**UNIVERSITE DE BLIDA - BLIDA 1**
**Faculté des Sciences**
Département d'Informatique

## THESE DE DOCTORAT

en Informatique

Spécialité : Génie des Systèmes Informatiques

# CONCEPTION DE SYSTEME EMBARQUE BASÉE SUR LES CONCEPTS DE L'ARCHITECTURE LOGICIELLE

Par

Abdelhakim BAOUYA

Devant le jury composé de :

| | | |
|---|---|---|
| H. ABED | Professeur, U. de BLIDA | PRÉSIDENT |
| N. BOUSTIA | Maître de conférences, U. de BLIDA | EXAMINATEUR |
| N. BENBLIDIA | Professeur, U. de BLIDA | EXAMINATEUR |
| D. BENNOUAR | Maître de conférences, U.de BOUIRA | DIRECTEUR DE THÈSE |
| O. AIT MOHAMED | Professeur, U. de CANADA | CODIRECTEUR DE THÈSE |
| W.HIDOUCI | Professeur, E.S.I., ALGER | INVITE |

Blida, Avril 2016

UNIVERSITY OF BLIDA - BLIDA 1

# ABSTRACT

**Embedded Systems Design based On the Concepts of Software Architecture**

by Abdelhakim BAOUYA

Today's systems are constructed from a set of interconnected components to produce a behavior including one or more tasks. The constraints imposed on such systems in terms of functionality, reliability, cost and Time-to-Market are very strict. Unfortunately, their development is becoming a hard process. Thus, in one side the applications are becoming more complex especially for real-time embedded systems, in other side, the Market demand for high-quality systems is becoming a big challenge. Recently, formal validation of computer systems are emerging especially to predict the system behavior and ensure its correctness before the implementation.

The aim of this thesis is to provide a practical and formal framework that enables probabilistic assessment and functional requirements verification on a system modeled by SysML internal blocks diagrams. Our main contribution is a novel approach to automatically verify probabilistic system behavior under time constraints based on their functional requirement. The design verification is based on Probabilistic Model Checking. To handle the deployment process, our approach allows automatic deployment-space exploration that maximizes the system-life duration with respect to the hardware platform characteristics. To demonstrate the effectiveness of our approach, we apply our methodology on academia as well as on automotive case studies. . . .

**Keywords**: Software Architecture; Model Checking; SysML; Temporal Logic; Embedded Systems; Real-Time

**جامعة البليدة - البليدة 1**

# ملخص

## تصميم الأجهزة المحمولة على أساس مفاهيم هندسة البرمجيات

**باوية عبد الحكيم**

الأنظـــمة الحـــالية تشيد من مجموعة من المركبـــات المتصلة لإنتاج السلوك بما في ذلك المهام الوظيفية. للأسـف القيود المفروضة على هذه الأنظمة من حيث الأداء الوظـــيفي، والموثوقية والتكلفة والوقتـ أصبحت عملية صعبة من جهة و من جهة أخرى التطـبيقات أصـــبحت أكثر تعقيدا خصوصا المحـــمولة و الطـــلب في السـوق لأنظـمة ذات جودة عـــالية صار تـــحديا كبيرا. في الآونـة الأخيرة، لاحظنا ظهور المصادقة ريـاضيا على أنظمة الكمـبيوتر للرد على قيود المصمم خصوصا للتنبؤ بسلوك النـــظام قبل التنـــفيذ.

والهدف من هذه الرســـالة هو توفير الإطار الرسمـــي الذي يســـمح التقيـيم الاحتمـــالي والتحقق من المتطلبات الوظيفية على الأنظمة المهندسة باستعـــمال - نظام لغة النمذجة- مسـاهمتنا الرئيسية هي توفير نهج جديد للتحقق آليا من سـلوك النظم الاحتـــمالية في ظل ضيق الوقت على أسـاس احتياجاتـــهم الوظيـفيـــة. ويستـنـد التحقق من التصـــــميم على تقـنيـات فـحص نموذج احتمالي. لتحليل السلوك ، يتم التعبير عن الخصائص في المنطق الزمني. للتـعامل مع عملية توزيع المركبات، هذا المنـهج يسمح التوزيع آلـيـا لاكـــتشاف فضـاء التوزيـع الذي يزيـد مـن حياة النظـــام باحتساب خصائص الأجهزة. وللتدليل على فعالية المنهج، نطبق أسلوبنا على الأمثلة الأكاديمية وكذلك أنظمة السيارات.

**دلالات :** هندسة البرمجيات ، المصادقة الرياضية، الأجهزة المحمولة، المنطق الزمني

UNIVERSITY DE BLIDA - BLIDA 1

# RESUME

**Conception de systèmes embarqués basée sur les concepts de l'architecture logicielle**

par Abdelhakim BAOUYA

Aujourd'hui, les systèmes sont construits à partir d'un ensemble de composants interconnectés pour produire un comportement incluant une ou plusieurs tâches. Les contraintes imposées à ces systèmes en termes de fonctionnalité, de fiabilité, de coût et de délais de mise sur le marché sont très stricts. Malheureusement, leurs développement devient un processus difficile, parceque les applications embarquées a temps réel sont de plus en plus complexes et la demande pour des systèmes de haute qualité est en train de devenir un grand défi. Récemment, la validation formelle a pu répondre aux contraintes des developpeurs afin de prédire le comportement du système et garantir son exactitude avant sa mise en œuvre.

L'objectif de cette thèse est de fournir un framework formel qui permet l'évaluation probabiliste et la vérification des exigences fonctionnelles sur un système modélisé par SysML internal blocks diagrams. Notre principale contribution est de fournir une nouvelle approche pour vérifier automatiquement le comportement des systèmes probabilistes sous contraintes temporelles en fonction de leurs exigences fonctionnelles. La vérification de la conception est basée sur les techniques de vérification des modèles probabilistes dont les propriétés sont exprimées en logique temporelle. Aussi, notre approche permet l'exploration automatique de l'espace de déploiement qui maximise la durée de vie du système on considérant les caractéristiques de

la plate-forme matérielle. Pour démontrer l'efficacité de notre approche, nous appliquons cette méthode sur un exemple académique et aussi sur les systèmes automobiles. ...

**Mots clés**: Architecture logicielle; Vérification formelle; SysML; Logique temporelle; Systèmes embarqués; Temps réel

# ACKNOWLEDGEMENTS

I would like to express my special appreciation and thanks to my advisors Dr. Djamal Bennouar, Dr. Otmane Ait Mohamed and Dr. Samir Ouchani, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. I would also like to thank my committee members, professor Abed Hafida, professor Boustia Nahrimane, professor Benblidia Nadjia and professor Hidouci Walid for serving as my committee members even at hardship. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

A special thanks to my family. Words cannot express how grateful I am to my mother, and father for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal.

. . .

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# INTRODUCTION

In System Engineering, SysML (System Modeling Language)is a standard language [2] developed by the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE). SysML is widely used for specification, analysis, design and verification of a broad range of complex systems (e.g. Hardware, software, information work flow). SysML reuses a subset of UML2 artifacts (Unified Modeling Language) [3] and provides additional extensions to specify requirements such as probability and components deployment. Behavioral specification is achieved in SysML using three diagrams: State Machine, Communication and Activity Diagram. Particularly, SysML activity diagrams is used to model system behaviors. In addition, systems' composition is supported by the call behavior and send/receive artifacts.

Constraints on software development in terms of functionality, performance, reliability and time to market are becoming more stringent. Therefore, software development reveals several challenges. Indeed, if in one side applications are becoming increasingly complex, in the other side the market pressure for rapid deployment of these systems makes designs a challenge. One major challenge in software development process is to advance errors detection at early stages of life cycles. Researchers at the National Institute of Standards and Technology (NIST) attest: about 50 percent of software development budgets go to testing, yet flaws in software still

cost the U.S. economy $59.5 billion annually [4]. It has been shown in Fig.1 [1] that the cost of system repairing during the maintenance in software development life cycle is approximately 67%. Therefore, acceleration of verification and maintenance process at preliminary design is extremely beneficial as compared to fixing them at the testing phase.

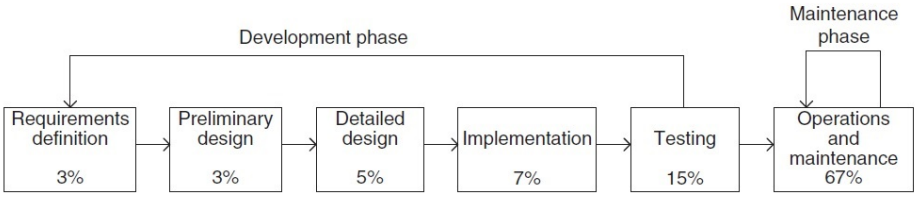



FIGURE 1: Traditional Software Design Life Cycle [1]

Recently, formal verification methods have become essential tools for developing safety-critical systems, for whose behavioral correctness is a main concern. These methods require a mathematical expertise for specifying what the system ought to do and verifying it with respect to the requirements. There are mainly two ways for doing formal verification: *Theorem proving* and *model checking*.

Theorem proving needs to be performed by people with skills (Experts) in order to solve the problem. In addition, this approach needs a precise description of the problem written in logical form and the user needs to think carefully about the problem with deeper understanding in order to produce an appropriate formulation. However, there are significant implementations of this approach such as HOL [5], Coq [6].

Model checking is popular formal verification approach in software and hardware industry. For instance, SLAM [7] which is a Microsoft research project uses a model checking to verify device drivers are conformed to their Application Programming Interface (API) specification. According to [8], Model Checking is automatic and usually very fast. The user does not need to construct a correctness proof. If the

specification is not satisfied, the Model Checker will produce a counterexample execution trace that shows why the specification does not hold. In addition, temporal Logics can easily express many of the properties that are needed for reasoning. This verification method focuses on either *qualitative* or *quantitative* properties [9]. The qualitative properties assert that certain event will happen surely or not. The quantitative properties are based on the probability or expectation of a certain event (e.g. the probability of the system failure in the next $t$ time units is 0.85). *Probabilistic model checking* is an effective technique to verify probabilistically the satisfiability of a given property. In this thesis we use PRISM language for probabilistic model checker [10] where the properties can be expressed in Probabilistic Computation tree logic (PCTL) or in Continuous-stochastic logic (CSL). Note that the properties prescribe what the system should do, and what it should not do, whereas the model description addresses how the system behaves.

Owing to the fact that it is never enough to just ensure the functional correctness of a given system, ensuring the software deployment (i.e. currently known as partitioning) is a real challenge. The process consists in distribution of a software components on different physical locations with respect to the requirements. The core of the process is based on the functional correctness of the composed activity diagrams of the interacting components by taking into account the the physical component's failure at early design life cycles. Layali et al. [11] assert that hardware faults have a negative impacts on the programs (i.e. software).

It is extremely important to provide a mechanism employing quantitative techniques for deployment evaluation of software based on their design models. In this thesis, we address the issue of functional assessment of software/systems modeled in SysML activity diagrams. The goal is to gauge how well a product is meeting its functional requirements. Since we use model-checking technique, temporal logic is used to express functional properties.

# 1 Problem statement

Current research initiatives focus mainly on qualitative model checking of SysML to ensure the correctness of systems functionality. Most of the proposed approaches are intended to verify the probabilistic **or** the timed system behavior[12], [13], [14], [15], [16], [17], [18]. Furthermore, software-deployment is rarely checked in UML and SysML behavioral diagrams. Since SysML is a young modeling language that extends UML with system features, only few related works exist. Herein, this thesis aims at investigating and answering two fundamental questions:

1. How to verify and evaluate the functional properties in SysML activity diagrams under time constraints?
2. How to choose the best deployment configuration using probabilistic model checking?

# 2 Objectives

The main goal of this thesis is to check and to quantify the functionality of probabilistic systems at design level by taking into account the system architecture and its functional constraints. Thus, we propose a SysML language for components and behavior specification. In addition, we use MARTE profile to specify the hardware and the real-time characteristics. The objectives of this thesis are summarized as follows:

1. Providing an efficient verification framework to evaluate the functional correctness of probabilistic systems modeled in SysML under time constraints,
2. Demonstrating the efficiency of model checking in deployment-space exploration where activity diagrams are partitioned on the software components.

# 3 Proposed methodology

This section describes our framework to specify and to verify the functionality of systems modeled by using SysML blocks and activity diagrams. Our proposed framework targets systems composed of hardware and software components. Furthermore, it automatically evaluates the reliability of systems at the design level by verifying the distributed activity diagram on the different software components (i.e. Blocks). The output of the software components are annotated with probabilities that are extracted from the hardware platform characteristics. The proposed framework depicted in Fig.2 is based on model checking, and it develops three main concepts: extracting the individual reliability of the hardware platform, extracting the semantics of the studied diagrams, coding the semantics in Prism input language and verifying the satisfiability of the reliability properties.

The SysML activity diagrams is used to specify the behavior of the studied system. The language allows the allocations of the artifacts on different software components (i.e. Partitions) where the interactions is modeled by transition lines that can cross the partition canvas. In addition, when two software components are distributed on different processors the cross lines are annotated with probability values.

For verification, we extract the formal semantics of SysML activity diagrams. This helps to encode easily diagrams into the input language of the PRISM model checker. To ensure the correctness of our proposed approach, we prove the soundness of model to model in the framework.

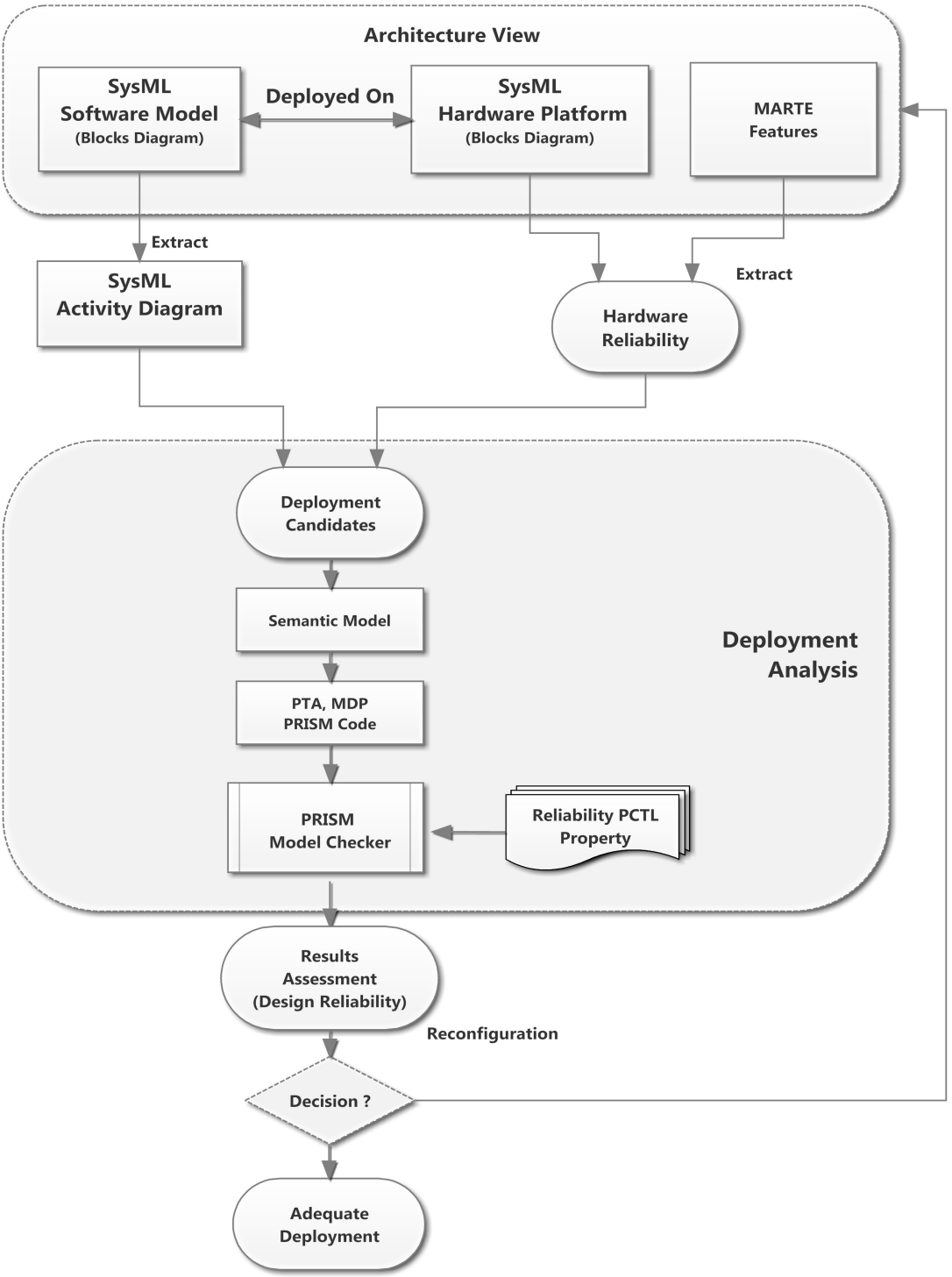



FIGURE 2: Methodology for software design and deployment

# 4 Thesis contributions

The main contributions of this thesis are:

1. **Probabilistic and timed verification framework**. We propose a practical and formal framework to specify and to verify the timed and probabilistic systems modeled by SysML activity diagrams,

2. **Formal semantics of SysML activity diagrams under time constraints**. We formalize SysML activity diagrams enhanced with probability and time by giving an adequate meaning that can be supported by the existing model checking tools. As we study behavioral diagrams, we developed a calculus based on the structural operational semantics.

3. **Formal semantics of probabilistic and timed automata in PRISM language**. We formalize PRISM model by providing a syntax and the operational semantics that support the main operators of PRISM.

4. **Mapping soundness proof**. Our verification mechanism uses PRISM as verification tool. It encodes the semantics of SysML activity diagrams under time constraints in the input language of PRISM. We prove the soundness of the verification by showing that our encoding preserves the satisfiability of the existing properties.

5. **Software deployment**. We propose SysML internal blocks diagrams to specify the architecture of software components and its deployment with activity diagrams that express the functional behavior. The original model is enriched with sub set of real-time characteristics.

# CHAPTER 1

# BACKGROUND

## 1.1 Introduction

System-level design and verification is two steps procedure starting from modeling, ending on verification. The first step consists on design view interpreted in formal language with associated behavior that rules the system operations. The original design needs to be converted to other formalism acceptable by model checker tools. Modeling is accompanied with a set of designer requirements that need to be verified for decision making. These requirements are expressed in temporal logic. The verification process starts by exploring different paths (sequence of states) to check if the requirements are held else, a counterexample is produced prohibiting system failures.

## 1.2 Transition system

Models of systems describe their behavior in an accurate and an unambiguous way. They are mostly expressed using finite-state automata, consisting of a finite set of states and a set of transitions. Mainly, we cite: Markov Decision Processes [10], Probabilistic Timed Automata (PTA)[39], Continuous Time Markov Chains (CTMC) [20] and Contin-uous Time Markov Decision Processes (CTMDP)[20]. Next, we define Probabilistic and Timed-Probabilistic systems, then we introduce the system-level modeling language used in our approach.

### 1.2.1 Probabilistic systems

Probabilistic Transition Systems [9] is a version of transition systems (automata) that support the probabilistic decision. Precisely, a probabilistic automata (PA) that exhibits both probabilistic and nondeterministic features. Definition 2.1 illustrates a PA where $Dist(S)$ denotes the set of convex distributions over S and $\mu = [..., s_i \rightarrow p_i, ...]$ is a distribution in Dist(S) that assigns a probability $p_i$ to a state $s_i$.

**Definition 2.1** (Probabilistic Automaton). A probabilistic automaton is a tuple $M = (\overline{s}, S, L, \sum, \delta)$, where:

- $\overline{s}$ is an initial state, such that $\overline{s} \in S$,
- S is a finite set of states,
- L: $S \rightarrow 2^{AP}$ is a labeling function that assigns each state to a set of atomic propositions taken from the set of atomic propositions (AP),
- $\sum$ is a finite set of actions,

- $\delta : S \times \sum \rightarrow Dist(S)$ is a probabilistic transition function assigning for each $s \in S$ and $\alpha \in \sum$ a probabilistic distribution $\mu \in Dist(S)$.

### 1.2.2 Timed probabilistic systems

Timed Probabilistic Systems [39] are an extended version of probabilistic transition system that support the time and probabilistic decision. More specifically, Probabilistic Timed Automata (PTA) is a Probabilistic Automata equipped with a finite set of real-valued clock variables called *clocks*. Conditions on the values of the clocks are used as enabling transition conditions (i.e., guards) of actions: only if the condition is fulfilled the action is enabled and capable of being taken; otherwise, the action is disabled. Conditions which depend on clock values are called clock constraints which are used to limit the amount of time that may be spent in a given location. The following definition prescribes how constraints over clocks are defined.

**Definition 2.2** (Clock Constraint). A clock constraint over set $X$ of clocks is by the following grammar

$$g ::= x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g$$

where $c \in \mathbb{N}$ and $x \in X$. Let $CC(X)$ denotes the set of clock constraints over $X$.

A Probabilistic Timed Automata (PTA) is a Probabilistic Automata with clock variables. The clocks are used to formulate the real-time assumptions on system behavior. An edge in a PTA is labeled with a guard (when is it allowed to take an edge?), an action (what is performed when taking the edge?), and a set of clocks (which clocks are to be reset?). A state is equipped with an invariant that constrains the amount of time that may be spent in that location. The formal definition is:

**Definition 2.3** Probabilistic Timed Automata (PTA). PTA is a tuple M = $<\overline{s}$, S, X, Act, Inv, Prob , L $>$, where:

- $\overline{s}$ is an initial state, such that $\overline{s} \in$ S ,
- S is a finite set of states,
- X is a set of clocks,
- Act is a set of actions,
- *Inv* :$S \rightarrow$ *CC*(*X*) is an invariant condition, imposes restrictions on the allowable values of clock variable,
- Prob: S $\times$ $Act \rightarrow$ *Dist*($2^X \times$ $S$) is a probabilistic transition function assigning for each $s \in S$ and $\alpha \in Act$ a probabilistic distribution $\mu \in Dist(2^X \times S)$.
- L : S $\rightarrow$ $2^{AP}$ is a labeling function mapping for each state a set of atomic propositions.

Transitions (Edges) in PTA are labeled with tuple $(g, \alpha)$ where g is the clock constraint of the PTA, $\alpha$ is an action. The intuitive interpretation of $s \xrightarrow{g,\alpha}_p s'$ is that the PTA can move from state s to state *s*′ when clock constraint $g$ holds. Besides, when moving from state s to *s*′, any clock will be reset to zero and action $\alpha$ is performed according to the *distribution* $\mu \in Dist(2^X \times S)$. Function *Inv* assigns to each state a state invariant that specifies how long the PTA may stay there (maximum elapsing time). For state s, *Inv*(s) constrains the amount of time that may be spent in s.

A location in PTA [39] is a pair $(s, \vartheta) \in \mathbb{R}_{\geq 0}$ such that for a set X of clocks, $\vartheta$ is a clock valuation with $\vartheta : X \rightarrow \mathbb{R}_{\geq 0}$, assigning to each clock $x \in X$ its current value $\vartheta(x)$. In any location $(s, \vartheta)$, either a certain amount of time $t_s \in \mathbb{R}_{\geq 0}$ elapses, or an action $\alpha \in Act$ is performed. If time elapses, the choice of $t_s$ requires that the invariant *Inv* remains continuously satisfied while time passes i.e.$\vartheta(t_s) \models$ *Inv*$(s)$. The resulting location after this transition is $(s, \vartheta + t_s)$. In the case where an action is performed, an action $\alpha$ can only be chosen if it is enabled i.e. when clock constraint g holds,

$\vartheta(t_s) \models g$. Once an enabled action $\alpha$ is chosen, a set of clocks will be reset to zero and a successor location are selected at random, according to the distribution $\mu_{(s,\vartheta+t_s)} \in Prob(s, \alpha)$.

There are two possible ways in which a PTA [39] can proceed by taking a transition in the PTA (Action transition) or by letting time progress while remains in a state (Delay transition):

- Action transition : $(s, \vartheta) \xrightarrow{\alpha} (s', \vartheta')$ if the following conditions hold:
  (a) there is a transition $s \xrightarrow{g,\alpha}_p s' \;\; t_s \in \mathbb{R}$
  (b) $\vartheta + t_s \models g$
  (c) $\vartheta' = (v + t_{s'})[X := 0]$
  (d) $\vartheta + t_s \models \mathit{Inv}(s)$
- Delay transition : $(s, \vartheta) \xrightarrow{d} (s, \vartheta + d), d \in \mathbb{R}$
  (e) if $\vartheta + d \models \mathit{Inv}(s)$

For a transition that corresponds to (a) traversing a transition $s \xrightarrow{g,\alpha}_p s'$ in PTA it must hold that (b) $\vartheta$ satisfies the clock constraint g (ensuring the transition is enabled), and (c) the new clock valuation in $s'$ is reset, should (d) satisfy the state invariant of $s$ ($\vartheta + t_s \vDash Inv(s)$). To remain in the same state (e) if the location invariant remains true while time progresses.

A reward structure are defined at the level of PTA equivalently referred to as costs or prices using a pair r = ($r_S$, $r_{Act}$), where $r_S : S \to R^{\geq 0}$ is a function assigning to each state the rate at which rewards are accumulated as time passes in that location and $r_{Act} : S \times Act \to R^{\geq 0}$ is a function assigning the reward of executing each action in each state.

**Definition 3** (Parallel composition of PTAs).
The parallel composition of PTAs $M_1 = (\overline{s}_1, S_1, X_1, Act_1, inv_1, Prob_1, L_1), M_2 = (\overline{s}_2,$

$S_2$, $X_2$, $Act_2$, $inv_2$, $Prob_2$, $L_2$) is the PTA M= $M_1||M_2 = (\overline{s}_1 \times \overline{s}_2, S_1 \times S_2, X_1 \cup X_2, Act_1 \cup Act_2, inv$, Prob, L(s1) $\cup$ L(s2)): where $Prob(S1 \times S2, Act1 \cup Act2)$ is the set of transitions, such that that one of the following requirements is met [39].

1. $s_1 \xrightarrow{t_1,\alpha} \mu_{(s_1,\vartheta+t_1)}, s_2 \xrightarrow{t_2,\alpha} \mu_{(s_2,\vartheta+t_2)}$, and $\alpha \in Act_1 \cap Act_2$ , $t_1 \in X_1$, $t_2 \in X_2$,
2. $s_1 \xrightarrow{t_1,\alpha} \mu_{(s_1,\vartheta+t_1)}, \mu_{(s_2,\emptyset)} = [s_2 \mapsto 1]$, and $\alpha \in Act_1 \setminus Act_2$, $t_1 \in X_1$,
3. $\mu_{(s_1,\emptyset)} = [s_1 \mapsto 1], s_2 \xrightarrow{t_2,\alpha} \mu_{(s_2,\vartheta+t_2)}$, and $\alpha \in Act_2 \setminus Act_1$, $t_2 \in X_2$.

If PTA $M_i$ has associated with rewards structure $(r_s^i, r_{Act}^i)$, then the reward structure of r=$(r_s, r_{Act})$ for $M_1||M_2$ :

1. $r_s^i(s1, s2) = r_s^1(s1) + r_s^2(s2), r_{Act}((s1, s2), \alpha) = r_{Act}^1(s1, \alpha) + r_{Act}^2(s2, \alpha)$, and $\alpha \in Act_1 \cap Act_2$
2. $r_{Act}((s1, s2), \alpha) = r_{Act}^1(s1, \alpha)$, and $\alpha \in Act_1 \setminus Act_2$
3. $r_{Act}((s1, s2), \alpha) = r_{Act}^2(s2, \alpha)$, and $\alpha \in Act_2 \setminus Act_1$

As example, we present in Fig.1.1 the probabilistic timed automata of selected behavior in CSMA-CD Protocol [10] designed for networks with a single channel and specifies the behavior of stations with the aim of minimizing simultaneous use of the channel (data collision). The probabilistic timed automaton representing a single station is given below. $\lambda$ to the time to send a data packet. The actions in each node are described by $send_i$, $end_i$ and $busy_i$. each transition depends on the clock node constraints $x_i$.

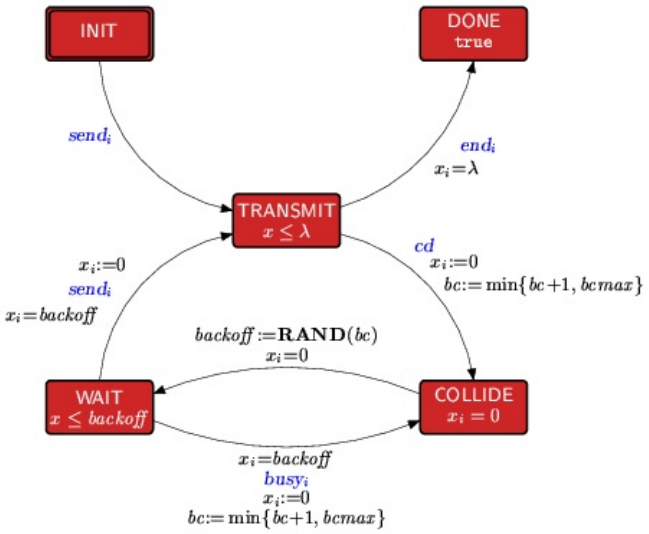



FIGURE 1.1: CSMA station

## 1.3 Software Architecture

Software product is a combination of a set of procedures, modules and objects that provides some functionality. For the sake of being precise; architecture is defined to give some organization of different elements of the software. Therefore, it is essential that the software product be designed to address the full set of functional behaviors that must be exhibited by the final product. The software architecture represents the decomposition of requirements into the functions and subfunctions that are necessary to provide the specified behavior and performance characteristics [19]. This decomposition follows different styles established by the software researchers community and cited by Garlan et al. [20]:

- Module style: A module is an implementation unit that provides a coherent set of responsibilities. A module might take a form of a class or a collection of classes. Each module has a set of properties express the informations associated with modules. Mainly relations are based on compositions.

- Component-and-connector (C&C) style: A component is one of the principal

processing units of the executing system. Component might be services, processes, class, object, server, client. A connector is the interaction mechanism among components. Connectors includes call communication protocols such as remote method invocation. Connector is a component with communication as behavior.

- Allocation style : Describes the mapping of software units to elements of environment in which software is developed or executed. For example: mapping the software in C language to the Micro-blaze Processor on field-programmable gate arrays (FPGAs). The goal of allocation style is to compare the properties of the software elements with properties provided by the environment to determine whether the allocation will be successful or not[21].

In our methodology, the concepts of system modeling are based on Component-and-connector (C&C) and Allocation styles. Components represent the principal computational elements that are presents at runtime. Components have interfaces called ports. A Port defines a specific point of interactions of a component with the rest of the assembly (i.e. a set of components). The example in Fig.1.2 show the internal and external representation of component following SaveCCM component model [22] where the main observed elements: 1)A delegation port that maps the internal port to the external port, 2) SaveCCM standard component, 3) linking required to provided interface (i.e. Attachment), 4) A wrapper component which is a kind of an envelope, 5) event port which is a kind of a port that mainly support boolean types. We note that the component model has its proper formal notation referred as Architecture Description languages (ADLs), it provides graphical and textual representation. ADLs are generally defined with their underling semantic and grammar for automation in case of architecture analysis. In case of C&C and allocation styles the automated tool can extract the relevant informations from software components (i.e. Tasks) such as minimum and maximum execution time with processors characteristics to establish a best tasks scheduling.

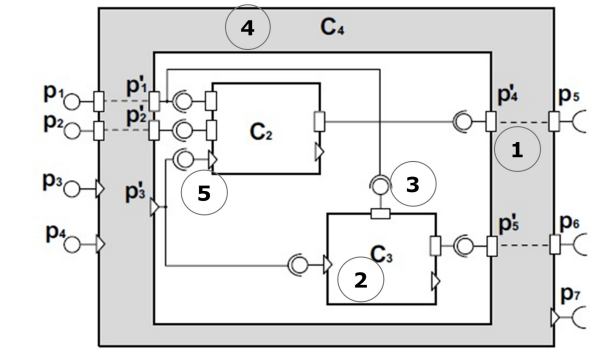



FIGURE 1.2: SaveCCM component model

Now, we surround the existing component models that allow us to represent the architecture of our assembly. The best existing work that surveys the component model was established by Hošek et al. [23]. We refer to [23] to surveys the recent works. AUTOSAR (Automotive Open System Architecture) [24] dedicated for the automotive industry, AUTOSAR distinguishes atomic software components, representing pieces of functionality, and compositions, representing logical interconnection of components. An atomic component is limited to one electronic control unit (ECU). However, real-time properties are not clearly specified. Pin Component technology was dedicated [25] to design and implement predictable real-time software with support for the UML state-charts semantic. Each Pin component consists of a container and a custom code. Real-time features are provided via a support of an underlying external commercial environment. SaveCCM [22] dedicated for automotive embedded system specification. Three kinds of component exists basic component, switches (i.e. connector role) and assemblies (i.e. a set of component) with support for the timed state-charts semantic. BIP component model (behavior, interaction, priority ) was principally dedicated for modeling real time embedded software. The BIP model [26] uses connectors to specify possible interactions between components and priorities to select among possible interactions with support for the state-charts semantic. Architecture Analysis and Description language (AADL) language [27] dedicated firstly for the avionic industry. AADL allows the specification of the software and the hardware component. The software components are thread, data,

TABLE 1.1: Comparison of Component Models

| Component Model | Behavior | Extendability |
|---|---|---|
| SaveCCM | Timed State Chart | |
| PIN | UML State Charts | |
| AUTOSAR | | |
| BIP | State Charts | |
| AADL | Modes, Behavior Annex | √ |
| SysML | Activity, Sequence, State machine | √ |

subprograms and hardware covers the physical platform such as processor, buses and memory. The behavior is expressed through modes and ”behavior annex“ which is a kind of state chart. However, for extendability; the user need to define its proper grammar. In literature we found a lot of methodologies based on UML for component specification such as MoPCoM co-design methodology [28], ENOSYS[29] and COMPLEX[30] due to its extendability based on the MARTE profile standard. System Modeling Language (SysML) [2] reuses a set of UML diagrams (Component diagrams, class diagrams) with additional properties to express probability and allocations and it is strongly recommended since its support three forms of behavior specification: (State machine diagram, Activity diagram and Sequence diagram) in addition to there compositions. To sum up, Table.1.1 compares the surveyed component model based on their behavior expressibility and extendability. We observe that SysML component model is more expressive than the other component-based Models.

## 1.4 SysML Behavioral Diagrams

In this section, we explore the SysML behavioral diagrams. The diagrams considered are: *Sequence*, *State Machine* and *Activity* behavioral diagrams.

### 1.4.1 Sequence diagrams

Sequence diagrams [31] describe the interactions within a system (i.e. an instance of the block that owns the interaction) using communicating entities. An interaction is a communication based on the exchange of messages in the form of operation calls or signals arranged in a temporal order. The principal structural feature of an interaction is the lifeline. A lifeline represents the relevant lifetime of the interaction's owning block where message is a communication type between two lifelines. The Diagrams offer possibility to model accurately the message exchange using the asynchronous and synchronous messages using control structure (loop and conditions). In addition, time can be represented explicitly on sequence diagrams (time and duration observation).

### 1.4.2 State machine diagrams

The UML 2 state machine diagram is reused by SysML specification [31]. State machine are used to describe the state-dependent behavior of a block throughout its life cycle in terms of its states and the transitions between them. State machines are normally owned by blocks and execute within the context of an instance of that block. The states in any one region are exclusive; that is, when the region is active, exactly its initial pseudostate become active. When a state is entered, an (optional) entry behavior (e.g., an activity) is executed. Similarly on exit, an optional exit behavior is executed. While in a state, a state machine can execute a *do behavior*. A region also normally has a final state that, when active, signifies that the region has completed. Change of state is effected by transitions that connect a source state to a target state. Transitions are defined by triggers, guards, and effects. The trigger indicates an event that can cause a transition from the source state, the guard is evaluated in order to test whether the transition is valid, and the effect is a behavior executed

once the transition is triggered. Triggers may be based on a variety of events such as the expiration of a timer, or the receipt of a signal. In addition to initial and final pseudostate, control nodes supports a junction, choice, join, fork, terminate and history pseudostate node.

### 1.4.3 Activity diagrams

In SysML, an activity is a formalism for describing behavior that specifies the transformation of inputs to outputs through a controlled sequence of actions [31]. The activity diagram is the primary representation for modeling flow-based behavior in SysML. Moreover, UML activity diagrams [32] can capture the behavior of a process or system in a wide variety of domains for use cases detailing computational, business, and other workflow processes as well as modeling in general. An activity may be composed of a set of actions coordinated sequentially and/or concurrently. Furthermore, the activity may also involve synchronization and/or branching. These features can be enabled by using control nodes including fork, join, decision, and merge. Activity diagram is essentially employed for verification and described in Chapter 2.

## 1.5 System Requirements Specification

This section introduces (propositional) temporal logic ; a logical formalism that is suited for specifying requirements. The syntax and semantics of linear temporal logic are defined. Two commonly used temporal logics are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) . LTL is called linear, because the qualitative notion of time is path-based and viewed to be linear: at each moment of time there is

only one possible successor state and thus each time moment has a unique possible future [9]. Whereas in CTL the qualitative notion of time is state-based and it is possible to quantify over the paths departing from a given state [9].

### 1.5.1 Temporal Logic

Time in temporal logic is not explicitly mentioned. CTL [33] is an important branching temporal logic that is sufficiently expressive for the formulation of an important set of system properties. Branching time refers to the fact that at each moment there may be several different possible futures. Thus, the future is nondeterministic as any of these paths can be considered as the future path. The temporal operators in branching temporal logic allow the expression of properties of some or all computations that start in a state. To that end, it supports an existential path quantifier (denoted $\exists$) and a universal path quantifier (denoted $\forall$). The second temporal operators that reason over states, where X means “next state” and U means “until”. CTL *state formulae* over the set AP of atomic proposition are formed according to the following grammar:

$$\varphi ::= true \mid ap \mid \varphi \wedge \varphi \mid \neg\varphi \mid \forall\psi \mid \exists\psi$$

where ap ranges over a countable set of atomic formulas. CTL path formulae are formed according to the following grammar:

$$\psi ::= X\varphi \mid \varphi_1 \cup \varphi_2$$

where $\varphi$, $\varphi_1$ and $\varphi_2$ are state formulae. Contrarily to CTL, LTL has no path quantifiers such as $\forall$ and $\exists$. However, LTL has same operators as CTL such as X “next state”

and U “until”. The syntax for LTL is expressed in BNF as follows [9]:

$$\varphi ::= true \mid ap \mid \varphi \wedge \varphi \mid \neg\varphi \mid X\varphi \mid \varphi_1 \cup \varphi_2$$

where ap is any atomic proposition. An LTL formula is evaluated on a single path, or on a set of paths. A formula f holds on a set of paths if it holds on every path in the set [**?** ].

### 1.5.2 Probabilistic Temporal Logic

To verify probabilistic systems, we present a temporal logic for the formal specification of quantitative properties of PA and PTAs. The basis for this is the temporal logic PCTL [10] [34], a probabilistic extension of the logic CTL [33] which has been proposed for specifying properties of both PAs [35] and discrete-time Markov chains [36]. The syntax of temporal logic is given by the following grammar:

$\varphi$ :: =$true \mid ap \mid \varphi \wedge \varphi \mid \neg\varphi \mid P_{\bowtie p}[\psi] \mid R^{r}_{\bowtie q}[\rho]$,
$\psi$ :: = $\varphi \cup^{\leq k} \varphi \mid \varphi \cup \varphi$ ,
$\rho$ :: = $I^{=k} \mid C^{\leq k} \mid F\ \varphi$

Where ”ap” is an atomic proposition, P is a probabilistic operator and R is a reward. Operator $P_{\bowtie p}[\psi]$ means that the probability of path formula $\psi$ being true always satisfies the bound $\bowtie$ p, p $\in$ [0, 1]. $R^{r}_{\bowtie q}[\rho]$ means that the expected value of reward function $\rho$ on reward structure r meets the bound $\bowtie$ q, q $\in \mathbb{Q}$. ”$\bowtie$”$\in <, \leq, >, \geq$. ”$\wedge$” represents the conjunction operator and ”$\neg$ ” is the negation operator. Two paths formulas are included bound until $\varphi 1 \cup \varphi 2$ and time-bound until $\varphi 1 \cup^{\leq k} \varphi 2$ . Bound until means that a state satisfying $\varphi$2 is eventually reached and that, at every time-instant

prior to that, $\varphi 1$ is satisfied. The time-bounded variant has the same meaning, where the occurrence of $\varphi 2$ occur within time k. The reward operator I$^{=k}$ refers to the reward of the current state at time instant k, C$^{\leq k}$ refers to the total reward accumulated up until time point k, and F $\varphi$ to the total reward accumulated until a state satisfying $\varphi$ is reached, e.g:

- R$^{time}_{max}$ =?[F success] : "What is the expected reward accumulated before the system successfully terminates?".
- P$min$ =?[true $\cup^{\leq 100}$ Complete] : The minimum probability of the system eventually completing its execution successfully after 100 time units?".

Other useful operators can be derived such as:

- $true \equiv \neg false$
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$
- $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \iff \varphi_2 \equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$
- Future: $F\varphi \equiv true \cup \varphi$ or $F^{\leq k} \equiv true \cup^{\leq k} \varphi$ and k $> 0$.
- Generally: $G\varphi \equiv \neg(F\neg\varphi)$ or $G^{\leq k}\varphi \equiv \neg(F^{\leq k}\neg\varphi)$ and $k \geq 0$.
- $P_{\geq p}[G\varphi] = P_{\leq 1-p}[F\neg\varphi]$.

Let P is PTA and [[P]]=($S$, $\overline{s}$, $\mathbb{T}$, Act, lab, Step$_P$) its semantics where S is a set of states, $\overline{s}$ is the initial state, $\mathbb{T}$ is a set of clocks, lab is a labeling function and Step$_P$ : $S \times Act \longrightarrow Dist(2^x \times S)$ is a probabilistic transition function and let **r** denotes the reward structure over [[P]] corresponding to the reward structure over P. The satisfaction relation of a PCTL formula [37]is denoted by "$\models$" and defined as follows where $(s, \vartheta)$ is a location and *Path* is a sequence of states:

- $s, \vartheta \models$ true is always satisfied,
- $s, \vartheta \models ap \iff ap \in L(s)$ and L is a labeling function,

- $s, \vartheta \vDash \varphi_1 \wedge \varphi_2 \Longleftrightarrow s, \vartheta \vDash \varphi_1 \wedge s, \vartheta \vDash \varphi_2$,
- $s, \vartheta \vDash \neg\varphi \Longleftrightarrow s, \vartheta \nvDash \varphi$,
- $s, \vartheta \vDash P_{\bowtie p}[\psi] \Longleftrightarrow Prob^{\sigma}_{[[P]]}(\pi \in Path_{s,\vartheta} | \pi \vDash \psi) \bowtie p$, for all $\sigma \in Adv_{[[P]]}$,
- $s, \vartheta \vDash R^{r}_{\bowtie q}[\rho] \Longleftrightarrow Exp^{\sigma}_{[[P]]}(rew(r, \rho)) \bowtie q$, for all $\sigma \in Adv_{[[P]]}$.

Where for any finite path $\pi$ of[[P]]:

- $\pi \vDash \varphi_1$ U$^{\leq k}\varphi_2 \Longleftrightarrow$ There is a position $(i, t)$ of $\pi$ such that $\pi(i) + t \vDash \varphi_2$and $dur_{\pi}(i) + t \leq k$ and $\pi(j) + t' \vDash \varphi_1 \vee \varphi_2$ for all positions $(j, t') \prec (i, t)$ of $\pi$ ,
- $\pi \vDash \varphi_1$ U $\varphi_2 \Longleftrightarrow$ There is a position $(i, t)$ of $\pi$ such that $\pi(i) + t \vDash \varphi_2$ and $\pi(j) + t' \vDash \varphi_1 \vee \varphi_2$ for all positions $(j, t') \prec (i, t)$ of $\pi$.

For reward structure r=$(r_S, r_{Act})$ over [[P]], the random variable $rew(r, \rho)$ over infinite paths of [[P]] is defined as follows:

$$rew(r, I^{\leq k})(\pi) = r_S(\pi(j_k)),$$

$$rew(r, C^{\leq k})(\pi) = \textstyle\sum_{i=0}^{j_k - 1} r(\pi, i) + (k - dur_{\pi}(j_k)).r_S(\pi(j_k)),$$

$$rew(r, F\varphi)(\pi) = \begin{cases} \sum_{i=0}^{j_\phi - 1} r(\pi, i) + t_\phi.r_S(\pi(j_\phi)) & if (j_\phi, t_\phi)\ exists, \\ \infty & otherwise \end{cases}$$

where $j_0 = 0, j_k = max\{i \mid dur_{\pi}(i) < k\}$ for $k > 0$ and, when it exists, $(j_\phi, t_\phi)$ is the minimum position under the ordering $\prec$ such that $\pi(j_\phi) + t_\phi \vDash \phi$.

## 1.6 Verification Approaches

In this section, we present the verification approaches needed for both non-probabilistic and probabilistic systems.

### 1.6.1 Non-Probabilistic verification

For a transition system M, the verification procedure determines the subset of states from S with respect to a property $\varphi$ of the logic presented in Section 1.5.1 ,i.e., determining $Sat(\varphi) = def\{s \in S | s \models \varphi\}$, where S is the set of states. As an example, Algorithm 1 returns the set of states that satisfies the CTL formula $\exists[\varphi_1 \cup \varphi_2]$. First, it looks for the states that satisfy the formula $\varphi_2$. Then, it searches backward for states satisfying the formula $\varphi_1$ [9].

Algorithm 1 [9] Computation of the satisfaction sets of CTL property = $\exists[\varphi_1 \cup \varphi_2]$.
**Input** : Transition system M and CTL property = $\exists[\varphi_1 \cup \varphi_2]$
**Output** : T a set of state satisfying $\varphi$

```
1 T := Sat(φ2); (* compute the greatest fixed point *)
2 While { s ∈ T | Post(s) ∩ T = ∅} ≠ ∅ do
3 let s ∈ {s ∈ T | Post(s) ∩ T = ∅};
4 T := T \ {s};
5 end do;
6 return T ;
```

### 1.6.2 Probabilistic Verification

In this section, we present the computation technique used in symbolic model checker PRISM [10]. In our work, the model checking methodology is based on probabilistic timed Automata. The technique is based on:

- Digital clocks based on MDP,
- Abstraction refinement with stochastic games.

To show that, we select PTA defined in Definition 2.2 as a special formalism of probabilistic timed automata that exhibits both probabilistic and nondeterministic behaviors. To reason formally about MDPs (PTA based digital clocks), we need a probabilistic space over it. And, as it is a nondeterministic behavior, the adversary notion is introduced to decide which action should be chosen in any state of the MDP. In general, the choice is made depending on the history execution of the MDP. The Definition 2.4 describes the adversary function.

**Definition 2.4** (Adversary). An adversary of an MDP $M = (S, s, \alpha_M, \delta_M, L)$ is a function $\sigma : FPathM \rightarrow Dist(\alpha_M)$ that maps every finite path of a system into a distribution, where:

- $FPath_M$ is a finite set of states,
- $Dist(\alpha_M)$ is a labeled function assigning to each state of the automaton the set of atomic propositions that are true in that state.

Probabilistic reachability refers to the minimum/maximum probability with which a given set of states of a probabilistic system ($T \subseteq S$) can be reached from a particular state (s). To this end, reachs (T) is the set of paths that start from s and contain a state from T, and $IPathM$,s defines the infinite paths starting from a state s in M.

reachs (T) = $\{\pi \in IPathM, s|\pi(i) \in Tandi \in \mathbb{N}\}$

where i is the (countable) set of all finite paths from s ending in T, and each element of this union is measurable through three ways: value iteration, the linear programming problem or policy iteration [**?** ]. This is equivalent to computing the probabilistic bounds of the reached paths:

$$P^{min}_{M,s}(reach_s(T)) = Inf_{s \in Adv_M} Probs^{\sigma}_{M,s}(s, \psi)$$

$$P^{max}_{M,s}(reach_s(T)) = Sup_{s \in Adv_M} Probs^{\sigma}_{M,s}(s, \psi)$$

Digital clocks restricts the standard continuous-time semantics of a PTA so that only time transitions of duration 1 occur. This means that clocks take only integer, rather than real values. Using this fact, and knowing there is a maximal constant cx to which each clock *x* is compared in PTA M and property $\varphi$, we can again build and analyse a finite-state MDP. The digital clock semantics of M and $\varphi$ is defined as for the standard semantics, except that the rule for time transitions restricts durations to 1, and each clock *x* can increase to at most *V(x)* + 1. Formally, the digital clock semantics of a PTA M is defined as:

There are two possible ways in which a PTA based digital clock can proceed by taking a transition Or by letting time progress while remains in a state (Delay transition):

- Action transition : $(s, \vartheta) \xrightarrow{\alpha} (s', \vartheta')$ if the following conditions hold:
  - (a) there is a transition $s \xrightarrow{g,\alpha}_p s' \;\; t_s \in \mathbb{R}$
  - (b) $\vartheta + 1 \models g$
  - (c) $\vartheta' = (v + t_{s'})[X := 0]$
  - (d) $\vartheta + 1 \models \mathit{Inv}(s)$
- Delay transition : $(s, \vartheta) \xrightarrow{1} (s, \vartheta + 1), d \in \mathbb{R}$
  - (e) if $\vartheta + 1 \models \mathit{Inv}(s)$

The second model checking technique we discuss for PTAs is abstraction refinement using stochastic games [38]. This approach uses the game-based notion of abstraction. This approach [37]is to build an abstraction of a large or infinite-state MDP based on a finite partition of its state space. Abstractions take the form of

stochastic two-player games, a generalization of MDPs in which there are two distinct types of nondeterminism, each controlled by a separate player. This technique is adapted by an algorithm to compute exact (minimum or maximum) reachability probabilities for PTAs. First, a reachability graph is constructed, based on a successor operation that returns the set of states that can be reached by performing an action and then letting time pass. The abstraction is repeatedly analyzed and refined until the exact required probabilities are obtained.

## 1.7 Probabilistic Verification Tools

In this section we compare existing probabilistic model checkers. Our selection was based on two main features: the kind of models that they can support, and the temporal logic that they use to specify properties.

According to Table.1.2 we compare the well known used tools for embedded systems verification. Through our research, we found that PRISM is mainly used for verification because it is not only endowed with all verification techniques but due to its scalability. We have shown that the tool recently apply two methods for design abstraction namely Stochastic Games and Digital Clock. In addition, the models can be described using the PRISM language as discrete-time Markov chains, continuous-time Markov chains, Markov decision processes (MDPs) or probabilistic timed automata (PTA) and we can see here that PRISM is a complete tool,

TABLE 1.2: Model Checkers vs. Supported Formal Models

| Tool | CTMC | PTA | PA | TA | State Explosion |
|---|---|---|---|---|---|
| Fortuna [40] | | Maximum Probability | √ | | Digital Clock |
| NuSMV [41] | | | | Programmable Clock | |
| ProLog [42] | | | | Digital Clock | |
| ProbLog [43] | | Digital Clock | √ | Digital Clock | |
| UPPAAL [44] | | | | √ | |
| MRMC [20] | √ | | | | |
| Modest [45] | √ | √ | √ | √ | |
| PRISM | √ | √ | √ | √ | Stochastic Games/Digital Clock |

## 1.8 Conclusion

In this chapter, we introduced the concepts used in software architecture and we provided a comparison between the well known component models. In addition, we provided the main concepts for system behavioral verification needed for the rest of the thesis. Also, we presented a comparison between the existing semantic models, temporal logics, the verification techniques, and the existing verification tools. In the next chapter, we propose a model-checking based methodology for the verification of SysML activity diagrams under time constraints.

# CHAPTER 2

# A QUANTITATIVE VERIFICATION FRAMEWORK OF SYSML ACTIVITY DIAGRAMS UNDER TIME CONSTRAINTS

## 2.1 Introduction

Time-constrained and probabilistic verification approaches has gained a great importance in system behavior validation including avionic, transport risk assessment, automotive systems and industrial process controllers. They enable the evaluation of system behavior according to the design requirements and ensure their correctness before any implementation. Due to the difficulty of analyzing, modeling and verifying these large scale systems, we introduce a novel verification framework based on PRISM probabilistic model checker that takes the SysML activity diagram under time constraints as input and produce their equivalent timed probabilistic automata that is expressed in PRISM language. To check the functional correctness of the system under test, the properties are expressed in PCTL temporal logic. To prove the soundness of our mapping approach, we capture the underlying semantics of

both SysML activity diagrams and their generated PRISM code. We found that the timed probabilistic equivalence relation between both semantics preserve the satisfaction of the system requirements. We present digital camera as a case study to illustrate the applicability of the proposed approach and to demonstrate its efficiency by analyzing performability properties.

## 2.2 Related work

In this section, we provide the state-of-the-art related to the verification of behavioral models then we contrast them to our proposed approach.

Ermeson et al.[12],[13] use SysML language and MARTE UML Profile (Modeling and Analysis of Real-Time and Embedded systems) to specify ERTSs (Embedded Real-time Systems) constraints such as execution time and energy [39]. They map only the states and the transitions into ETPN (Time Petri Net with Energy constraints). Their approach is restricted to a subset of artifacts with control flow (data flow is missed). Michal and Marian[14] propose a verification and simulation of UML state machine. For this purpose, two mapping mechanisms were defined. The first consists on mapping the original model to Petri Network (PN) for verification according to the requirements. When the requirements are satisfied, the second mapping occurs to generate VHDL or Verilog description for simulation. The data on each transition is considered as a trigger for a new state. The formalization is restricted on Petri Networks. Huang et al.[40] and Knorreck et al.[41] propose a verification of SysML State Machine Diagram by extending the model with MARTE [39] to express the execution time. The tool has as input the state machine diagram and as output Timed Automata (TA) expressed in UPPAAL syntax [42]. UPPAAL uses CTL (Computational Tree Logic) properties to check if the model is satisfied with liveness and safety properties. In [43], Akerholm et al. define a mapping rules that takes as

input a SaveCCM component Model [22] and extracts different tasks. The results are encoded in UPPAAL Syntax. Cafe et al.[15] develop a framework that maps the SysML diagrams describing the heterogeneous system (hardware and software) to SystemC language. The inputs are a set of SysML diagrams such as block diagram, parametric diagram and behavioral diagram (e.g. Activity or state machine diagram). The result is simulated to check the satisfaction of timing and functional requirement. In [44], the simulation based verification is based on Simulink [45] but timing verification is done by UPPAAL. In [46], Yan et al. use real-time model checking tool UPPAAL to find the possible best throughput and response time of activity diagram extended by MARTE annotations. However, The mapping rules concern a subset of activity diagrams artifacts and did not include a call behavior or sub activities. Lasnier et al.[27] develop a framework for automatic generation of embedded real time applications in C/C++/ADA. The input is AADL language (Architecture Analysis and design Language) [47] with typed software components (e.g. threads, data) and hardware components (e.g. processor, bus, memory). In addition, The design model is enriched with time execution properties (e.g. computation time, priority, deadline and scheduling algorithms) [48]. A scheduling tool is used for timing estimation and C/C++/ADA code is generated for simulation. Yosr et al.[16] propose a probabilistic verification of SysML activity diagram where the execution time of actions are represented as constraints (i.e. A note artifact in SysML Activity diagram). The diagram is translated to its corresponding Discrete-Time Markov chain (DTMC) and use PRISM model-checker for performance evaluation using PCTL. The approach is restricted on a sub set of SysML activity diagram constructs with control flow (data flow is missed). However, the mapping soundness was not proved. Debbabi et al.[49] translate SysML activity diagrams into the input language of the probabilistic model-checker PRISM. The authors present the algorithm implementing the translation of SysML activity diagrams into Markov Decision Process (MDP) supported by PRISM model-checker. The mapping approach is limited to only the control flow artifacts without including objects and time except in [50] where objects are included. AccordingOuchani et al.[18], The selected artifacts have been

formalized and a verification algorithm has been proposed for mapping these artifacts to PRISM language. The transformation result is a probabilistic automata to be checked by PRISM. The mapping approach is sound but the approach is limited to only the control flow artifacts without including object and time. Grobelna et al.[51] present an approach to verify the correctness of UML activity diagrams for logic controller specification especially for embedded systems design. The authors formalize the activity diagram using rule-based logic model [52]. The result model is mapped to NuSMV[53] in order to to verify system model against behavioral properties expressed in Linear Temporal Logic (LTL) formulas to detects some errors. When system is error free, the transformation of logical model into synthesizable model in VHDL language. The same approach is done on state machine diagram by Rodriguez et al.[54]. However, the soundness of the approach was not proved as in [55] and [56]. Anuarul et al.[57] introduce a methodology based on probabilistic model checking to analyze the dependability and performability properties of safety-critical systems especially in aerospace applications. Starting from a high-level description of a model, a Continuous-Time Markov Chains (CTMC) is constructed from the Control Data Flow Graph (CDFG) with different sets of available components and their possible failures, fault recovery and repairs in the radiation environment. The methodology is based on PRISM model checker tool to model and evaluate dependability, performability and area trade-offs between available design options using PCTL.

### 2.2.1 Comparison

As a summary, in Table.2.1 we compare our framework to the existing works by taking into consideration five criteria: SysML language, time constraints, data workflow, formalization, soundness and automation. The SysML criteria shows if an approach covers the probabilistic systems. The time Constraint criteria shows if the

approach includes the time specification at the design level. The data workflow criteria indicates if data objects exist at specification level. The formalization criteria confirms if the approach presents a semantics and formalizes the studied diagrams. The soundness feature shows if the mapping soundness of the studied approach is proved. The automation criteria checks if the presented approach provides a tool. From the comparison, we observe that only few works formalize the behavioral diagrams, including data workflow and time constraint at the specification level. Our works has for objective to support: data workflow, time constraints, and formalize the SysML activity diagram. In addition, we implement a tool that allows the automatic verification.

## 2.3 Preliminaries

### 2.3.1 SysML Activity Diagram

SysML Activity diagram is a graph-based diagram where activity nodes are connected by activity edges. The activity diagrams is a primary representation for modeling flow based behavior in [2]. Fig.2.1 shows the set of interesting artifacts used for verification in our framework. The artifacts consist of activity nodes, activity edges, activity control and constraints. Activity nodes have three types: activity invocation, objects and control nodes. Activity invocation includes action, call behavior, send/receive signals (objects). Activity control includes: flow initial, flow final, activity final, fork, merge and join node. Activity edge includes: control flow and object flow. Object flow can connect the output pin of one action to the input pin of next action to enable the passage of tokens. Control flow provides additional constraints on when and in which order the action within an activity will be executed. A control token on an incoming control flow enables the execution of an action and offers a control

TABLE 2.1: Comparison with the Existing Works

| Approach | SysML | Time Constraint | Data workflow | Formalization | Soundness | Automation |
|---|---|---|---|---|---|---|
| [55], [51], [54]. | | | | | | √ |
| [50]. | √ | √ | √ | | | √ |
| [27], [43], [12], [13], [41]. | √ | | | | | √ |
| [57], [56], [44], [46]. | | √ | | | | √ |
| [18]. | √ | | | √ | √ | √ |
| [40], [16]. | √ | | | √ | | √ |
| [15], [49]. | √ | | | | | √ |
| Our | √ | √ | √ | √ | √ | √ |

token on outgoing control flow when action completes its execution. Control nodes such as join, fork, merge, decision, initial and final are used to control the routing of control token over edges and specify the sequence of actions (concurrency, synchronization). In addition, the constraints can be used as SysML notes stereotyped with ”Local Post Condition” and ”Local Pre Condition”. Next, we define the rules for actions to begin and end execution during the activity workflow and we show the expression manner of time in activity diagram.

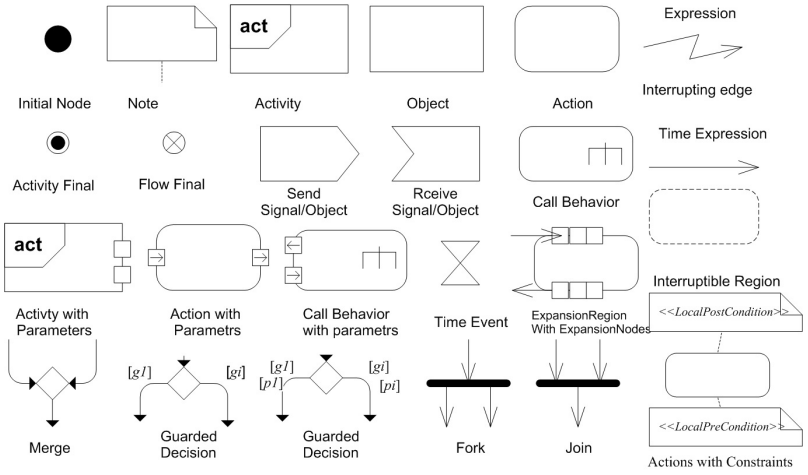



FIGURE 2.1: A sub set of SysML activity diagram artifacts

#### 2.3.1.1 Actions Execution

The following rules [31] describe the requirements for actions to begin and end execution during the activity workflow:

a. The action owned by the activity must be executed.

b. The action is executed if the number of tokens at each required input pins is equal or greater than its lower multiplicity bound.

c. The action is executed if a token is available in each of the action’s incoming control flow.

d. Once these perquisites are met, the action will start executing and tokens at its input pins are available for consumption.

#### 2.3.1.2 Time Expression using SysML/MARTE

[31], [16] and [32] provide a manner to add the time constraints to actions with specialized form of constraint that can be used to specify the time duration of an action's execution. The constraint is shown using standard constraint notation: a note attached to the action which is constrained (see Fig.2.2). However, annotation of time constraints on top of SysML activity diagrams is not clearly defined in the standard [2].

According to OMG SysML [2] the actions can be triggered using wait time action artifact (Time Event). Wait time action artifact becomes enabled when control token arrives on its incoming control flow and the precedent action has finished. If the time event specified by execution time elapses then it completes immediately and offer token to its outgoing control flow, else waits for that time event to occur. If the Time event has not an incoming edge then it becomes enabled as soon as the activity begins executing.

MARTE is a new UML profile standardized by the OMG [39] aims to replace the UML profile SPT (Profile for Schedulability, Performance and Time) MARTE was defined to make quantitative predictions regarding real-time and embedded features of systems taking account both hardware and software characteristics. The *core concepts* are design and analysis parts. The design parts are used to model real time embedded systems. On the other hand, the analysis parts provide facilities to support the resources analysis such as execution time, energy and memory usage. In our thesis, we use execution time for quantitative verification.

Fig.2.2 illustrates how the probability value is specified on the outgoing edges of the decision nodes testing their corresponding guards. In addition, the time is specified using MARTE profile with the stereotype $\ll resourceUsage \gg$.

The action TurnOn requires exactly 2 units of time to terminate; Action AutoFocus terminates within the interval ]1,2[, The action TakePicture' execution time is negligible. To model probabilistic systems, the probabilities are assigned to edges emanating from decision nodes where the assigned values should sum up to 1. For instance, the decision node testing the guard **memFull** has the following semantic interpretation: the probability is equal to 0.2 that the outcome of the decision node will be (memFull=true) and the corresponding edge traversed.

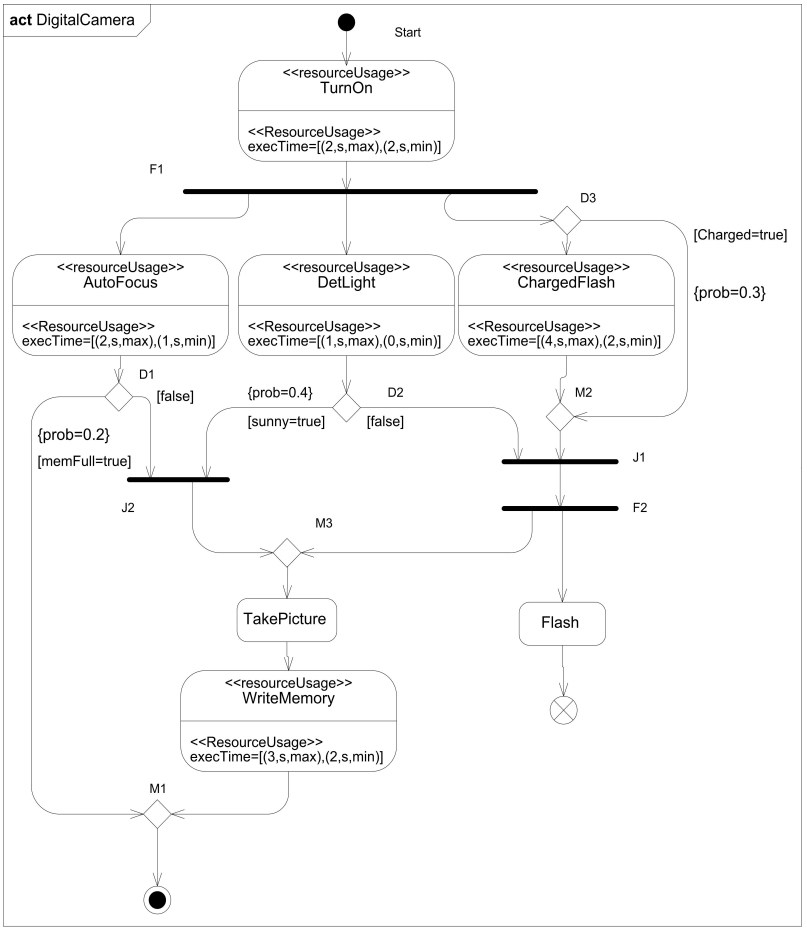



FIGURE 2.2: Digital camera activity diagram design

## 2.4 SysML activity diagram formalization

In this section, we formalize SysML activity diagram with extended artifacts by providing an adequate calculus.

### 2.4.1 SysML activity diagrams syntax

Based on textual specification of SysML [2] we formalize the SysML activity diagram by developing its Calculus : Timing Activity Calculus (TAC). In Table.2.2, each SysML artifact is represented and described formally by its related TAC term. The TAC terms in Fig.2.3 extends NuAC defined by [32] and enhanced by [18] adding the time and data. In this calculus we distinguish between two syntactic concepts: Marked and Unmarked terms. A marked term are activity diagrams with tokens. The unmarked terms correspond to the static structure of the activity diagram. A marked term denotes the configuration of diagram in the workflow process.

To support tokens we augment the "Over bar" operator with integer value n such that the $\overline{\mathcal{N}^n}$ denotes the term $\mathcal{N}$ marked with n tokens with the convention that $\overline{\mathcal{N}^1} = \overline{\mathcal{N}}$ and $\overline{\mathcal{N}^0} = \mathcal{N}$ . Multiple tokens can be observed when the loop encompass a fork in its body. Furthermore, we use a prefix label "*l* :" for each node to uniquely reference it in the case of a backward flow connection. Particularly, labels are useful for connecting multiple incoming flows towards merge and join nodes. Let $\mathcal{L}$ be a collection of labels ranged over by $l; l_0; l_1$,.. and $\mathcal{N}$ be any node (except initial) in the activity diagram. We write $l : \mathcal{N}$ to denote a l-labeled activity node $\mathcal{N}$. It is important to note that nodes with multiple incoming edges (e.g. join and merge) are visited as many times as they have incoming edges. Thus, as a syntactic convention we use only a label (i.e. *l*) to express a TAC term if its related node is encountered already.

We denote by *D*$(A, g, \mathcal{N}, \mathcal{N})$ and *D*$(A, p, g, \mathcal{N}, \mathcal{N})$ to express a non-probabilistic and a probabilistic decision, respectively. For the workflow observation on SysML activity diagrams, we use structural operational semantics [58] and [59] to formally describe how the computation steps of TAC atomic terms take place. We define **Act** as the set of actions labeling the transitions (i.e. the alphabet of TAC, to be distinguished from action nodes in activity diagrams). An element $\alpha$ is the label of the executing action node or x(y) inputs an object name on x and stores it in y. An element **t** is the time for action transition and **p** be probability values such that $p \in ]0, 1[$ . The general form of a transition is $A \xrightarrow{t,\alpha}_{p} A'$. The probability value specifies the likelihood of a given transition to occur and it is denoted by $P(A, t, \alpha, A')$. The case of p = 1 presents a non-probabilistic transition and it is denoted simply by $A \xrightarrow{t,\alpha} A'$. For simplicity, we denote by A[$\mathcal{N}$] to specify $\mathcal{N}$ as a sub-term of A and by $|A|$ to denote a term A without tokens. For the call behavior case of $a \uparrow \mathcal{N}$, we denote A[$a \uparrow \mathcal{N}$] by $A \uparrow_a \mathcal{N}$ . In the sequel, we describe the operational semantic rules of the TAC calculus.

1. **INT-1** $l : i \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : \bar{i} \rightarrowtail \mathcal{N}$ This axiom introduces the execution of $\mathcal{A}$ by putting a token on *i* .

2. **INT-2** $l : \bar{i} \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : i \rightarrowtail \overline{\mathcal{N}}$ This axiom propagates the token in the marked term *i* into its outgoing $\mathcal{N}$.

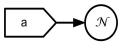
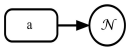
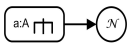
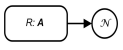
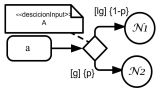
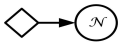
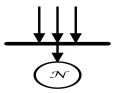
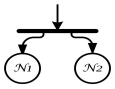
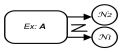
3. **ACT-1** $\forall n > 0, m \geq 0$ $l : \overline{\overline{a^m} \rightarrowtail \mathcal{N}}^n \xrightarrow{l}_1 l : \overline{\overline{a^{m+1}} \rightarrowtail \mathcal{N}}^{n-1}$ This axiom propagates the token from the global marked term to *a*.

4. **ACT-2** $\overline{\overline{a^{m+1}} \rightarrowtail \mathcal{N}}^n \xrightarrow{t,\alpha}_1 l : \overline{\overline{a^m} \rightarrowtail \overline{\mathcal{N}}}^n$ This axiom propagates the token from the marked term a to $\mathcal{N}$.

5. **ACT-3** $\dfrac{\mathcal{N} \xrightarrow{b(y)} \mathcal{N}'}{l:\overline{\overline{a^m} \rightarrowtail \mathcal{N}}^n \xrightarrow{b(y)}_1 l:\overline{\overline{a^m} \rightarrowtail \overline{\mathcal{N}'}}^n}$ The derivation rule ACT-3 illustrates the evolution of term $\overline{a^m} \rightarrowtail \mathcal{N}^n$ when $\alpha$ = b(y) inputs a name on b and stores it in y.

6. **EXP-1** $l : \overline{EX(p, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l'}_p l : \overline{EX(p, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$ EXP-1 derivation rule propagates the token from the global marked term to $\mathcal{N}_1$ with probability p .

TABLE 2.2: TAC Terms of SysML Activity Diagram Artifacts.

| Artifacts | TAC terms | Description |
|---|---|---|
|  | $l : i \rightarrowtail \mathcal{N}$ | Initial node is activated when a diagram is invoked. |
|  | $l : \odot$ | Activity final node stops the diagram' execution. |
|  | $l : \otimes$ | Flow final node kills its related path' execution. |
|  | $l : a?v \rightarrowtail \mathcal{N}$ | Receive node is used to receive a signal/object. |
|  | $l : a!v \rightarrowtail \mathcal{N}$ | Send node is used to send a signal/object. |
|  | $l : a \rightarrowtail \mathcal{N}$ | Action node defines an atomic action. |
|  | $l : a \uparrow B \rightarrowtail \mathcal{N}$ | Call behavior node invokes a new behavior. |
|  | $l : R \uparrow A \rightarrowtail \mathcal{N}$ | Region node invokes a sub actions behavior. |
|  | $l : D(\mathcal{A}, p, g, \mathcal{N}_1, \mathcal{N}_2)$ | Decision node with a call behavior $\mathcal{A}$, a convex distribution $\{p, 1-p\}$ and guarded edges $\{g, \neg g\}$. |
|  | $l : M(x) \rightarrowtail \mathcal{N}$ | Merge node specifies the continuation and $x = \{x_1, x_2\}$ is a set of input flows. |
|  | $l : J(x) \rightarrowtail \mathcal{N}$ | Join node presents the synchronization where $x = \{x_1, x_2\}$ is a set of input pins. |
|  | $l : F(\mathcal{N}_1, \mathcal{N}_2)$ | Fork node models the concurrency that begins multiple parallel control threads. |
|  | $l : Ex(p, \mathcal{N}_1, \mathcal{N}_2)$ | InterruptibleActivityRegion invokes a sub behavior that can be interrupted with probability=1-p. |

7. **BEH-0** $\forall n > 0\ l : \overline{a \uparrow \mathcal{A}^n} \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : \overline{a \uparrow \mathcal{A}^{n-1}} \rightarrowtail \mathcal{N}$ This axiom propagates the token from the global marked term to *a*.

8. **BEH-1** $\frac{\mathcal{A}=l':i\rightarrowtail\mathcal{N}\ \mathcal{A}'=l':\bar{i}\rightarrowtail\mathcal{N}}{l:\overline{a\uparrow\mathcal{A}^n}\rightarrowtail\mathcal{N}\xrightarrow{l}_1 l:\overline{a}\uparrow\overline{\mathcal{A}'^{n-1}}\rightarrowtail\mathcal{N}}$ BEH-1 axiom introduces the execution of the behavior $\mathcal{A}$ related to a.

9. **BEH-2** $\frac{\mathcal{A}[\overline{l':\odot}]\xrightarrow{l'}_1|\mathcal{A}|}{l:\overline{a\uparrow\mathcal{A}^n}\rightarrowtail\mathcal{N}\xrightarrow{l'}_1 l:\overline{a\uparrow|\mathcal{A}|^n}\rightarrowtail\mathcal{N}}$ The derivation rule BEH-2 finishes the execution of a call behavior and moves the token to the succeeding term $\mathcal{N}$.

10. **BEH-3** $\frac{\mathcal{A}\xrightarrow{t,\alpha}_p\mathcal{A}'}{l:\overline{a\uparrow\mathcal{A}^n}\rightarrowtail\mathcal{N}\xrightarrow{t,\alpha}_p l:\overline{a}\uparrow\overline{\mathcal{A}'^n}\rightarrowtail\mathcal{N}}$ The derivation rules BEH-3 and BEH-4 present the effect on $\overline{a \uparrow \mathcal{A}^n}$ when $\mathcal{A}$ or $\mathcal{N}$ executes an action a with a probability p.

11. **BEH-4** $\frac{\mathcal{N}\xrightarrow{t,\alpha}_1\mathcal{N}'}{l:\overline{a\uparrow\mathcal{A}^n}\rightarrowtail\mathcal{N}\xrightarrow{t,\alpha}_p l:\overline{a\uparrow\mathcal{A}^n}\rightarrowtail\mathcal{N}'}$

12. **FRK-1** $\forall n > 0$ *l* : $\overline{F(\mathcal{N}_1, \mathcal{N}_2)} \xrightarrow{l}_1$ *l* : $\overline{F(\overline{\mathcal{N}_1}, \overline{\mathcal{N}_2})}$ The FRK-1 axiom shows the multiplicity of the arriving tokens according to the outgoing sub-terms

13. **FRK-2** $\frac{\mathcal{N}_1 \xrightarrow{t,\alpha} \mathcal{N}_1'}{l{:}\overline{F(\mathcal{N}_1,\mathcal{N}_2)} \xrightarrow{t,\alpha} l{:}\overline{F(\mathcal{N}_1',\mathcal{N}_2)}}$ The FRK-2 derivation rule illustrates the changes on a fork term when its outgoing execute an action.

14. **DEC-1** $\forall n > 0$ *l* : $\overline{D(g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g,\alpha}$ *l* : $\overline{D(g, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$ The axiom DEC-1 describes a non-probabilistic decision where a token flows through the edge satisfying its guard.

15. **DEC-2** $\forall n > 0$ *l* : $\overline{D(p, g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g,\alpha}_p$ *l* : $\overline{D(p, g, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$ The axiom DEC-2 describes a probabilistic decision where a token flows through the edge satisfying its guard with probability p.

16. **DEC-3** $\frac{\mathcal{A}{=}l{:}i{\rightarrowtail}\mathcal{N}\,\mathcal{A}'{=}l'{:}\bar{i}{\rightarrowtail}\mathcal{N}}{l{:}\overline{D(\mathcal{A},p,g,\mathcal{N}_1,\mathcal{N}_2)}^n \xrightarrow{l}_1 l{:}\overline{D(\mathcal{A}',p,g,\mathcal{N}_1,\mathcal{N}_2)}^{n-1}}$ DEC-3 axiom shows a transition of probability one to initiate an invoked behavior.

17. **DEC-4** $\frac{\mathcal{A}[\overline{l':\odot}] \xrightarrow{l'}_1 |\mathcal{A}|}{l{:}\overline{D(\mathcal{A},p,g,\mathcal{N}_1,\mathcal{N}_2)}^n \xrightarrow{g,l'}_p l{:}\overline{D(|\mathcal{A}|,p,g,\overline{\mathcal{N}_1},\mathcal{N}_2)}^n}$ DEC-4 derivation rule shows the termination of a behavior with a transition of probability one and how a token can flow from a behavior call execution to a guarded path with a probability value .

18. **DEC-5** $\frac{\mathcal{N}_1 \xrightarrow{t,\alpha} \mathcal{N}_1'}{l{:}\overline{D(\mathcal{A},p,g,\mathcal{N}_1,\mathcal{N}_2)}^n \xrightarrow{t,\alpha} l{:}\overline{D(\mathcal{A}',p,g,\mathcal{N}_1',\mathcal{N}_2)}^n}$ DEC-5 shows the evolution of a decision term when one of its behavior has been changed under time t.

19. **MRG-1** *l* : $\overline{\mathcal{N} \rightarrowtail l' : M(x, y)}^n \xrightarrow{l}$ *l* : $\overline{\mathcal{N} \rightarrowtail l' : M(\overline{x}, y)}^n$ MRG-1 is a transition with a probability of value 1 to put a token coming from the sub-term $\mathcal{N}$ on a merge labeled by l.

20. **MRG-2** *l* : $\overline{l' : M(\overline{x}, \overline{y}) \rightarrowtail \mathcal{N}}^n \xrightarrow{l}$ *l* : $\overline{l' : M(x, \overline{y}) \rightarrowtail \overline{\mathcal{N}}}^n$ MRG-2 is a transition with a probability of value 1 to present a token flowing from a merge labeled by l to the sub-term $\mathcal{N}$.

21. **MRG-3** *l* : $\mathcal{A}[l' : M(x, y) \rightarrowtail \mathcal{N}, \overline{l_x}] \xrightarrow{l}$ *l* : $\mathcal{A}[l' : M(\overline{x}, y) \rightarrowtail \mathcal{N}, l_x]$ MRG-3 shows the merge enabled when token arrived atone of its pins

22. **MRG-4** $\frac{\mathcal{N} \overset{t,\alpha}{\rightarrow} \mathcal{N}'}{l:\overline{M(x,y) \rightarrowtail \mathcal{N}}^{n} \overset{t,\alpha}{\rightarrow} l:\overline{M(x,y) \rightarrowtail \mathcal{N}'}^{n}}$ The derivation rules MRG-4 presents the subsequence of *l* : $\overline{M(x,y) \rightarrowtail \mathcal{N}}$ when $\mathcal{N}$ executes an action $\alpha$ with a probability p.

23. **JOIN-1** *l* : $\overline{\overline{\mathcal{N}} \rightarrowtail l' : J(x,y)}^{n} \overset{l}{\rightarrow}$ *l* : $\overline{\mathcal{N} \rightarrowtail l' : J(\overline{x},y)}^{n}$ JOIN-1 represents a transition with a probability of value 1 to activate the pin x in a join labeled by l

24. **JOIN-2** *l* : $\overline{l : J(\overline{x},\overline{y}) \rightarrowtail \mathcal{N}}^{n} \overset{\tau}{\rightarrow}$ *l* : $\overline{l : J(x,y) \rightarrowtail \overline{\mathcal{N}}}^{n}$ JOIN-2 represents a transition with a probability of value 1 to move a token in join to the sub-term $\mathcal{N}$

25. **JOIN-3** *l* : $\mathcal{A}[l' : J(x,y) \rightarrowtail \mathcal{N}, \overline{l_x}] \overset{\tau}{\rightarrow}$ *l* : $\mathcal{A}[l' : J(\overline{x},y) \rightarrowtail \mathcal{N}, l_x]$ JOIN-3 shows the join input enabled when token arrived at one of its pins

26. **JOIN-4** $\frac{\mathcal{N} \overset{t,\alpha}{\rightarrow} \mathcal{N}'}{l:\overline{J(x,y) \rightarrowtail \mathcal{N}}^{n} \overset{t,\alpha}{\rightarrow} l:\overline{J(x,y) \rightarrowtail \mathcal{N}'}^{n}}$The derivation rule JON-4 presents the subsequence of *l* : $\overline{J(x,y) \rightarrowtail \mathcal{N}}$ when N executes an action a with a probability p.

27. **SND** *l* : $\overline{a!v}^{n} \rightarrowtail \mathcal{N} \overset{l}{\rightarrow}$ *l* : $\overline{a!v}^{n-1} \rightarrowtail \overline{\mathcal{N}}$ SND describes the evolution of the token after sending an object $v$.

28. **FFIN** $\mathcal{A}[l : \overline{\otimes}] \overset{l}{\rightarrow} \mathcal{A}[l : \otimes]$ This axiom states that if the sub-term $l : \otimes$ is reached in $\mathcal{A}$ then a transition of probability one is enabled to produce a term describing the termination of a flow.

29. **AFIN** $\mathcal{A}[l : \overline{\odot}] \overset{l}{\rightarrow} |\mathcal{A}|$ This axiom states that if the sub-term $l : \odot$ is reached then no action is taken later by destroying all tokens.

30. **PRG** $\frac{\mathcal{N} \overset{t,\alpha}{\rightarrow} \mathcal{N}'}{\mathcal{A}[\mathcal{N}] \overset{t,\alpha}{\rightarrow} \mathcal{A}[\mathcal{N}']}$ PRG derivation rules preserve the evolution when a sub-term $\mathcal{N}$ evolves to $\mathcal{N}$' by executing the action $\alpha$ with a probability p under time t.

The semantics of SysML activity diagrams expressed using $\mathcal{A}$ is the result of the defined inference rules. The semantics can be described in terms of PTA as stipulated by Definition 3.1.

$\mathcal{A} ::= \epsilon \mid l : \overline{i}^{n} \rightarrowtail \mathcal{N}$
$\mathcal{N} ::= \overline{\mathcal{N}} \mid l : F(\mathcal{N},\mathcal{N}) \mid l : Ex(\mathcal{A},p,\mathcal{N},\mathcal{N}) \mid l : D(\mathcal{A},p,g,\mathcal{N},\mathcal{N}) \mid \overline{\mathcal{O}}^{n} \rightarrowtail \mathcal{N} \mid I : \odot \mid I : \otimes$
$\mathcal{O} ::= a\mathcal{B} \mid R \uparrow \mathcal{A} \mid j(x_1,x_2) \mid M(x_1,x_2)$
$\mathcal{B} ::= \uparrow \mathcal{A} \mid !v \mid ?v \mid \epsilon$

FIGURE 2.3: Syntax of Timing Activity Calculus (TAC).

**Definition 3.1** *(TAC-PTA).* A Probabilistic timed automata of TAC term A is the tuple $M_{\mathcal{A}}$ = $<\overline{s}$, *S*, *X*, *Act*, *Inv*, *Prob* , *L* $>$, where:

- $\overline{s}$ is an initial state, such that $L(\overline{s}) = \left\{ l : \overline{i} \rightarrowtail \mathcal{N} \right\}$ ,
- S is a finite set of states reachable from $\overline{s}$, such that, $S = \left\{ s_{i:0 \leq i \leq n} | L(s_i) = \left\{ \overline{\mathcal{N}} \right\} \right\}$
- *X* is a set of clocks,
- Act is a set of actions including three types: label of the executing action node, x(y) inputs an object name on x and stores it in y ,
- *Inv* :$S \rightarrow$ *CC*(*X*) is an invariant condition (i.e. Constraints over clocks X),
- Prob: S $\times$ $Act \rightarrow$ *Dist*($2^{x} \times$ $S$) is a probabilistic transition function assigning for each $s \in S$ and $\alpha \in Act$ a probabilistic distribution $\mu$ where:
  - For each S' $\subseteq$ S and $t \in X$ such that S'=$\left\{ si : 0 \leq i \leq n : s \xrightarrow{t,\alpha}_{pi} s_i \right\}$ , each transition $s \xrightarrow{t,\alpha}_{pi} s_i$ satisfies one TAC semantic rule and $\mu(S', \vartheta) = \sum_{i=0}^{n} p_i = \sum_{i=0}^{n} \mu(s_i, \vartheta + t) = 1$ .
  - For each transition $s \xrightarrow{t,\alpha}_{1} s''$ satisfies one TAC semantic rule and $\mu(S'', \vartheta + t) = 1$
- L : S $\rightarrow 2^{[[\mathcal{L}]]}$ is a labeling function where: $[[\mathcal{L}]] = \{true, false\}$.

## 2.5 PRISM formalization

In this section, our formalization focus on probabilistic timed automata (PTA) that extends the standard probabilistic automata (PA) considered as appropriate semantic model for SysML Activity Diagram [60]. The PRISM model checker supports the PTA with ability to model real-time behavior by adding real-valued clocks (i.e. clocks variable) which increases with time and can be reset (i.e. updated).

A Timed Probabilistic System (TPS) that represents a PRISM program (P) is composed of a set of "m" modules $(m > 0)$. The state of each module is defined by the evaluation of its local variables $V_L$. The global state of the system is defined as the union of the evaluation of local and global variables: V=$V_L \cup V_G$. The behavior of each module is described as a set of statements in the form of:

$$[a]g \rightarrow p_1 : u_1 .. + p_n : u_n$$

Where ***a*** is an (optional)action labeling the command, the **g** is a predicate consists of boolean, integer and clock variables and propositional logic operators. ***p*** is a probability. The update ***u*** is a set of evaluated variables expressed as conjunction of assignments $(V'_j = val_j)$&..&$(V'_k = val_k)$ where $V_j \in V_L \cup V_G$ and $val_j$ are values evaluated via expressions denoted by "eval" eval: V $\rightarrow \mathbb{R} \cup \{True, False\}$. The formal definition of a command is given in Definition 3.2.

**Definition 3.2** A PRISM command is a tuple c = $<$ a, g, u $>$, where:

- "a" is an action label.
- "g" is a predicate over V.
- "u" $= \{(p_i, u_i)\}$ $\exists m > 1, i < m,\ 0 > p_i > 1, \sum_i^m p_i = 1$ and u = $\{(v, \textit{eval}(v)) : v \in V_l\}$.

| | |
|---|---|
| $P$ | ::= $PTA < variables >< eval >< modules >< system >$ |
| $< variables >$ | ::= $\epsilon \mid < KindOfVariables > v :< VariableType > initinit(v);< variables >$ |
| $< KindOfVariables >$ | ::= $\epsilon \mid Global$ |
| $< VariableType >$ | ::= $bool \mid int \mid clock \mid double \mid [min, .., max]$ |
| $< Modules >$ | ::= $modulename < variables >< ModuleInvaraint >< ModuleBehavior > endModule$ |
| $< ModuleInvaraint >$ | ::= $invariant < InvariantSet > endinvriant$ |
| $< InvariantSet >$ | ::= $(g => Inv) \mid (g => Inv)\& < InvariantSet >$ |
| $< ModuleBehavior >$ | ::= $\epsilon \mid [a] : g \longrightarrow < update >; < ModuleBehavior >$ |
| $< update >$ | ::= $< p >:< eval >; \mid < p >:< eval > + < update >$ |
| $< eval >$ | ::= $(v' = eval(V)) \mid (v' = eval(V))\& < eval >$ |
| $< p >$ | ::= $\epsilon \mid p :$ |
| $< system >$ | ::= $system < AlgebraExpression > endSystem$ |
| $< AlgebraExpression >$ | ::= $\epsilon \mid name\ CSPExpr\ name; < AlgebraExpression >$ |

FIGURE 2.4: The Syntax of PRISM Probabilistic Timed Automata

The set of commands are associated with modules that are parts of a system and it definition is given in Definition 3.3.

**Definition 3.3** A PRISM module is tuple M = $< V_l$, $I_l$, *Inv*, C$>$, where:

- $V_l$ is a set of local variable associated with a module,
- *Inv* is a time constraint of the form $v_l \bowtie d/ \bowtie \in \{\leq, \geq\}$ and d $\in \mathbb{N}$,
- $I_l$ is the initial value of $V_l$.
- C= $\{c_i, 0 < i \leq k\}$is a set of commands that define the module behavior.

To describe the composition between different modules, PRISM uses CSP communication sequential process operators [61] such as Synchronization, Interleaving, Parallel Interface, Hiding and Renaming. Definition 3.4 provides a formal definition of PRISM system.

**Definition 3.4** A PRISM system is tuple P = $<$V, I, exp, M, CSPexp$>$, where:

- V = $V_g \coprod_{(i=1)}^{m} V_{li}$ is the union of a set local and global variables.
- $I_g$ is initial values of global variables.
- *exp* is a set of global logic operators.
- M is a set of modules composing a System.
- *CSPexp* is CSP algebraic expression.

### 2.5.1 PRISM Syntax

The syntax of PRISM PTA is defined by the BNF grammar presented in Fig.2.4. To clarify the syntax we define the following:

- $[min...max]$ is a range of values such that $min, max \in \mathbb{N}$ and min $<$ max,
- $p \in ]0,\ 1[$ is a probability value,
- eval is an evaluation expression that can be composed of the following operators: $+, -; *, /, <, <=, >, >=, !, |, =>, g?a : b$,
- $val \in \mathbb{R} \cup \{true, false\}$ is a value given by the function $eval$,
- $v$ is a string describing a variable $(v \in V)$ and $init(v)$ is its initial value,
- name is a string describing the module name. For the module i; name is denoted by $M_i$,
- $Invariant$ impose restrictions on the allowable values of clock variable,
- $CSPexp$ is a CSP expression composed of the following operators $||,|| \ |,|[a, b, ..]|,/ \{a, b, ..\}$ , and $\{a \leftarrow b, c \leftarrow d, ...\}$.

### 2.5.2 PRISM semantics

The probabilistic timed automata of a PRISM program P is based on the atomic semantics of a command “c” denoted by [[c]]. The latter is a set of transitions defined as follows: $[[c]] = \{(s, a, \mu)|s \models g\}$ where $\mu$ is a distribution over S such that $\mu(s, \vartheta + v_t) = \{|0 \leq p_i \leq 1, v \in V, s'(v) = eval(V)|\}$.

Definition 3.5 stipulates the formal definition of PRISM probabilistic timed automata denoted by $M_P$. The states of $M_P$ take the form $\langle V1, ..., Vn, eval\rangle$. The stepwise behavior of $M_P$ is described by the operational semantic rules provided as follows:

1. **INIT** $\langle V_i, init(V_i)\rangle \rightarrow \langle V_i([[init(V_i)]]), -\rangle$INIT initializes variables. For a module $M_i$, init returns the initial value of the local variable $v_i \in V_i$.

2. **LOOP** $\langle V_i, -\rangle \rightarrow \langle V_i\rangle$ This axiom presents a loop in a state without changing variables' evaluations. It can be applied to avoid a deadlock.

3. **UPDATE** $\langle V_i, v_i' = eval(V)\rangle \rightarrow \langle V_i([[v_i]])\rangle$ UPDATE axiom describes the execution of a simple assignment for a given variable $v_i$. Its evaluation is updated in $V_i$ of $M_i$ .

4. **CNJ-UPD** $\left\langle V, v_i' = eval(V) \wedge v_j' = eval(V)\right\rangle \rightarrow \langle V([[v_i]], [[v_j]])\rangle$ CNJ-UPD implements the conjunction of a set of assignments.

5. **PRB-UPD1** $\langle V_i, p : v_i' = eval(V)\rangle \rightarrow_p \langle V_i([[v_i]])\rangle\ 0 < p < 1$ .

6. **PRB-UPD2** $\left\langle V, p : v_i' = eval(V) \wedge v_j' = eval(V)\right\rangle \rightarrow_p \langle V([[v_i]], [[v_j]])\rangle\ 0 < p < 1$ PRB-UPD1 and PRB-UPD2 describe probabilistic updates.

7. **ENB-CMD1** $\frac{V \models g, Inv(V)}{\langle V, M([a]g \rightarrow p_i : u_i)\rangle \rightarrow \mu}$ ENB-CMD1 enables the execution of a probabilistic command.

8. **ENB-CMD2** $\frac{V \models g, Inv(V)\ \ V \not\models g', Inv'(V)}{\langle V, [a]g \rightarrow u; [a']g' \rightarrow u'\rangle \xrightarrow{a} \langle V([[u]]), [a']g' \rightarrow u'\rangle}$ ENB-CMD2 enables the execution of a command in a module.

9. **ENB-CMD3** $\frac{V \models g, Inv(V)\ \ V \vDash g', Inv'(V)}{\langle V, [a]g \rightarrow u; [a']g' \rightarrow u'\rangle \xrightarrow{a} \langle V([[u]]), [a']g' \rightarrow u'\rangle}$. ENB-CMD3 solves the nondeterminism in a module by following a policy.

10. **SYNC** $\frac{\langle V_i, c_i\rangle \xrightarrow{a} \mu_i\ \ \langle V_j, c_j\rangle \xrightarrow{a} \mu_j}{\langle V_i \cup V_j, M_i || M_j\rangle \xrightarrow{a} \mu_i.\mu_j}$ SYNC derivation rule permits the synchronization between modules on a given action a.

11. **INTERL** $\frac{\langle V_i, M_i(c_i)\rangle \xrightarrow{a_j} \mu}{\langle V, M_i ||\ | M_j\rangle \xrightarrow{a_j} \mu}$ INTERL derivation rule describes the interleaving between modules.

**Definition 3.5** *(PRISM-PTA)*. A Probabilistic timed automata of PRISM program p is the tuple $M_p$ = $<s_i$, *S*, *X*, *Act*, *Inv*, *Prob* , *L* $>$, where:

- $s_i$ is an initial state, such that $L(s_i) = [[init(V_i)]]$ ,
- S is a finite set of states reachable from $s_i$, such that, $S = \{s_{i:0 \leq i \leq n} | L(s_i) \in \{AP\}\}$
- *X* is a set of PRISM clocks variables,
- Act is a finite set of actions,
- *Inv* : imposes restrictions on the allowable values of clock variable,
- Prob: S $\times$ $Act \rightarrow$ *Dist*($2^x \times$ $S$) is a probabilistic transition function assigning for each $s \in S$ and $a \in Act$ a probabilistic distribution $\mu$ where: For each s $\in$ S, $v \in V$ is a PRISM variable, $a \in Act$ and $v_t \in V$ is a clock variable such that $s(v, v_t) \xrightarrow{a} \mu_{(v,\vartheta+v_t)}$ , $\vartheta + v_t \vDash Inv(v)$ and $s(v, v_t) \vDash g$.
- L : S $\rightarrow 2^{AP}$ is a labeling function that assigns for each state a set of valuated propositions.

## 2.6 The verification approach

This section describes the transformation of SysML activity diagrams A into a PTA written in PRISM input language. Algorithm.1 illustrates the transformation algorithm T that takes $\mathcal{A}$ as input and returns its PRISM code by PrismCode. The diagram $\mathcal{A}$ is visited using a depth-first search procedure (DFS) and the algorithm's output produces PRISM synchronized modules.

First, the initial node is pushed into the stack of nodes denoted by Nodes (line 7). While the stack is not empty (line 8$-$29), the algorithm pops a node from the stack into the current node denoted by cNode (line 9). The current node is added into the

Algorithm 1 Transformation Algorithm T of SysML Activity Diagrams into PRISM Code.

**Input** : SysML Activity diagram A
**Output** : PRISM Code

```
1 Nodes : Stack;
2 cNode, fNode as Node;
3 nNodes, vNodes List_Of_Node;
4 maxDuration, minDuration : Integer;
5 action as Action;
6 ProcedureT(A)
7 Nodes.push(init); ◄ Push the initial node ∈ A
8 While Nodes.notEmpty()
9 cNode:= Nodes.pop(); ◄ Pull out the first node
10 If(cNode not in vNode)
11 Then
12 vNodes.add(cNode); ◄ The current is considered as visited
13 nNodes:=cNode.Successors();◄ Store the successors in buffer list
14
15 if(cNode.hasDuration()) ◄ set Invariant
16 Then
17         maxDuration:= cNode.maxDuration();
18         PrismCode.add(ϕ(cNode,maxDuration));
19 End if
20
21 minDuration:= cNode.minDuration();
22 PrismCode.add(Γ(cNode,nNodes,minDuration,action));
23 End if
24
25 For all(n in nNodes)
26 Then Nodes.add(n); ◄ Add the successors to the stack
27 End For
28         nNodes.clear(); ◄ Empty the buffer list and search ends
29 End While
30
31 End Procedure T
32
```

list vNode of visited nodes (line 12) if it is not already visited (line 10). *PrismCode* is constructed by calling the function $\Gamma$ and $\phi$. The first has four arguments which are the current node, its successors, the minimum duration and action type (line 22). The second has two arguments which are the current node and the maximum duration to impose the maximal clocks supported by the state (line 18). The explored successors are pushed into the stack nodes (line $25-27$). The algorithm terminates when all nodes are visited.

The function $\Gamma$ presented in Listing.2.1 and Listing.2.2 produces the appropriate PRISM command for each TAC term. The action label of a command is the label of

its related term “n”. The guard of this command depends on how the term is activated and minimal clock valuation. The flag related to the term is its label l that is initialized to false except for the initial node it is true which conforms to the premise of the TAC rule “INIT-1”. The updates of the command deactivate the propositions of the term, activate that ones related to its successors, reset the clock variable of its successors and assign an object values to its successors inputs pins. For a term $n \in \mathcal{A}$, we define three useful functions are: $L(n)$; $S(\mathcal{A}_i)$, and $E(\mathcal{A}_i)$ that return the label of a term n, the initial and the final terms of the diagram $\mathcal{A}_i$, respectively. For example, the call behavior action “$l : a \uparrow \mathcal{A}_i$” (line 25) produces two commands (line 29), and it calls the function $\Gamma$’ (line 56). The first command in (line 29) synchronizes with the first command in (line 60) produced by the function $\Gamma$’ in the action l from the diagram $\mathcal{A}$. Similarly, the second command in (line 29) synchronizes with the command of (line 64) in the action L($E(\mathcal{A}_i)$) from the diagram $\mathcal{A}_i$. The first synchronization represents the TAC rule BH-1 where the second represents the rule BH-2. The function $\Gamma$’ is similar to the function $\Gamma$ except for the initial and the final nodes as shown in (line 58) and (line 62), respectively. The region behavior calls the sub actions within the region in the iterative manner Listing.2.2 (line 3). After each execution an object is produced, the number of execution has the size of the collection in the output. Due to the *unsupported collection type* in PRISM we use the the simple object type integer, double. Thanks for prism renaming ability, we can rename the sub module name end its variables to produce multiple objects. At the end of Region execution, multiple parallel commands are synchronized and objects are assigned Listing.2.2 (line 7). The region is interrupted Listing.2.2 (Line 10), including accept event actions in the region, when a token traverses an interrupting edge. At this point the interrupting token has left the region and is not terminated Listing.2.2 (Line 31). The $\phi$ function Listing.2.2 (line 37- 42) associate with each action the maximum integer number supported by the clock variable (i.e. clock invariant). The generated PRISM fragment of each diagram $\mathcal{A}_i$ is bounded by two PRISM primitives: the module head “Module$\mathcal{A}_i$”, and the module termination “endmodule”.

```
1  Γ : A ⟶ P
2  Γ(A) = ∀n ∈ A , L(n = (i)) = true , L(n ≠ (i)) = false
3  l : i ⇀ N ⟹
4  in
5  {[l]l ⟶ (l′ = false)&(L(N)′ = true);} ∪ Γ(L(N))
6  end
7  l : M(x, y) ⇀ N ⟹
8  in
9  {[l_x]l_x ⟶ (l′_x = false)&(L(N)′ = true);} ∪ {[l_y]l_y ⟶ (l′_y = false)&(L(N)′ = true);} ∪Γ(L(N))
10 end
11 l : J(x, y) ⇀ N ⟹
12 in
13 {[l]l_x ∧ l_y ⟶ (l′_y = false)&(l′_x = false)&(L(N)′ = true);} ∪ Γ(L(N))
14 end
15 l : F(N_1, N_2) ⟹
16 in
17 {[l]l ⟶ (l′ = false)&(L(N_1)′ = true)&(L(N_2)′ = true);} ∪ Γ(L(N_1)) ∪ Γ(L(N_2))
18 end
19 l : D(A, p, g, N_1, N_2) ⟹
20 in
21 {[l]l → p : (l′ = false)&(l_g)′ = true) + 1 − p : (l′ = false)&(l_¬g)′ = true);} ∪ Γ(L(N_1)) ∪ Γ(L(N_2))
22 ∪ {[l_g]l_g ∧ g → (l′_g = false)&(L(N_1)′ = true)} ∪ {[l_¬g]l_¬g ∧ ¬g → (l′_¬g = false)&(L(N_2)′ = true)}
23 end
24
25 l : a ↑ A_i ⇀^α N, case(α) of
26
27 b(y), for z ∈ A_i ⟹
28 in
29 {[l]l ⟶ (l′ = false);} ∪ {[L(E(A_i))]E(A_i) ⟶ (l′ = false)&(y′ = z)&(L(N)′ = true);} ∪ Γ′(A_i) ∪ Γ(L(N))
30 end
31 otherwise
32 in
33 {[l]l ⟶ (l′ = false);} ∪ {[L(E(A_i))]E(A_i) ⟶ (l′ = false)&(L(N)′ = true);} ∪ Γ′(A_i)
34 end
35
36 l : ⊙ ⟹
37 in
38 [l] → &_{l∈L}(l′ = false);
39 end
40
41 l : ⊗ ⟹
42 in
43 [l] → (l′ = false);
44 end
45
46 l : a ⇀^{t>c,α} N, case(α) of
47 b(y), for z ∈ a, ⟹
48 in
49 {[l_bz]l&(t > c) ⟶ (l′ = false)&(y′ = z)&(L(N)′ = true);} ∪ Γ(L(N))
50 end
51 otherwise
52 in
53 {[l_α]l&(t > c) ⟶ (l′_α = false)&(L(N)′ = true);} ∪ Γ(L(N))
54 end
55
56 Γ′ : A → P
57 Γ′(A_i) = ∀m ∈ A , L(m) = false
58 l : i ⇀ N ⟹
59 in
60 {[l]l ⟶ (L(S(A_i))′ = true);} {[L(S(A_i))]L(S(A_i)) ⟶ (L(N)′ = true);} ∪ Γ(L(N))
61 end
62 l : ⊙ ⟹
63 in
64 {[L(E(A_i))]L(E(A_i)) ⟶ (L(E(A_i))′ = false);}
65 end
66 otherwise Γ(A)
67
```

LISTING 2.1: Generating PRISM Commands Function-Part1

```
1  Γ′ : A → P
2
3  l : R ↑ A_j ↦^α N, case(α) of
4
5  b(y_j), for z_j1, z_j2, ....z_jsize ∈ A_j ⟹
6  in
7  {[l]l ⟶ (l′ = false);} ∪ {[L(E(A_j))]E(A_j) ⟶ (l′ = false)&_{j1..jsize}(y′_j = z_j)&(L(N)′ = true);}
8  ∪Γ″(A_j) ∪ Γ(L(N))
9  end
10 l : Ex(p, N_Excp, N_2) ⟹
11 in
12 {[l]l ⟶ p : (l′ = false)&(L(N_2)′ = true) + 1 − p : (l′ = false)&(L(N_Excp)′ = true);} ∪
13 Γ(L(N_2)) ∪ Γ(L(N_Excp))
14 end
15
16 Γ″ : A → P
17 Γ″(A_j) = ∀m ∈ A , L(m) = false
18 l : a ↣ N ⟹
19 in
20 {[l]l ⟶ (L(S(A_j))′ = true);} {[L(S(A_j))]L(S(A_j)) ⟶ (L(N)′ = true);} ∪ Γ(L(N))
21 end
22
23 l : a ↣ E(A_j) ⟹
24 in
25
26 {[l]l ⟶ (l′ = false)&(L(E(A_j))′ = true);}
27 {[L(E(A_j))]L(E(A_j)) ⟶ (L(E(A_j))′ = false);}
28 end
29
30
31 l_Excp : a?v ↣ N ⟹ // Exception handling for line 10
32 in
33 {[l_Excp]l_Excp ⟶ (l′_Excp = false)&(L(N)′ = true);} ∪ Γ(L(N))
34 end
35
36
37 φ : A → P
38 φ(A) = ∀m ∈ A , t_l clock variable , c_l integer constant
39 l : a ↣ N ⟹
40 in
41 {(l = true) ⟹ (t_l <= c_l)&φ(N)}
42 end
43
```

LISTING 2.2: Generating PRISM Commands Function-Part2

## 2.7 The transformation soundness

Our aim is to prove the soundness of the transformation algorithm $\Gamma$ by showing that the proposed algorithm preserves the satisfiability of PCTL properties. Let $\mathcal{A}$ be a TAC term and $M_{\mathcal{A}}$ is its corresponding PTA constructed by the TAC operational semantics denoted by $\mathcal{S}$ such that $\mathcal{S}(\mathcal{A})$ = $M_{\mathcal{A}}$. For the program $\mathcal{P}$ resulting after transformation rules, Let $M_p$ its corresponding PTA constructed by PRISM operational semantics denoted $\mathcal{S}'$ such that $\mathcal{S}'(\mathcal{P}) = M_p$. As illustrated in Fig.2.5, proving the soundness of $\Gamma$ algorithm is to find the adequate relation $\mathcal{R}$ between $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$.

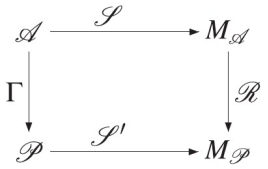



FIGURE 2.5: The Transformation Soundness.

To define the relation $M_{\mathcal{A}}\mathcal{R}M_{\mathcal{P}}$, we have to establish a step by step correspondence between $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$. First, we introduce the notion of the timed probabilistic bisimulation relation [62] [63] in definition 3.7 and 3.8.This relation is based on the probabilistic equivalence relation $\mathcal{R}$ defined in Definition 3.6 where $\delta/\mathcal{R}$ denotes the quotient space of $\delta$ with respect to $\mathcal{R}$ and $\equiv_{\mathcal{R}}$ is the lifting of $\mathcal{R}$ to a probabilistic space.

**Definition 3.6** (The equivalence $\equiv_{\mathcal{R}}$). If $\mathcal{R}$ is an equivalence on $\delta$, then the induced equivalence $\equiv_{\mathcal{R}}$ on Dist($\delta \times 2^x$) is given by: $\mu_{\equiv_{\mathcal{R}}}\mu'$ iff $\mu(\delta, \vartheta + d) \equiv_{\mathcal{R}} \mu(\delta, \vartheta + d')$.

**Definition 3.7** (Timed Probabilistic Bisimulation Relation). A binary relation $\mathcal{R}$ over the set of states of PTAs is timed bisimulation iff whenever $s_1\mathcal{R}s_2$, $\alpha$ is an action and d is a delay:

- if $s_1 \stackrel{d,\alpha}{\rightarrow} \mu(s_1, \vartheta + d)$ there is a transition $s_2 \stackrel{d',\alpha}{\rightarrow} \mu(s_2, \vartheta + d')$, such that $s_1 \mathcal{R} s_2$. The delay *d* can be different from *d'*;
- two states s, s' are time probabilistic bisimilar, written $s \sim s'$, iff there is a timed probabilistic bisimulation related to them.

**Definition 3.8** (Timed Probabilistic Bisimulation of PTAs). Probabilistic Timed automata $A_1$ and $A_2$ are timed probabilistic bisimilar denoted ($A \sim A'$) iff their initial states in the union of the probabilistic timed transition systems T($A_1$)and T($A_2$) generated by $A_1$ and $A_2$are timed probabilistic bisimilar.

For our proof, we stipulate herein the mapping relation $\mathcal{R}$ denoted by $M_{\mathcal{A}} \mathcal{R} M_{\mathcal{P}}$ between a TAC term $\mathcal{A}$ and its corresponding PRISM term $\mathcal{P}$.

**Definition 3.9** (Mapping relation). The relation $M_{\mathcal{A}} \mathcal{R} M_{\mathcal{P}}$ between a TAC term $\mathcal{A}$ and a PRISM term $\mathcal{P}$ such that $\Gamma(\mathcal{A}) = \mathcal{P}$ is a timed probabilistic bisimulation relation.

Finally, proving that $\Gamma$ is sound means showing the existence of a timed probabilistic bisimulation between $M_A$ and $M_P$.

**Lemma1** (Soundness). The mapping algorithm $\Gamma$ is sound, *i.e.* $M_{\mathcal{A}} \sim M_{\mathcal{P}}$.

**Proof 1**: We prove $M_{\mathcal{A}} \sim M_{\mathcal{P}}$ by following a structural induction on TAC terms and their related PRISM terms. For that, let $s_1, s_1' \in \mathcal{S}_A$ and $s_2, s_2' \in \mathcal{S}_P$. We distinguish the following cases where L(s) takes different values:

1. L($s_1$) = $l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{i, a\} \Longrightarrow \exists s_1 \stackrel{d,\alpha}{\longrightarrow}_1 s_1'$, L($s_1$')= $l : x \rightarrowtail \overline{\mathcal{N}}$. For L($s_2$) = $\Gamma$(L($s_1$)), we have L($s_2$)=$\langle L(x), \neg L(\mathcal{N}) \rangle$ then $\exists s_2 \stackrel{d',\alpha}{\longrightarrow}_1 s_2'$ where $L(s_2') = \langle \neg L(x), L(\mathcal{N}) \rangle$.
2. L($s_1$) = $l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{a!v, a?v\} \Longrightarrow \exists s_1 \stackrel{\alpha}{\rightarrow}_1 s_1'$, L($s_1$')= $l : x \rightarrowtail \overline{\mathcal{N}}$. For L($s_2$) = $\Gamma$(L($s_1$)), we have L($s_2$)=$\langle L(x), \neg L(\mathcal{N}) \rangle$ then $\exists s_2 \stackrel{\alpha}{\rightarrow}_1 s_2'$ where $L(s_2') =$

$\langle \neg L(x), L(\mathcal{N}) \rangle$.

3. L($s_1$) = $l : \overline{D(g_1, \mathcal{N}_1, \mathcal{N}_2)}^n$ then $\exists s_1 \xrightarrow{g_1,\alpha}_1 s_1'$, L($s_1$')= $l : \overline{D(g_1, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. For L($s_2$) = $\Gamma$(L($s_1$)), we have L($s_2$)=$\langle l, \neg l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$ then $\exists s_2 \xrightarrow{g_1,\alpha}_1 s_2'$ where $L(s_2') = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$.

4. L($s_1$) = $l : \overline{\odot}$ then $\exists s_1 \xrightarrow{\alpha}_1 s_1'$, L($s_1$')= $l : \odot$. For L($s_2$) = $\Gamma$(L($s_1$)), we have L($s_2$)=$\langle l \rangle$ then $\exists s_2 \xrightarrow{\alpha}_1 s_2'$ where $\forall l_i \in \mathcal{L} : L(s_2') = \langle \neg l_i \rangle$.

5. L($s_1$) = $l : \overline{\otimes}$ then $\exists s_1 \xrightarrow{\alpha}_1 s_1'$, L($s_1$')= $l : \otimes$. For L($s_2$) = $\Gamma$(L($s_1$)), we have L($s_2$)=$\langle l \rangle$ then $\exists s_2 \xrightarrow{\alpha}_1 s_2'$ where $L(s_2') = \langle \neg l \rangle$.

From the obtained results, we found that $\mu(s_1, \vartheta + d) = \mu(s_2, \vartheta + d') = 1$ then $s_1 \sim s_2$. In addition, the unique initial state of $M_{\mathcal{A}}$ is always corresponding to the unique initial state in $M_{\mathcal{P}}$. By studying all TAC terms, we find that $M_{\mathcal{A}} \sim M_{\mathcal{P}}$, which confirms that Lemma 1 holds.

In the following, we show that the mapping relation preserves the satisfiability of PCTL properties. This means, if a PCTL property is satisfied in the resulting model by a mapped function $\Gamma$ then it is satisfied by the original one.

**Proposition 1** (PCTL preservation). For two PTAs $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$ such that $\Gamma(\mathcal{A}) = \mathcal{P}$ where $M_{\mathcal{A}} \sim M_{\mathcal{P}}$. For a PCTL property $\phi$, then: $(M_{\mathcal{A}} \vDash \phi) \Longleftrightarrow (M_{\mathcal{P}} \vDash \phi)$.

**Proof 2**. The preservation of PCTL properties is proved by induction on the PCTL structure and its semantics. Since $M_{\mathcal{A}} \sim M_{\mathcal{P}}$ and by relying to the semantics of each PCTL operator $\zeta \in \{\mathsf{U}, \mathsf{U}^{\leq k}, \mathsf{I}^{=k}, \mathsf{C}^{\leq k}, \mathsf{F}, \mathsf{P}_{\bowtie p}, \mathsf{R}_{\bowtie q}\}$, we find that $(M_{\mathcal{A}} \vDash \zeta) \Longleftrightarrow (M_{\mathcal{P}} \vDash \zeta)$ which means: $(M_{\mathcal{A}} \vDash \phi) \Longleftrightarrow (M_{\mathcal{P}} \vDash \phi)$

## 2.8 Implementation and experimental results

In this section, we apply our verification framework on Digital camera case study [32]. The related SysML activity diagrams are modeled on Topcased2 then mapped into Prism code via our Java implementation. Listing.2.3 shows a simplified code for the Digital Camera module. In the purpose of providing experimental results demonstrating the efficiency and the validity of our approach, we verify four system functional requirements. They are expressed in PCTL as follows:

1. The maximum probability value that the *TakePicture* action should not be activated if either the memory is full memFull=true or the AutoFocus action is still ongoing. T is a constant value referring to the time bound

$$Pmax =?[F^{\leq T}(memFull|AutoFocus)\&TakePicture] \tag{2.1}$$

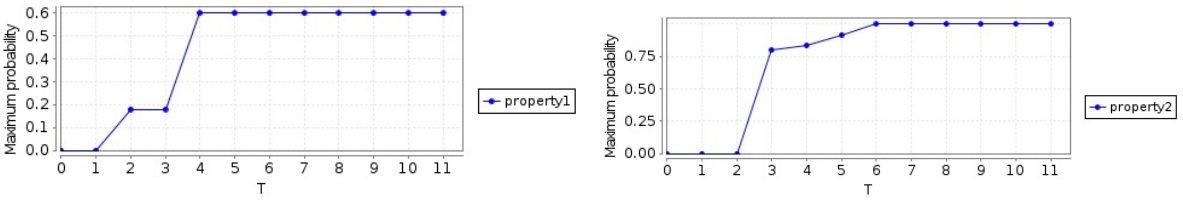
2. The maximum probability to complete all tasks after turning on the camera. T is a constant value referring to the time bound

$$Pmax =?[F^{\leq T}Final] \tag{2.2}$$

3. The minimum expected time that the *TurnOff* action should be activated after turning on the camera.

$$R\{"time"\}min =?[F(TurnOff)] \tag{2.3}$$

The verification results of the above two properties are done using an i7 CPU 2.67GHz with 8.0GB of RAM and shown in Fig.2.6. For different values of time “T”, Fig.2.6a shows that the verification result for property 1 converges to 0.6 after 4 time units. Fig.2.6b shows that the verification result for property 2 converges to 0.916 after 6




(A) Property1 (B) Property2

FIGURE 2.6: The Verification of PCTL Properties on the Digital Camera

time units. For the third property, the minimum reward or minimum expected time that the *TurnOff* action should be activated after turning on the camera is equal to 3.448 time units.

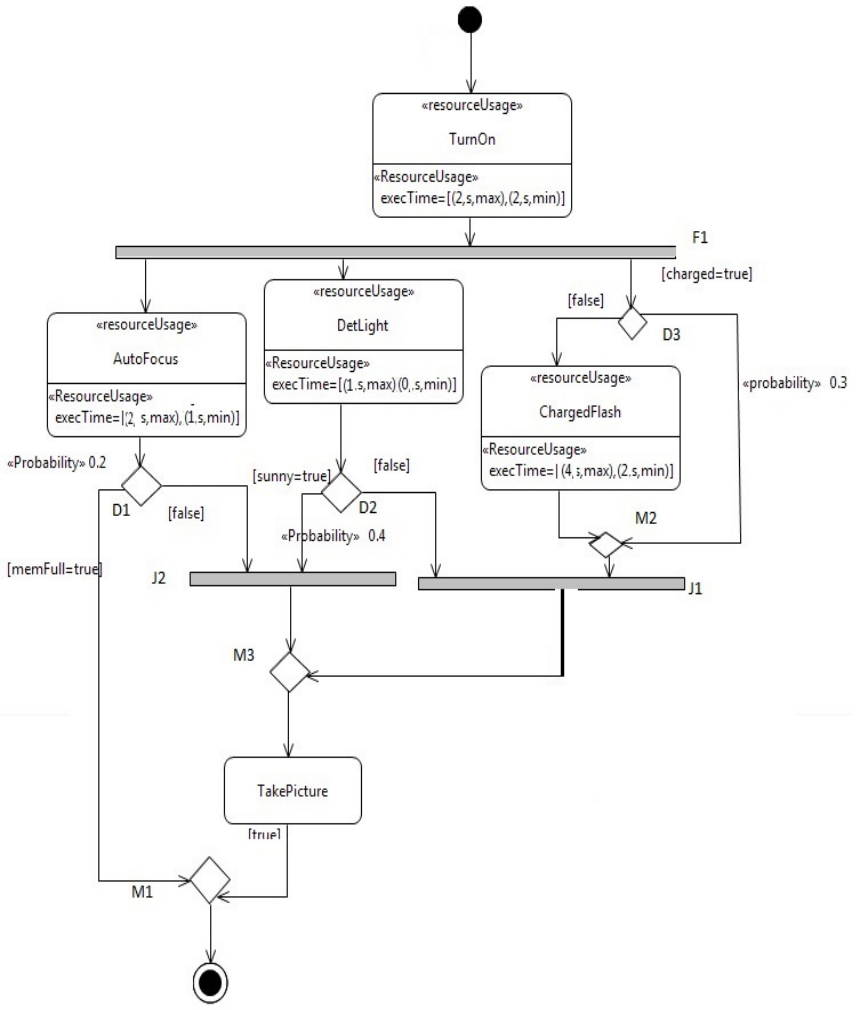



FIGURE 2.7: The abstract SysML activity diagram for Property 4.

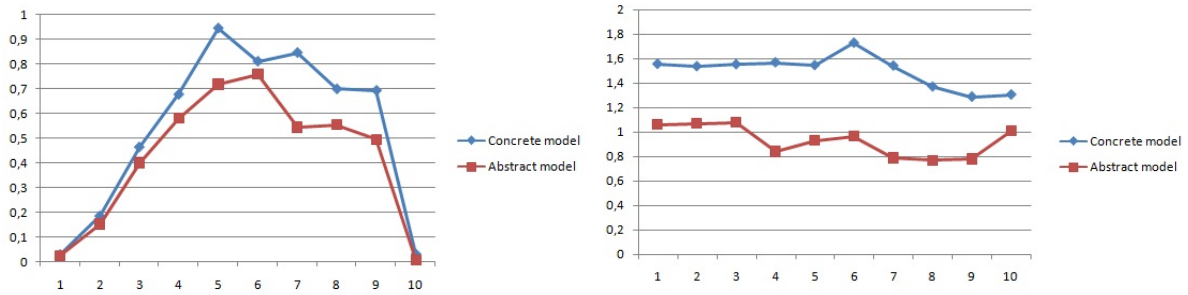
TABLE 2.3: Verification results for Property 2.4

| Time Interval | Concrete model | | Abstract model | | Results |
|---|---|---|---|---|---|
| | Tv | Tc | Tv | Tc | |
| 1 | 0.03 | 1.554 | 0.0026 | 1.059 | 0 |
| 2 | 0.186 | 1.535 | 0.154 | 1.068 | 0.18 |
| 3 | 0.465 | 1.551 | 0.401 | 1.077 | 0.26 |
| 4 | 0.678 | 1.567 | 0.58 | 0.839 | 0.679 |
| 5 | 0.946 | 1.544 | 0.716 | 0.928 | 0.679 |
| 6 | 0.812 | 1.728 | 0.757 | 0.965 | 0.476 |
| 7 | 0.847 | 1.54 | 0.543 | 0.788 | 0.476 |
| 8 | 0.7 | 1.368 | 0.555 | 0.767 | 0.476 |
| 9 | 0.694 | 1.285 | 0.495 | 0.779 | 0.476 |
| 10 | 0.032 | 1.303 | 0.01 | 1.007 | 0 |

Now, we want to verify the abstraction effects over SysML Activity diagram described above Fig.2.2 to cope with state explosion problem when we check the property

$$Pmax =?[F^{\leq T} TakePicture] \tag{2.4}$$

For this purpose we apply the algorithm and rules defined by [64]. This abstraction approach depends on the activity diagram and the PCTL properties as input and produces an abstracted model. However, the abstraction rules do not focus on time constraints. To take the advantages of the algorithm we hide the state (Activity Action) that does not appear in PCTL property proposition set such as: Flash, TurnOff, FinalFlow and Fork2 except those are constrained. The result is shown in Fig.2.7 and Table.2.3 presents its different verification results in function of time interval T. To evaluate the verification cost, we measure the time required for verifying a given property, denoted by **Tv** and time required to construct the model denoted by **Tc**. The results are depicted in Fig.2.8. The number of states and transitions for the concrete model are 9364 states and 23653 transitions, respectively and 4580 states, 12221 transitions for the abstracted model. As conclusion, we notice that the abstraction rules preserves the results while verification and construction time are optimized.




(A) Verification Time (B) Construction Time

FIGURE 2.8: The abstraction effects on Digital Camera Activity diagram.

## 2.9 Conclusion

In this chapter, we presented a formal verification approach of systems modeled by using SysML activity diagram. The proposed approach use SysML activity diagram with time annotations using MARTE profile to evaluate the system behavior at different periods of time. The approach consists on capturing the activity diagram with time features as probabilistic timed automata supported by PRISM language. We proposed a calculus dedicated to SysML Activity diagrams that captures precisely the underlying semantics. In addition, we formalized PRISM language and showing its semantics. Moreover, we proved the soundness of our proposed approach by defining the relation between the semantics of the mapped diagrams and the resulting PRISM models. By this relation, we proved the preservation of the satisfiability of PCTL properties. We applied our approach on a simple case study: digital Camera where time and probability were evaluated. In the next chapter, we propose a deployment based probabilistic model checking to verify the embedded software against the reliability property.

```
pta
module Activity1
Initial : bool init true;
TurnOn : bool init false;
Fork1 : bool init false;
Fork2 : bool init false;
AutoFocus : bool init false;
charged : bool init false;
      ........................
x1 : clock;
x2 : clock;
x3 : clock;
x4 : clock;
x5 : clock;
invariant (TurnOn =true => x1<=2)& (ChargeFlash =true => x4<=4) &
(AutoFocus =true => x3<=2) &(DetLight =true => x2<=1) &
(WriteMem =true => x5<=3) endinvariant

[Initial] Initial -> (Initial'=false) & (TurnOn'=true) & (x1'=0);
[TurnOn] TurnOn & x1>=2 -> (TurnOn'=false) & (Fork1'=true);

[Fork1] Fork1 -> (Fork1'=false) & (DetLight'=true) & (AutoFocus'=true) &
(D3'=true) & (x2'=0) & (x3'=0);
[D3] D3 -> 0.3 : (charged'=true) & (D3'=false)
+ 0.7 :(Notcharged'=true) & (D3'=false);
[ ] charged=true -> (charged'=false)  & (M21'=true);
[ ] Notcharged=true -> (Notcharged'=false) & (ChargeFlash'=true) & (x4'=0);
[ChargeFlash] ChargeFlash&x4>=2  -> (ChargeFlash'=false) & (M22'=true);
      ........................................
[J1] J1=true -> (J1'=false) & (Fork2'=true);
[Fork2] Fork2 -> (Fork2'=false) & (Flash'=true) & (M31'=true);
[Flash] Flash -> (Flash'=false) & (FinalFlow'=true);
[FinalFlow] FinalFlow -> (FinalFlow'=false);
[J2] J2=true -> (J2'=false) & (M32'=true);
[M31] M31=true -> (M31'=false) & (TakePicture'=true);
[M32] M32=true -> (M32'=false) & (TakePicture'=true);
[TakePicture] TakePicture -> (TakePicture'=false) & (WriteMem'=true) & (x5'=0);

[WriteMem] WriteMem&x5>=2  -> (WriteMem'=false) & (M12'=true);
[Final] Final -> (Initial'=false)& (TurnOn'=false)& (Fork1'=false)&
(Fork2'=false)& (AutoFocus'=false)& (charged'=false)&
      ..................................
endmodule

rewards "time"
true : 1;
endrewards
```

LISTING 2.3: Digital Camera PRISM code fragment

# CHAPTER 3

# RELIABILITY ANALYSIS BASED PROBABILISTIC MODEL CHECKING TO OPTIMIZE SOFTWARE DEPLOYMENT IN EMBEDDED SYSTEMS

## 3.1 Introduction

Embedded systems is a mixture of software and hardware components that span a wide range from a small platform of sensors and actuators to a distributed systems consisting in lot of interacting nodes. Recently, software parts gain more importance in designing of such systems- it implements the complex system functionality and its deployments is very hard, laborious and time-consuming. To efficiently exploit the physical platforms in the software development process, we introduce a novel approach for deployment decision making based on PRISM probabilistic model checker that takes the software components and the physical platform to produce different configurations. To check the best one, the configuration should satisfy

reliability properties that are expressed in PCTL temporal logic. We present automotive control system as case study to illustrate the applicability of the proposed approach.

## 3.2 Embedded Software Deployment

Software deployment is a complex procedure known to be NP hard [65], the process consists in distribution of a software components on different physical locations with respect to the requirements. The term *component* refers to an operational unit or modules consisting on a set of operations where the assembled components represent the system. Note that the component-based development approach is a proven concept for managing complexity [22], and has been used also in Embedded System and in Software development for a while [30]. In case of software-hardware deployment, we have to distinguish two set of components; hardware and software communicating using a proper interface. [66] and [67] give a state of the art related to the deployment concepts and existed strategies in deployment-space exploration.

Reliability is one of the quality attributes for the achievement of the deployment with minimum failures. This issue is addressed by [68] for service-oriented deployment based on reliability assessment called DISNIX with proper OS to deploy a heterogeneous components (i.e. services) in a network of machines. In our approach, we want to optimize the deployment with system configuration reliability as respected threshold (i.e. mapping of software component to hardware host).

Recently, formal methods have become essential tools for developing safety-critical

systems, where its behavioral correctness is a main concern. These methods require a mathematical expertise for specifying what the system ought to do and verifying it with respect to the requirements. One of the interesting method is a Probabilistic Model Checking [9].

This chapter investigates the use of probabilistic model checking during the design process, with the goal of automatic *deployment-space exploration*. To guide the search for an optimal deployment, the specification (i.e. Architectural view) is transformed into PRISM input language as *Probabilistic-Timed Automata* (PTA) or in *Markov-Decision Process* MDP model that allows estimation by checking a probabilistic property that expresses the reliability. Fig.2 in chapter.2 depicts the different steps for the deployment-space exploration. The specification consists on the SysML internal blocks diagram (IBD) of both software and hardware components enriched with MARTE features (Modeling and Analysis of Real-Time and Embedded systems) [39] to express some software and hardware characteristics. The second step consists on *Reliability* extraction from different hardware components to annotate the SysML activity diagrams with probabilities. To illustrate the use of transformations in this process, we study the deployment of automotive control system with respect to some constraints that represent the deployment restrictions.

## 3.3 Automotive Control Systems

Many cities are more and more overcrowded which lead to the growing accidents and unpredicted emergencies. In response to that, the engineers have to open a ways to improve the transportation safety. In this respect, several innovative and cost-effective solution are emerging to simplify our daily-life so-called Automotive Control Systems (ACS). The evolution of *Automotive Control Systems* is enabled by recent advances in computing and sensing technologies as well as advances

in estimation and control theory. The Automotive Control Systems are the *active safety functions* [69] that encompass all features intended to prevent accidents. The main active functions that are studied in our chapter are: Anti-lock Brake (ABS) and Adaptive-Cruise Control (ACC) Subsystems.

### 3.3.1 Adaptive Cruise Control

ACC (Adaptive Cruise Control) [69] simplifies the task of driving a car because it relieves the driver of the mentally demanding task of keeping a check on the car's speed. The main function of ACC is keeping the speed constant at the setting selected by the driver (i.e. Absence of vehicle in front). In Fig.3.1, if the vehicle in front is traveling at a constant speed, a car fitted with ACC will follow it at the same speed and a virtually constant distance. That is because the distance between the two vehicles is at least within a broad speed range. Here the speed change is automatic without the need for driver intervention.

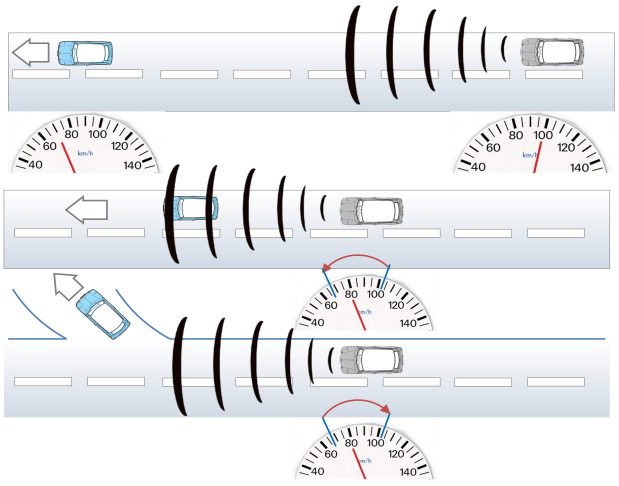


FIGURE 3.1: Adaptive Cruise Control System

### 3.3.2 Anti lock Brake System

Wheels of a vehicle may lock up under braking due to wet or slippery road surfaces then the vehicle become uncontrollable and leave the road. The anti-lock braking system (ABS) detects if one or more wheels are about to lock up under braking and if so, makes sure that the brake pressure remains constant or is reduced. As a consequence, the vehicle can be braked or stopped quickly and safely. The electronic core unit processes the informations received from the brake paddle according to defined mathematical procedures. The results of those calculations are inputs to the emergency stop detection (see Section 3.7).

## 3.4 Related Work

In this section, we depict the recent works related to the deployment-space exploration optimization, partitioning and allocation then we compare them with our proposed approach.

Meedeniya et al. [70] present an approach for reliability optimization in case of software deployment. The authors propose a hardware and software constraints (i.e. physical failures, software workload, deployment restrictions) that are considered as inputs for the reliability function. The approach uses Genetic Algorithms (GA) [71] to optimize the reliability function and applied it on automotive systems. This approach is dedicated to be integrated as framework for AADL language (Architecture Analysis and design Language) [47] but, the authors did not explain how they integrated it in the top of language like extended properties since AADL support it. In addition, the authors does not explain how the algorithm is applied on specification as

in [72]. The same approach is depicted in [73] to find software-to-hardware assignments that maximize the reliability of the system with respect to constraints. These constraints include memory, physical platform failure and communication between software components. The deployment-space exploration is based on Ant Colony Optimization (ACO) combined with constraint programming (CP). Herrera et al.[30] present the COMPLEX approach for hardware/software partitioning and allocation to specific physical platform (i.e. software and hardware). The design level is based on UML component diagram with additional annotation to specify a deployment-space allocation (DSE). However, the authors do not provide any information about algorithm or strategy applied. do Nascimento et al.[74] present a framework for hardware/software code generation based on Model-Driven Engineering (MDE) proposed by the Object Management Group (OMG). The approach starts from UML class and sequence diagrams annotated using MARTE profile. For design-space exploration, the authors propose a model-to-model transformation to abstract the architectural specification. The generated model is considered as input for H-SPEX DSE tool [75]. The core of the tool is based on Ant Colony Optimization. However, the authors do not provide any information about the optimization strategy that is hidden from the user. Etienne et al.[29] present ENOSYS design flow for the Modeling and Synthesis of Embedded Systems. The approach consists of four consecutive steps respectively named Modeling, Synthesis, Source Code Optimization and Design Space Exploration. The modeling step is based on UML Composite diagrams (i.e. Components representations) with the core behavior using state machine or activity diagrams. The composite diagram is enriched with set of annotations using MARTE profile to express the software and hardware components and allocations. Automatic partitioning of the system into software and hardware components occurs when software code is generated and executed to obtain performance, area and early power figures. The process is executed until the required constraints are met. The disadvantages of the approach is the process of exploration based on code level instead on design level that leads to errors and time consuming. In [76], the process is a same as [29] where decision is made at synthesis level. Besnard et al.[77]

present a verification and validation framework of embedded software using AADL and its behavioral annex. The specification is based on AADL language(Architecture Analysis and design Language) [47] and Simulink [45] for functional behavior. The authors implemented a systematic translation into the multi-clocked synchronous semantics of the Signal data-flow language [78] that enables timing analysis in case of Multiprocessor partitioning with respect to timing constraints. However, the approach soundness is not proved. Nath and Datta[79] address a partitioning of JPEG encoder (i.e. applied for obtaining high quality output from continuous-tone images) into software and hardware components. The approach is based on multi-objectives optimization taking account mainly the execution time, the memory requirement, the power consumption and the software modules that must be equal to the number of processors. The process is based on genetic algorithm. However, the authors do not explain how the hardware and software component are specified at early stage of the design.

### 3.4.1 Comparison

As a summary, in Table.3.1 we compare our framework to the existing works by taking consideration six criteria: The specification language, the addressed problem, the applied algorithm, formalization, soundness and automation. The Specification criteria shows if the design is based on clear user view (i.e.Languages and Models). The second criteria focus on the decision problem (i.e. deployment and partitioning) that the authors want to solve. Applied algorithm criteria indicates the procedure used to solve the decision problem. Formalization criteria confirms if the approach presents a semantics and formalizes the studied diagrams. Soundness feature shows if the mapping of the studied approach is proved. Automation criteria checks if the presented approach provides a tool. From the comparison, we observe

that only few works formalize the specification diagrams, including the model checking as decision tool for deployment case. Our work has for objective to support: component-based specification using SysML/MARTE formalizing the SysML behavior diagram (chapter.2). In addition, we implement the tool that allows the automatic deployment decision.

## 3.5 SysML Diagrams

The system specification described in this chapter is characterized by following a component-oriented approach [80] for the deployment of software components. In component-based approach, the system is built by the composition of multiples blocks (i.e. components) interacting with each others through a well defined interfaces. Blocks provide and require services to function correctly. These services are reported in the interfaces that are implemented by the components. In addition, the specification of behavior is based on activity diagrams (chapter.2).

### 3.5.1 SysML Internal Blocks Diagram

The **block** [31] is the fundamental modular unit for describing system structure in SysML. It can define a type of logical or physical entity (e.g., a system); a hardware, software. Blocks are often used to describe reusable components that can be used in many systems. Internal block diagram (IBD) has a relationship to the Blocks diagram, it expresses aspects of a system's structure that complements the aspects conveyed on blocks diagrams. An IBD conveys how the blocks must be assembled to create a valid instance. It also shows how an instance of that block must be connected to external entities to create a valid instance of the system as a whole

TABLE 3.1: Comparison with the existing approaches

| Approach | Specification | Problem | Algorithm | Formalization | Soundness | Automation |
|---|---|---|---|---|---|---|
| [30] | UML/MARTE | Partitioning | | | | √ |
| [73] | | Deployment | Ant Colony Optimisation | | | √ |
| [70] | | Deployment | Genetic Algorithm | | | √ |
| [74] | UML/MARTE | Deployment | Ant Colony Optimisation | √ | | √ |
| [72] | | Allocation | Genetic Algorithm | | | |
| [29] | UML/MARTE | Partitioning | | | | √ |
| [77] | AADL/Simulink | Allocation | Scheduling Algorithm | | | √ |
| [79] | | Partitioning | Genetic Algorithm | | | √ |
| [76] | UML/MARTE | Deployment | | | | √ |
| Our | SysML/MARTE | Deployment | Model Checking | √ | √ | √ |

[81]. Additionally, When the blocks are defined and designed, they are endowed with behavior to express the core operations of implemented interfaces. There are three main behavioral formalisms in SysML: activity, state machines and interaction diagrams.

The system under design is specified by a SysML IBD before starting the deployment. The graphical notations of SysML helps designers for the specification of embedded systems in easy way. However, the description needs a relevant information about the type of the blocks containing the system such as software and hardware blocks. Thus, we use MARTE profile [39].

The interfaces can be modeled using MARTE stereotype “Client-Server port” included in the sub-profile Generic Component Model (GCM) (See Fig.3.2). Software components are modeled by the MARTE stereotype “RtUnit” included in the MARTE sub profile High-Level Application Modeling (HLAM). The *isMain* attribute denote the main application. In this case, the main attribute has to include the function that acts as trigger for complete application execution. Each RtUnit component in Fig.3.3 represents a thread and executed by the processor. The attribute *memorySize* can be added to express the amount of static memory requires for each software component. We use the MARTE stereotype “allocate” to assign the software components to the processors. The application components are interconnected by means of connectors that represent the communicating channels. Communications are established through ports that embody the provided/required interfaces of each hardware-software component. The software component connectors can be annotated by MARTE stereotype “CommunicationStep” with attribute *msgSize* to refer to the size of messages exchanged between the components included in the MARTE sub profile Generic Quantitative Analysis Modeling(GQAM). Hardware components communicate through a bus. The bus represents a connector component that is stereotyped using MARTE “HW_Media” from sub profile Hardware

Resource Modeling (HRM). In addition, more attributes can be added for analysis such as *throughput* using sub profile GQAM. The processor is the main component in the hardware board stereotyped using MARTE “HW_Processor” from sub profile HRM (See Fig.3.4) and some attributes can be added like *mips* that characterizes the throughput. Processors can be annotated by“ExecutionHost” from GQAM with attribute *memorySize* to express the processor capacity. Finally, to annotate each hardware components with probability of success and failure of data transmission, we use “GaStep” from GQAM with attribute *prob*. In our chapter,*prob* represent the success of execution (i.e. Reliability). Posadas et al.[76] present depicts the main features used to specify the embedded systems.

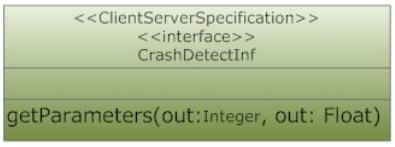



FIGURE 3.2: Example of interface

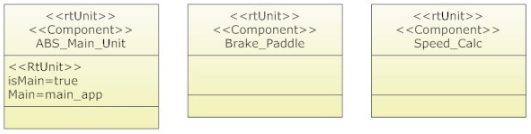



FIGURE 3.3: Example of Software Components

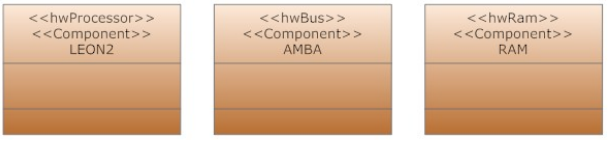



FIGURE 3.4: Example of Hardware Components

### 3.5.2 Linking Behavior to Blocks Using Partitions

A set of activity diagram artifacts in Chapter.2 can be grouped into an activity partition (also known as a *swimlane*) [31] that is used to indicate responsibility for execution of those nodes. A typical case is when an activity partition represents a block and indicates that any behaviors invoked by call actions in that partition are the responsibility of the block. Activity partitions are depicted as rectangular symbols that

physically encompass the action symbols and other activity nodes within the partition (the so-called “swimlan” notation). In our chapter, we link each sub set of actions to each blocks that are part of the blocks (BD) and internal blocks diagrams (IBD). The figure in 3.5 presents an example where different activity artifacts are allocated to three blocks. The transition crossing the blocks can be annotated with probability that is derived from hardware properties.

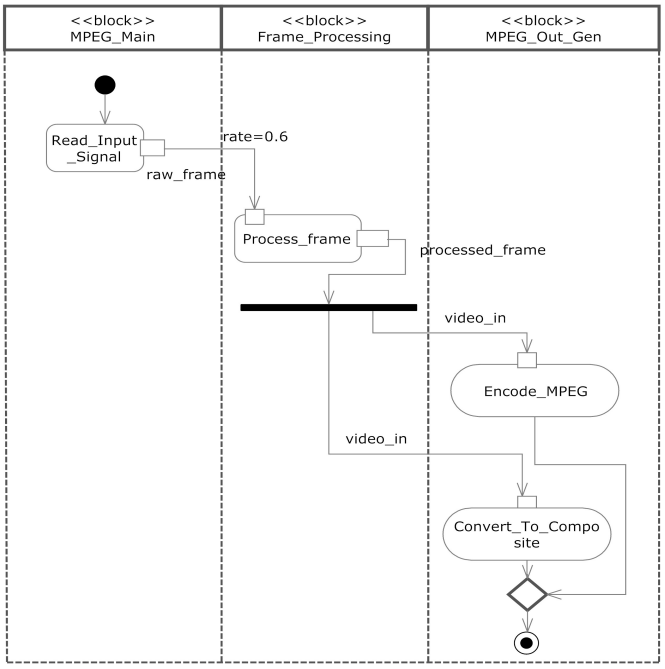



FIGURE 3.5: Example of activity allocation to blocks

## 3.6 The Deployment Problem

The deployment optimization consists on automatically explores the states space of possible allocations of software components to hardware components according to the constraints addressed in this section, and returns the set of near-optimal candidates.

To evaluate a single deployment we use some specification metrics defined in [70] for the software and the platform architecture in addition to the reliability of each physical components (e.g. Processor, Bus).

The constraints established in this section corresponds to the components type. For instance, the processor representation corresponds to the $<< HwProcessor >>$ or $<< ComputingRessource >>$ annotations.

### 3.6.1 Deployment Quality Measure

The quality metric of the approach depends on the reliability of physical hardware (Processors and buses). The software failures are not considered in the approach. We consider that the software failures do not influence the optimization process then they are abstracted.

The reliability of each processor allows populating our design model with probabilities ; means that the activity diagram is characterized by reliability on associated interrupting edges of the partitioned diagram.

The objective of the chapter is to find the best *Deployment Configuration Candidate* that satisfy the requirement in term of reliability. The best deployment configuration consist on the exploration of all candidates that maximizes the life duration and postpone the system failure. Here, the process is called: *Reconfiguration* [82]. The semantics of configuration can be described in terms of PTA or MDP as stipulated in previous chapter.2 where the definition is:

**Definition 4.1** (Reconfiguration). Reconfigurations are denoted by transitions M $\stackrel{\alpha}{\rightarrow}$

M′ meaning that the execution of $\alpha$ on the configuration M produces a new configuration M′. The action $\alpha$ consists on software components redeployment and probabilities redefinition.

We can now define a deployment plan as a sequence of reconfiguration actions from one deployment to another.

**Definition 4.2** (Deployment problem). A deployment problem consists on a sequence of reconfigurations to find the suitable one $M_i$ according to the requirement (i.e. PCTL properties) $M_0 \overset{\alpha_0}{\rightarrow}_{p_0} M_1 \overset{\alpha_1}{\rightarrow}_{p_1} ... \overset{\alpha_{m-1}}{\rightarrow}_{p_{m-1}} M_m$, for $0 \leq i \leq m$.

## 3.7 Experimental Results

In order to implement our methodology for deployment-space exploration, an Eclipse plug-in has been developed. The graphical tool used for creating SysML/MARTE models is Eclipse TopCased. After the generation of the corresponding XML file of the main design, The plug-in parses the document and generates multiple(s) PRISM code in MDP or PTA model in order to check the configuration against the PCTL property. Finally, the plug-in returns a message for the close-configuration that satisfies our requirements. Our approach is illustrated on the case study of an embedded automotive control system designed according to [70]. The objective of the optimization is to maximize the life duration and postpone the system failure. To assess the performance of model checking on automotive systems, two software components are deployed on dedicated hardware architecture namely the Anti-lock Brake System (ABS) and Adaptive Cruise Control (ACC). In our specification we simplify the design in order to not clutter the content.

**Hardware Architecture**: The hardware architecture used for the case study consists of four processors and three buses connecting the processors together, see Fig.3.7.

TABLE 3.2: Parameters of buses and processors

| Hardware | *Reliability*[%] |
|---|---|
| PR 0 | 80 |
| PR 1 | 70 |
| PR 2 | 90 |
| PR 3 | 60 |
| Bus 0 | 82.3 |
| Bus 1 | 72.99 |
| Bus 2 | 88.06 |

Table.3.2 contains the Reliability parameters of each processor and buses. We refer to PR as processors.

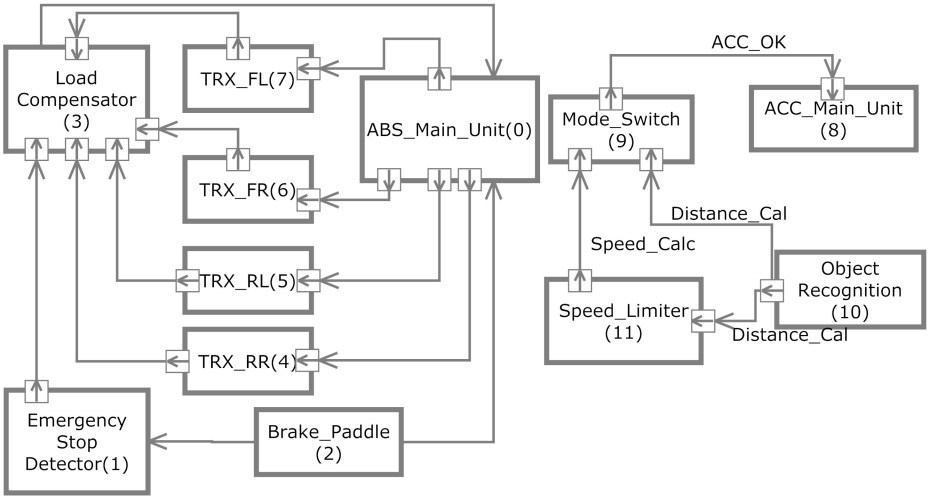



FIGURE 3.6: Adaptive Cruise Control (Right) and Anti-lock Brake System (Left)

**Anti-lock Brake System (ABS)**: used in modern cars to minimize hazards associated with skidding and loss of control due to locked wheel during braking. The software architecture of the ABS is depicted in Fig.3.6(Left) and Fig.3.8(Associated Activity diagram) (components 0-7). The ABS Main unit is the major decision unit regarding the braking levels for individual wheels, while Load Compensator unit assists with computing adjustment factors from wheel load sensor inputs. Components 4-7 represent transceiver software components associated with each wheel, and communicate with sensors and brake actuators. *Brake Paddle* is the software

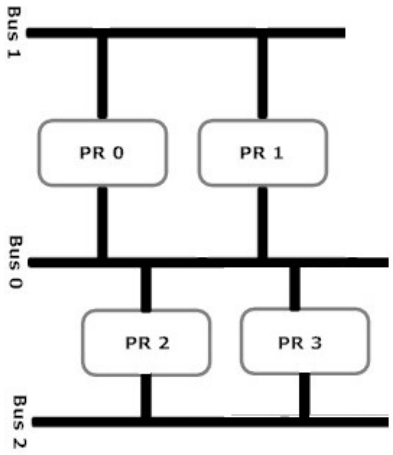



FIGURE 3.7: Hardware topology

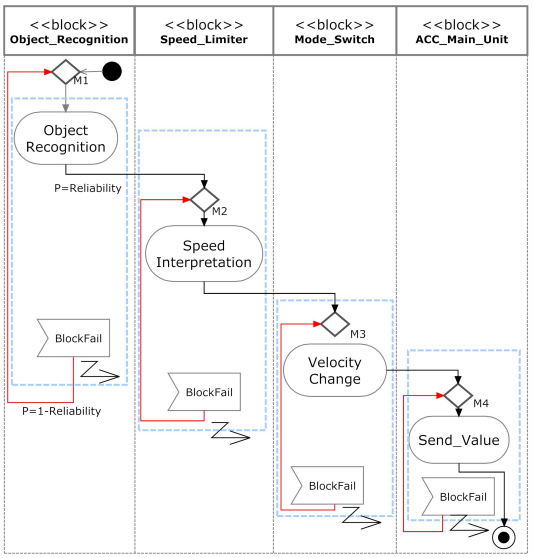



FIGURE 3.8: Activity diagram for Adaptive Cruise Control

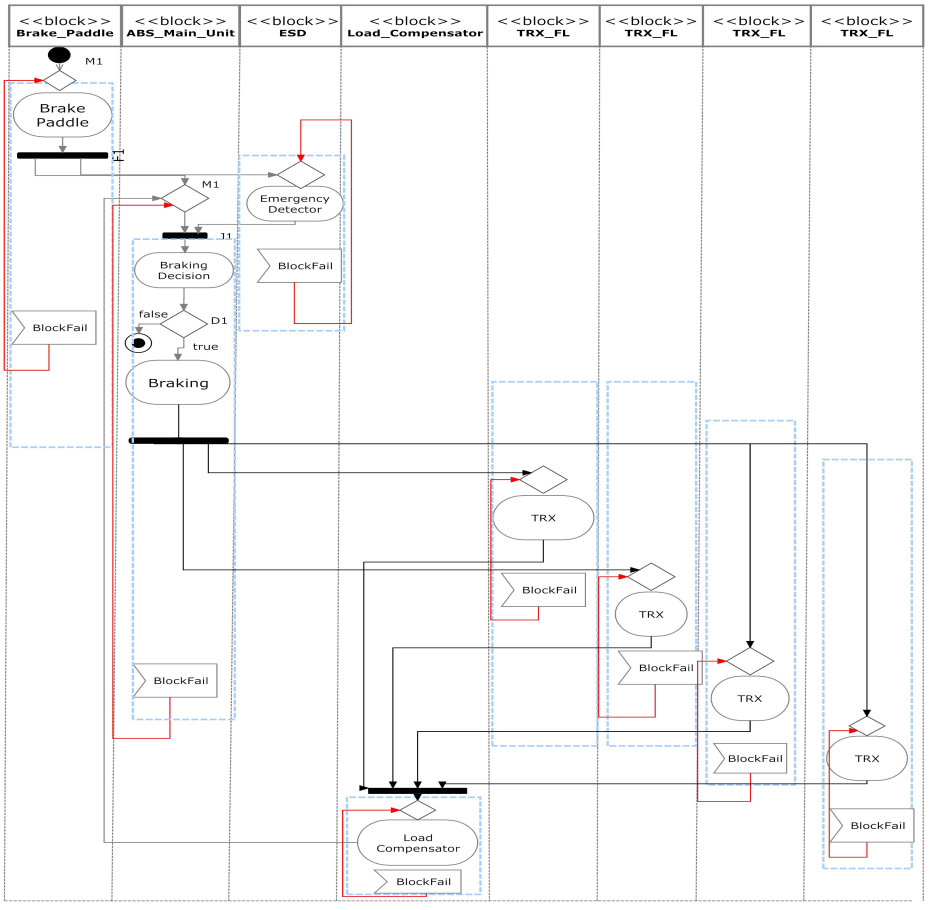



FIGURE 3.9: Activity diagram for Anti-lock Brake System (Left)

component that reads from the *Paddle Sensor* and sends the data to the emergency stop detection software module.

TABLE 3.3: Allocation results

| Hardware | Components |
|---|---|
| PR 0 | 4,6,11,9,7 |
| PR 1 | 2,5,1 |
| PR 2 | 3,0 |
| PR 3 | 8,10 |

**Adaptive Cruise Control (ACC)**: The aim of this component is to avoid crashes by reducing speed once the slower vehicle in front is detected. The main software components used by ACC are components 8-11 as depicted in Fig.3.6 (Right) and Fig.3.9(Associated Activity diagram). Service initialization is possible at Object Recognition software that communicates with sensors. Speed Limit, Mode Switch and Brake Paddle contributes to triggering of the service. The captured data are processed by the ACC Main Unit and Human Machine Interface to communicate with actuators.

### 3.7.1 Evaluation

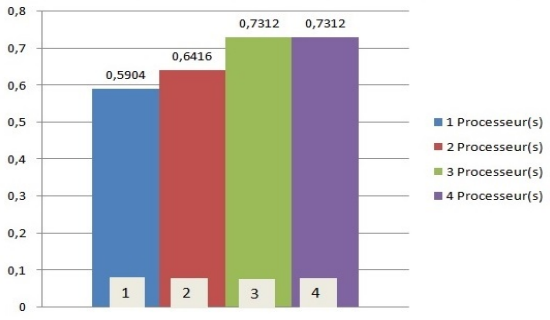



FIGURE 3.10: ABS Reliability

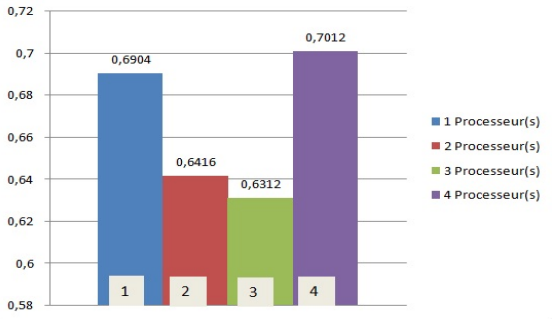



FIGURE 3.11: ACC Reliability

Despite the low number of components and states (i.e.PRISM states) generated after mapping, the deployment-space exploration took 14000 seconds which is approximately two hours and a half. Our plug-in records each result of checked model

```
mdp
const double P1;
const double P2;
const double P3;
const double P4;
module ACC
Initial : bool init true;
Object_Recognition: bool init false;
Speed_Interprestation:bool init false;
Velocity_Change : bool init false;
Send_Value : bool init false;
Final: bool init false;

Proc_Fail: bool init false;
    .........

D1: bool init false;
D2: bool init false;
    ...........

M11: bool init false;
M12: bool init false;
M1: bool init false;
[Initial] Initial -> (Initial'=false)&(M11'=true);
[M11]M11 -> (M11'=false)&(M1'=true);
[M12]M12 -> (M12'=false)&(M1'=true);
[M1]M1 -> (M1'=false)&(Object_Recognition'=true);
[Object_Recognition] Object_Recognition -> 1.0:(D1'=true)&
(Object_Recognition'=false);

[D1] D1 -> P1:(M21'=true)& (D1'=false)+(1-P1):(Block1_Fail'=true)&(D1'=false);
[Block1_Fail] Block1_Fail-> (Block1_Fail'=false)& (M12'=true);
[Speed_Interpretation]Speed_Interpretation -> 1.0:(SpeedInterprestation'=false)
&(D2'=true);
[D2] D2 -> P2:(M31'=true)& (D2'=false)+(1-P2):(Block2_Fail'=true)&(D2'=false);
[Block2_Fail] Block2_Fail-> (Block2_Fail'=false)& (M22'=true);
[Velocity_Change] Velocity_Change -> 1.0:(Velocity_Change'=false)&
(D3'=true);
[D3] D3 -> P3:(M41'=true)& (D3'=false)+(1-P3):(Block3_Fail'=true)&(D3'=false);
[Block3_Fail] Block3_Fail-> (Block3_Fail'=false)& (M32'=true);
[Send_Value] Send_Value -> 1.0:(Send_Value'=false)&(D4'=true);
[D4] D4 -> P4:(Final'=true)& (D4'=false)+(1-P4):(Block3_Fail'=true)&(D4'=false);
[Block4_Fail] Block4_Fail-> (Block4_Fail'=false)& (M42'=true);
endmodule
label "ProcFail" = Block1_Fail|Block2_Fail|Block3_Fail|Block4_Fail;
```

LISTING 3.1: ACC PRISM code fragment

to get the minimum probability of each configuration candidates. During this process 65792 candidates was explored and performed on i7 CPU 2.67GHz with 8.0GB of RAM. The Listing.3.1 shows the PRISM code that can be parametrized using the constants values. The constant values are filled with our plug-in during the model checking process.

Reliability property to analyze ACC and ABS system using PRISM is; the probability that a ACC and ABS will need to be replaced by a new one in 129600 time steps:

$$Pmin =?[F \leq T\ (\text{“}Proc\_Fail\text{”})], T = 129600$$

*Proc_Fail = true* asserts that the processor is in failing state. We labeled our PRSIM code with the label ProcFail that corresponds to the failing state of each block:

$$label\text{“}Proc\_Fail\text{”} = Block1\_Fail|Block2\_Fail|Block3\_Fail|Block4\_Fail$$

The analysis results of reliability property obtained from PRISM are as follows: The result of the property in case of ACC is shown in Fig.3.10; the result of property in case of ABS is shown in Fig.3.11. To choose the best deployment; we have to maximize the life duration (i.e. After 129600 time steps) and postpone the system failure. In Table.3.3, we show the best software deployment candidate relative to the probabilistic results. The acceptable deployment in Fig.3.10 shows that the processor failure occurs after 129600 time steps with probability near to 64.16% where the acceptable deployment in Fig.3.11, the processor failure occurs after 129600 time steps with probability near to 63.12%. The cases where the blocks deployed on one processor are not considered in our approach.

## 3.8 Conclusion

In this chapter, we presented a case study for Multiprocessing deployment of Component based Architecture for embedded systems in automotive domain. The approach is based on Blocks/IBD diagrams for component organization with behavior expressed by SysML activity diagram. The approach consists on capturing the different deployment candidates in PRISM language. The near-optimal deployment is selected when the reliability property is satisfied especially when system failure is postpone in time. Compared to the existing works, which use Meta-heuristic algorithms , our approach leverages probabilistic modeling thus allowing the assessment of properties formally expressed in probabilistic temporal logic.

# CONCLUSION

In this thesis, we presented a formal verification approach of systems modeled by using SysML activity diagrams. The objective is to alleviate errors and failures that can emerge in system design-flow in order to reduce the cost of maintenance and repairing as soon as possible from the implementation. The proposed approach use SysML activity diagram with time annotations using MARTE profile to evaluate the system behavior at different periods of time. Compared to Grobelna et al.[51], which use NuSMV[53] for state reachability in activity diagram, our approach leverages probabilistic and timed modeling, allowing the assessment of properties expressed in probabilistic temporal logic. With respect to Ouchani et al.[18] which use a probabilistic model to check the SysML activity diagram with difficulty to predict the system behavior due to the static description, we employ a probabilistic and timed model capturing system governed by time constraints. Moreover, in contrast to Marinescu et al.[44], which use C programs as component behavior; only components (without extracting C behavior) are mapped to UPPAAL that result in difficulties to evaluate accurately the system (the core behavior is hidden), our approach make decisions at specification level using SysML activity diagram.

To deal with the partitioning problem, we presented a deployment-exploration approach of embedded software modeled by using SysML internal blocks diagram (IBD). The first objective is to validate a deployment in case of hardware failures that generally emerge at the implementation level. The proposed approach uses SysML internal blocks diagram with additional real time system annotations using MARTE profile. Compared to Meedeniya et al.[70] and Thiruvady et al.[73], which use genetic algorithms, our approach leverages probabilistic modeling thus allowing the assessment of properties formally expressed in probabilistic temporal logic. With respect to Besnard et al.[77] which use a scheduling algorithm for software partitioning to check the AADL Language results in difficulty to obtain a good partitions due to the constraints limited to time, we employ a probabilistic and timed model capturing system governed by hardware and software constraints (i.e. time and probability). Moreover, in contrast to Etienne et al.[29], which is based on code generated for performance estimations, area and early power figures. Our approach extracts the relevant informations directly from the specification and mapped the behavior to PRISM for model checking.

The research contribution of this work consists on capturing the activity diagrams with time and probability features as probabilistic timed automata supported by PRISM language. We proposed a calculus dedicated to SysML Activity diagrams that captures precisely the underlying semantics. In addition, we formalized PRISM language and showing its semantics. Moreover, we proved the soundness of our proposed approach by defining the relation between the semantics of the mapped diagrams and the resulting PRISM models. By this relation, we proved the preservation of the satisfiability of PCTL properties. We have shown the effectiveness of our approach in realistic case study: Embedded automotive control system where its behavior is evaluated.

The practical advantages of the proposed approach in the context of deployment,

consists on providing key decision for the acceptable configuration via probabilistic model checking. In addition, the results of our approach using PRISM can be plotted as graphs that can be inspected for interpretations and anomalies detection. The limitations of the proposed approach relate to the size of the state-space associated with a corresponding model.

## Future work

The presented work can be extended by investigating several directions. First, our approach can be extended to support more system features such as energy and support more diagrams such as timing and sequence diagrams. Concerning the specification part of the framework, the requirements could be expressed in dedicated diagram such as requirement diagrams of SysML. Also, we desire to build a strategy to develop a platform for system of systems (SoS). For verification-time minimization part, three direction can be followed; either to develop a new verification algorithm based on powerful engine or explore the statistical model checking which proved its usefulness or develop a new abstraction algorithm that minimizes the constructed graph model. In case of deployment, not only reliability could be handled but also availability and maintainability requirements of embedded systems. In addition, low level view could be generated automatically from our original model at high abstraction view into C/C++/SystemC with a precise hardware characteristics for validation and accurate estimations on real-time system behavior. Finally, more complex systems could be validated by our framework.

# Appendix A

# ABBREVIATIONS

| | |
|---|---|
| **BNF** | Backus-Naur Form |
| **CTL** | Computation Tree Logic |
| **CTMC** | Continuous Time Markov Chains |
| **DTMC** | Discrete Time Markov Chains |
| **ERTS** | Embedded Real-time System |
| **INCOSE** | International Council on Systems Engineering |
| **LTL** | Linear Temporal Logic |
| **MDP** | Markov Decision Processes |
| **MARTE** | Modeling and Analysis of Real-Time and Embedded systems |
| **NuAC** | New Activity Calculus |
| **OMG** | Object Management Group |
| **PCTL** | Probabilistic Computation Tree Logic |
| **PRISM** | PRobabilistIc Symbolic Model checker |
| **PA** | Probabilistic Automata |
| **PTA** | Probabilistic Timed Automata |
| **PN** | Petri Nets |
| **SPN** | Stochastic Petri Nets |
| **SysML** | Systems Modeling Language |
| **UML** | Unified Modeling Language |

# REFERENCES


1. Vaclav R ajlich. " software e volution a nd m aintenance". I n *P roceedings o f the o n Future of Software Engineering*, FOSE 2014, New York, NY, U SA, 2014. ISBN 9 78-1-4503-2865-4. (2014), 133–144

2. *OMG Systems Modeling Language ( Object Management Group SysML)*. O. M. Group (Ed.), (2012).

3. *OMG Unified Modeling Language: Superstructure 2.1.2.Object Management Group*. O. M. Group (Ed.), (2007).

4. U.S Dept of Commerce. "Free nist software tool boosts detection of soft-warebugs".http://2010-2014.commerce.gov/blog/2010/11/09/free-nist-software-tool-boosts-detection-software-bugs, (2010).

5. Brigitte Pientka. "Proof pearl: The power of higher-order encodings in the logical framework lf". In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages . Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74590-7. (2001), 246–261.

6. Yves Bertot and Pierre Cast´eran. "*Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*". Springer, (2004).

7. Thomas Ball, Ella Bounimova, Rahul Kumar, and Vladimir Levin. "Slam2: Static driver verification with under 4"In *Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design*, FMCAD '10, Austin, TX, 2010. FMCAD Inc, (2010), 35–42.

8. EdmundM. Clarke. "The birth of model checking". In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages . Springer Berlin Heidelberg, 2008. ISBN 978- 3-540-69849-4. (2008), 1–26.

9. Christel Baier and Joost-Pieter Katoen."*Principles of Model Checking" (Representation and Mind Series)*.The MIT Press. ISBN 026202649X, 9780262026499, (2008).

10. Marta Kwiatkowska, Gethin Norman, and David Parker. "Prism 4.0: Verification of probabilistic real-time systems". In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011. ISBN 978-3-642- 22109-5. doi: 10.1007/978-3-642-22110-1 47, (2011), 585–591.

11. Rashid Layali, Pattabiraman Karthik., and Gopalakrishnan Sathish. "Characterizing the impact of intermittent hardware faults on programs". *Reliability, IEEE Transactions on*, 64(1), March 2015. ISSN 0018-9529. doi: 10.1109/ TR.2014.2363152, V.6., (2015), 297–310.

12. Andrade Ermeson, Maciel Paulo Romero Martins, Callou Gustavo Rau de Almeida, and Nogueira Bruno. "A methodology for mapping sysml activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints". In *Digital Society, 2009. ICDS '09. Third International Conference on*, ( 2009), 266–271.

13. Andrade Ermeson, Maciel Paulo Romero Martins, Callou Gustavo Rau de Almeida, and Nogueira Bruno."Mapping sysml state machine diagram

to time petri net for analysis and verification of embedded real-time systems with energy constraints". In *Advances in Electronics and Micro-electronics, 2008. ENICS '08. International Conference on*, (2008). 1–6,

14. Doligalski Michal and Adamski Marian. "Uml state machine implementation in fpga devices by means of dual model and verilog". In *Industrial Informatics (IN-DIN), 11th IEEE International Conference on, (2013),* 177–184,

15. Daniel Chaves Cafe, Filipe Vinci dos Santos, Cecile Hardebolle, Christophe Jacquet, and Frederic Boulanger. "Multi-paradigm semantics for simulating sysml models using systemc-ams". In *Specification Design Languages (FDL), 2013 Fo-rum on*, (2013), 1–8.

16. Jarraya Yosr, Soeanu Andrei, Debbabi Mourad, and Hassaine Fawzi. "Automatic verification and performance analysis of time-constrained sysml activity diagrams". In *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops* (2007), 515-522.

17. George Renu and Samuel Philip. "Improving design quality by automatic verification of activity diagram syntax". In *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*, (2012). 303–308

18. Samir Ouchani, Otmane A¨ıt Mohamed, and Mourad Debbabi. "A formal verifica-tion framework for sysml activity diagrams". *Expert Systems with Applications*, ISSN 0957-4174. V.6, (2014), 2713 – 2728.

19. Norman Gethin, Parker David, and Sproston Jeremy. "Model checking for probabilistic timed automata". *Formal Methods in System Design*. ISSN 0925-9856. doi: 10.1007/s10703-012-0177-x, V.2, (2013), 164–190..

20. Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. "The ins and outs of the probabilistic model checker mrmc". *Performance Evaluation*, V.2, (2011), 90–104.

21. Richard Schmidt. "Section 1. software engineering fundamentals". In Richard F. Schmidt, editor, *Software Engineering*. Morgan Kaufmann, Boston. ISBN 978-0-12-407768-3. , (2013), 1 – 6

22. David Garlan, Felix Bachmann, James Ivers, Judith Stafford, Len Bass, Paul Clements, and Paulo Merson. "*Documenting Software Architectures: Views and Beyond*". Addison-Wesley Professional, 2nd edition, (2010).

23. Ian Gorton. Understanding software architecture. In *Essential Software Architectur*s. Springer Berlin Heidelberg, (2011). ISBN 978-3-642-19175-6, (2011) ,1-16.

24. Jan Carlson, John H°akansson, and Paul Pettersson. Saveccm: An analysable component model for real-time systems. *Electronic Notes in Theoretical Computer Science*. ISSN 1571-0661. doi: http://dx.doi. org/10.1016/ j.entcs.2006.05.019. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005) Proceedings of the In-ternational Workshop on Formal Aspects of Component Software, (2006), 127 – 140.

25. Petr Hosek, Tomas Pop, Tomas Bures, Petr Hnetynka, and Michal Malohlava. Comparison of component frameworks for real-time embedded systems. In Lars Grunske, Ralf Reussner, and Frantisek Plasil, editors, *Component-Based Software Engineering*, volume 6092 of *Lecture Notes in Computer Science*,

pages 21–36. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13237-7. doi: 10.1007/978-3-642-13238-4 _2, (2010, 21-36.

26. Galib Krdzalic and Alexander Driss. Software architecture without autosar. *Auto Tech Review*. ISSN 2250-3390. doi: 10.1365/s40112-014-0593-y, (2014), V.4.

27. Scott Hissam, James Ivers, Daniel Plakosh, and Kurt Wallnau. Pin component technology (v1.0) and its c interface. Technical Report CMU/SEI-2005-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, (2005).

28. Basu Ananda, Bensalem Saddek, Bozga Marius, Combaz Jacques, Jaber Mohamad, Thanh-Hung Nguyen, and Sifakis Joseph. Rigorous component-based system design using the bip framework. *Software, IEEE*. ISSN 0740-7459. doi: 10.1109/MS.2011.27, V.23,(2011), 41–48,.

29. Gilles Lasnier, Bechir Zalila, Laurent Pautet, and J´erome Hugues. Ocarina : An environment for aadl models analysis and automatic code generation for high integrity applications. In Fabrice Kordon and Yvon Kermarrec, editors, *Reliable Software Technologies – Ada-Europe 2009*, volume 5570 of *Lecture Notes in Computer Science*, . Springer Berlin Heidelberg, ISBN 978-3-642-01923-4. doi: 10.1007/978-3-642-01924-1_ 17, (2009), 237–250.

30. Vidal Jorgiano, de Lamotte Florent, Gogniat Guy, Soulard Philippe, and Diguet Jean-Philippe. A co-design approach for embedded system modeling and code generation with uml and marte. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 226–231, April 2009. doi: 10.1109/DATE.2009.5090662.

31. Brosse Etienne, Quadri Imran Rafiq, Sadovykh Andrey, Ieromnimon Frank, Kritharidis Dimitrios, Catrou Rafael, and Sarlotte Michel. Enosys fp7 eu project:

An integrated modeling and synthesis flow for embedded systems design. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–5, July 2012. doi: 10.1109/ReCoSoC.2012.6322880.

32. Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The {COMPLEX} methodology for uml/marte modeling and design space exploration of embedded systems. *Journal of Systems Architecture*, 60(1):55 – 78, 2014. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2013.10.003.

33. Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0123743796, 9780080558363, 9780123743794.

34. Mourad Debbabi, Fawzi Hassane, Yosr Jarraya, Andrei Soeanu, and Luay Alawneh. *Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 3642152279, 9783642152276.

35. Clarke Edmund and E. Allen Emerson. “design and synthesis of synchronization skeletons using branching-time temporal logic”. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag. ISBN 3-540-11212-X.

36. Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. In Kim Guldstrand Larsen and Peter Niebert, editors, *Formal Modeling and Analysis of Timed Systems*, volume 2791 of *Lecture Notes in Computer Science*, pages 105–120. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-21671-1. doi: 10.1007/978-3-540-40903-8_9.

37. Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In P.S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-60692-5. doi: 10.1007/3-540-60692-0_70.

38. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. ISSN 0934-5043. doi: 10.1007/BF01211866.

39. Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. A game based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010. ISSN 0925-9856. doi: 10.1007/s10703-010-0097-6.

40. Jasper Berendsen, Taolue Chen, and David N. Jansen. *Theory and Applications of Models of Computation: 6th Annual Conference, TAMC 2009, Changsha, China, May 18-22, 2009. Proceedings*, chapter Undecidability of Cost-Bounded Reachability in Priced Probabilistic Timed Automata, pages 128–137. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02017-9. doi: 10.1007/978-3-642-02017-9_16. URL http://dx.doi.org/10.1007/978-3-642-02017-9_16.

41. Alessandro Cimatti, Clarke Edmund, Fausto Giunchiglia, and Marco Roveri. Nusmv: A new symbolic model verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 495–499, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66202-2.

42. Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.

43. Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

44. Gerd Behrmann, Alexandre David, and KimG. Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23068-7. doi: 10.1007/978-3-540-30080-9_7.

45. Arnd Hartmanns. Modest – a unified language for quantitative models. In *FDL*, pages 44–51, September 2012.

46. Frédéric Mallet and Robert de Simone. Marte: A profile for rt/e systems modeling, analysis–and simulation? In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 43:1–43:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.

47. Xiaopu Huang, Qingqing Sun, Jiangwei Li, and Tian Zhang. Mde-based verification of sysml state machine diagram by uppaal. In Yuyu Yuan, Xu Wu, and Yueming Lu, editors, *Trustworthy Computing and Services*, volume 320 of *Communications in Computer and Information Science*, pages 490–497. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-35794-7. doi: 10.1007/978-3-642-35795-4_62.

48. Daniel Knorreck, Ludovic Apvrille, and Pierre de Saqui-Sannes. Tepe: A sysml language for time-constrained property modeling and formal verification. *SIGSOFT Softw. Eng. Notes*, 36(1):1–8, January 2011. ISSN 0163-5948. doi: 10.1145/1921532.1921556.

49. Mikael Akerholm, Jan Carlson, Johan Fredriksson, Hans Hansson, John Håkansson, Anders Möller, Paul Pettersson, and Massimo Tivoli. The save approach to component-based development of vehicular systems. *J. Syst. Softw.*, 80(5):655–667, May 2007. ISSN 0164-1212. doi: 10.1016/j.jss.2006.08.016.

50. Raluca Marinescu, Henrik Kaijser, Marius Mikucionis, Cristina Seceleanu, Henrik Lonn, and Alexandre David. "Analyzing industrial architectural models by simulation and model-checking". In Cyrille Artho and Peter Csaba ˜A–lveczky, editors, *Formal Techniques for Safety-Critical Systems*, volume 476 of *Communications in Computer and Information Science*, pages 189–205. Springer International Publishing, 2015. ISBN 978-3-319-17580-5. doi: 10.1007/978-3-319-17581-2_13.

51. Simulink Link. Simulink. http://www.mathworks.com/simulink.

52. Gaogao Yan, Xue-Yang Zhu, Rongjie Yan, and Guangyuan Li. F"ormal through-put and response time analysis of marte models". In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering*, volume 8829 of *Lec-ture Notes in Computer Science*, pages 430–445. Springer InternationalPub-lishing, 2014.ISBN978-3-319-11736-2. _

53. Peter Feiler. "Model-based validation of safety-critical embedded systems". In *Aerospace Conference, 2010 IEEE*, pages 1–10, March 2010. doi: 10.1109/AERO.2010.5446809.

54. Singhoff Frank, Legrand J´erˆome, Nana Laurent Tchamnda, and Marc´e Lionel. "Cheddar: A flexible real time scheduling framework". *Ada Lett.*, XXIV(4):1–8, November 2004. ISSN 1094-3641. doi: 10.1145/1046191.1032298.

55. Mourad Debbabi, Fawzi Hassa¨ıne, Yosr Jarraya, Andrei Soeanu, and Luay Alawneh. "Probabilistic model checking of sysml activity diagrams". In *Verification and Validation in Systems Engineering*, pages 153–166. Springer Berlin Heidel-berg, 2010. ISBN 978-3-642-15227-6. _ doi: 10.1007/978-3-642-15228-3 9.

56. Ouchani Samir, Jarraya Yosr, and Ait Mohamed Otmane. "Model-based systems security quantification". In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 142–149, July 2011.

57. Iwona Grobelna, Micha? Grobelny, and Marian Adamski. "Model checking of uml activity diagrams in logic controllers design". In Wojciech Zamojski, Jacek Mazurkiewicz, Jaros?aw Sugier, Tomasz Walkowiak, and Janusz Kacprzyk, edi-tors, *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30 ? July 4, 2014, Brunow, Poland*, volume 286 of *Advances in Intelligent Systems and Computing*, pages 233–242. Springer International Publishing, 2014. ISBN 978-3-319-07012-4.

58. Iwona Grobelna. "Formal verification of logic controller specification by means of model checking". Lecture Notes in Control and Computer Science. Springer International Publishing, 2013.

59. RicardoJ. Rodriguez, LarsAke Fredlund, Anngel Herranz, and Julio Marino. "Execution and verification of uml state machines with erlang". In Dimitra Gi-annakopoulou and Gwen SalaÃn, editors, *Software Engineering and Formal Methods*, volume 8702 of *Lecture Notes in Computer Science*, pages 284–289. Springer International Publishing, 2014. ISBN 978-3-319-10430-0. doi: 10.1007/978-3-319-10431-7_22.

60. Shuang Liu, Yang Liu, Jun Sun, Manchun Zheng, Bimlesh Wadhwa, and Jin Song Dong. "Usmmc: A self-contained model checker for uml state ma-chines". In *Proceedings of the 2013 9th Joint Meeting on Foundations of Soft-ware Engineering*, ESEC/FSE 2013, pages 623–626, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2237-9. doi: 10.1145/2491411.2494595.

61. Thomas Noll. Safety, dependability and performance analysis of aerospace sys-tems. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for*

*Safety-Critical Systems*, volume 476 of *Communications in Computer and Information Science*, pages 17–31. Springer International Publishing, 2015. ISBN 978-3-319-17580-5. doi: 10.1007/978-3-319-17581-2_2.

62. Hoque Khaza Anuarul, Ait Mohamed Otmane, Savaria Yvon, and Thibeault Claude. "Early analysis of soft error effects for aerospace applications using probabilistic model checking". In Cyrille Artho and Peter Csaba ¨Olveczky, editors, *Formal Techniques for Safety-Critical Systems*, volume 419 of *Communications in Computer and Information Science*, pages 54–70. Springer International Pub-lishing, 2014. ISBN 978-3-319-05415-5. _

63. Robin Milner. *C"ommunicating and Mobile Systems: The Pi-calculus"*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0-521-65869-1.

64. Gethin Norman, Catuscia Palamidessi, David Parker, and Peng Wu. "Model checking the probabilistic $\pi$-calculus". In *Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07)*, pages 169–178. IEEE Computer Society, 2007.

65. Samir Ouchani, Ait Mohamed Otmane, and Debbabi Mourad. "A probabilistic ver-ification framework of sysml activity diagrams". In *Intelligent Software Methodolo-gies, Tools and Techniques (SoMeT), 2013 IEEE 12th International Conference on*, pages 165–170, Sept 2013.

66. C. A. R. Hoare. "*Communicating Sequential Processes*." Prentice-Hall, Inc., Up-per Saddle River, NJ, USA, 1985. ISBN 0-13-153271-5.

67. Mordechai Ben-Menachem. "Reactive systems: Modelling, specification and verification"; is written by l. aceto, et al; and published by cambridge university press; distributed by cambridge university press; © 2007, (hardback), isbn 978- 0-521-87546-2, pp. 300. *SIGSOFT Softw. Eng. Notes*, 35(4):34–35, July 2010. ISSN 0163-5948. doi: 10.1145/1811226.1811243.

68. Roberto Segala. "A compositional trace-based semantics for probabilistic automata". In Insup Lee and ScottA. Smolka, editors, *CONCUR '95: Con-currency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 234–248. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-60218-7. doi: 10.1007/3-540-60218-6 17.

69. Samir Ouchani, Otmane A¨ıt Mohamed, and Mourad Debbabi. "A property-based abstraction framework for sysml activity diagrams". *Know.-Based Syst.*, . ISSN 0950-7051, V.56, 328–343, (2014).

70. Garey Michael and Johnson David. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

71. Jean-Paul Arcangeli, Raja Boujbel, and S´ebastien Leriche. "Automatic deploy-ment of distributed software systems: Definitions and state of the art". *Jour-nal of Systems and Software*, 103(0):198 – 218, 2015. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2015.01.040.

72. T. Saxena and G. Karsai. "A meta-framework for design space exploration". In *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, (2011) 71–80.

73. Sander van der Burg and Eelco Dolstra. "Disnix: A toolset for distributed deployment. *Science of Computer Programming"*, ISSN 0167- 6423. doi: http://dx.doi.org/10.1016/j.scico.2012.03.006. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010), 52 – 69, 2014..

74. Bernhard Mattes. Occupant protection systems. In Konrad Reif, editor, *Brakes,* "*Brake Control and Driver Assistance Systems"*, Bosch Professional Automotive

Information, pages 162–179. Springer Fachmedien Wiesbaden, 2014. ISBN 978-3-658-03977-6. doi: 10.1007/978-3-658-03978-3_14.

75. Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. "Reliability-driven deployment optimization for embedded systems". *Journal of Systems and Software*, 84(5):835 – 846, 2011. ISSN 0164-1212. doi: http: //dx.doi.org/10.1016/j.jss.2011.01.004.

76. Melanie Mitchell and Melanie Mitchell. "*An Introduction to Genetic Algorithms"*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-13316-4.

77. Gookhyun Kim, Jinhee Park, and Jongmoon Baik. "An effective approach to identifying optimal software reliability allocation with consideration of multiple constraints". In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, (2012), 541–546.

78. Dhananjay Thiruvady, I. Moser, Aldeida Aleti, and Asef Nazari. "Constraint programming and ant colony system for the component deployment problem". *Procedia Computer Science*, 29(0):1937 – 1947, 2014. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2014.05.178. 2014 International Conference on Computational Science (2014), 1937 – 1947.

79. Francisco Assis Moreira do Nascimento, Oliveira, and Fl´avioRech Wagner. "A model-driven engineering framework for embedded systems design". *Innovations in Systems and Software Engineering*, 2012. ISSN 1614- 5046. 8(1):19–33.

80. Marcio Oliveira, Eduardo Bri˜ao, and Fl´avio Nascimento, Franciscoand Wagner. "Model driven engineering for mpsoc design space exploration." In *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design*, SBCCI '07, New York, NY, USA. ACM. ISBN 978-1-59593-816-9. (2007), 81–86.

81. Hector Posadas, Pablo Penil, Alejandro Nicolas, and Eugenio Villar. "Auto-matic synthesis of embedded 4SW0 for evaluating physical implementation alternatives from uml/marte models supporting memory space separation". *Microelectronics Journal*. ISSN 0026-2692. doi: http://dx.doi.org/10.1016/j.mejo.2013.11.003. DCIS'12 Special Issue, (2014),1281 – 1291.

82. Loıc Besnard, Adnan Bouakaz, Thierry Gautier, Paul Le Guernic, Yue Ma, Jean-Pierre Talpin, and Huafeng Yu. "Timed behavioural modelling and affine scheduling of embedded software architectures in the 4AADL0 using polychrony". *Science of Computer Programming*, 106(0):54 – 77, 2015. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2014.05.014. Special Issue: Architecture-Driven Semantic Analysis of Embedded Systems, (2015), 54-77.

83. Yue Ma, Huafeng Yu, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin, Lo¨ıc Besnard, and Maurice Heitz. "Toward polychronous analysis and validation for timed software architectures in aadl". In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, San Jose, CA, USA, 2013. EDA Consortium. ISBN 978-1-4503-2153-2, (2013), 1173–1178 .

84. Pankaj Kumar Nath and Dilip Datta. "Multi-objective hardware–software partitioning of embedded systems: A case study of 4JPEG0 encoder". *Applied Soft Computing*. ISSN 1568-4946. ( 2014), 30 – 41.

85. Clemens Szyperski. "*Component Software: Beyond Object-Oriented Programming"*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition. ISBN 0201745720, (2002).

86. Lenny Delligatti. "*SysML Distilled: A Brief Guide to the Systems Modeling Lan-guage"*. Addison-Wesley Professional, 1st edition. (2010)

87. Tudor Lascu, Jacopo Mauro, and Gianluigi Zavattaro. "A utomatic component deployment in the presence of circular dependencies". In Jos´e Luiz Fi-adeiro, Zhiming Liu, and Jinyun Xue, editors, *Formal Aspects of Component Software*, volume 8348 of *Lecture Notes in Computer Science*. Springer International Publishing. ISBN 978-3-319-07601-0. (2014), 254-272