after, the probability function is the cluster in grey next to the green cluster. Although, at the same time, the transition “Bits received” can be also have other function if the user needs them, for example, to send out a message that all the bits have been received-then the user just will need to create an output arc towards a place that counts how many times this transition has been fired-. But it is important to remark that it is not necessary strictly speaking.

It is important to state here that only one processor will be useful for this model, as it can be observed:

If in the model more than one processor is working, then all the four clusters of receiving data work independently: at first sight it should not be a problem, but then all of the bits that are processed go to P16 so, for example, if P7 sends 5 bits, and P27 send 3 bits the transitions “Bits received” and “Everything processed” will be fire, without really processing a whole package of bits of information from the same sensor.

If the user needs more processors can choose the other model show in this master thesis, the second one can easily work with as many processor as needed, and it was initially design with this purpose.

But, if the user wants to use more than one processor in this model, then the user can make a modification: it is needed to copy the cluster green three times- it can be done with the “copy tool” in the blinder “Create” located in the “tool box”- and then he can separate the clusters in different pages so it is more visual. What this means is to create again four times –or the number of times that is needed- the flow of the information and the processing of the bits in independent clusters. They will be just connected by the use of the same processors –shared resources- in the places I0, I1, I2, I3 and I4.

It should be emphasised, even if this modification is done in this model, then all the processors should be the same prototype, because the time that is added in each transition called “Processing bits1” or “Processing bits2” or the others ones, does not depend on the type of processor that you create. If this requirement is desirable for the model, then it should be chosen the other model, or use programming functions like the ones implemented in the other models.

3.2.6 Alarm

It has been created another system that could simulate an external alarm, and it can be done with some variations: it could either an alarm that is fired using a function to create a random time or an alarm that is fired in a specific spam of time. In this case the first option was chosen.

If the figure 20 is observed, it will be seen the main process of the alarm that is simulated by the places A1 and A2, and the transitions Tw –Time waiting- and Tsa-Time start again-.

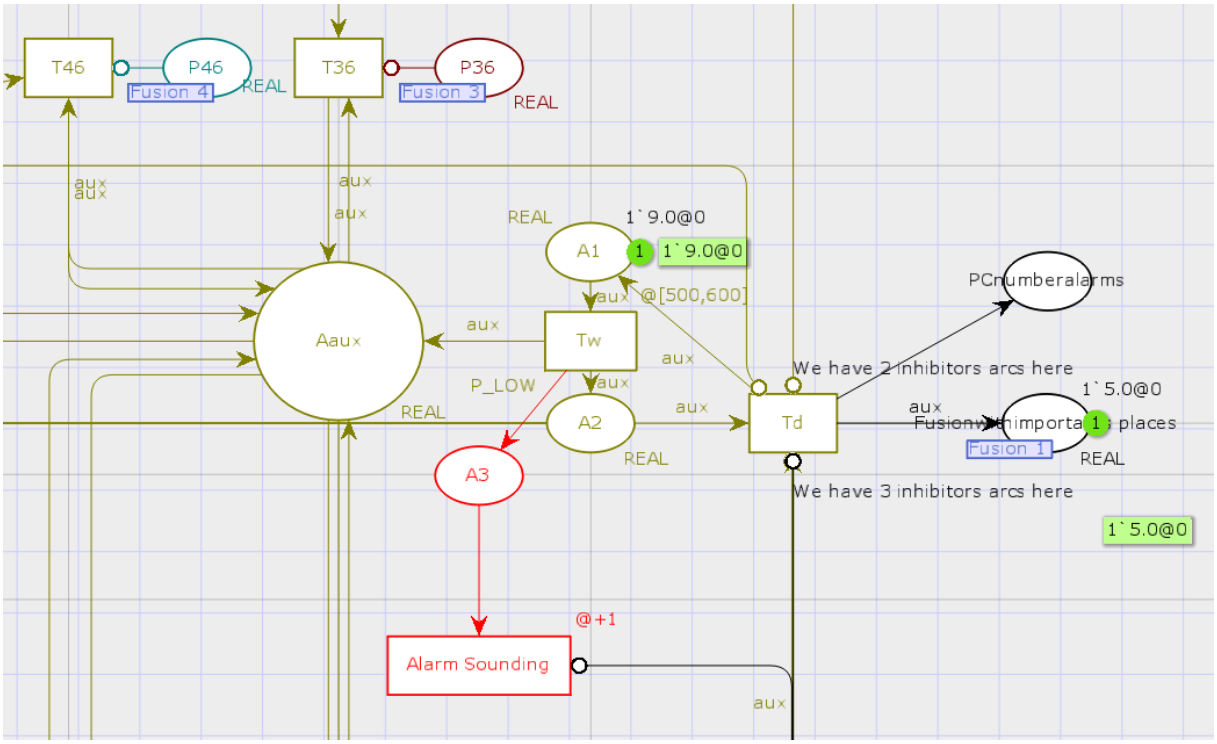

Figure 20: Alarm cluster

The alarm system will start in the place A1, with an initial token: a real number defined as a 9.0 for the alarm system. This can be seen because it is written in the place, 1`9.0@0, in the upper-right side, and it is available at the time 0,-@0-.

The transition Tw will give it a random time between the numbers that are written in its upper-right side, @ [5000, 6000] in this case and two tokens will be generated and will go to A2 and Aaux. The first one will make the alarm sound, represented as a transition in red, when the token that has been created is available. This token will be available at the time that will be generated after adding the initial time that have the token in the previous place-A1- to the random time added by the transition Tw.

As it can be seen, there is a detail in the transition that simulates the alarm, the place A2 is the place that simulates the state of “Alarm Sounding”, when the token is here it means that the system is down and the alarm should be sounding, but the transition that sounds the alarm is the transition fired by the place A3. This has been made in order to differentiate the state of system down of the alarm sounding that, although they happened at the same time, they are not the same.

And if they remain in the same place then the token would have to choose between fire the alarm or simulate that the event passed-fire Td- and both things are needed in the

model, not just one. Furthermore, if the system of the figure 21 is tried, then it is possible that an infinitive loop would be created: the transition Alarm sounding will be always available.

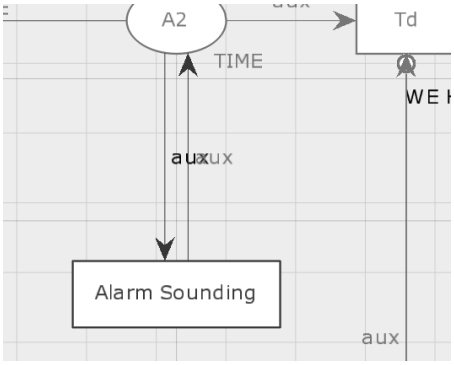

Figure 21: Alarm loop modification

In this model it is supposed that the alarm sounds because of a mistake made in the sending or the processing of the bits, so the alarm will stop when the system advance to the next message. If the user wishes to create an alarm that is generated because of the fall of one component that needs to be repair, or a similar condition, then the user can implement the solution in the figure 23 and it will be explained in the next paragraphs.

Following the description of the alarm, the other token that is created will go to the place Aaux; this place will allow the transitions that have around it to delete the Boolean tokens that represent the bits. This is done because of the supposition of the alarm: The firing of the alarm means that there is a conflict with the sending of the bits and therefore they were not properly processed, so they must be deleted before the transition “Everything processed” takes place.

Whenever the place Aaux has an available token in it, the transitions T6, T16, T26, T36 and T46 will able to be fired in case there are any Boolean tokens in either the places that are waiting to be processed –like H11 in the hardware1- or P16.

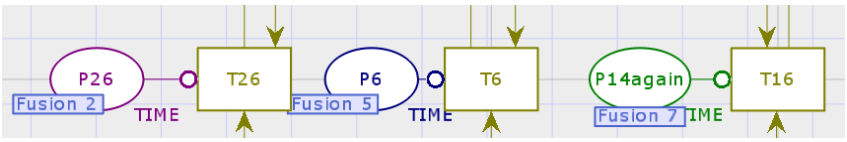

Figure 22: Alarm auxiliary transitions

These places are connected with an inhibitor arc to the transition Td so that Td does not get fired when there are tokens in these places. Thus, when all the tokens that represent the bits are deleted then the alarm will stop sounding and the alarm process will start again.

There are two more places, “PC number alarms” and “fusion with important places”, the first one is a counting place, it has been created to show how many times the alarm has been fired. A monitor can be added to the place if the monitor analysis is going to be used.

The second place is a fusion place with the places I0, I1, I2, I3, and I4, this place generates a token that simulates a processor so that the system can start again its usual service after the alarm has been fired.

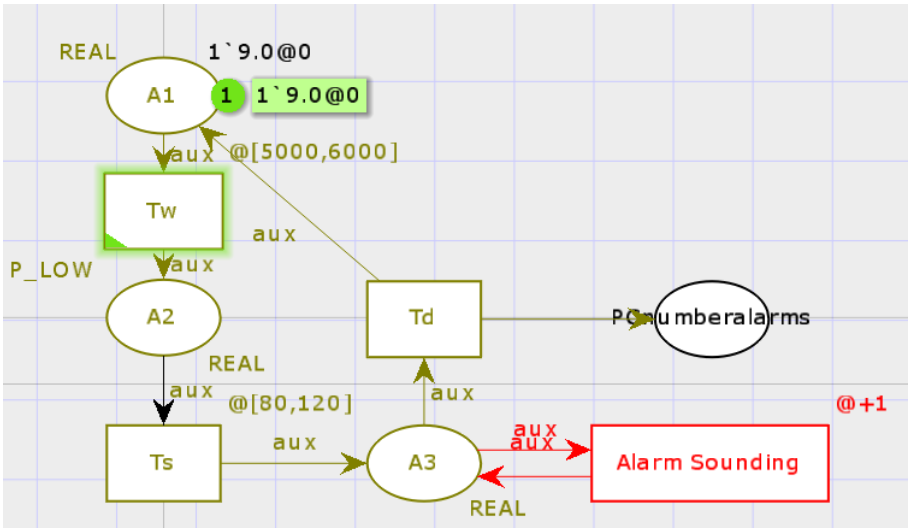

Figure 23: Alarm cluster modification

Another detail worth revealing are the inhibitors arcs that connect A2 with all the transitions that simulate the process of the information, which it is absolutely logical: if the alarm is fired because of a problem sending or processing the information then the transitions that simulate this problems should not work, there is a problem with them and must be solved before start again, and that is what the alarm cluster simulates.

Regarding the modification that was mention before, the main change to realize in the figure 23 is the addition of the place A3 and the transition Ts. The initial transition gives the token the specific time that will make him available for the next transition, Ts, and this transition does the same as the former one, it gives the token another random time, this time between 80 and 120 that will be the time that the alarm will be sounding. This time the alarm transition also needs to add a unit of time to the token if the infinitive loop is to be avoided. When the time comes and the transitions Td and Alarm are enable at the same time, then one of them will be fired randomly. If it is fired the transition alarm sounding the situation will happen again, but if it is Td then the flow of the alarm will start again.

It is important to remark that there are other ways of simulate how an alarm works depending on what is needed, it was just showed here two of the most common systems that are to be found in a real-time system.

In this case, the alarm is a random, asynchronous event, when it is fired it is supposed that the communication has failed and then all the bits that are being sent at that moment have to be deleted.

3.2.7 Lines

This is also a random, asynchronous event called lines, but lightly modifications are showed.

This case simulates the use of two lines for the transmission of the bits, A and B, in case one of them fails.

The supposition is the next one: A starts working as the line A is the main line in the model, and when A fails then the line B starts working until the line A has been repaired and everything can keep working as usual. It has been stated that the line B will have a specific number of uses and cannot be repaired: this is to simulate the difference between qualities of the lines that can be used in a real system, between the main and expensive line and the auxiliary and cheap line.

The number 3 that appears at the figure is the number related with the arc that links the place “Line B” and “T Work B”, this number shows the number of tokens that “Line B” should have to allow “T Work B” to be fired.

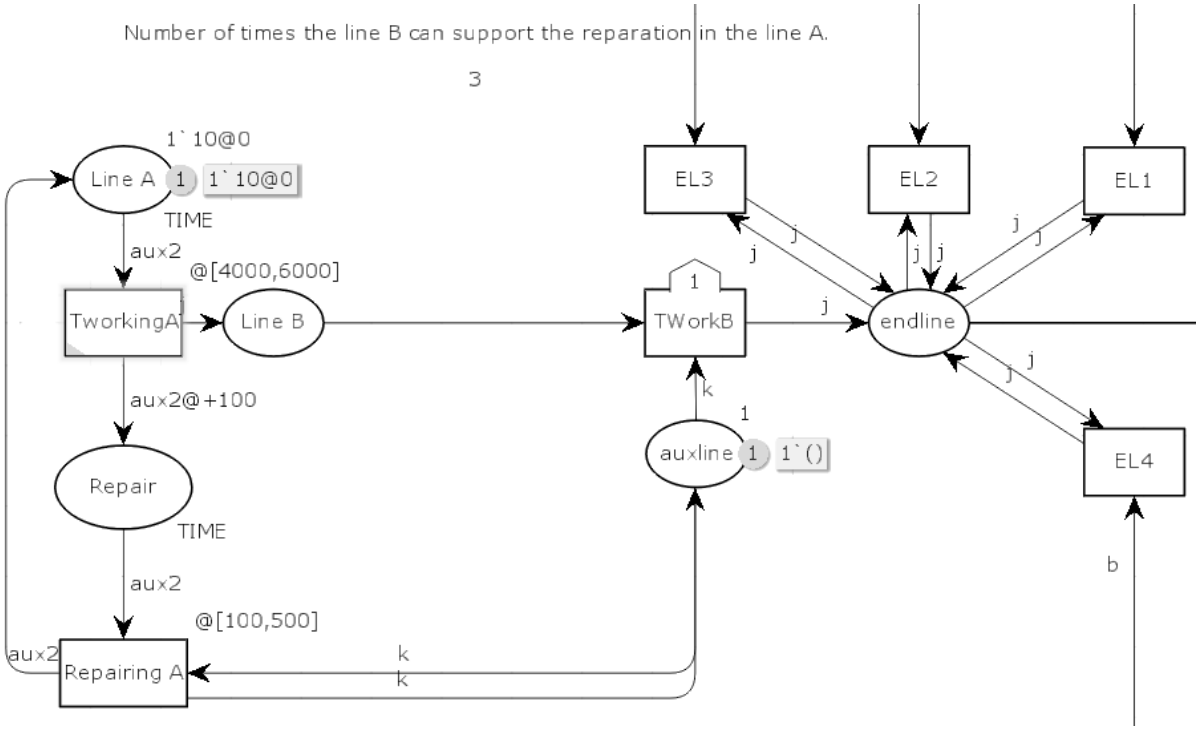

Figure 24: Cluster lines

This process starts with the line A working, simulated by the initial token the place called “Line A”. The breakdown of the Line A is simulated as a random time between 4000 and 6000 units of time, and it can be change by writing other numbers in the upper-right side of the transition “T working A”. This transition will generate two tokens that will go to the next places: “Repair” and “Line B”. The token in the place “Repair” will remain here until the line A is repaired, this has been stated that will last between 100 and 500 units of times – it was done using the same way that was used before- and when the line A is ready to work again then the token will go to the start and a new loop will start again.

Looking now to the right side of this sub-model, it will be seen the simulation of the work of the line B. The line B will always work after the line A is down, and when the line B works three times then it will enable the transition “T Work B” and this transition will create a token that will simulate the complete breakdown of both lines. A token will remain in the place “end line”. This place has the same function as the place A2 in the alarm cluster; this is the place that simulated the sounding of the alarm. This place enables all the transitions that has around it to destroy all the tokens that are being sending or processing: the lines are down and the system is unable to keep working.

The token in the place “auxline” is the one that allows the line A to repair. Thus, when the line B is also down, this token is also taken by the transition “T Work B” and the line A will remain always in the place “repair”.

In case that both transitions are enable at the same time and the token contained in the place “auxline” have to choose, two characteristics have been created: The arc that connects the transition “T working A” with the place adds time to the token, so that this token will be always available 100 units of time after the other token is available, so here one transition is given priority over the other.

Figure 25: Detail constraint

This can also be done by writing a higher priority in the transition that is needed, but this way it is showed another ways of creating priorities, -this could be useful if the user needs to work with software that does not implement priorities in itself-. The second characteristic that has been created is the double arc that joins “auxline” and “Repairing A”. This double arc is used as a mean to unable the transition “Repairing A” when the transition “T Work B” is fired and therefore all the lines are down.

Moreover, a constraint has been created: This constraint will allow firing the transition once in the simulation. It is the constraint showed in the picture 25 and it is placed in the transition “T Work B”. In fact this constraint is not rigorously needed, this function is already made by the token in the place “auxline”, but it is showed here as a double check.

3.2.8 Probability

There are some mathematical functions that are provided in case the user needs them. The most common ones to simulate a real system would be Exponential, Normal, Poisson, Student and Uniform, between others.

In this system it is implemented the normal function to simulate the probability of process wrongly a message. As it can be seen, the transition T90 is enable by two places, the left one is the place that enables the transition when the main system generates a token to allow the function probability perform.

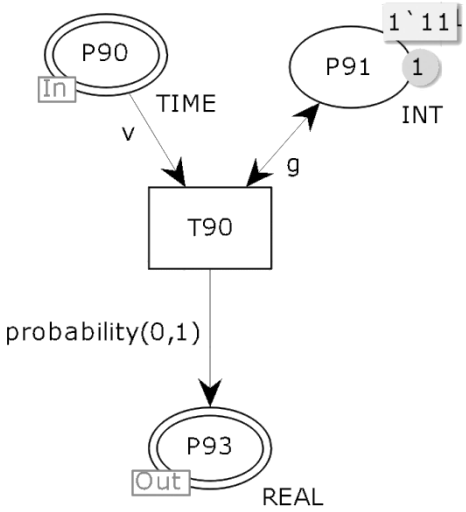

Figure 26: Implementation mathematical functions

The place on the right is an integer place that can be utilized for introducing the values of its tokens in the function probability: if the function probability, instead of being written as probability (0, 1), could have a “g” instead of a 1, the value that the function probability will take would be the one that the place P91 holds, this is, the variable “g” that would represent the number 11.

This is done to allow the user generate more easily the requirements of his own system. As it can be observed, the arc that joins the transition and the place aims to both directions. This is another way of showing the double arc makes the transition require a token from the place and then it returns the same token to the place. It could be also a

```
fun probability (n:int, i:int) =
    let
        val realn = Real.fromInt n
        val reali = Real.fromInt i
        val c=reali*reali
        val rv = normal(realn,c)
    in
        rv
end;
```

Figure 27: Function probability

checking place.

The function probability receives two integers, that in this model it is just given it 0 and 1, and the function convert the model to real numbers, using the functions showed in the picture 27, then the variance is created by multiplying the standard variation that was introduced before in the function, and therefore, a real random variable is created calling the function normal. This is the value that the function and the arc will return.

```
normal(n:real, v^2:real) : real
```

Figure 28: Detail normal function

The change of the integer values to real values is done because the function normal defined by the program CPN Tools showed in the picture 28 requires it in the definition that offers. More information about this function and other ones can be found in the web page or the manual of the program.

The main system will send a token to allow the creation of a random real token, this would be required just before all the bits are processed, this token is utilized in this case to decide whether or not the package of bits that is being sent is properly processed.

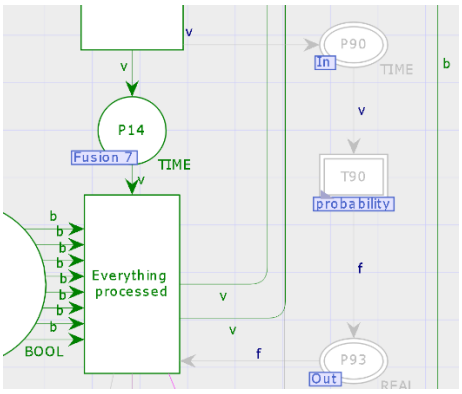

Figure 29: Implementation function probability

3.2.9 Checking the bits sent

This will be check by the system by just creating an instruction “if” in an arc, and as it is intended to know how many work and how many did not work out two arcs were created, it can be seen in the next figure 30.

Here the “if condition” creates a token in the place or not. It can be seen that if the variable f is lower or equal than 1.28 then a token is created in the place that counts the number of tokens that is properly received, otherwise it will remain empty.

It was created the opposite instruction for another place, so both correct and mistaken sent message can be count by the user when the simulation finishes. Or it can also be added a monitor to these place –a Mark Size monitor will be enough- and it could be check in the performance analysis generated by the program.

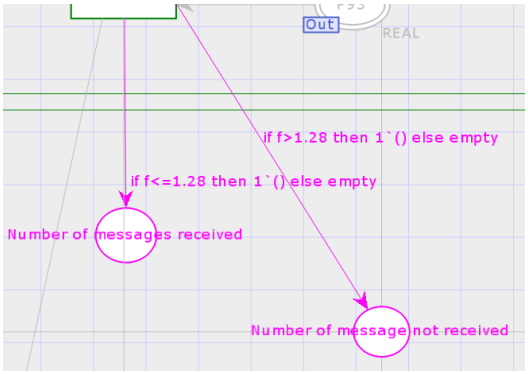

Figure 30: Checking sent messages

This 1.28 unit that divides the correct and incorrect tokens is the probability of 90% of being sent correctly.

3.2.10 Modification

In this example it was not created an instruction in case that the information is inadequately received, but this may be needed. If the user wants to implement an instruction that sends again the information then he should create an inhibitor arc for the transition “Everything processed”, so that it is not fired, he should also delete the tokens that remain in the place P16 -the place that goes just before the former transition- and make the hardware cluster that sent the information to repeat the operation again.

It is showed in the picture 31, where the places that give the order of sending again the information are the IN places, each one with the color of the cluster that will send the order. The place in pink would not allow the transition “Everything processed” to be fired and will destroy the tokens of P16.

The places IN will generate the information again in each hardware cluster, could be at the very first beginning or just for processing again, this would depend on the system the user is working with, but the order sent to the hardware will contain the time that has already been spent, because the token that is sent is a real that is timed, for example 1.0 if the cluster to repeat the action is the cluster 1.

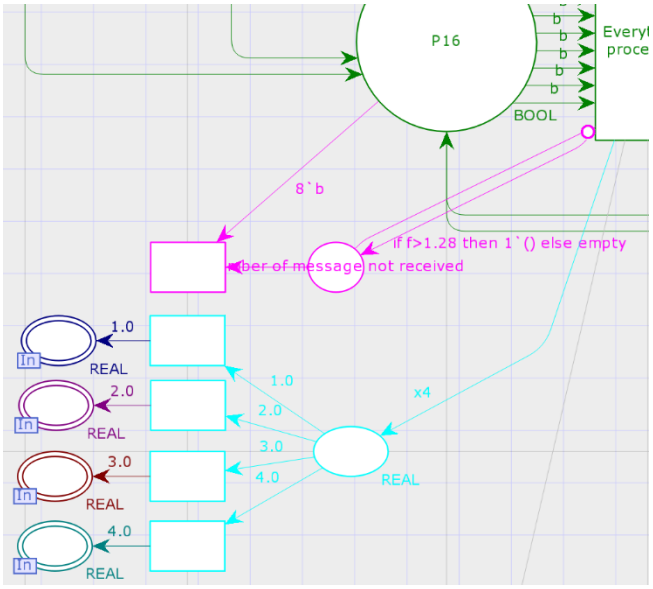

Figure31: Modification checking information

This figure 32 is exactly the same as the figure 30 with one slightly difference, the places are not UNIT places, they are REAL places. Thus, each token that these places save, each one of them will be saved with the time that fired the transition "Everything processed". This way the user, either looking at it or creating a monitor, will be able to know the exact time.

The figures 30 and 32 do not count the messages that are lost because of the alarm or the failure of the lines, but it can be easily implemented. If the user needs to know these details then he can create the places to check them or a monitor to receive the information in the reports.

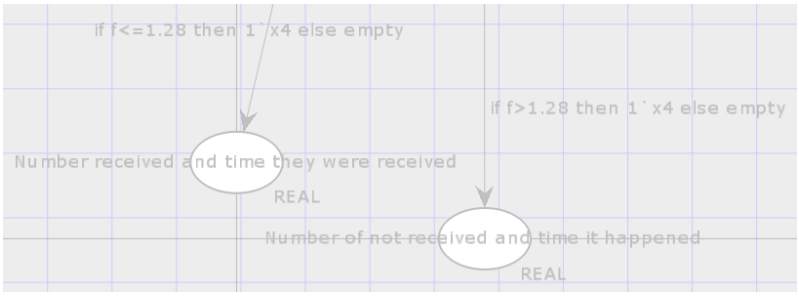

Figure 32: Checking sent messages model 2

3.3 Model 2

The goal of this model is to accurately simulate the behaviour of a communication system. The following model is a modification of the former one so that more processors can be simulated.

3.3.1 Suppositions

In this simulation the followings suppositions were adopted:

- The information is carried by bits-the bits are Boolean units-.
- There are four clusters of transitions and places that simulate sensors or units that creates information.
- The timed, coloured and stochastic extensions of the Petri nets language are used.
- It is possible to use as many processors as needed, but all processors must have the same features.
- Pages hardware 1, 2, 3 and 4 represent the creation and flow of the messages.
- Page probability represents the function probability.
- The main page shows the system and the relationships between pages.
- Each token has a number that shows where each tokens belongs:
    - Number 1: is a token of cluster hardware number 1.
    - Number 2: is a token of cluster hardware number 2.
    - Number 3: is a token of cluster hardware number 3.
    - Number 4: is a token of cluster hardware number 4.
    - Number 5: is a token that represents the processor.
    - Number 9: is a token that represents the flow of the states of the event-simulator called alarm.
    - Number 10: is a token that represent the flow of the states of the event-simulator called lines.
    - Number 11: is a token that enables the probability function.

This model will be useful for simulate a low-level communication protocol, although instead of bits the supposition could be bytes, Kbytes or bigger groups of bits. The model

presents the limitation that just processors with the same features can be used, and therefore it will be just useful for medium or small systems that possess this characteristic.

On the other hand this model is extremely easy to understand as it is highly visual and jus some used of the ML language has been implemented in the model. The list provides the user with a wider range of tools to study the behaviour of the model under different circumstances.

This model would be recommended to simulate a real-time communication system or a similar system with the same principles-like a producer-consumer system -. The only condition the processors must have is that they must share the same features.

3.3.2 Modification to the former model if more are to be used for the analysis of the information can be held.

The following modification can be done to the model showed before if monitors are to be used for analysing the information. The difference between using monitors or not was chosen based on the facility to understand the flow of the information and the easy way to comprehend this information when it is deliver by the simulation.

But if the user prefers the computer to analyses it and suggests one protocol policy, then the user can utilize the modification that it showed below.

This latter model also allows the use of more than one processor, but the processors must have the same characteristics. This is due to the fact that the addition of time is always the same independently of the processor that is used. The CPN Tools just allows adding one option of time for the same action, independently of its components or characteristics. If the user needs to use different processors or make the flow of the information depending on the components of the model then the last Petri net model showed latter would be suitable for this goal.

Here it is going to be described the modification and the changes that have been done in the new simulation model.

3.3.3 Main system

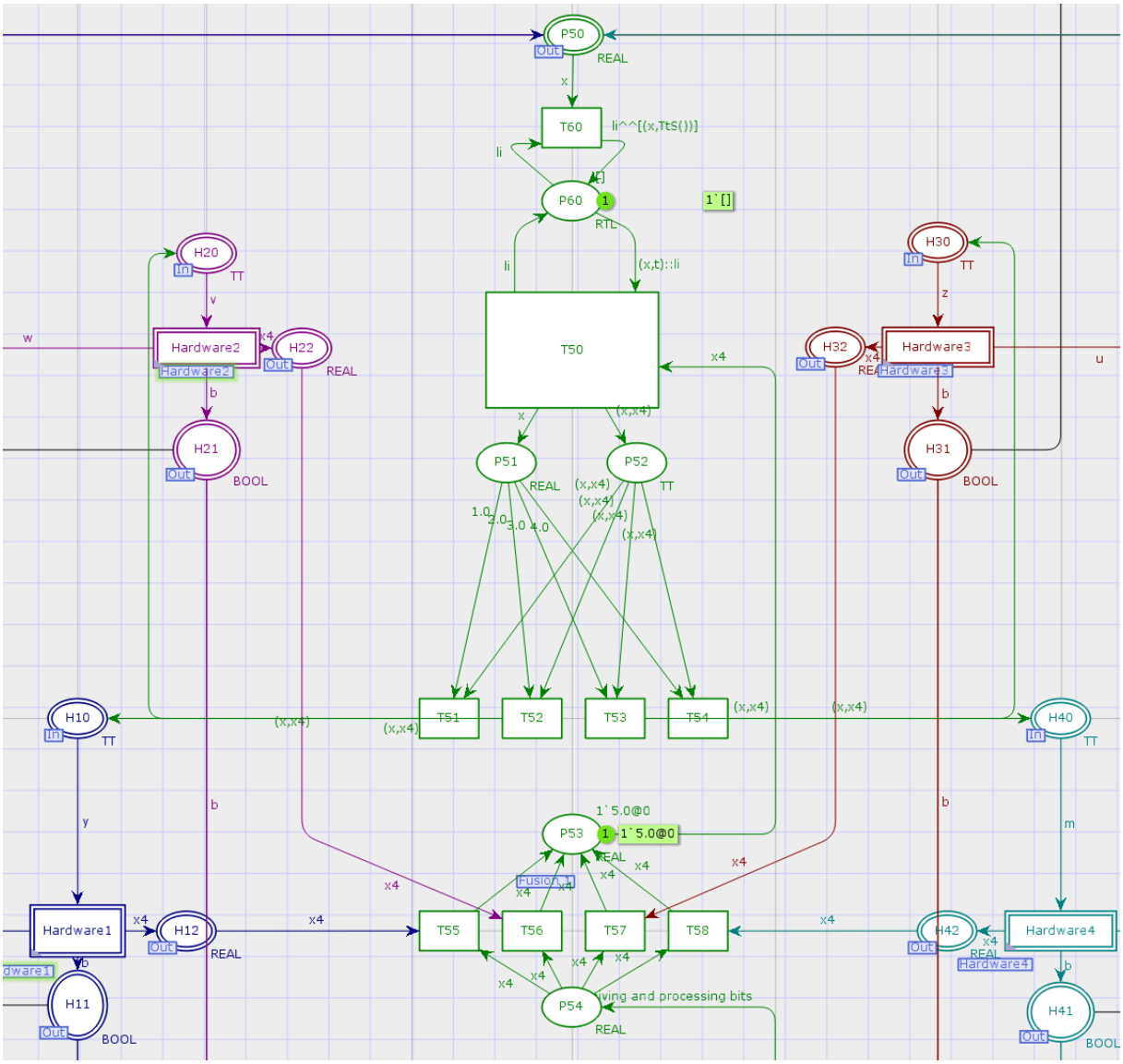

Figure 33: Model 2

As it can be seen in the figure 33 the clusters of places and transitions called hardware have been reduced because its former functions are now done by the green cluster called “System”. The main process is the same, the flow of information is inside every hardware cluster and when the information is codify and ready to be processed a token is sent to the place P50. As no changes have been made to these clusters called hardware then they are not showed here, there were showed in the figure 13. There a slight modification that will be presented latter.

In the second model it has been created a place that can hold a list, this is P60. This is important to the user because it will be attached a monitor, called “List Length Data Collector” that will analyse the information that is waiting for a processor to be free.

This tool will be useful because it will provide us with information like average time of waiting, average number of tokens waiting, number of tokens that have passed through it, maximum and minimum number of tokens in the place at the same time, number of times a transition has been fire-in case it is a transition- and it can be find more advanced information, if the user needs it.

For analysing it will be utilized just the characteristics mention before, and, in some cases, it will also be utilized the list of all the transitions that take place and some statistics that are created with the simulation, like the 90%,95% and 99% Half Length.

The list is made with the transition T60, in here the transition adds to the list the two values that are needed for the list. It was defined before the list as a product of real and string, so this is what was added there: it was added a real, x, and a string, that it is created by the function TtS () –converts the time to a string-.

The list is the place where the queue will be made, so it will also define which communication protocol will be applied: like FIFO-First In First Out- or LIFO-First In Last Out-. The specific constructions of these models are showed in the figures 34 and 35.

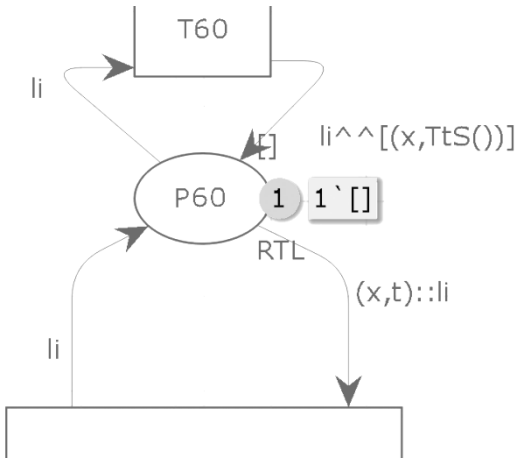

Figure 34: FIFO configuration

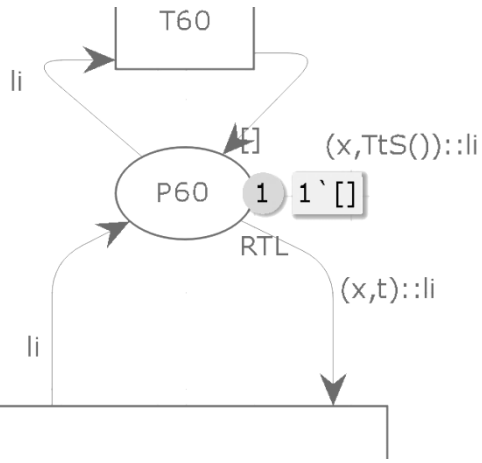

Figure 35: LIFO configuration

The place initially contents an empty list,
and the arc adds each new component at the bottom of the list, and the arc that takes them out of the list takes the first one: this is, first-in first-out -FIFO- the protocol used in the list. Other protocols used with these models will be explained in detail when the behaviour of every model is studied.

The place P60 will receive all the tokens when the information is ready, and therefore, if there is not one processor for each system that sends information, this will be the place when the messages will remain until one processor is free and can be used with the next information token.

A message will leave the list, or the queue, when one processor is free and it can operate with the message. The processors in this model are represented by the real number 5.0.

The next step is to attach the processor with the message: this could have been done with another list, but a combination of transitions and places was needed to choose the path that follows each message. What it is done here is differentiate between the times it takes to process the messages, independently of the processors, as it was explained before.

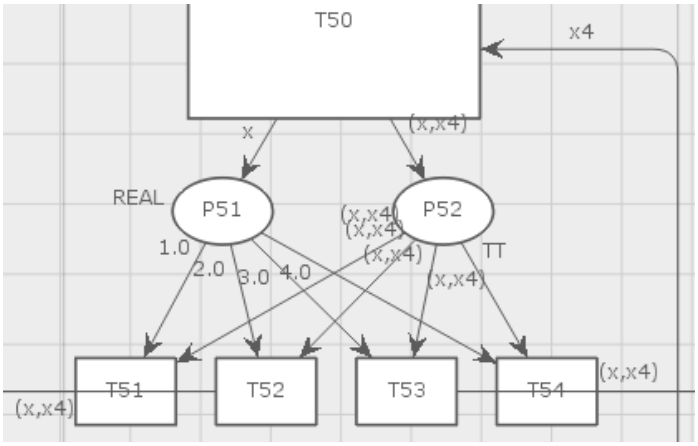

Figure 36: Detail selection

Therefore, in figure 36 it can observed that T50 leads to the places P51 and P52, and these two places are followed by four transitions, and this six elements leads towards the correspondent system that have to go every message with the processor. One example could be a message coming from the cluster called “Hardware1”, then the place P51 will have a token with the number 1.0, which will just enable the transition T51, and this transition will lead the token 1.0 from P51 and the token that is contained in the place P52 towards the cluster called “Hardware1”.

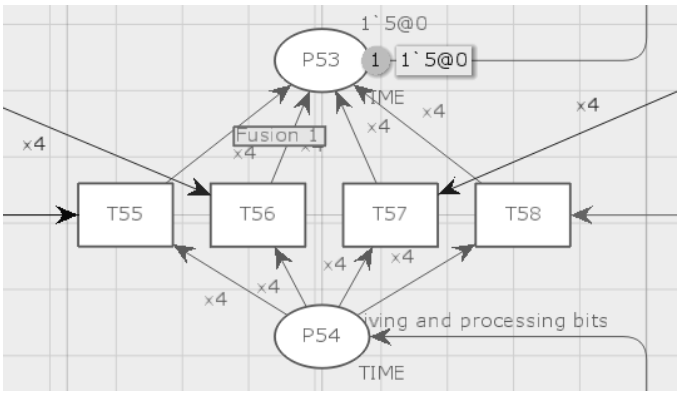

Figure 37: Detail processors

Looking now at the figure 37 it will be seen how the processor are taken back from the message they were processing and the processor will be set free.

Every time a message finishes the processor is set free and it is able to take the next message. Between the different ways of simulate this condition that it has been chosen the following one:

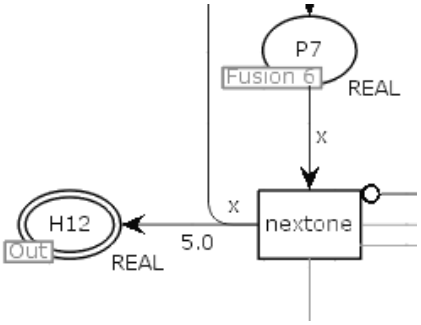

Figure 38: Modification hardware setting a processor free

At its end, each hardware cluster creates a real number 5.0 that represents a processor. This processor is placed in one of the places needed for the transitions in the figure 37 for generating a new processor; the other condition is generated when all the bits have been processed.

Therefore, a token processor is created and the system can keep working as in its first step.

3.3.4 Modification to the second model

If it is going to be used another communication protocol, for example if it is intended to create priorities one messages over others, then it can be programmed using ML language if the user has experience with it, or it can also be done with a combination of places and transitions, which is the modification that will be depicted because it is easier to understand for a non- experienced ML programmer, and ML language is implemented in the third created model.

This four transitions and one place are enough to generate priorities for the messages, as it can be seen in the picture 39, there is a number in the lower-left side of the four transitions –between 1000 and 1003 all of them- and this priorities show that the "tmes1" is the transition that should go before any of the others, as it has a lower number: it is as a position, the first position is fired before the second position.

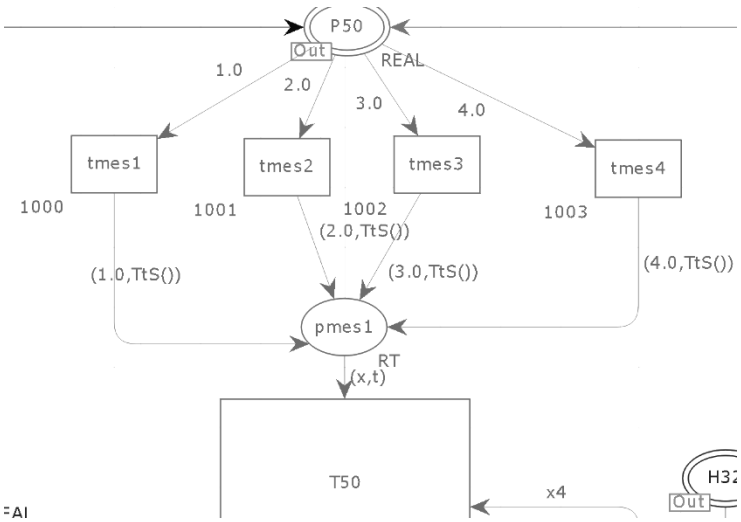

Figure 39: Priorities

After the priority is given to the messages they will go to the place “pmes1” where they will remain until a processor is free and can take one of them.

Another way of create a stronger priority would be to make every one of this transitions to add a time to each message, so, if it is wished that the message 1 has priority above the others, then it can be made the transition “tmes1” to add less time to the message 1 than the others transitions would add to the other messages.

In order not to fake the model the time that it is added to every message should be taken from a latter transition that adds time: for example, if it is added one unit of time to the message 1, then it should be reduced the time that is generated in the cluster “Hardware1” in one unit of time. This way the system will always processed before the messages that was decided that has priority over the others, just by using this detail.

Another modification that could be made to the model could be choosing one processor for one message, for example, if it is said that the message 1 should only be used with the processor 5 then it can be written this condition in the upper-left side of the transitions depicted in the picture 40. It can be seen in the transition T51.

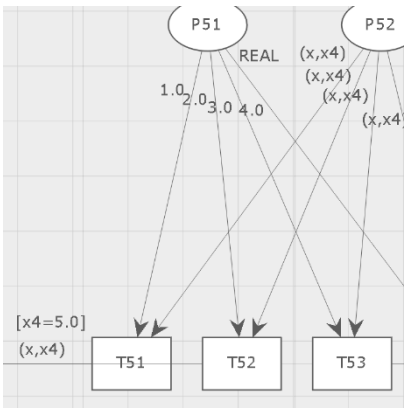

Figure 40: Detail selection

The former modifications were made to the first model in order to use the monitors to analyse the information provided by the simulations. Another advantage of this modified model is that more than one processor can be used, although all the processor have the same characteristics.

As not changes have been made to the other clusters of places and transitions the pictures of these have not been offered in this description of the model.

3.4 Model 3

The goal of this model is to accurately simulate the behaviour of a communication system.

3.4.1 Suppositions

In this simulation the followings suppositions were adopted:

- The information is carried by packages of bits-the packages of bits are Boolean units-
- The information arrives randomly to the system; there is no need to simulate sensors.
- The timed, coloured and stochastic extensions of the Petri nets language are used.
- It is possible to use as many processors as needed, and all processor can have different features.
- Each kind of information processed by each processor generates different times of process.
- There are three pages: Main system, Message flow and Probability.
- Each token has a number or a string that indicates where each tokens belongs:
    - String mes1: is a token that simulates the information1.
    - String mes2: is a token that simulates the information2.
    - String mes3: is a token that simulates the information3.
    - String mes4: is a token that simulates the information4.
    - String proc1: is a token that simulates the processor 1.
    - String proc2: is a token that simulates the processor 2.
    - String proc3: is a token that simulates the processor 3.
    - Number 9: is a token that represents the flow of the states of the event-simulator called alarm.
    - Number 10: is a token that represent the flow of the states of the event-simulator called lines.
    - Number 11: is a token that enables the probability function.

This model will be useful for simulating a low-level communication protocol, although instead of bits the supposition could be bytes, Kbytes or bigger groups of bits. The model

ML language has been implemented in the model and it is necessary to read it to understand the behaviour of the system. In fact, in this third model, although the Petri nets language is used, the whole model was created thinking how to implement ML language so that the Petri nets language would not be so widely used.

Also both lists provide the user with a wider range of tools to study the behaviour of the model under different circumstances.

This model would be recommended to simulate a real-time communication system, or a similar system, with the same principles - like a producer-consumer system -.

3.4.2 Third model

In the third model that is generated here it is possible to create different processors and different messages and add a time to every message depending on the message that was generated and the processor that is utilized for working with the information. Thus, this model is more capable than the former ones, but therefore it has a slight disadvantage that will be described in detail in the description of the model.

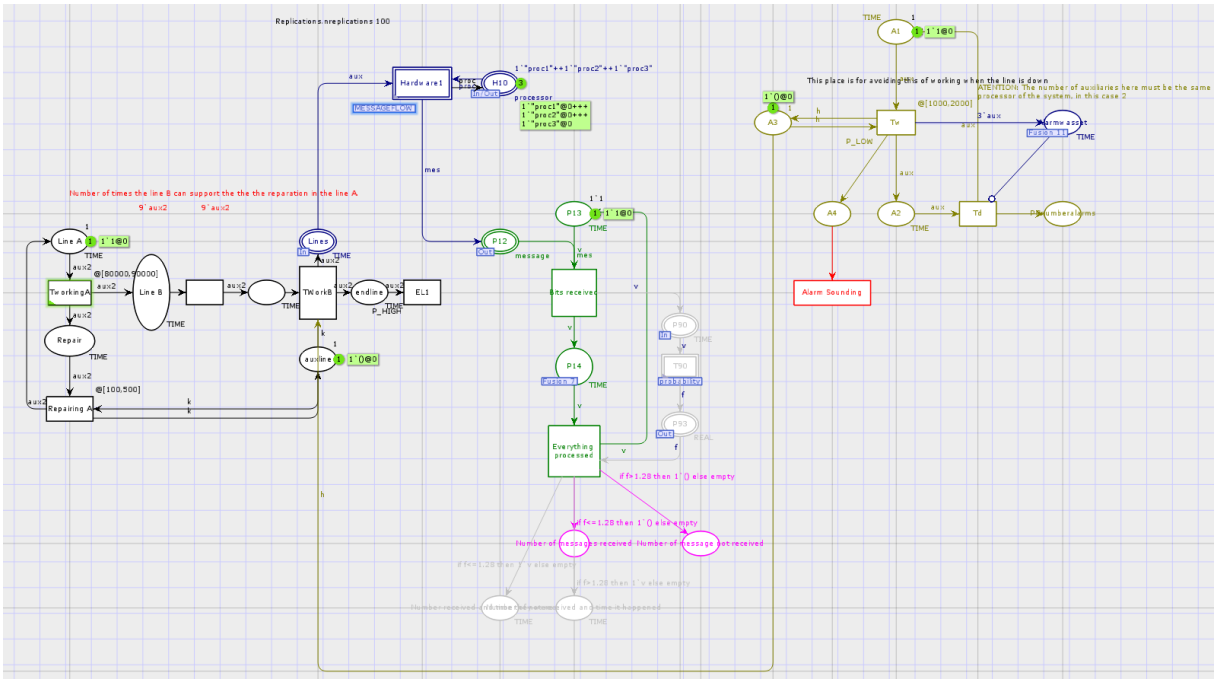

Figure 41: Model 3

3.4.3 Main system

In the third model there are more functions and more ML language than in the previous ones -making the model more capable and more complex- and not so many places and transitions are needed as the functions and the language make its previous tasks.

As it can be seen in the picture 41, the biggest change in the system have been the clusters “Hardware”, when there were four before there is one now, and this one is even smaller in this page. There are three places connected to the transition MESSAGE FLOW: one is the place H10, although for this place there is no need to stay in this page, as it can be seen that all its tokens go in and out of the transition “Hardware1”, but this way it can be seen from this page the number of processor that are being used in the model.

The other places related with the main transition are P12 that leads the message towards the transitions “Bits received” and “Everything processed” as it was done before with the preceding models and exactly for the same purpose: It is intended to count the number of messages that have been sent by just looking at one place, in this case the place in light grey and pink after “Everything processed”.

The other place called lines has the same goal as in the former models: it will stop the flow of the messages when the lines are down.

The other clusters that are seen in the system page are the ones that simulates the use of the lines and the one that simulates the use of the alarms. Regarding the first one, they are exactly the same as the clusters that were used in the earlier models with two slight differences: As just one “Hardware” cluster is used, then it is just needed one place to stop the only “Hardware” cluster, in this case the place called Lines.

It can also be seen that there are one transition and one place more between the place Line B and the transition T Work B, but before explaining its meaning it should be remembered how this worked briefly: when the line A is down the line B starts working until the line A can work again, and B just can work an specific number of times-it was written 3 times in the preceding models-. If it is written 3`aux2 in the arc that connects “Line B ” with the blank transition and it is also written 3`aux2 between the blank place and the transition “T Work B” then the simulator of the lines would need 9 times of the line A to be down to stop working.

Following the description of the system, it can be seen that the alarm cluster has also change, the main change is the blue place called “Alarm was set” and the attention should be paid to the arc that generates the tokens to this place, the number of tokens generated must be equal to the number of processors that are used in the whole system: this place will delete all the messages that will be processing information at the time that the alarm sounds, so, if the number of processor is changed to study a different communication protocol or just to study a more economic system, then this number must be always changed.

The other places and transition in the alarm cluster have the same purpose as in the previous models but one place. The place called A3 was not in the other representations, this place is here as a check place, this place will allow the alarm to be fired until the lines are down. When the lines are down, the transition that stops the system –T Work B- will take the token of the place A3 and, therefore, the alarm will not be able to be fire again and the system will not have any enabled transitions.

3.4.4 Message flow:

The message flow is completely different than the messages flows of the previous representations- what was called “Hardware clusters”-. In the third model there is just one cluster called message flow because only one is needed.

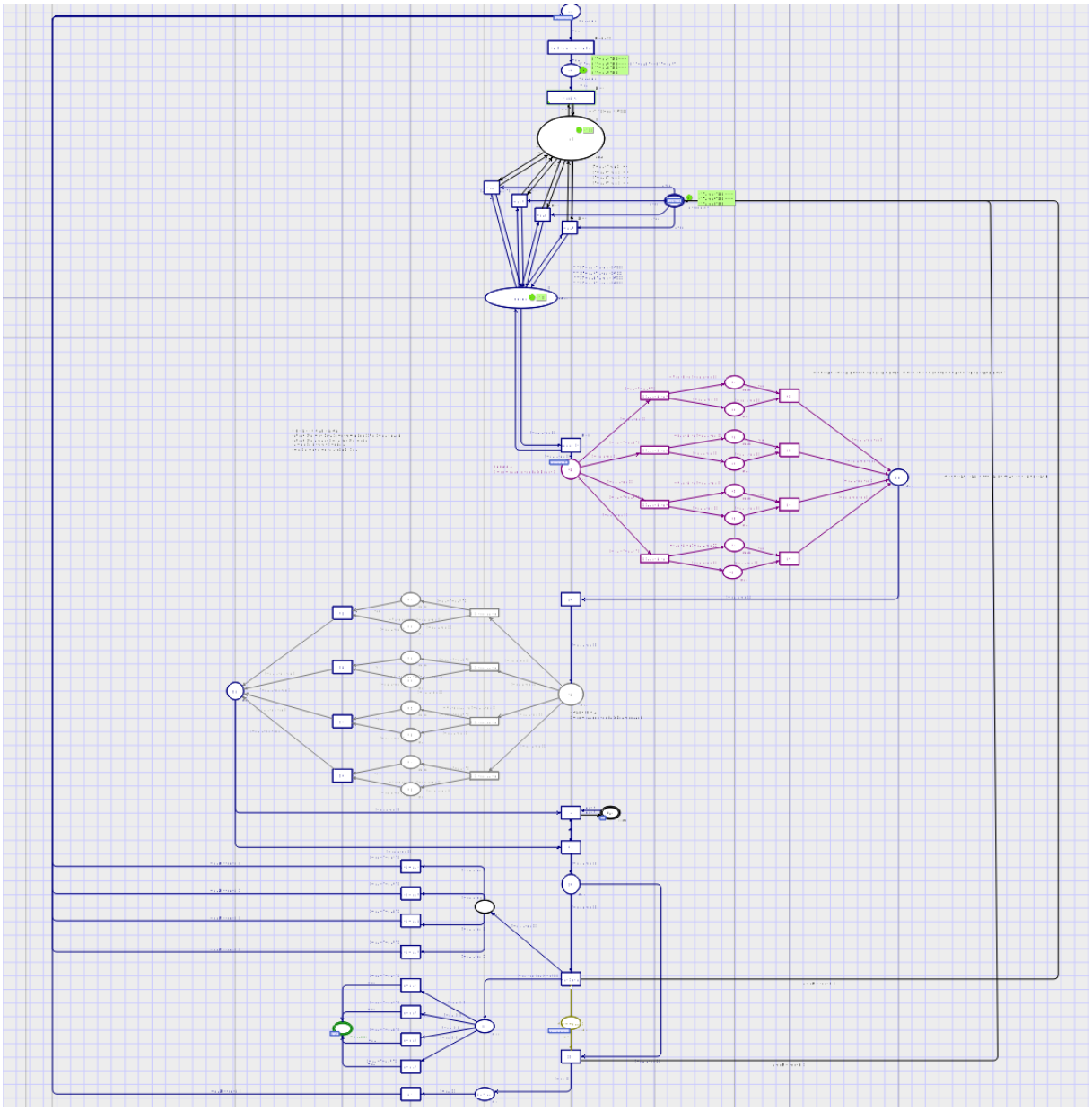

Figure 42: Message flow

The picture 42 that is showed above represents the whole flow of the message, but every cluster of places and transitions will be explained.

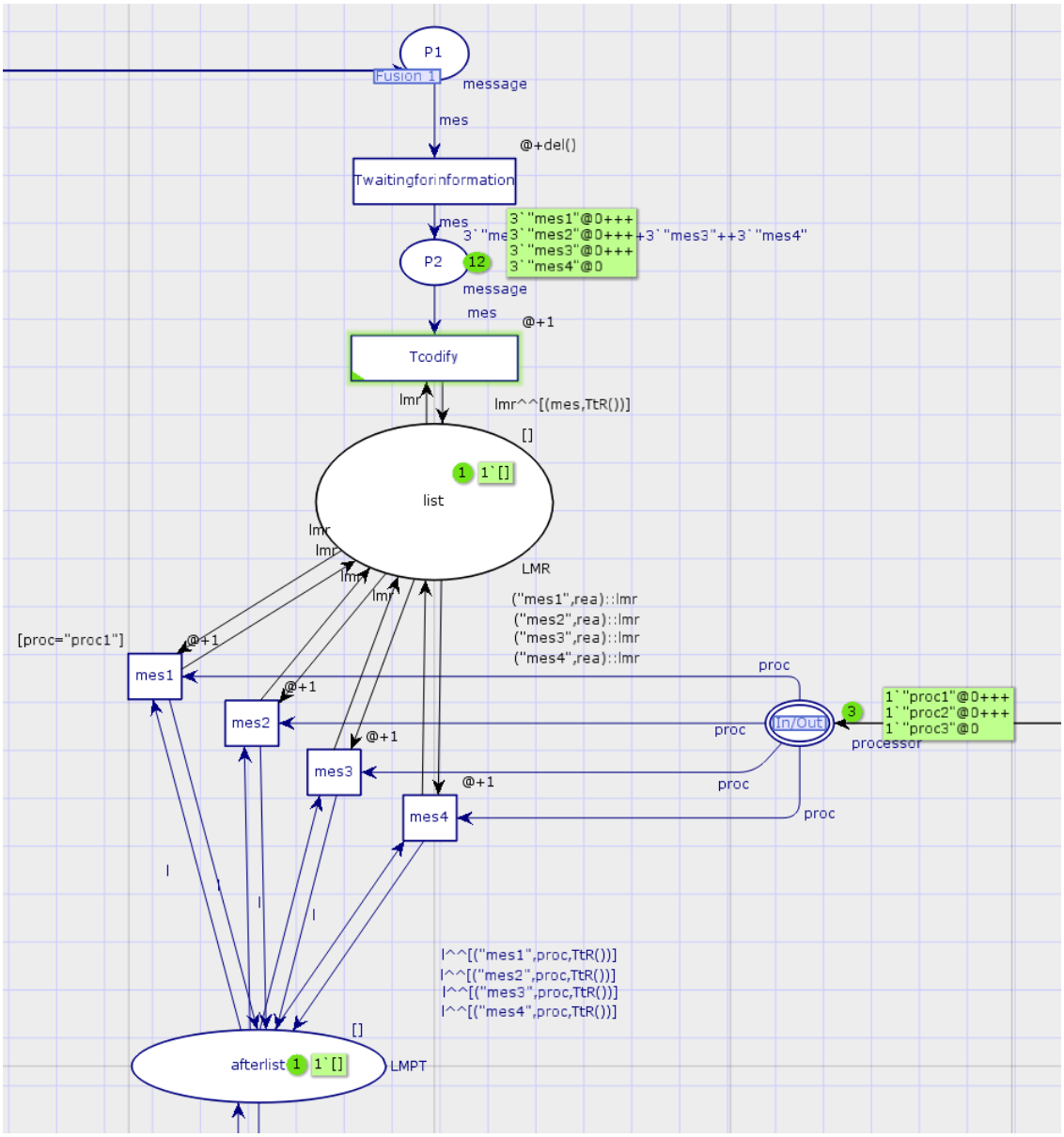

Figure 43: Queue

In this picture 43 can be seen the beginning of the flow of the information. The messages are initially in P2, it is supposed that the sensors of the real system have already the information and they are waiting for the system that process the information to be turn on. Thus there are 12 messages in the places P2 because the information has already been generated. If the information will be generated then the message could start in the place P1 and they will pass to the place P2 after the transition “T waiting for information” adds to each message a random time –depending on the time that the messages need to be generated- through the function del(). Every time that a message is processed will be lead to the place P1 and this place would choose randomly the time that last in being generated again.

Briefly it will be explain the function del (), the function del () uses a function inside itself that choses a random number of a color set-the function is colorset.ran () and the colorset is the kind of unit it is being used: could be an integer, a real number, a Boolean, -.

The colorset delay is a colorset defined by ourselves and it is an integer between 1 and 6.

After this explanation it should be paid attention to the next step: when the messages passes the transition “T codify” they would join the list that have been created in the place called “list”. As in the previous models, this is one of the most important places:

It is important to remind here that a real-time system are being studied, and most of these systems have limited resources –in this case limited processors- and the queues are formed because of the limitation in the resources. So, if some characteristics are changed, like the communication protocol or the time needed by the processors in order to try to improve the system, then it should be studied how the queue changes with every one of the changes that is made: whether the queue is larger or shorter, how long in average the tokens are kept waiting till one shared resource can be associated to one of them, which are the maximum number of tokens in the queue,…

At the same time that the tokens passed to the list, the list also attached to each token the time when they have passed to the list in the variable rea, which is a real number. Therefore the list is composed of the kind of message sent and a real number for each message.

After this list the next step is one out of the four transitions that are available to the place list. If it is observed the right side of the picture it will be noticed that the expressions of the arcs that connect the transitions with the place only allows one kind of message: the first one allows message 1, the second one allows message 2, etc… in order to create one transition for each message between the place “list” and the place “after list”.

This was made because it allows the user to generate a communication protocol, for instance, as it can be seen in the upper-right side of the transition “mes1”, this message just can be processed by the processor 1, as it is showed in the picture 43 .The place that holds the tags In/Out is the place where the processor remain until one message requires them in order to be processed.

After the messages pass through these transition the processor is attached to the message and they go to the place “after list” that was also declared as a list, but it is not necessary that this place is a list.

3.4.5 Sending cluster

Then the tokens arrive to the first big cluster of transitions and places. It is important to remember that this third model was created because the user may have different processor with different characteristics and therefore he might need to add different units of times depending on the kind of message and the processor that are used. This purple cluster is called “sending” because it will be the one that adds the units of time to the message because of the time that it last to send the information. In other words, depending on each messages and each processor assigned, the time that the system needs to send the information varies and, therefore, this cluster will add the proper time span.

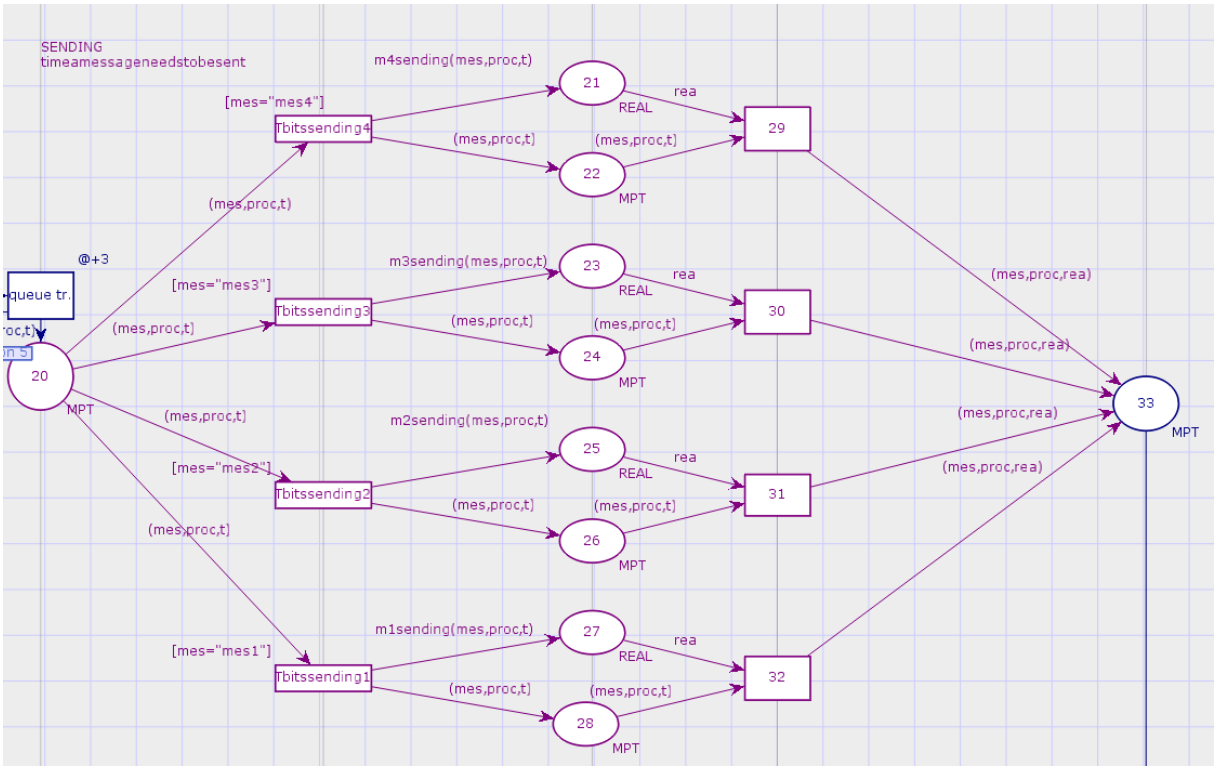

Figure 44: Cluster sending

Two things are need to be remarked here:

- It is quite likely that the time that the message needs to be sent depends more on the kind of connection that links the sensors and the processors than the kind of processor itself, this characteristics can easily be implemented attaching to the message another information like the kind of cable that is used and making another function that will add time depending on this new feature.

-A question that could be asked here could be why the information is generated and the processor is attached to the message, and after that, the time is added to the message. It seems more logical that the sending time should be added before it is attached to the processor.

If it is considered -like in this case- that the processor is a feature that affects the time, then it is needed to attach the processor before. But, even if this is not an important parameter, it could be true that the sending of the information could happen before the processor, but, if it is added the time in the simulation after the attachment of the processor, it can be seen that in the system, between the moment that the information is generated and the moment that the message has reached the appropriate processor, no changes are made and therefore it can be used this addition of time.

In the purple cluster “sending” it can be seen that each message is isolated using the condition in the upper-left side of the transitions. It could also be differentiated by processors instead of messages, or any other component that would be useful to add to this system.

As it has already been differentiated by messages it is needed to add the time now depending on the processor that is used for each one of the messages already split. Therefore it can be written a function in the arc -right after the separation of the messages- that will add a specific time for a specific feature of message and processor.

For this purpose it was chosen case functions, as they are the more suitable ones for choosing between different actions. In this case it would distinguish between the different times that the different processor will last for each message. One example of these functions can be seen in the picture 45.

```
fun m1sending(mes,proc,t)=
  case proc of
  proc1=>t+1.2 |
  proc2=>t+1.3 |
  proc3=>t+1.4;
```

Figure 45: Function example

This function is written in the arc that connects the transition that only admits the message 1, then this function will add to the real variable “t” the time that each processor spends receiving this message: 1.2 units of time for the processor 1,…etc.

As it was said before, it can be chosen different components in which the user can base the different time that is added to the process –instead of processor for the action of sending it could be used the kind or the quality or the kind of connection between the sensors and the processors -.

But, if two or more components will be utilized for adding specific times, then the user would have to decide how to split the different components in order to add the specific time. One option would be to create a nested function for each component, for example, in the function that was used before for splitting the processors. For this purpose the user could create another case function inside each case of processor, this time splitting for messages -or any other component –and after this, it would be a function that first splits the

processors, and afterwards, splits by messages and adds an specific time to each combination of components.

This way is more easy than the one used in this model, and less time is consumed in create the proper model, but, as the Petri Nets was conceived as a graphical tool it was decided that the model would be created following this attitude of working.

If the user prefers to make the model more visual and it is desirable to check the model when the simulation is running, it might be preferable to split the components of the model using the basic tools of the Petri Nets: places and transitions. If there are two or more components that are needed to split in the model, then the user could split each component using the condition of the transitions that limited the components that are able to fire each transition-the upper-right side that can be seen in the picture 46 -.

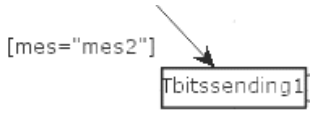

Figure 46: Detail constraints

Then, this would be the first component that would be split-in this example the message- and it would be needed a transition for each kind of component. In this example there are four kind of message so four transitions have been created.

The next step would be to split again depending on the next component. The user should again create a transition for each kind of component – in this case three transitions would be created as three processors are being used -. The user should keep splitting this way until there are no more components. And in the end of this process the final arc would be the one that would add the specific time for the specific combination of components.

Some observations should be made regarding this approach of splitting. The first thing is that, for a small number of components, the cluster of transitions and places grows incredibly fast. The first time that the system splits the components used a transition that is needed for each kind of one of them - four kind of messages, four transitions needed - and, in the next step, each one of this transitions should be connected with the same number of transitions as kinds of components there are - in this case, as there are three processors then each one of the former transitions should be connected to three transitions-. This makes the system very big for just for a reduced number of components.

Therefore, a mixed solution would be recommended here, like the one implemented in the third model, using both functions and transition to split the components, the functions in order not to make the model really big and the transitions to be able to check the system when the simulation is running by just looking at the program.

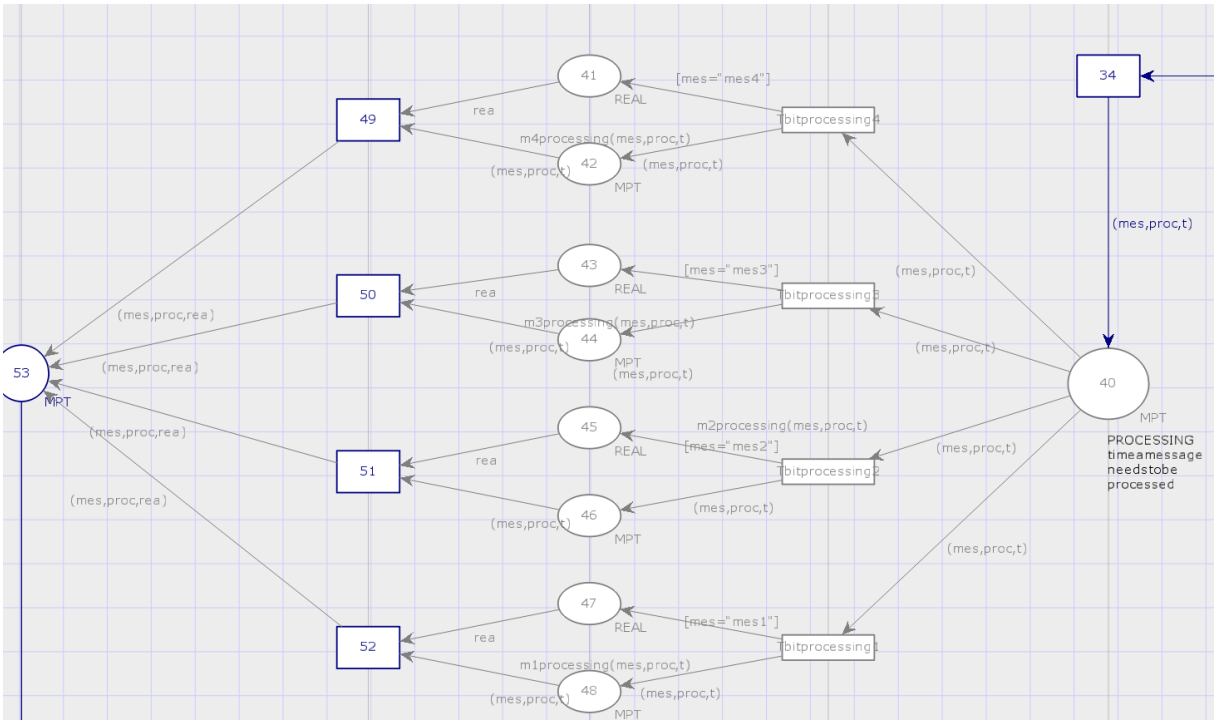

Figure 47: Cluster processing

At the end, either by functions, transitions or a mixed solution, there will be always specific combinations of elements with its specific addition of time.

3.4.6 Processing cluster

After the sending time is added to this message, it will arrive to the next grey cluster, the “processing” cluster. This cluster has the same number of places and transitions as the sending cluster, as it makes the same function as the previous one: it splits the message in order to add a specific time for each specific combination of components.

It does exactly the same as the cluster before him, but in this case the functions are different because the processing time is different of the sending time.

An example of these functions can be found in the picture 48.

```
fun m4processing(mes,proc,t)=
  case proc of
  proc1=>t+0.25 |
  proc2=>t+0.37|
  proc3=>t+1.33;
```

Figure 48: Function processing

After the times of sending and processing have been added to the message that is all time that is used for the communication system, and at this point, the message should either disappear or go back to the initial state where it could be used again when more information is ready to be sent.

3.4.6 Modifications

But, there are some modifications that should be included in the model so that it can be analysed all the available information. What's more, it would be also needed a protocol for the events that could happen, like when the alarm is fired or the lines that connect the sensors and the processors are down.

3.4.6.1 Lines down

The first modification that will be explained would be the situation when the lines are down. When the line A has been down several times –this number is written in the arc that connects "Line B" with "T work b", it was already explained in the alarm cluster of the first model: the way this event works is the same as it was in the first model-. So, when the line A has been down a number of times chosen by the user, and therefore the line B has worked that number of times and at the last time the line B breaks down, then the whole system has to stop.

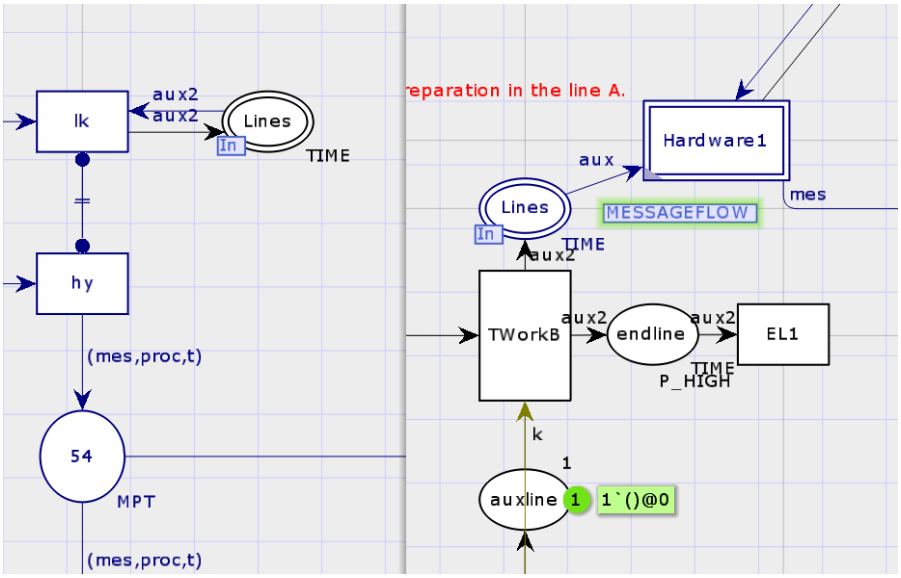

Figure 49: Detail relationships pages

The system simulate this in the following way: the transition “T Work B” is the transition that creates a token that will be introduced in the page “Message flow” and that will delete all the messages that are being processed at that moment.

At the picture it can be seen that on the right side is found group of elements of the page “system” and on the left side a group of elements of the page “Message flow”. The place Lines in both pages is the same, it is the connection between the pages.

What this place does -when the lines are down and the transition “T Work B” has created a token in the place “lines”- is to delete all the messages that come to it. The place “lk” and “hy” are fired by the same previous place-place 53-, and these two transitions are connected by a constraint arc that avoids that both transitions can be fired at the same time. So, when one transition is ready to be fired the other one is unable. This means that the transition “hy” will be always ready, until the transition “lk” is available. The transition “lk” will be available when the place “Lines” has its token in it.

The place “Lines” works as a check place; this means that its function is to make the transition available when the conditions are met.

The transition “hy” is the transition that would follow the messages when the system is working. This transition leads to the place 54, and these place will lead the messages to two transitions, depending on whether or not the event “alarm” is fired or not.

The picture 49 presents the relationship between the places and transitions for this case.

3.4.6.2 Alarm modification

How the alarm is fired was already explained in the page “system”, and it is also the same behaviour as the alarm in the previous models, with the slightly difference that the number of processor that you use in the whole model should be the number that must be write in the arc that links the transition “Tw” with the place “Alarm was set”. –For a better understanding go over the picture 20 and 23-.

The place “Alarm was set” is also a check place, this place will lead the message to the right transition depending on whether or not the alarm is fired. If the alarm is not sounding and everything is working the message will go to the transition “next one” and this transition will generate two tokens that will go to the places 55 and 56 that will be depicted latter.

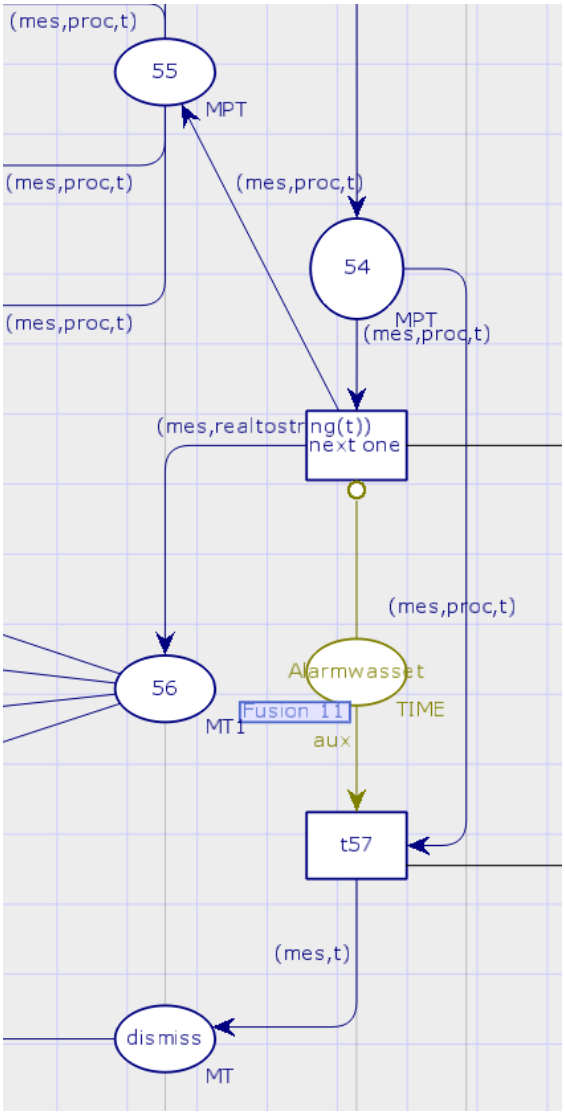

Figure 50: Detail message flow

If the alarm was set up there will be the same number of tokens in the place “alarm was set” as processors are in the whole system. This is because it was considered that the three processors are working at the time that the alarm is set, because the time that the processors are not working are much smaller than the spans of time when they are working.

Otherwise, if more precision is needed, what the user would need for a more accurate approximation would be three ways, instead of one way, one for each processor. If this option is chosen it would be easy to determine when the processor is working or not, but the system will grow much bigger, at least three times bigger.

Either way, both the transitions “next one” and “T57” have a black arc that sets free the processors by giving them back to the initial place of the processor-place In/Out.

If the alarm is fired then the transition “T57” will lead the messages to the place “dismiss” and this place will send them to the initial state, where they will be kept waiting until the time comes that they are generated again by the sensors-go over picture 51-.

The places 55 and 56 could have been created in just one of them, but it was chosen to segregate because they will be utilized for accomplishing different goals.

The place 55 also separates the information sent by messages, although it was not necessary, it was made to make the system more visual.

Every one of these transitions will lead the tokens “message” to the initial step -- go over picture 42 - and the arcs that connect the transitions with the place “P1” will add the time that has been saved in the real variable “t” to the color set “message”, that it is of the kind string rename as a message.

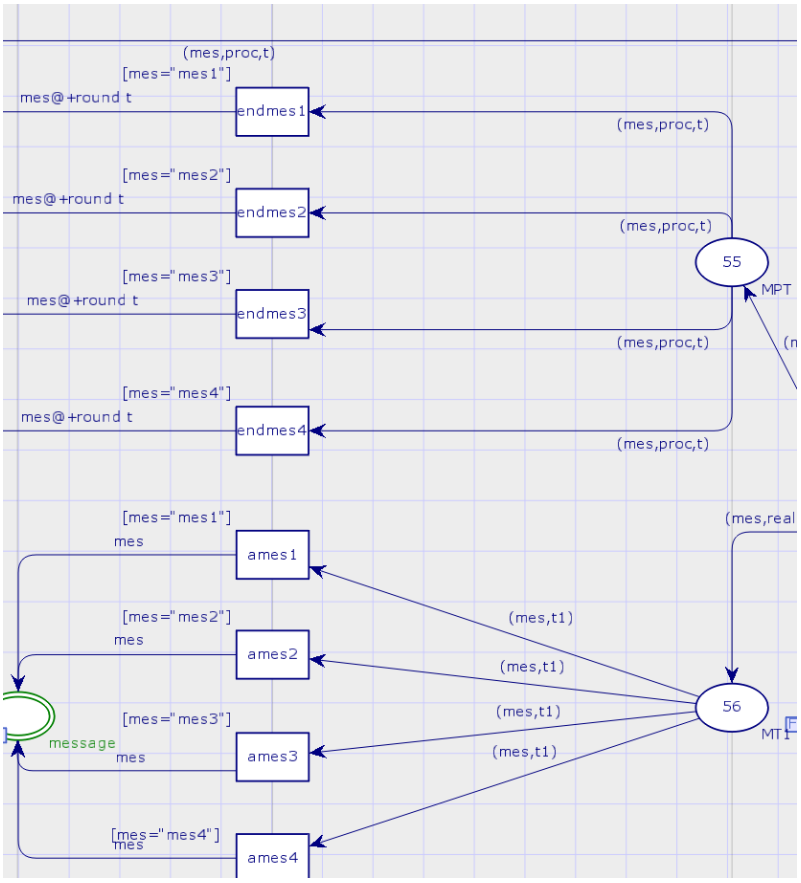

Figure 51: Analysis messages

In the transitions that follow the place 56 is necessary to differentiate for each message. A suitable monitor will be attached to every one of these transitions in order to allow the computer analyse the information that the model generates.

It will be attached a monitor that will count the number of times each transition is fired, so that it will be known which message is faster and which one is slower with the characteristics that are set in the system.

Afterwards, the messages will go to the place that has no name in the picture 51–this is another place that connects the different pages of the net- and this place will lead the messages to the green cluster. It can be seen with detail how it is related with the main system in the picture 41.

3.4.7 Time when using more than one variable to select the specific time

As it has been seen in the description of the model, the addition of time it is not added in each transition –representing the duration of an action- or in each arc-representing the duration that last sending information-.

If it is chosen to use just the Petri nets graphical model without using any ML programming language the time could be add as in the former models in the transitions and arcs and the final reports would be as accurate as the ones in the former models.

But, on the other hand, if it is chosen to use ML programming language not to make the model so big in order not to use so many transitions and places, then each transition and arc can have a function that defines with option represents the messages that has arrived and it will add the specific time needed. For example, if there is just a transition for the cluster “processing” it will be needed a function in this transition that could decide how much time add to the message depending on the kind of processor and the kind of message that has arrived. This is, 4.25 units of times for mes1 and proc2, for example. And then the system will go to the next step.

If there are more than one variable the function needs to have nested functions, maybe “if” functions or “case” functions, and if there are a lot of variables then it is possible that everything is decided in a big function with lots of nested functions inside it.

The problem found here is that CPN tools would not allow nested functions, or at least not the nested functions that were introduced, so it was needed another solution. When the CPN tools becomes open-source programme, or an experienced ML programmer is required for creating a proper nested ML function, then this model would be more powerful and relevant by just adding this detail.

In order to add the time without using nested function it has been created a real variable that is attached with the message and the processor and every time is added in this variable. It is added in the functions m_i processing and m_i sending in order to make it easier for the user to change the characteristics in case different combinations of elements are been tested.

The problem using this configuration for adding time is that, for the program, the time does not pass, because for the program the time is represented by the number that follows the @, the time that it is not being modified because nested functions are not being used.

This implies that, in this case, the time will be just in four steps –T waiting for information, T codify, the transitions that are after the list and the arc that adds the time located in the variable “t” -. The time that is added in the clusters processing and sending does not count for the program, it will be all added at the end, in the arc that adds the variable “t”. This indicates that the clusters sending and processing for one message will be process by the program as if they did not last any unit time, because this time is added to the “computer time” –the time showed with the @-.

Therefore, although the cluster processing will be executed right after the cluster sending for each message, it might be possible that after the sending cluster, there is another message that should be process before in the sending cluster than the first message is processed in the processing cluster.

An example, if mes1 is available at 3 and mes2 is available at 4 units of time, and after the sending cluster it is added to the variable “t” 3 units of times: as this time is added to the real variable “t” and not to the program time –the real number that follows the @- then, for the program, the mes1 would still have 3 units of time instead of 6. Therefore the mes1 will be executed in the cluster processing before the mes2 is executed in the cluster sending.

This detail is to remain in the mind of the user when observing the execution of the program.

This is also the reason why it would be more accurate and relevant the use of nested functions that this solution, although regarding the reports and the simulation of the system this features does not affect the accurateness of the simulation.

4. DISCUSSION AND IMPLICATION OF THE RESULTS

All the models have monitors that analyses the information and present it to the user so that further improvements can be done in the models.

For each model three alternative communication protocols have been executed, and each one of them have been simulated three times: the Petri nets systems are a graphical modelling tool that represent nondeterministic systems, more than one running simulation is needed.

These are presented to study briefly in the cases exhibited here so that the information can be analysed.

In case the programmer decides use this tool, it should be borne in mind that more than three simulations should be run and that the number of steps should be modified. This will provide a wider range of information: the reliability of the performance of a system cannot needs more than three simulations.

This case it is aimed to show that the tools that have been created are able and have the potential to analyse and improve the performance of real-time system; in this case with this goal three simulations will be enough if they show relevant information.

Overflow

CPN tools has a determined data space to save the information that represent the time for each token. This means that it is possible to overflow this data if enough steps are executed.

4.1 Model 1

The monitors of the model one are just in the final transition of each hardware cluster that shows how many times the transition has been fired; this is, how many messages have been processed.

4.1.1 Simulation 1 model 1

With this model the first three simulations have the characteristics of the description in the model1.

Data 1

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	16	16	1.000000	1	1
mes2	16	16	1.000000	1	1
mes3	16	16	1.000000	1	1
mes4	20	20	1.000000	1	1

Simulation steps executed: 2000

Model time: 8751

Data 2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	18	18	1.000000	1	1
mes2	16	16	1.000000	1	1
mes3	19	19	1.000000	1	1
mes4	15	15	1.000000	1	1

Simulation steps executed: 2000

Model time: 8913

Data 3

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	18	18	1.000000	1	1
mes2	20	20	1.000000	1	1
mes3	18	18	1.000000	1	1
mes4	12	12	1.000000	1	1

Simulation steps executed: 2000

Model time: 8864

As it can be observed, for 2000 steps executed the time of the system, although being nondeterministic, the time is very stable: it is around 8700 and 8900 units of time, this is a variation of 2, 27% of the time.

The fastest one is the first one when all the messages have been more regular; although in all of them 67 messages have been processed.

In the three simulations diverse combinations of messages has been displayed as it is nondeterministic, but in all three of them 67 messages have been processed.

4.1.2 Simulation 2 model 1

In the second simulation the message 1 is considered to be slower than the rest of the other messages

Data 4

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	9	9	1.000000	1	1
mes2	20	20	1.000000	1	1
mes3	22	22	1.000000	1	1
mes4	17	17	1.000000	1	1

Simulation steps executed: 2000

Model time: 8933

Data 5

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	10	10	1.000000	1	1
mes2	13	13	1.000000	1	1
mes3	22	22	1.000000	1	1
mes4	23	23	1.000000	1	1

Simulation steps executed: 2000

Model time: 8893

Data 6

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	10	10	1.000000	1	1
mes2	17	17	1.000000	1	1
mes3	17	17	1.000000	1	1
mes4	24	24	1.000000	1	1

Simulation steps executed: 2000

Model time: 8962

As expected, this simulation shows that the message 1 is the slower one and the one that has the lower number of simulations.

In this case the time models are more stable, between 8893 and 8962, showing that moderate the speed makes the system slower: in this case the variation of time is 0, 75%.

This case would be better than the second time in case that synchronisation with other system is required - it would be more accurate- and it would also be more appropriate in case that it is a kind of message is less important than the rest, this is, for example, that messages 2 and 4 carry critical information and it is desired that they are more inspected than the others.

4.1.3 Simulation 3 model 1

In the third simulation it was consider that the hasrdware1 is always sending information - there is no random time of creation of message1, it is always running - the message 2 is 10 times slower in the random time that creates the messages, the message 3 is 20 times slower and the message 4 is 100 times slower - all of them in the random time that is needed to create them-.

Data 7

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	19	19	1.000000	1	1
mes2	20	20	1.000000	1	1
mes3	19	19	1.000000	1	1
mes4	10	10	1.000000	1	1

Simulation steps executed: 2000
Model time: 8777

Data 8

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	18	18	1.000000	1	1
mes2	16	16	1.000000	1	1
mes3	24	24	1.000000	1	1
mes4	10	10	1.000000	1	1

Simulation steps executed: 2000

Model time: 8885

Data 9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
All_the_messages	67	67	1.000000	1	1
mes1	22	22	1.000000	1	1
mes2	17	17	1.000000	1	1
mes3	20	20	1.000000	1	1
mes4	9	9	1.000000	1	1

Simulation steps executed: 2000

Model time: 8909

As expected, the message 4 is slower than the other messages, but just around 50% slower. Therefore, as so much variation of creation of messages just affects the execution of them by around half of it, it is not strange that the others variations introduced are not really important. It can be seen that barely affects the execution of the messages: just in one execution out of three the message 1 is the fastest one, this implies that changes of creation of messages of around 10 and 20 times is not really significant: this information shows that slower and maybe cheaper elements can be obtained without really affect the performance of this model.

Furthermore, in these simulations it is showed that the amount of messages processed in the same number of steps is exactly the same as the simulations executed before, what's more, the time, although not so regular as in the second simulation, does not presents significant changes, it runs between 8909 and 8777, which means a variation of 1,49%, even less than the first one.

4.2 Model 2

The monitors of the model two are in the final transition of each hardware cluster that shows how many times the transition has been fired; this is how many messages have been processed.

This model has a queue, that is marked with the monitor QUEUE, a monitor that counts the average time that a processor stays without being used called PROCESSORS and a monitor called TIMEWORKING that represent the time the processors are working: if the average times of processors and time working are added they result 1,00 this is, 100% of the time available for the processors.

Finally, the monitor called EVERYMESSPROCESSED counts the number of messages that have been correctly processed by the system.

4.2.1 Simulation 1 model 2 FIFO

Data 10

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	128	1.410661	0	3
Marking_size_System'PROCESSORS	126	0.000718	0	1
Marking_size_System'TIMEWORKING	126	0.999282	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.00000	1	1

Simulation steps executed: 2000 Model time: 11143

Data 11

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	128	1.257019	0	3
Marking_size_System'PROCESSORS	126	0.000720	0	1
Marking_size_System'TIMEWORKING	126	0.999280	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.000000	1	1

Simulation steps executed: 2000 Model time: 11112

Data 12

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	128	1.356465	0	3
Marking_size_System'PROCESSORS	126	0.000716	0	1
Marking_size_System'TIMEWORKING	126	0.999284	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.000000	1	1

Simulation steps executed: 2000 Model time: 11168

The aim of this analysis is to increase the effectiveness of the system by changing some characteristics and the communication protocol. The reports will show the behaviour and the user should aim to boost the service of the queue, the less the queue is used the less that the information is waiting to be processed: in this case, the average used of the queue is 1,33 units.

The next goal that should bear in mind the user would be to reduce the number showed in PROCESSORS as much as possible: this number represents the average number of processor that remains free waiting for receive the information.

Logically, for the same number of steps it will be desired a number in EVERYMESSPROCESSED as high as possible, but this is achieved by improving the used of the former two monitors.

Finally the monitor called TIMEWORKING should be as close to 1 as possible, meaning that it is more effective.

These simulations have been executed with the communication protocol FIFO, first in first out, and it can be sees that the average is queue 1,33, processors 0,000718, time working 0,999282 and 60 messages processed. The average time for 2000 steps is 11153.

Now it should be compared these results with another protocol communications.

4.2.2 Simulation 2 model 2 LIFO

In this case the LIFO communication protocol is used - last in fist out-.

Data 13

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	127	1.479081	0	3
Marking_size_System'PROCESSORS	126	0.000702	0	1
Marking_size_System'TIMEWORKING	126	0.999298	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.000000	1	1

Simulation steps executed: 2000 Model time: 11401

Data 14

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	128	1.608866	0	3
Marking_size_System'PROCESSORS	126	0.000698	0	1
Marking_size_System'TIMEWORKING	126	0.999302	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.000000	1	1

Simulation steps executed: 2000 Model time: 11459

Data 15

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
List_length_dc_System'QUEUE	128	1.660197	0	3
Marking_size_System'PROCESSORS	126	0.000690	0	1
Marking_size_System'TIMEWORKING	126	0.999310	0	2

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_System'EVERYMESSPROCESSED	60	60	1.000000	1	1

Simulation steps executed: 2000 Model time: 11592

If the average numbers are calculated it will be obtained that QUEUE is 1,58, PROCESSORS is 0,00069, TIMEWORKING 0,999303 and the number of messages processed are the same as the ones before.

Compared with the communication protocol FIFO it can be observed that the average time spent by the messages in the queue has increased 18% and the used of the processor, although it has not vary significantly, it is slightly better in the LIFO communication protocol. The monitor TIMEWORKING represents an improvement, and it is clearly seen that it is much faster the FIFO communication protocol than the LIFO communication protocol: for 2000 steps the slowest simulation of the FIFO needs 11168 in contrast with the fastest LIFO communication protocol that needs 11401.

It is clear with these results that the FIFO communication protocol should be applied instead of the LIFO for this system, though the processors have a slightly better used rate in the LIFO protocol.

4.2.3 Simulation 3 model 2 LIFO modified

The following communication protocol followed is LIFO, because as it presents better results FIFO this is an attempt to improve LIFO in case it is needed this communication protocol in case the system requires it.

In this new simulation message1 was made slow and message4 was made fast: a priority of 100 was defined for message 1 and a priority of 10000 was defined for message 4; as the standard priority is 1000 this means that message 1 has ten times more priorities than

usual and message 4 ten times less priorities: between them message 1 has 100 more priority than message 4.

Data 16

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
LIST	132	1.298891	0	3
PROCESSORS	130	0.002512	0	1

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
EVERYTHINGPROCESSED	62	62	1.000000	1	1
MES1	13	13	1.000000	1	1
MES2	17	17	1.000000	1	1
MES3	16	16	1.000000	1	1
MES4	17	17	1.000000	1	1

Simulation steps executed: 2000 Model time: 11546

Data 17

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
LIST	132	1.413000	0	3
PROCESSORS	130	0.000694	0	1

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
EVERYTHINGPROCESSED	62	62	1.000000	1	1
MES1	8	8	1.000000	1	1
MES2	17	17	1.000000	1	1
MES3	19	19	1.000000	1	1
MES4	19	19	1.000000	1	1

Simulation steps executed: 2000 Model time: 11523

Data 18

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
LIST	133	1.482887	0	3
PROCESSORS	130	0.000695	0	1

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
EVERYTHINGPROCESSED	62	62	1.000000	1	1
MES1	14	14	1.000000	1	1
MES2	18	18	1.000000	1	1
MES3	15	15	1.000000	1	1
MES4	16	16	1.000000	1	1

Simulation steps executed: 2000 Model time: 11512

The average values of the variables are 1,4 tokens per unit of time in the list, 62 messages processed, 0,0013 free processors per unit of time, 11527 units of time for 2000 steps executed.

As it can be seen compare with the former models, the length of the list of the LIFO modified is better than the LIFO and comes close to the performance of the list in the FIFO model, though this one is more effective. It can be observed that is much more stable than any other one before, in the three simulations executed the time just vary 0, 29% in 2000 steps: this model would have to be seriously consider in case the synchrony of time is a critical factor - this case this clearly state the usefulness of the tool for analysing real-time concurrent systems, it has been showed by just modifying some characteristics in the model that this tool can save identify the best system for synchronising- .

Although it processes more tokens than the former models - two more-, the FIFO model is faster because the average time is lower in its case, but still this represents a development in the LIFO model: it is two messages faster in a similar amount of time, which is logical because the used of the list have been enhanced.

Furthermore, it is interesting to remark that the least executed message is message1, as it is the most priority one it always enter before to the list and as the list follow the LIFO communication protocol the message 1 is the last one in leaving the list. And for message 4 the same behaviour is applied but in an inverse way.

4.3 Model 3

The monitor QUEUE evaluates the number of times a message is waiting in the queue and the monitor PROCESSORS counts the average time a processor is free and it is not being used.

There is also a monitor that counts the number of times that a message is processed.

In this case it should be bear in mind that there are 12 messages in the system, 3 of each kind.

4.3.1 Model 3 simulation 1 FIFO

Data 19

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	0	3
QUEUE	170	8.999998	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	11	11	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	10	10	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	10	10	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	12	12	1.000000	1	1

Simulation steps executed: 1350 Model time: 158824309

Data 20

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	1	3
QUEUE	170	8.999998	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	12	12	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	11	11	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	9	9	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	11	11	1.000000	1	1

Simulation steps executed: 1350 Model time: 157705828

Data 21

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	0	3
QUEUE	170	8.999998	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	11	11	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	12	12	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	10	10	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	10	10	1.000000	1	1

Simulation steps executed: 1350 Model time: 157146587

The main difference that can be observed between these simulations and the simulations made in the other models is the time that takes to this third model to run. This is due to the fact that in the third model longer times have been written in the functions that add the time to each combination of processor and messages. This should not affect any of the conclusions as the comparisons have to be made between different simulations of the

same model, the goal of the simulations is to find the best communication protocols and the best combination of characteristics of the model.

Logically, in the third model the time is higher and the messages that have been executed are less than in the others models, in this case it is 43 messages for all the simulations.

The difference between the counters PROCESSORS and QUEUE is the number of messages that are waiting in the queue at the moment that the simulations stop - 1350 steps-.

As fewer messages are processed the variations in the queues are smaller. The same consideration has to be made to the spans of time that the processors are waiting.

Having said that it is observed that there are not variations in the monitors that evaluate neither the queues nor the processors, this shows a very stable system, this means that the speed of the messages compared with the speed of creation and availability -there is always a message available as there are 12 messages and just 3 processors-.

The average time is 157892241, with a variation between the higher and the lower of 1,06%, which is approximate to the variation of the models executed before.

This is why there are not variations in those monitors, but, as it was said before, the comparisons and main conclusion have to be drawn from the contrasts between diverse modifications of the same model.

4.3.2 Model 3 simulation 2 LIFO

Data 22

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	0	3
QUEUE	171	8.999998	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	0	0	0.000000	0	0
Count_trans_occur_MESSAGEFLOW'ames2_1	10	10	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	13	13	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	18	18	1.000000	1	1

Simulation steps executed: 1350 Model time: 166094436

Data 23

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	0	3
QUEUE	170	8.999999	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	5	5	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	18	18	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	11	11	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	9	9	1.000000	1	1

Simulation steps executed: 1350 Model time: 183430892

Data 24

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	162	3.000000	0	3
QUEUE	170	8.999998	0	9

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	9	9	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	12	12	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	13	13	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	9	9	1.000000	1	1

Simulation steps executed: 1350 Model time: 166653676

As it happened before, there is such an availability of messages that no variations can be observed in the monitors QUEUE and PROCESSORS, because the last number that changes in the monitor PROCESSORS simulation is so irrelevant that can be ignored: this is a model that simulates an approximate behaviour of a system; that variation is insignificant in this case.

On the other hand, it can be seen that the message1 is always the slowest message, even in the first simulation it did not send any information. This shows a problem when using LIFO communication protocols: when there are much more consumers than producers the first consumers that enters the queue will not leave it, what's more, in this case these first elements that entered the queue will remain there because the time that last an element in being processed and being generated again by the sensors is less time that the time needed for the queue to get empty.

In the simulations is not showed, but if a simulation is made manually, and it is created a new message that is called "trial", for example, that is added to the queue at the very beginning of the simulation, it would be showed by the monitors that this message cannot be processed: it always remains in the queue.

As if this was not disadvantage enough, this communication protocol it is also slower, the average time is 172059668 for the same number of steps executed, and it presents a variation of 10,07%, this is, ten times higher.

In case both models were being studied in order to choose one to apply for a real system, the FIFO model presents more desirable performance and advantages.

4.3.3 Model 3 simulation 3FIFO with modifications

Data 25

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	99	3.000000	0	3
QUEUE	110	9.779770	0	11

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	8	8	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	9	9	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	5	5	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	5	5	1.000000	1	1

Simulation steps executed: 900 Model time: 89500502

Data 26

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	99	3.000000	0	3
QUEUE	109	10.060449	0	11

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	9	9	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	8	8	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	6	6	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	5	5	1.000000	1	1

Simulation steps executed: 900 Model time: 68618256

Data 27

Timed statistics				
Name	**Count**	**Avrg**	**Min**	**Max**
PROCESSORS	100	3.000000	0	3
QUEUE	109	9.821630	0	11

Untimed statistics					
Name	**Count**	**Sum**	**Avrg**	**Min**	**Max**
Count_trans_occur_MESSAGEFLOW'ames1_1	8	8	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames2_1	9	9	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames3_1	6	6	1.000000	1	1
Count_trans_occur_MESSAGEFLOW'ames4_1	6	6	1.000000	1	1

Simulation steps executed: 900 Model time: 187587767

As the LIFO model showed before would have for this model the disadvantage that some messages will remain in the queue without being processed, a modification of the FIFO model is presented here.

In this case conditions were added to simulation: message 3 and 4 could just be processed by the processor number 1. This condition was intended in order to let free more processors to the messages 1 and 2 and study the performance of the system. The results did show the improvement of the number of messages 1 and 2 processed regarding the total amount of processed messages, but this model presents several disadvantages compared with FIFO without modification:

First it was not possible to executed 2000 steps because the model is slower and the memory data in the CPN tools reserved for the representation of times overflowed. This is, the units of time needed to represent the time that takes 2000 steps are more than the units available in the software CPN tools.

As stated before, the system is slower and it is also less stable: for 900 steps it is needed an average number of units of 92362756 units. The variation in just 900 steps is 128%.

Furthermore, the number of messages per unit of time in the queue has increased, an average of 9, 89, and logically has also increased, although slightly, the free time of the processors.

As a result of this reports, in this model it would be recommended try not to attached one processor to just one or two message, as it has been seen the consequences that it has in its performance.

As a matter of fact, the first FIFO communication protocol would be the only one recommended for this system, showing the other two executed here such important disadvantages.

5 DISCUSSION AND CONCLUSION

5.1 Introduction

Petri nets graphical visual easiness jointly with ML programming language has been successfully applied to simulate concurrent, real-time systems. Although its real power are discrete event systems, the combination with the programming language has proved that it is a powerful tool for the simulation of different models and for the analysis of the information, although this information could be more detailed.

5.2Goals conclusion

The main goals of the master thesis were satisfactory achieved:

- It was possible to model a concurrent, stochastic, real-time system that could represent the behavior of a multi-processor and multi-sensor communication protocol
- The CPN software has showed itself a useful tool to model and analyze the information, being recommended for its suitability for real-time systems.

5.3 Implications

The CPN tools has proved itself to be a useful tool to simulate different models that can be applied either to create the best combination of elements for new systems or to improve existing systems changing communication protocols.

As it was seen in the example simulations, it is easy, fast and effective to make all the relevant changes to the models.

It combines the visual simplicity of understanding of the Petri nets graphical design with the ML programming language, which removes the limitations that present the graphical representation of the Petri nets and can be utilized for small, medium and big system, within some margins.

It would be a tool recommended for beginners with real-time, concurrent systems for its easy visualization - although some ML language should be known- and it would be also recommended because it is a fast and effective modelling tool: the models analyzed could be created by an experienced programmer in one or two days.

5.4 Limitations

Petri nets together with a programming language, as it is presented in CPN tools, can effectively model small, medium and big real-time systems, with some limitations:

When modeling big systems the Petri nets graphical system grows incredibly high and, although in theory it is possible to simulate any system independently of its size, practically it is barely achievable. This disadvantage can be avoided by using the programming language ML, but, the bigger the system is, the more ML language will be needed, and the less Petri nets graphical tool will be implemented.

The goal for using Petri nets instead of any programming language is that it creates an easily understandable model, really visually fast, that has the advantage of fastest understanding than any programming language when the programmers are change

This is imperative and it is the main reason why CPN tools would be choose instead of a programming language: the main time consumption task in improving software it is the time that the programmer spends understanding the program; as Petri nets is a graphical tool, it can decreased appreciably the time consumed in this task.

This problem can be seen in the third model: if more variables are added to select a different combination of variables -like instead of just choosing between messages and processor it could be also possible to choose between day and night, or line A, B or C-, then it can either increase the clusters of transitions and places making the model really big - and impossible to create if 15 or 20 variables more are added- or it can also be possible to use ML programming language.

And here remains the limitation: if programming language is utilized for differentiating 15 or 20 variables then there is no point in using Petri nets graphical because it will no longer represent the flow of the system: this will be made by the functions implemented by the ML language and then the user should spend time understanding the language and the program, it will have lose its ease of visual understanding.

One detail: these 15 or 20 variables that make the system practically unreachable have to be used at the same time; in other words, just when it is needed to specified all the 15 variables at the same time - this is when it is needed to know that it is message1 with processor 2 in the line C at midnight, and the frequency of the electricity is 50Hz and more specifications at the same time-.

If the 15 variables are not needed to specify at the same time, but, for example, it is needed 4 variables now, five at the next step, and 3 in each of the two next steps, then it can and it would be recommended to use CPN tools.

In other words, if it is needed for the first operation two variables -message 1 and processor3-, for the second operation three - line C, 50hz and midnight -,… then CPN tools would be an effective and suitable tool.

Another limitation, although it is not software limitation, it is that the programming language in CPN tools is ML language. This is not a common taught language at universities, so it is possible that some time is required to learn to use this language.

5.5 Recommendations and Further investigation

Regardless the former limitations, this work can definitely be developed further.

Further investigation could be creating more complex models that suits better a real system. Nowadays in the information age –also known as computer or digital age- there are a lot examples where these tools could be applied.

In this models nested functions were implemented but did not work, one easy way of improving the models are creating these functions.

Another field that would open the way to more students would be change the ML language of the CPN tools for a more common one known by most of the students, like C++, matlab or a similar or common programming language.

It might be also interesting to compare the tools -like CPN tools - available for concurrent systems and its suitability for real-time systems.

Regarding CPN tools, it would be helpful if there were more options to display the information recorded by the monitors. As CPN tools is going to be an open-source program, it could easily been implemented.

It would be also interesting to verify the created models using the tool showed in [6] and [9].

References

1. Murata T. (1989): Petri nets: properties, analysis and applications, Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580.

2. Mikhail Zarubin (2014): Petri nets for modelling and calculations.

3. A. Spiteri Staines (2012): A Colored Petri Net for the France-Paris Metro, international journal of computers. Issue 2, Volume 6, 2012

4. Falko Bause, Pieter S Kritzinger (2002): Stochastic Petri Nets: An Introduction to the Theory. ISBN: 3-528-15535-3.

5. M.H. Jansen-Vullers, M. Netjes (2006): Business Process Simulation - A Tool Survey.

6. W.M.P. van der Aalst, A.H.M. ter Hofstede (2000): Verification of Workflow Task Structures: A Petri-net-based approach.

7. Mor Peleg, Daniel Rubin, Russ b. Altman (2005): Using Petri Net Tools to Study Properties and Dynamics of Biological Systems. J Am Med Inform Assoc. 2005; 12:181–199. DOI 10.1197/jamia.M1637

8. Vlatka Hlupic, Vesna Bosilj-Vuksic (2001): Petri Nets and IDEF Diagrams: Applicability and Efficacy for Business Process Modelling.

9. Pedro M. González del Foyo, José Reinaldo Silva (2011): Some Issues in Real-Time Systems. Verification Using Time Petri Nets. J. of the Braz. Soc. of Mech. Sci. & Eng. Copyright Ó 2011 by ABCM October-December 2011, Vol. XXXIII, No. 4 / 467

10. Claudine Chaouiya (2007): Petri net modelling of biological networks. Briefings in bioinformatics. vol 8. No 4. 210 ^219

11. K. Jensen, L.M. Kristensen (2009): Coloured Petri Nets. DOI 10.1007/b95112_1, Chapter 1, Introduction to Modelling and Validation.

12. Santiago C. Pérez (2006): Modelación, simulación de funcionamiento y evaluación de prestaciones de protocolos de red con redes de petri desarrollo de herramientas académicas de enseñanza.

13. Pedro Solana González, Margarita Alonso Martínez, Daniel Pérez González (2007): Analisis y modelado con redes de workflow del proceso de tratamiento de experiencias operativas. XX Congreso anual de AEDEM, Vol. 1, 2007 (Ponencias), pág. 77.

14. Carlos Castellanos (2006): Consideraciones para el modelado de sistemas mediante Redes de Petri Vol 27, No 2 (2006).

15. Green, D.C. (1991): Data communication / D. C. Green

16. Zurawski Richard, Zhou MengChu (1994): Petri Nets and Industrial Applications: A Tutorial. IEEE. transactions on industrial electronics, vol. 41, no. 6, December 1994

17. Description of Petri Nets
http://www.scholarpedia.org/article/Petri_net

18. Web page of CPN Tools
http://cpntools.org/

19. Petri net world: online services for the international Petri Nets community
http://www.informatik.uni-hamburg.de/TGI/PetriNets/index.html

20. Murillo, Luis Diego (2008): Redes de Petri: Modelado e implementación de algoritmos para autómatas programables. Tecnología en Marcha, Vol. 21, N.° 4, Octubre-Diciembre 2008, pp. 102-125

ANEXES

Figure: declarations model 1

```
secondmodelpriorities.cpn
  Step: 0
  Time: 0
  Options
  History
  Declarations
    Standard priorities
    Standard declarations
      colset UNIT = unit;
      colset BOOL = bool;
      colset INT = int;
      colset INTINF = intinf;
      colset TIME = time timed;
      colset REAL = real timed;
      colset STRING = string;
      var b:BOOL;
      var x,x1,x2,x4,w,w1,w2,w3,u,u1,u2,u3,l,l1,l2,l3,aux,aux2,aux3,a,d :REAL;
      var j,k:UNIT;
      colset TT=product REAL*REAL timed;
      colset T= string timed;
      colset RT= product REAL*T;
      colset RTL= list RT;
      var li:RTL;
      var t: T;
      var y,v,m,z:TT;
      var x3,x5: INTINF;
      var f:REAL;
      var g:INT;
      colset delay = int with 25..75;
      fun del() = delay.ran();
      fun probability (n:int, i:int) =
          let
              val realn = Real.fromInt n
              val reali = Real.fromInt i
              val c=reali*reali
              val rv = normal(realn,c)
          in
              rv
          end;
      fun Mtime()=ModelTime.time():time;
      fun TtS()=ModelTime.toString(Mtime()):string;
  Monitors
    EVERYTHINGPROCESSED
    MES1
    MES2
    MES3
    MES4
    PROCESSORS
    mes1
    mes2
    mes3
    mes4
  System
    Hardware1
    Hardware2
    Hardware3
    Hardware4
    probability
```

Figure: Declarations model 2

```
thirdmodel.cpn
  Step: 0
  Time: 0
  Options
  History
  Declarations
    Standard priorities
    Standard declarations
      colset UNIT = unit timed;
      colset BOOL = bool;
      colset INT = int;
      colset INTINF = intinf;
      colset TIME = time timed;
      colset REAL = real timed;
      colset STRING = string;
      var b1,b2,b3,b4:BOOL;
      var y,x,x1,x2,x3,z,w,w1,w2,w3,v,u,u1,u2,u3,l1,l2,l3,m,aux,aux2,aux3,a,d :TIME;
      var rea:REAL;
      var j,h,k:UNIT;
      colset T=real timed;
      var t:T;
      colset T1=string;
      var t1:T1;
      colset message=string timed;
      var mes:message;
      colset processor= string timed;
      var proc:processor;
      colset MPT=product message*processor*T timed;
      var mpt:MPT;
      colset MPTB=product MPT*BOOL timed;
      colset MT=product message*T timed;
      var mt:MT;
      colset MT1
      colset LMPT=list MPT;
      colset MR=product message* REAL;
      colset LMR=list MR;
      var lmr: LMR;
      var l:LMPT;
      var f:REAL;
      var g:INT;
      colset delay = int with 1..6;
      fun del() = delay.ran();
      fun probability
      fun expTime
      fun Mtime()=ModelTime.time():time;
      fun delaysending
      fun m1sending
      fun m2sending
      fun m3sending
      fun m4sending
      fun m1processing
      fun m2processing
      fun m3processing
      fun m4processing(mes,proc,t)=
        case proc of
        proc1=>t+0.25 |
        proc2=>t+0.37|
        proc3=>t+1.33;
      fun TtR()=ModelTime.toReal(Mtime()):real;
      fun RtS(rea)=Real.toString rea:string ;
      fun StoR(t1:string)=ModelTime.maketime(t1):time;
      fun realtostring(r)=Real.toString r:string;
  Monitors
  SYSTEM
    probability
    MESSAGEFLOW
```

colset MR=product message* REAL;

Figure: Declarations model 3