Final Thesis

Instrumentation of timed automata for formal verification of timed properties

by

Mikael Hagman

LITH-IDA/DS-EX-ING--07/004--SE

2007-09-24

Examinator: Daniel Karlsson
Zebo Peng
Instrumentation of timed automata for formal verification of timed properties

Mikael Hagman

Embedded systems are used in many technical products of today. The tendency also points to the fact that they are in many ways becoming more and more complex as technology advances. Systems like advanced avionics, air bags, ABS brakes or any real-time embedded system requires reliability, correctness and timeliness. This puts hard pressure on designers, analyzers and developers. The need for high performance and non failing systems has therefore led to a growing interest in modeling and verification of component-based embedded systems in order to reduce costs and simplify design and development. The solution proposed by the Embedded Systems Lab at Linköping University is the modeling language PRES+, Petri Net based Representation for Embedded Systems. PRES+ models are then translated into timed automata, TA, which is used by the UPPAAL verification tool. To be able to verify timing properties the translated TA model must be instrumented with certain timers, called clocks. These clocks must be reset in a manner reflected by the property to be verified.

This thesis will provide a solution to the problem and also give the reader necessary information in order to understand the theoretical background needed. The thesis will also show the reader the importance of modeling and time verification in the development of embedded systems. A simple example is used to describe and visualize the benefit regarding real-time embedded systems as well as the importance of the ability to verify these systems.

The conclusion drawn stresses the fact that high development costs, possible gain of human lives and the problems in developing complex systems only emphasize the need for easy to handle and intuitive verification methods.

Keywords
Embedded systems, formal verification, Petri Net, PRES+, Timed Automata
Abstract

Embedded systems are used in many technical products of today. The tendency also points to the fact that they are in many ways becoming more and more complex as technology advances. Systems like advanced avionics, air bags, ABS brakes or any real-time embedded system requires reliability, correctness and timeliness. This puts hard pressure on designers, analyzers and developers. The need for high performance and non failing systems has therefore led to a growing interest in modeling and verification of component-based embedded systems in order to reduce costs and simplify design and development.

The solution proposed by the Embedded Systems Lab at Linköping University is the modeling language PRES+, Petri Net based Representation for Embedded Systems. PRES+ models are then translated into timed automata, TA, which is used by the UPPAAL verification tool. To be able to verify timing properties the translated TA model must be instrumented with certain timers, called clocks. These clocks must be reset in a manner reflected by the property to be verified.

This thesis will provide a solution to the problem and also give the reader necessary information in order to understand the theoretical background needed. The thesis will also show the reader the importance of modeling and time verification in the development of embedded systems. A simple example is used to describe and visualize the benefit regarding real-time embedded systems as well as the importance of the ability to verify these systems. The conclusion drawn stresses the fact that high development costs, possible gain of human lives and the problems in developing complex systems only emphasize the need for easy-to-handle and intuitive verification methods.
Acknowledgements

I would first like to thank my supervisor, Daniel Karlsson PhD at IDA ESLab, (The Department of Computer and Information Science, Embedded Systems Lab Linköping University), for his guidance and support during these months. Your help was invaluable when I embarked on my journey in previously unknown territory, (that is Petri Nets and the concept of Timed Automata), that I never had come in contact with before. I would also like to thank Tommy Olsson, also at IDA for his help regarding questions about C++ implementation. I would also like to thank my family, including my kid brother, who helped me with many tings during my work. Last, but not least, I would like to direct a special thanks to my girlfriend and life partner, Wiktoria Glad, who during this time stood behind me all the way.
CONTENTS

1 INTRODUCTION ............................................................................................................. 4
  1.1 MOTIVATIONAL EXAMPLE ...................................................................................... 5
  1.2 PROBLEM ............................................................................................................... 6
  1.3 SOLUTION ............................................................................................................... 6
  1.4 LIMITATIONS ....................................................................................................... 6
  1.5 TARGET GROUP .................................................................................................... 7
  1.6 READING INSTRUCTIONS ..................................................................................... 7

2 BACKGROUND .............................................................................................................. 8
  2.1 PETRI NET ............................................................................................................. 8
    2.1.1 Mathematical definition .................................................................................. 8
    2.1.2 Petri Net as a graphical modeling tool ............................................................ 9

3 MODEL CHECKING PRES+ MODELS AND TIMED AUTOMATA .............................. 11
  3.1 BASIC DEFINITION OF PRES+ ............................................................................ 11
  3.2 THE MODEL CHECKING ENVIRONMENT ............................................................... 12
  3.3 DESCRIPTION AND DYNAMIC BEHAVIOR OF PRES+ ...................................... 13
  3.4 COMPUTATION TREE LOGIC .............................................................................. 15
  3.5 TIMED AUTOMATA WITH UPPAAL ................................................................. 16
    3.5.1 Timed Automata ............................................................................................ 16

4 PROBLEM FORMULATION AND SOLUTION ............................................................ 21
  4.1 IMPLEMENTATION REQUIREMENTS SPECIFICATION ..................................... 21
    4.1.1 Formula description and requirements ............................................................ 21
    4.1.2 Algorithms ..................................................................................................... 22

5 CONCLUSIONS AND FUTURE WORK ..................................................................... 28
  5.1 CONCLUSIONS .................................................................................................... 28
  5.2 FUTURE WORK .................................................................................................... 28

6 REFERENCES ............................................................................................................. 29

7 APPENDICES 1 – Abbreviations .............................................................................. 30

8 APPENDICES 2 – Notations ....................................................................................... 31
Chapter 1

1 Introduction

Today, most people in the industrial developed parts of the world, daily come in contact with more or less sophisticated items containing one or several systems which we refer to as embedded systems. The concept of embedded systems origins from the early 1970s from embedded computers and embedded computer systems, ECS [1] by the United States Air Force when they needed to find a distinction from mechanical systems containing computers from general purpose computers and commercial business information systems. At this time of age it was difficult to develop or buy these new specialized systems since a public law from 1965, mostly aimed at controlling purchases of general-purpose computer systems used for business applications rather then military-oriented special systems. The rule was that all component parts, had to be developed as a whole system and not as independent components. Thus the only legal system available at the time was to use a data-processing system approach to buy computers for electro-mechanical systems. This in turn, led to the following and the first of all, definition of embedded computer systems, according to the author [1]:

“A computer system that is integral to an electromechanical system such as a combat weapon system; tactical system; air-craft, ship, missile, spacecraft, certain command and control systems; and civilian systems such as an automated rapid transit system. Embedded computer systems are primarily differentiated from automatic data processing systems (ADPs) by how they are developed, acquired and operated in a using systems [sic!]. The key attributes of an ECS are

- It is physically incorporated into a larger system whose primary function is not data processing.
- It is integral to such a larger system from a design, procurement, and operations viewpoint.
- Its output generally includes information, control signals, and computer data.”

There are many definitions of embedded systems today but they do have a single concept in common, they are all special-purpose systems designed to perform a special task. Once they are programmed they generally perform this task without the possibility for the user to manipulate the function. In present time, embedded systems can be found in everything from cellular phones, dishwashers and handheld games to sophisticated avionic systems, both military and civilian. All of them use some sort of embedded system to control different behavior or functions. It does not take too much of an imagination to understand that some of these, like avionics systems for example, can become very complex. This complexity also makes them difficult and expensive to design and develop. There are several problems to overcome before a system is ready for the market. Most systems, especially embedded ones, need to meet requirements of correctness, meaning that it has to meet the specifications of any mathematical algorithm or in- and output correctness of a function. Since many embedded systems also are so called hard Real-Time systems (RTS) they also have to meet specifications regarding timeliness, which means that every part of the system has to react within a given time restriction or else fail. Failure to meet these time restrictions is not an option.
Developing new software for a complex system is not something you do in a day, most of the time is spent on mainly three parts. Approximately 33 percent of the cost is spent on analysis and design. 50 percent on testing and only just over 16 percent is spent on the actual coding process [2]. Fixing errors after the system is ready is even more costly and time consuming. Therefore it would be a great help if there is a possibility to simulate and verify the behavior of a system before actually building it since this could reduce some of the problems and cost in the design- and development process. To be able to do that, we do need easy-to-handle, and intuitive tools for modeling and verification of embedded systems. When it comes to verification methods there are basically two ways one could use, formal and informal. Formal verification in turn consists of several approaches. It is based on mathematical (logical) models, methods and theorems. In this thesis, formal verification will be the only method mentioned. Model checking, which is an instance of formal verification is, even though there competitive methods such as theorem proving and others, one of the most common one used in the industry today. It is known for its efficiency and relatively simple use.

This chapter will give an overview of the motivation and objective of this thesis. At the end of this chapter a general view of the structure of this thesis will be presented.

1.1 Motivational example

Imagine a car traveling at high speed on a road, an obstacle suddenly appears and the driver of the car hits the brakes hard and furiously turns the steering wheel in an attempt to avoid the obstacle on the road. If the car is an older vehicle, without any safety systems, this would probably cause it to slide off the road and crash, or something even worse.

However on most of the cars of today, several different security systems would have taken control at this point. These are advanced systems, designed to manage the functioning of the brakes, the suspension system, the seatbelts, airbags and so forth. The first couple of them designed to prevent the car from sliding off the road, and if it still would, the following systems would attempt minimizing the danger of severe injuries for the driver of the car and to the possible passengers. Let us take a hypothetical look at the first two systems mentioned above. It is not too farfetched to assume that these two systems work in conjunction with each other.

The Antilock Braking System (ABS) working to prevent the car’s brakes to lock also distributes the brake power amongst the four tires to ensure that the wheels with best contact to the road gets the maximal brake power. Meanwhile the suspension also tries to keep the chassis as stable as possible by for example working as an anti yaw system to the car. The input to most, if not all, of the safety systems contains one or several sensors located at different places of the car. It is easy to see that if these two systems did not work together in a correct order and of course responding within any given time restriction, the result could be catastrophic.

If the driver in a panic situation, hits the brakes hard and steers hard to the left two things would happen. The suspension sensors would react by sending signals to the system telling it to harden the suspension on the right side of the car to straighten the chassis up to a safe level. At the same time the brake sensors would, if necessary, send signals to the system to ease the braking power to prevent them from locking and thereby avoiding a sliding movement across the road. If the systems react in a correct manner, chances are quite good that an accident could be avoided. If they do not, the result may be that the car rolls or slides right of the road.
An even more intuitive example would be, if the accident, despite the ABS system attempt to avoid it, still happened. If the airbag then did not react in time and instead failed to meet its deadline or if the seatbelts failed to tighten their grip on the driver and the passengers at the right time, the possible outcome would not be hard to imagine. These two examples mentioned above only emphasize the need for efficient time verification tools to ensure correctness and timeliness in systems like ABS or airbags. I will return to this example later on in chapter 3.

1.2 Problem
When developing a new embedded system, it is most useful to make a model of it first. This could be done in PRES+, which is a Petri Net based representing tool for Embedded Systems. However, since PRES+ itself can not measure time these models have to be translated into something that is able to perform these measurements, in order to enable time verification of the model. Since the restrictions are stipulated by logical formulas they also have to be included in this translation. These logical formulas expresses characteristics that has to be verified, and refers to concepts such as places, transitions etc, in PRES+. The problem is therefore to find a way to enable time verification according to the time restrictions stipulated.

1.3 Solution
Embedded systems can, as mentioned above, be represented in PRES+. When model checking, these models are translated into timed automata, the input language of the underlying model checker. This translating method already exists.

However, in the verification of these models must be translated into concepts that are used in timed automata, especially time restrictions. This can be achieved by adding clocks where appropriate in the timed automata. These clocks are used to measure the time restrictions stipulated by the logical formulas.

1.4 Limitations
In order to verify an embedded system one has to choose a verification technique. In this thesis model checking is used. As input to specify a system’s behavior, logical formulas usually are used and there are many types of logical formulas that could be used to do this. This thesis however only addresses two types of them. These two types of formulas are as follows:

- $\text{AF}_{\leq x} p$. This formula stipulates that, given a certain time, $x$ there has to be a token present at place $p$.

- $\text{AG} (p \rightarrow \text{AF}_{\leq x} q)$. This formula stipulates that, if there is a token present in place $p$, there must also be a token present in place $q$, after $x$ time units.

Any further development could naturally be extended to include more complex formulas than the two sets mentioned above.
1.5 Target group
This thesis focus on readers that have some knowledge in embedded systems and formal verification of such systems. Any reader, who with a general interest in the subject should however be able to read this thesis.

1.6 Reading instructions

- Chapter one gives a brief introduction, the problem itself, a motivational example and a suggested solution.

- Chapter two will give the reader some preliminary knowledge and some basic definitions regarding Petri Nets.

- Chapter three introduces our Model Checking environment, knowledge and definition of PRES+ and introduces the concept of Timed Automata.

- Chapter four presents the solution suggested and points to the actual implementation of the solutions to the problem given in this thesis.

- Chapter five contains the conclusions drawn and also presents future work.
Chapter 2

2 Background

In this chapter the theoretical background for the thesis is presented. As mentioned in the
problem and solution description, we use PRES+ as the system modeling tool. Since PRES+
is an extension of Petri Net, there will be a brief introduction of Petri Nets followed by an
introduction of PRES+. When the model is subjected for verification it has to be translated
into something that is understandable by a verification tool, namely Timed Automata (TA).
The program used in this case is the UPPAAL verification tool [4]. UPPAAL is a result from
a joint research team from Uppsala University, Uppsala Sweden, and Aalborg University,
Aalborg Denmark. Chapter 2.3 gives a brief understanding of how logical formulas are used
to describe properties. Finally a shorter introduction of the concept of timed automata is also
presented at the end of this chapter.

2.1 Petri Net

The concept of Petri Nets origins from the year 1962 when Carl Adam Petri [7] submitted his
work at the Technische Universität, Darmstadt, Germany. Since then, progress has been made
in this area and today Petri Nets, even though there are other fields of applications, also is a
very useful graphical tool for modeling systems.

2.1.1 Mathematical definition

Petri Nets are formally defined as a 5-tuple $N = \{P, T, I, O, M_0\}$, where

$P = \{P_1, P_2, \ldots, P_n\}$ Where $P$ is a finite, non-empty set of places.

$T = \{T_1, T_2, \ldots, T_m\}$ Where $T$ is a finite, non-empty set of transitions.

$I \subseteq P \times T$ is a finite non-empty set of input arcs which define the flow relation between the
places and transitions.

$O \subseteq T \times P$ is a finite non-empty set of output arcs which define the flow relation between the
places and transitions.

$M_0$ is the initial marking of the net. A marking $M(P): \rightarrow \{0, 1\}$ is a function that denotes the
presence or the absence of tokens in places in the net. If there is a token present then
$M(P) = 1$, if a token does not exists then $M(P) = 0$. 
2.1.2 Petri Net as a graphical modeling tool

A Petri Net consists of **Places**, **Transitions** and **Arcs** that bind the places with the corresponding transition. There are two kinds of arcs that are relevant in this thesis, **Input**- and **Output** arcs. The output arc leads from a transition to a place while the input arc does the opposite. In figure 1, we can see how the places, arcs, transitions and so forth are being presented graphically.

![Diagram of Petri Net components](image)

- **Place**
- **Token**
- **Arc**
- **Transition**

**Figure 1**: Graphical presentation.

A simple Petri Net (PN) is also presented in figure 2, this shows the places and how they are connected to the transitions via the arcs.

![Diagram of a simple Petri Net](image)

**Figure 2**: A simple Petri Net.

Each place ($P_m$), denoted as circles is connected to a transition ($T_n$), denoted as filled rectangles. In order to verify the behavior of a system each place is also set with a **marking** that describes whether a place has a **token** present or not. The token is presented as a filled dot.
inside the place it is associated with (see place $P_0$ in the figure). The dynamic behavior of the
net depends on the markings. In a simple term, one could say that if there is a token present,
in all input places of a transition, then the requirements for that transition is fulfilled. More
generally we say that if a place has enough (i.e. one token in each input place) tokens then the
corresponding transition are enabled and therefore ready to fire. Once a transition has fired it
becomes an output place to next place in line, this place then containing the token, will hold
the new marking. This then enables the next transition, or transitions, in turn and so forth [8].
In this example (see figure 2) the new markings would be at place $P_1$ and $P_2$ which would set
the transitions $T_2$ and $T_3$ into an enabled state. There are problems with Classical Petri Nets
though, for instance one of the problems is that tokens themselves does not carry any
information, and therefore can not be evaluated, they also lack capability to perform time
measurements which makes them inadequate as the kind of verification tool we need.
Chapter 3

3 Model Checking PRES+ Models and Timed Automata

PRES+ which stands for (Petri net based Representation for Embedded Systems) is an extension of classical Petri Nets. This means that PRES+ therefore not only benefits from the functions of Petri Net but also introduces new features. This section will show us the environment the designers have to work with and give a brief introduction regarding the differences between classical Petri Net and PRES+. A shorter example of the dynamic behavior of the net is also presented.

3.1 Basic Definition of PRES+

Standard PRES+ has similar mathematical definitions as Petri Nets, but also as mentioned previously introduces new features. The graphical representation is however the same as in Petri Net, (see figure 1). In classical Petri Net tokens carry no information. In PRES+ however, tokens $k$ can represent both a value and a time stamp $k = (v, r)$, where $v$ represents a token value and $r$ a time stamp. Furthermore the transitions $T$ have functions $T_i(j)$ and may also contain time intervals. This means that for every transition $t \in T$, there exists a minimum delay $d^-$ and a maximum delay $d^+$, describing the execution time delay of any function associated with the transition [3].

Transitions can also have guards. A guard is a function of the input places that returns a boolean value whether it is true or not. If a place holds a token with a certain value, and there is a transition with a function (i.e. equal to, greater than, and so on), the guard would be true iff, (i.e. if and only if), the token value was within the transition function limits. A transition is not permitted to fire, that is enabled, unless its guard is satisfied. [5].

A Standard PRES+ model is as classical Petri Net, a 5-tuple $\Gamma = (P, T, I, O, M_0)$ where $P = \{P_1, P_2, \ldots, P_m\}$ Where $P$ is a finite, non-empty set of places. $T = \{T_1, T_2, \ldots, T_n\}$ Where $T$ is a finite, non-empty set of transitions. $I \subseteq P \times T$ is a finite non-empty set of input arcs which define the flow relation between the places and transitions. In this case, the flow from places to transitions.

$O \subseteq T \times P$ is a finite non-empty set of output arcs which define the flow relation between the places and transitions. In this case, the flow from transitions to places.

$M_0$ is the initial marking of the net. A marking $M: P \rightarrow \{0, 1\}$ is a function that denotes the presence or the absence of tokens in places in the net. If there is a token present then $M(P) = 1$, if a token does not exists then $M(P) = 0$. A place $p$ is marked if and only if $M(P) \neq 0$. 

11
3.2 The Model Checking Environment

Figure 3 presents an overview of the environment used in our proposed model checking approach [5]. As inputs to the model checker, (i.e. the UPPAAL model checker, briefly mentioned in section 3.5), we use a PRES+ model and a set of (T) CTL, (Timed CTL), properties. CTL or Computation Tree Logic is described in section 3.4. The highlighted areas in the figure show the parts of our model that this thesis deals with.

Since we do not want to “reinvent the wheel”, but want to be able to reuse already efficient existing model checkers, we use the UPPAAL model checker as mentioned earlier. In order to use this model checker we have to translate our PRES+ model into the language understood by UPPAAL, namely timed automata. Furthermore the set of CTL properties also has to be translated and used as input to our model checker. The highlighted arrow, leading from the (T) CTL property to the Translation module, shows one of the tasks this thesis includes. In order to be able to perform time measurements we have to introduce the CTL properties into the translation module as well.
3.3 Description and Dynamic Behavior of PRES+.

In this simple example, as described in figures 4a to 4c, the set of places ($P$) and transitions ($T$) are denoted as follows.

$$P = \{p_0, p_1, p_2, p_3\} \text{ and } T = \{t_1, t_2, t_3\}.$$  

Token values are presented within pointed brackets, $<>$, at the right top side of the places. Guards are presented at the upper right side at the transitions. Time limits are presented within brackets, $[\]$, at the left bottom side by the transitions. Functions are presented to the left upper side by the transitions. The token $k$, present in $p_0$ (as shown in figure 4a) is now carrying information consisting of, a value and a time stamp. The figure shows us that the token has a value of $k = 4$ and a time stamp with the value of $v = 0$. This token constitutes the initial marking $M_0$. The transition $t_1$ also has a time delay interval which gives the transitions execution time once it becomes enabled, in this case between 3 to 5 time units, and a guard that stipulates that the token value must be less or equal 4 in order to be enabled to fire. The following two figures continue to describe, in a very simple way, the dynamic behavior of the model in PRES+.

Figure 4a: A simple PRES+ net.
If these criteria are satisfied the transition is permitted to fire and the places $P_1$ and $P_2$ will each receive a token as shown in figure 4b. Both of these places now contain one token each with a value of 4. The next step depends on whether or not the restrictions in the following transitions $T_2$ and $T_3$ are satisfied. $T_2$ has a function besides its time delay interval that adds the value 5 to the token and $T_3$ has a guard associated with it. Both places $p_1$ and $p_2$ now have one token present respectively. We note however, that the guard associated with transition $T_3$ restricts the token value to $x \leq 3$, which means that only transition $T_2$ is enabled and thereby permitted to fire.

Figure 4c: The dynamic behavior of a PRES+ net, continued.

The place $P_3$ in figure 4c now therefore contains a token with the value 9. These additional features make PRES+ an intuitive and easy handled verification tool for Embedded Systems.
3.4 Computation Tree Logic

This section aims only to give a brief and simple basic knowledge of Computation Tree Logic and the use of logical formulas in order to give some understanding regarding theory later on in this thesis.

Computation Tree Logic, or CTL for short, is according to NIST [10], a propositional, branching-time temporal logic that allows formulas to be checked in linear time, making it suitable for model checking. We use these logical formulas to express a behavior over time and thus specifying the system we want to verify. CTL formulas consists of temporal operators like $\text{G}$ (globally), $\text{F}$ (future), $\text{X}$ (next step), $\text{U}$ (until) and $\text{R}$ (releases). In all cases these operators must also be preceded by a so called path quantifier $\text{A}$ (all) or $\text{E}$ (exists).

![Diagram](image_url)

**Figure 5:** Illustration of some possible CTL formulas.

In the first example (in figure 5) the universal path-quantifier $\text{A}$, followed by the operator $\text{G}$ states that for all possible futures (i.e. all computation paths) the property $p$ will always be true in every state in every possible future. The second example $\text{AF} p$ states that sometime in the future the property will exist amongst the possible paths. Finally the last example $\text{EF} p$ states a true property somewhere eventually along the possible computation paths.
Operator $X$, not shown in any example only looks one step ahead in the future but in contrary to the others it does not include the initial state. Without going too deep into the semantics we note furthermore the fact that formulas can be nested so more complex properties can be expressed. One example of this is the formula, $\text{AG} (p \rightarrow \text{AF}_{\leq x} q)$ which means that when $p$ is true then within equal or less then $x$ time units, $q$ also has to be true [5].

3.5 Timed Automata with UPPAAL

There is no need to reinvent the wheel again so we want to be able to reuse efficient and already existing model checkers. To accomplish this we have to translate the PRES+ model into a language the chosen model checker is able to understand. In this case the choice fell on the UPPAAL verification tool [4]. UPPAAL is the result of combined research between Uppsala University in Sweden and Aalborg University in Denmark.

UPPAAL is used to describe system behavior as networks of automata extended with clock and data variables. The tool itself consists of three major parts, a simulator, model checker and a description language and is thereby a useful tool for modeling, simulation and verification of real time systems such as embedded systems.

This chapter will give the reader, not a deep understanding about UPPAAL itself, but rather some understanding of timed automata and will focus on translating the PRES+ model [5] into timed automata.

3.5.1 Timed Automata

A timed automaton can be defined as a finite state machine (FSM) which has been equipped with a set of clocks. The timed automaton model was developed in the early 1990s by Lynch and Vaandrager. The model is a labeled transition system for components in real-time systems [9], and it uses trajectories to describe the evolution of the system state over time. In our case one timed automaton is created for each transition in our PRES+ model. It models what occurs in every transition. For each of the transitions in our model, a clock variable is also instantiated together with one global variable for every place in our model.
The global variable holds the value, if any, of the referred place. If no token exists the variable will be assigned a default value. Every automaton created consists of a number of locations, generally one more location than the number of inputs, (i.e. arcs that leads to the corresponding transition) has. If there is a guard present at the transition another location has to be added. This is done because the automata can be in different states. States indicate which location a token is placed in. A token present in the location en means there are tokens present at all input places of the transition and that the guard, if it exists, is satisfied. Any token located in enc describes the same except in this case the guard is not satisfied. Should a token be present in location s0 it implies that there is no token present in any input places corresponding to the transition, furthermore if there is a token at location s1 we note there is more than one input place to the transition, but only one of these places has a token present and so forth.

Each automata created to the corresponding transition will also be fitted with a clock variable, denoted $c_t$. Finally there is also a synchronization label associated with every transition, these are named as the transitions with the label “!” or “?” attached to the transition name. When a transition fires, the label “!” is taken for that transition, and at the same time this implies that all transitions labeled with a “?” has to be taken simultaneously [5].
As we can see the token is in the location $en$ which states all criteria regarding token values, and guard are satisfied and therefore the transition is ready to fire. We note that the upper time limit of the transition are reflected in the automata as a location invariant at location $en$, and the lower bound appear as guards on the transitions from $en$ to synch. This means that the token “has to leave” the state $en$ within $\leq 5$ time units, and that it can do so, earliest at $\geq 3$ time units. In figure 6a there is also a place named $synch$ present. This place would however normally not be there and is shown merely to visualize that in the translation process from PRES+ to timed automata, any edge (i.e. any input arc) with the location synch as target are used to set the condition, meaning the guard, the clocks and any existing relation values. The automata are then set to commit and the clock of the transition is set to zero. All of this happens before the transition is set with the label “!”. 

**Figure 7a:** A timed automata corresponding to transition $t_1$ in figure 6.
Figure 7b: A timed automata corresponding to transition $t_2$ in figure 6.

In figure 6 we note a guard, by the lower right side at the transition $t_2$ (which corresponds in figure 7b by $p_1 < 8$ at the upper arc leading to the location $\text{enc}$, and by $p_1 := 8$ by the arc that leads to the location $\text{en}$), associated with transition $t_2$, and therefore another state is added to the corresponding automata model namely $\text{enc}$. Here we see a token present in location $s_0$ which tells us there is one input arc to the transition $t_2$ but no token is present in the corresponding place. This also corresponds correctly to our PRES+ model in figure 6 which displays place $p_1$ as an input to transition $t_2$. When transition $t_1$ has fired, the result will be a change in the automata related to transition $t_2$. Either will the token be placed at the $\text{enc}$ location which means the guard is not satisfied or a token is placed in location $\text{en}$ and all restrictions are fulfilled and $t_2$ is ready to fire. Figure 7c describes the automata for transition $t_3$. The possible states for the automata are similar to the ones for the one corresponding to transition $t_2$ as we have the same amount of inputs to transition $t_3$ as we have to transition $t_2$. The differences here are the upper- and lower time limits and the function belonging to the transition.
Finally figure 7d describes the automata to the last transition $t_4$ from our PRES+ model. With only one place serving as input to the transition, the transition is similar to transition $t_1$ (figure 7a) and therefore can assume the same type of states as that transition.

**Figure 7c:** A timed automata corresponding to transition $t_3$ in figure 5.

**Figure 7d:** A timed automata corresponding to transition $t_4$ in figure 6.
Chapter 4

4 Problem formulation and solution

This chapter aims to give a detailed description of the requirements needed, a description of the problem and the suggested solution. The efficiency of PRES+ is well documented as a verification technique [5], and in conjunction with UPPAAL it becomes an easy to handle and intuitive tool. To make this easy to use, an intuitive graphical interface called Verpres has already been developed [6]. A translation module for translating Petri Nets into timed automata also exists.

4.1 Implementation Requirements Specification

As the motivational example in chapter 1 suggested, a tool for verifying embedded systems is needed in order to reduce design and development costs. Verpres, however, lacks the possibility to measure time restrictions stipulated by logical formulas and therefore has to be equipped to meet these demands. The main task is to add clocks to the timed automata resulting from the translation of PRES+ models, and reset them when necessary. The logical formulas therefore need to be introduced in the translation process in order to reset the clocks due to the restrictions that the formulas stipulate.

4.1.1 Formula description and requirements

There are several types of formulas that express time restrictions, in this thesis we will concentrate on two sets of such logical formulas.

1. \( \text{AF}_{\leq x} p \). Which stipulates that, given a certain time, \( x \) there has to be a token present at place \( p \).

2. \( \text{AG} (p \rightarrow \text{AF}_{\leq x} q) \). Which stipulates that, if there is a token present in place \( p \), there must also be a token present in place \( q \), at a given time \( x \).

Furthermore the places \( p \) and \( q \), can also be relations, i.e. \( pRv \) or \( qRv \), where \( R \in \{<, \leq, =, \geq, >\} \), and \( v \) is a given value. This implies that there has to be a token present in \( p \) or \( q \) with a value that satisfies the relation. The requirements will therefore be as follows.

- The formula is of the first type presented above.
  - A clock, initially set to zero, has to be added in the timed automata model, as well as in the timed query.

- The formula is of the second type presented above.
  - As in the first case, a clock has to be added. The clock also has to be set to zero, when and only when a token appears in place \( p \). (With consideration to any restrictions to the tokens value.)
4.1.2 Algorithms

A module which translates the model from PRES+ into timed automata already exists. In this module, though, some necessary steps need to be taken in order to enable time verification as stipulated by the logical formulas. The following algorithms, presented further down in this section, satisfy these demands. The motivational example mentioned in the introduction is used to visualize and exemplify the steps.

If we return to the motivational example from the introduction chapter 1.3 in the beginning of the thesis, we can use the flowchart in figure 8 to illustrate a simplified flow of events occurring when the driver steps on the brake. When the brakes are applied, a sensor recognizes whether or not the brakes lock. If they are locked the sensor gives a signal to the brake to loosen its grip if not it continues to apply the same pressure in order to make the car come to a halt. Meanwhile another sensor (perhaps connected to the speedometer) decides if the car still moves or not. If not the system has successfully filled its purpose and so forth.

![Flowchart](image)

**Figure 8:** A simple flowchart describing one of four brake sensors.
Figure 9 describes the PRES+ model for one of the brake sensors where place $p_0$ is the initial point. Depending on the value of the token (stipulated by the logical formula) the token can "travel" in two directions from place $p_1$. Let place $p_2$ indicate no brake lock, which means transition $t_3$ may be enabled. Let place $p_2$ indicate that the brake is locked and therefore a decreasing grip on the brake is necessary, if this is done transition $t_4$ may be enabled. This cycle of events repeats itself until the car has stopped moving. From place $p_4$ one could imagine a returning coupling to the sensor at place $p_0$ telling the sensor the brake is locked and therefore the grip has to decrease. From the place $p_5$ there should be further arc (not shown in this picture) that in conjunction with another system (i.e. a model for speed indication) leading to a transition and a place representing a halt of the vehicle. Observe that the place $p_0$ is marked with a dotted rectangle. Let us assume this place symbolizes the driver applying pressure to the brake pedal and thereby sending a signal $p_1$ representing a sensor that measures the power of pressure applied on the pedal. In place $p_0$ there also is a light grey colored dot. This dot is merely there to express and visualize the fact that the driver has stepped on the brake and the token in place $p_0$ has already transferred to place $p_1$. 

**Figure 9:** A simplified model of one of the brake systems.
In the driver’s attempt to stop the vehicle, the sensor $p_0$ measured the pressure denoted by the value 5 on the token. Our starting point in this verification example is however at place $p_1$.

The formula used in this example will be of the type $\text{AG} (p \rightarrow \text{AF} \leq x q)$, where $p = \text{place} \ p_1$ and with a relation $p < 6$, $x = 10$ time units and $q = \text{place} \ p_5$. This means that if there is a token present at place $p_1$, and the token has a value lesser then 6, then there has to be a token present in place $p_5$ within 10 time units. The token was also, as stipulated by the formula we want to verify in this case, assigned with a *time stamp* with the value of 1 time units. Since the formula stipulated a relation value of the token ($p_1 < 6$) this means that the clock (clock_cq), if the guard at transition $t_1$ is satisfied, should be set to zero.

1. **Implement a clock, clock_cq, to the timed automata model.**

Clock_cq is a clock that measures time from any given starting place, identified in step 2, to an ending point in the net. Which places being the starting and ending point is stipulated by the logical formula, meaning if there is a place $p_i$ in the model, and the place named in the antecedent of the formula has the same name, then this is the starting point. If there is an existing place $q_i$ which corresponds to the formula, then this will be the ending point in the model. Once it is established, in the translation module used, that the formula includes a time restriction the clock is added. Furthermore, if the place in question has a token present, and the token is a relation type the clock should be set to zero. If not, nothing should happen. In any case we must consequently add the additional clock, clock_cq, to the TA model.

![Figure 10: The corresponding TA-model to transition $t_1$ in figure 9.](image-url)
As an example we can look at the transition $t_1$ and its input, place $p_0$ (figure 9), we see that the token that previously was held in the place had the value of 5, and a time stamp with a value of 1. The transition $t_1$ has a guard that restricts the token value to be lesser or equal to 10, and a time delay interval between 1 and 4 time units. According to the formula we want to verify in our example all criteria were satisfied so far, and as a result (figure 9) the location $en$ was taken, meaning that transition $t_1$ was enabled to fire. If our formula had stipulated the place $p_0$ as the starting point then the clock$_{cq}$ would had been set to zero at the time the transition fired. Please note that the upper arc in the figure is dotted. This was done to illustrate the before, and after, state of the TA-model. Once fired there is no way that the TA-model can assume the $en$ state again.

2. Identify the place, stipulated by the antecedent of the implication in the formula.

For any given place, stipulated in the formula the translation module has to check if there is a place corresponding to this, or these places. If the place exists it will be identified. Otherwise, an invalid formula was given. As mentioned above, we want to verify the formula $AG (p_1 \rightarrow AF \leq_{10} p_3)$ in this example. This formula has a time limit. Therefore the clock clock$_{cq}$ was added in step 1. In our case $p_1$ is the starting point, as it is the place in the antecedent of the implication of the formula.

3. Identify the transitions that serve as input to the given place. Name this set of transitions $T_p$.

Each transition with one or more outgoing arcs that leads to places stipulated by the formula has to be identified in the translation module. In our example this would be the transition $t_1$ serving as the incoming transition to $p_1$.

4. For each transition in $T_p$, identified in step 3.

4.1. Identify the timed automata that correspond to the transition, this is named $At_p$.

The automaton, in figure 10, corresponding to the transition $t_1$ is identified by the translation module. This is done dynamically since the module iterates over each transition, $t_a$ in the net, and creates one corresponding automaton for every transition that serves as input to places.

4.2. Identify the output arc that leads from $en$ (that is the timed automata place that is set as enabled), with the place synch as target. Name this arc $b$.

When an output arc leads from the place $en$ with synch as the target it means that the transition is ready to fire. In our example, this would mean the widened (highlighted) transition in figure 10. In that case the arc leading from the place $en$ in the figure would be named $b$. 

25
4.3. **If the place \( p \), is a relation type (pRv).**

Arcs have to be identified because it is used to express a guard if the place is a relation type. (i.e. the transition will or will not be allowed to fire depending on if the restrictions, expressed by the guard, are satisfied or not) In this case this would mean that if the brake locks it would be place \( p_2 \) the corresponding automata for transition \( t_2 \) evaluated. If the brakes did not lock, then the place would be \( p_3 \), the corresponding automata for transition \( t_3 \) evaluated instead. In our motivational example we assumed that the value on the token present at place \( p_1 \) was 5, it also was of a relation type since the formula stipulated that \( p_1 < 6 \), this means that we have to add a guard to the TA.

4.3.1. **Add a new arc, \( b' \), in parallel and with the same attributes as \( b \). (let \( b' \) be a copy of \( b \))**

If we continue to look at transition \( t_1 \), the arc named \( b \) then being the arc leading from location \( en \), we want to add a new arc with the same attribute as \( b \). This has to be done since the antecedent of the formula is of a relational type, \( p_1 < 6 \), and we have to be able to express the possibility that the token value is not satisfied by that relation. Figure 11 shows the result of this operation, where there now are two arcs leading from the location \( en \), to \( synch \).

![Figure 11](image-url)

**Figure 11:** The correct automata, corresponding to transition \( t_1 \), with the new arc \( b' \) added.

The new arc gives us the possibility to set the clock \( clock_{cq} \) to zero only when needed and not else. Should there not be a relation value present the transition \( t_1 \) would still have to be able to fire, but without resetting \( clock_{cq} \) to zero. This is accomplished by using the complement to \( b \) as the transition to \( synch \) instead, this complement is added to the arc \( b' \) in step 4.3.3.
4.3.2. **Add a guard to \( b \) that expresses the relation \( pRv \).**

If a relation exists we need to add a guard to the arc in order to translate this into timed automata for UPPAAL to understand the restrictions given by the formula. The step 4.3 shows us that in our example a guard has to be added, the formula clearly stipulates a relation value, and therefore we add a guard to the arc \( b \) that expresses the relation. The guard in this case being \( p_1 < 6 \).

4.3.3. **Add a guard to \( b' \) that expresses the complement to the relation \( pRv \).**

At this step we actually add the complement to arc \( b' \) relation value for the automata, (step 4.3.1). We do this to avoid resetting clock_cq in these cases. In our example this means when the token held by place \( p_1 \geq 6 \), as shown in figure 10.

4.3.4. **Set the clock_cq to zero.**

Finally we need to reset the clock_cq to start the time measuring. This is done by adding an assignment to arc \( b \) as shown in the figure 11.

5. **Done.**
Chapter 5

5 Conclusions and Future Work

In this chapter a summary of this thesis is presented. At the end of the chapter suggestions of future work can be read.

5.1 Conclusions

In this thesis, an implementation of time measurement, in an already existing tool for modeling, simulation and verification of PRES+ models has been presented. PRES+, which is an extended timed Petri Net model, is very useful to represent embedded systems.

As mentioned in the motivational example it is easy to understand the benefits of verification models. If we return to the example with the car on the road and reflects for a minute on the different components that have to work together, and in a strict order to accomplish the tasks of keeping the car on the road, or if the system fails, try to minimize possible human injuries. The need to be able to verify the embedded system working as predicted is obvious. Each system has to, not only fulfill their own specifications, but also has to work in conjunction with each other. There would not be any meaning if the airbag fired after a crash already had taken place or if the antiskid system didn’t work together with the break control system.

A verification tool like this, presented in this thesis can be used to verify each of the components functioning as specified, especially considering the issue of timeliness. Contributions made by this thesis are the added possibility to now measure time as stipulated by a logical formula and enabling translation of the formulas into the TA model.

In the verification process the tool needs two inputs, one is the timed automata (TA) model and the other is a query that stipulates the given restrictions. The TA model itself measures the time restrictions for each transition, while the new implementations that this thesis has presented, measures time from a stipulated start point in a system.

5.2 Future work

The main contribution, to the verification tool mentioned, is the implementation of time measurement capability. This capability is however somewhat limited. Future work could extend the implemented functions to handle more complex formulas.
References

Encyclopedia of Software Engineering.
Copyright © 2002 by John Wiley & Sons, Inc. All rights reserved.
DOI: 10.1002/0471028959.sof102


Net Models and Timed Automata”,
http://citeseer.ist.psu.edu/cache/papers/cs/29069/http:zSzzSzwww.ida.liu.sezSzlabszSzseslab
zSzpublicationszSzpapzSzdbzSzRTCSA02.pdf/cortes02verification.pdf.
Downloaded June 16, 2007.

[4] Collaboration between the Basic Research in Computer Science at Aalborg
University(AAL) and the Department of Computer Systems (DoCS) at Uppsala University

Dissertation No 1017, Dept. of Computer and Information Science, Linköping University,


[7] www.informatik.uni-hamburg.de/TGI/PetriNets/faq/
Downloaded April 15, 2007.

based on a Petri Net Representation”, in Journal of Systems Architecture, pp. 49(12-

Downloaded April 2, 2007.

Downloaded April 15, 2007.
Appendices 1.

Abbreviations

ABS  Antilock Breaking System
ADP  Automatic Data Processing
CTL  Computation Tree Logic
ECS  Embedded Computer Systems
FSM  Finite State Machine
NIST US National Institute of Standards
PRES+ Petri-net based representation of Embedded Systems
RTS  Real Time Systems
TA   Timed Automata
TCTL Timed CTL
Appendices 2.

Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d^-, d^+$</td>
<td>Lower- respective upper bound of the time delay</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>An arbitrary PRES+ model</td>
</tr>
<tr>
<td>$I$</td>
<td>Set of input arcs</td>
</tr>
<tr>
<td>$k$</td>
<td>Token</td>
</tr>
<tr>
<td>$M$</td>
<td>Marking</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Initial marking</td>
</tr>
<tr>
<td>$M(p)$</td>
<td>Marking of a place $p$</td>
</tr>
<tr>
<td>$O$</td>
<td>Set of output arcs</td>
</tr>
<tr>
<td>$P$</td>
<td>Set of places</td>
</tr>
<tr>
<td>$p, P_m$</td>
<td>Place</td>
</tr>
<tr>
<td>$r$</td>
<td>Token timestamp</td>
</tr>
<tr>
<td>$T$</td>
<td>Set of transitions</td>
</tr>
<tr>
<td>$t, T_n$</td>
<td>Transition</td>
</tr>
<tr>
<td>$T(f)$</td>
<td>Function of transition $T$</td>
</tr>
<tr>
<td>$v$</td>
<td>Token value</td>
</tr>
</tbody>
</table>

Computation Tree Logic (CTL)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Universal path quantifier</td>
</tr>
<tr>
<td>$E$</td>
<td>Existential path quantifier</td>
</tr>
<tr>
<td>$F$</td>
<td>Temporal operator future</td>
</tr>
<tr>
<td>$G$</td>
<td>Temporal operator globally</td>
</tr>
<tr>
<td>$R$</td>
<td>Temporal operator releases</td>
</tr>
<tr>
<td>$U$</td>
<td>Temporal operator until</td>
</tr>
<tr>
<td>$X$</td>
<td>Temporal operator next step</td>
</tr>
</tbody>
</table>