Expert Systems with Applications 42 (2015) 7493–7510

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A quantitative verification framework of SysML activity diagrams under time constraints

Abdelhakim Baouya [a,*], Djamal Bennouar [b], Otmane Ait Mohamed [c], Samir Ouchani [d]

[a] CS Department, Saad Dahlab University, Blida, Algeria
[b] CS Department, University of Bouira, Algeria
[c] ECE Department, Concordia University, Montreal, Canada
[d] SnT Center, University of Luxembourg, Luxembourg

ARTICLE INFO

Article history:
Available online 29 May 2015

Keywords:
SysML activity diagram
Probabilistic Timed Automata
Model checking
PCTL

ABSTRACT

Time-constrained and probabilistic verification approaches gain 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 as input and produce their equivalent timed probabilistic automata that is/are 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 the 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 case study to illustrate the applicability of the proposed approach and to demonstrate its efficiency by analyzing a performability properties.

1. Introduction

Constraints on software development in terms of functionality, performance, reliability and time to market are becoming more stringent. Therefore, development reveals several challenges. Indeed, if in one side the applications are becoming increasingly complex, in the other side the market pressure for rapid development of these systems makes the task of their 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 (US Dept of Commerce, 2010). It has been shown in Fig. 1 (Rajlich, 2014) that the cost of system repairing during the maintenance in software development life cycle is approximately 67%. Therefore, accelerate the verification and maintenance process at preliminary design is extremely beneficial as compared to fixing them at the testing phase.

Recently, formal verification 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. There are mainly two ways for doing formal verification: *Theorem proving* and *model checking*.

Theorem proving needs to be operated by people with skills (Experts) in order to solve the problem. In addition, this approach need 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 is significant implementations of this approach such as HOL (Pientka, 2007), Coq (Bertot & Castéran, 2004).

Model checking is popular formal verification approach in software and hardware industry. For instance, SLAM (Ball, Bounimova, Kumar, & Levin, 2010) which is a Microsoft research project uses a model checking to verify device drivers are conform to their API specification. According to Clarke (2008), model checking is automatic and usually very fast. The user does not need to construct

* Corresponding author. Tel.: +213 794 005 852.
E-mail addresses: baouya.abdelhakim@gmail.com (A. Baouya), dbennouar@gmail.com (D. Bennouar), otmane.aitmohamed@concordia.ca (O.A. Mohamed), samir_ouchani@yahoo.com (S. Ouchani).

http://dx.doi.org/10.1016/j.eswa.2015.05.049
0957-4174/

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 (Baier & Katoen, 2008). 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 our paper we use PRISM language for probabilistic model checker (Kwiatkowska, Norman, & Parker, 2011) 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.

In System Engineering, SysML (system modeling language) is a standard language (OMG, 2012) 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) (OMG, 2007) and provides additional extensions to specify requirements such as probability. Behavioral specification is achieved in SysML using three diagrams: state machine, communication and activity diagram. Activity diagram is particularly described in our paper.

This paper proposes a probabilistic verification of SysML activity diagram that are guarded with time constraints. The proposed framework is depicted in Fig. 2, it takes SysML activity diagrams (OMG, 2012) as inputs to produce the probability of the PCTL property. The kernel of the framework is based on transforming the activity diagram to its equivalent Probabilistic-Timed Automata (PTA) expressed in PRISM language. We extract the adequate semantics model related to SysML activity diagrams. Then, we present the underlying semantics related to the produced PRISM model. Furthermore, we show that the relation between both semantics preserves the satisfiability of PCTL properties.

The remainder of this paper is structured as follows: Section 2 discusses related work. The preliminaries needed for our work are presented in the next section. Section 4 describes and formalizes the SysML activity diagrams. The semantics PRISM model checker is presented in Section 5. Section 6 provides a mapping algorithm of SysML activity diagrams into PRISM code and its soundness is proved in Section 7. Section 8 illustrates the application of our framework on a case study. Section 9 draws conclusions and lays out future work.

2. Related works

In this section, we depict the recent works related to the verification of behavioral models then we compare them with our proposed approach.

Andrade, Maciel, Callou, and Nogueira (2009) and Carneiro, Maciel, Callou, Tavares, and Nogueira (2008) 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 (Mallet & de Simone, 2008). They map only the states and the transitions into ETPN (Time Petri Net with Energy constraints). The approach is restricted on a sub set of artifacts with control flow (data flow is missed).

Doligalski and Adamski (2013) propose a verification and simulation of UML state machine. For this purpose, two mapping mechanisms are 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, Sun, Li, and Zhang (2013) propose a verification of SysML State Machine Diagram by extending the model with MARTE (Mallet & de Simone, 2008) to express the execution time. The tool has as input the sate machine diagram and as output Timed Automata (TA) expressed in UPPAAL syntax (Behrmann, David, & Larsen, 2004). UPPAAL uses CTL (Computational Tree Logic) properties to check if the model is satisfied with liveness and safety properties.

Cafe, Hardebolle, Jacquet, and Boulanger (2013) 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,

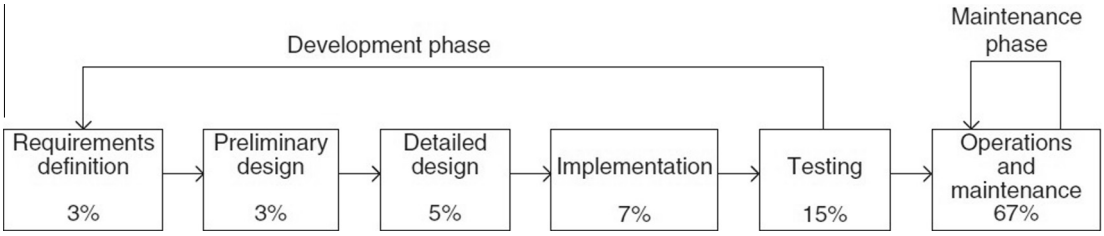


Fig. 1. Traditional software design life cycle (Rajlich, 2014).

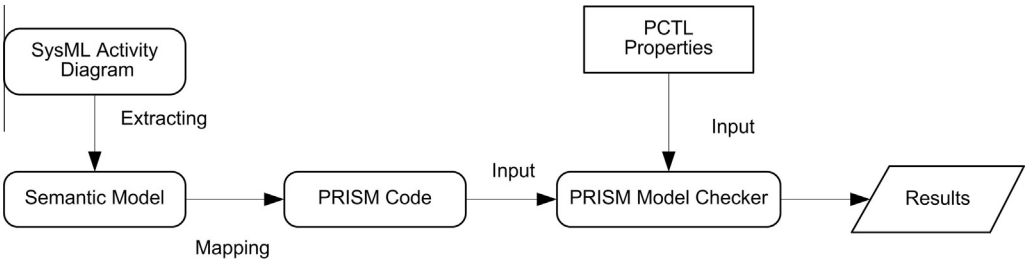


Fig. 2. Timed probabilistic verification framework for SysML activity diagram.

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.

kerholm et al. (2007) define a mapping rules that takes as input a SaveCCM component model (Carlson, Hkansson, & Pettersson, 2006) and extracts a different tasks in XML format using SaveToTimes tool. The results of the mapping is encoded in UPPAAL syntax. Using CTL properties, the model can be checked (e.g. Minimum execution time) to schedule the tasks (i.e. Components) of general assembly.

Jarraya, Soeanu, Debbabi, and Hassaine (2007) 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).

Ouchani, Mohamed, and Debbabi (2014b) propose a verification framework of SysML activity diagram. The authors address a sub set of SysML activity diagram artifacts with control flow. The different 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.

Debbabi, Hassane, Jarraya, Soeanu, and Alawneh (2010b) 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 object and time.

Ouchani, Jarraya, and Mohamed (2011) presents an approach based on probabilistic adversarial interactions between potential attackers and the system design models. These interactions result in a global model as composed SysML activity diagrams that support more additional artifacts than (Ouchani et al., 2014b) such as data objects. The result diagram is translated into PRISM language and a set of security and functional properties are verified against the composed model.

George and Samuel (2012) propose a syntax verification of UML activity diagram to ensure the correctness of a design that may be causing incorrect results or/and incorrect workflow. The algorithm is used to analyze a sub set of activity diagram constructs such as action nodes, decision nodes, fork nodes, join nodes, edges, branches, initial state, final states and guard conditions. The main purpose of the algorithm is prevent the errors at design phase by decomposing activity diagram in its basic elements and verify its individually such as components interconnection or identical guard condition at decision node.

Kaliappan, Koenig, and Kaliappan (2008) propose a verification approach for system workflow especially in communication protocol. The approach takes as input three UML diagrams: state machine diagram, activity diagram and sequence diagram. State machine diagram or activity diagram is converted into PROMELA code as a protocol model and its properties are derived from the sequence diagram as Linear Temporal Logic (LTL).

Lasnier, Zalila, Pautet, and Hugues (2009) 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) (Feiler, 2010) 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) (Singhoff, Legrand, Nana, & Marcé, 2004). For timing estimation and optimization, a scheduling tool is used based on software components timing constraints. After time optimization, the framework generates C/C++/ADA code that can be simulated.

Knorreck, Apvrille, and de Saqui-Sannes (2011) develop a framework for timing constraints verification for customized profile applied for real time system specification incorporating a combined UML and SysML diagrams. Each block diagram and its state machine is converted to corresponding automata in UPPAAL syntax. The properties are derived from parametric diagram as Computational Tree Logic (CTL).

Liu et al. (2013) propose a verification tool called UML State Machine Model Checking (USMMC). The tool verify UML state machine and processes verification using Linear Temporal Logic (LTL) properties. However, the mapping from UML state machine to USMMC input language is not given.

Grobelna, Grobelny, and Adamski (2014) 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 (Grobelna, 2013). The result model is mapped to NuSMV(Cimatti, Clarke, Giunchiglia, & Roveri, 1999) in order 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.

Hoque, Mohamed, Savaria, and Thibeault (2014) 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.

Noll (2015) presents the methodology that uses in COMPASS tool. The methodology targets aerospace systems and allows satisfactory for most safety–critical systems for aerospace on-board systems. The specification is based on the Architecture Analysis & Design Language (AADL). The analyses are mapped onto discrete, symbolic and probabilistic model checkers, but all of them are completely hidden away from the user by appropriate model-transformations. To model possible failures, the AADL model is enriched with behavioral error annex as state chart; when error occur, an error state is triggered. The toolset builds upon NuSMV (Cimatti et al., 1999), MRMC (Katoen, Zapreev, Hahn, Hermanns, & Jansen, 2011). Specifications are written in Probabilistic Computation Tree Logic (PCTL) and Continuous Stochastic Logic (CSL).

Abdulkhaleq and Wagner (2014) propose a method for safety analysis at the system level based on *System-Theoretic Process Analysis approach* (STPA). The process step for software development starts from specification level based on components interactions and identify the Unsafe Control Actions which are formalized into temporal logic. Symbolic Model Verifier (SMV) which was developed by McMillan (1992) is used to check the safety requirements. The limitation of the method is that the Component-based specification is not clearly described.

Marinescu et al. (2015) provide a framework for simulation and verification of EAST-ADL (Qureshi, Chen, Lönn, & Törngren, 2011). EAST-ADL is an architectural description language for automotive electronic systems. The specification level is a component-based architecture where behavior can be a set of xml-files and compiled

C-code or a Simulink model. The simulation is done by automatic transformation from the architectural model to a Simulink model. The verification is done by automatic transformation from the architectural model to a network of timed automata analysed by UPPAAL.

Jacobs and Simpson (2015) present a compositional approach to refinement and specification in the context of SysML. They consider the behavior of SysML blocks as a set of composed behavior of state machine and activity diagrams. The specification is mapped to a process-algebraic CSP where assertions about requirements can be proved or refuted (Jacobs & Simpson, 2013). The idea behind is to prove that the refinement is preserved and the automated analysis is computationally feasible.

Rodriguez, Fredlund, Herranz, and Marino (2014) present a tool for UML state machine diagram verification called UMerL. The tool is an interpreter of UML state machines implemented in Erlang (Fredlund & Svensson, 2007) that executes a system modelled as a collection of UML state machines. UMerL provides two functionalities: the system can be executed for simulation or verified. Using Linear Temporal Logic (LTL) the verification scenario can be checked and it provides a counterexample when an LTL property is violated.

Yan, Zhu, Yan, and Li (2014) use real-time model checking tool UPPAAL to find the possible best throughput and response time of MARTE models, and the best solution in the worst cases for both of them. The approach is based on adding the MARTE profile with its annotations (e.g. time processing) on UML diagrams such as a use case diagram, a deployment diagram and a set of activity diagrams. The result of mapping these three diagrams is a network of timed automata. Using Computation Tree Logic (CTL) formula, the properties on throughput and response time are formulated to check whether the network of timed automata satisfies the formulae. The mapping rules concerns a sub set of activity diagrams and did not include a call behavior or sub activities.

2.1. Comparison

As a summary, in Table 1 we compare our framework to the existing works by taking 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 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, include data workflow and time constraint at specification level. Our works has for objective to support: data workflow, time constraints, and formalize the SysML activity diagram. In addition, we implement the tool that allows the automatic verification.

3. Preliminaries

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 OMG (2012). Fig. 3 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 type: 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 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.

3.1.1. Actions execution

The following rules (Friedenthal, Moore, & Steiner, 2008) 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.

Table 1
Comparison with the existing works.

Approach	SysML	Time constraint	Data workflow	Formalization	Soundness	Automation
Liu et al. (2013), George and Samuel (2012), Grobelna et al. (2014), Abdulkhaleq and Wagner (2014) and Rodriguez et al. (2014)						√
Ouchani et al. (2011)	√	√	√			√
Lasnier et al. (2009), kerholm et al. (2007), Andrade et al. (2009), Carneiro et al. (2008) and Knorreck et al. (2011)	√					√
Kaliappan et al. (2008), Hoque et al. (2014), Noll (2015), Marinescu et al. (2015) and Yan et al. (2014)		√				√
Ouchani et al. (2014b)	√			√	√	√
Huang et al. (2013) and Jarraya et al. (2007)	√			√		√
Cafe et al. (2013) and Debbabi et al. (2010b)	√					√
Our	√	√	√	√	√	√

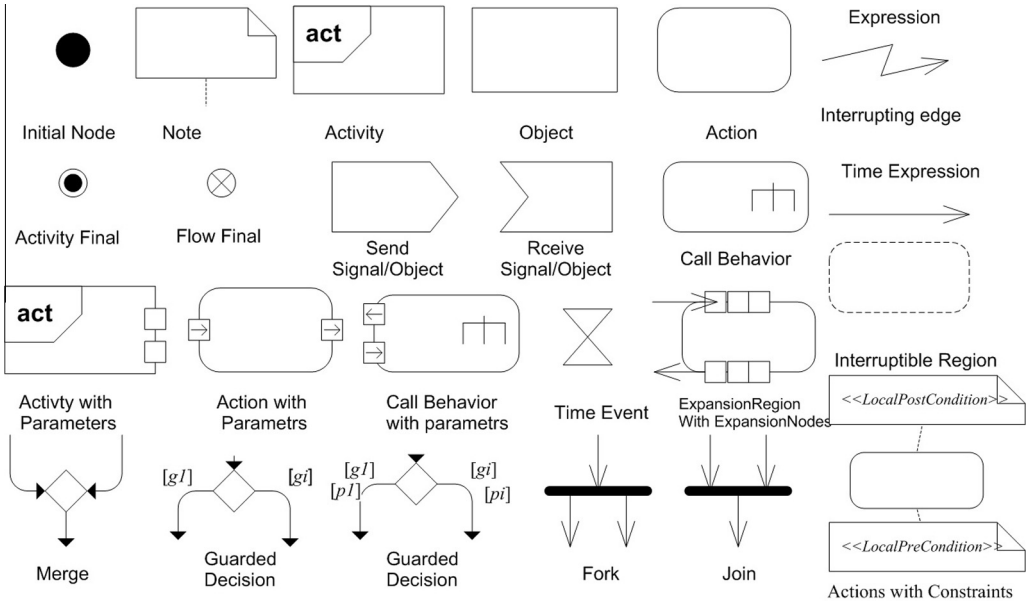


Fig. 3. A sub set of SysML activity diagram artifacts.

3.1.2. Time expression using SysML/MARTE

Friedenthal et al. (2008), Jarraya et al. (2007) and Debbabi, Hassane, Jarraya, Soeanu, and Alawneh (2010a) 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). However, annotation of time constraints on top of SysML activity diagrams is not clearly defined in the standard (OMG, 2012).

According to OMG SysML (OMG, 2012) 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 (Mallet & de Simone, 2008) 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 into 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 paper, we use execution time for quantitative verification.

Fig. 4 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 ⟨⟨*resourceUsage*⟩⟩.

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.

3.2. Probabilistic model checking

Model checking is a formal verification technique that given a finite-state model of a system and a formal property, systematically checks whether this property holds for that model. If not a counter example is generated. Probabilistic model checking is based on the analysis of systems that exhibit a probabilistic behavior. The model checked can be expressed as Discrete-Time Markov Chains (DTMC), Continuous-Time Markov chains (CTMC), Probabilistic Automata (PA) based on Markov Decision Process (MDP) or recently as Probabilistic Timed Automata (PTA) to describe a probabilistic model under time constraints. In this paper, we focus on Probabilistic Timed Automata (PTA).

Probabilistic Timed Automata (PTA) (Norman, Parker, & Sproston, 2013) are modeling formalism for systems that exhibit probabilistic, non-deterministic and real-time characteristic. Verification of PTAs permits analysis of a wide range of quantitative properties such as reliability and performance, e.g:

- The minimum probability to take a picture within 2 s;
- The minimum probability to take a picture within 2 s when flash is on.

PTAs can also be augmented with additional quantitative informations in the form of costs or rewards, e.g:

- The minimum expected time for completion of all tasks;
- The minimum expected energy to take a picture when flash is on.

This paper provides a formal definition of PTAs to specify and verify properties such as those listed above. 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 conditions (i.e., guards) of actions: only if the condition is fulfilled the action is enabled

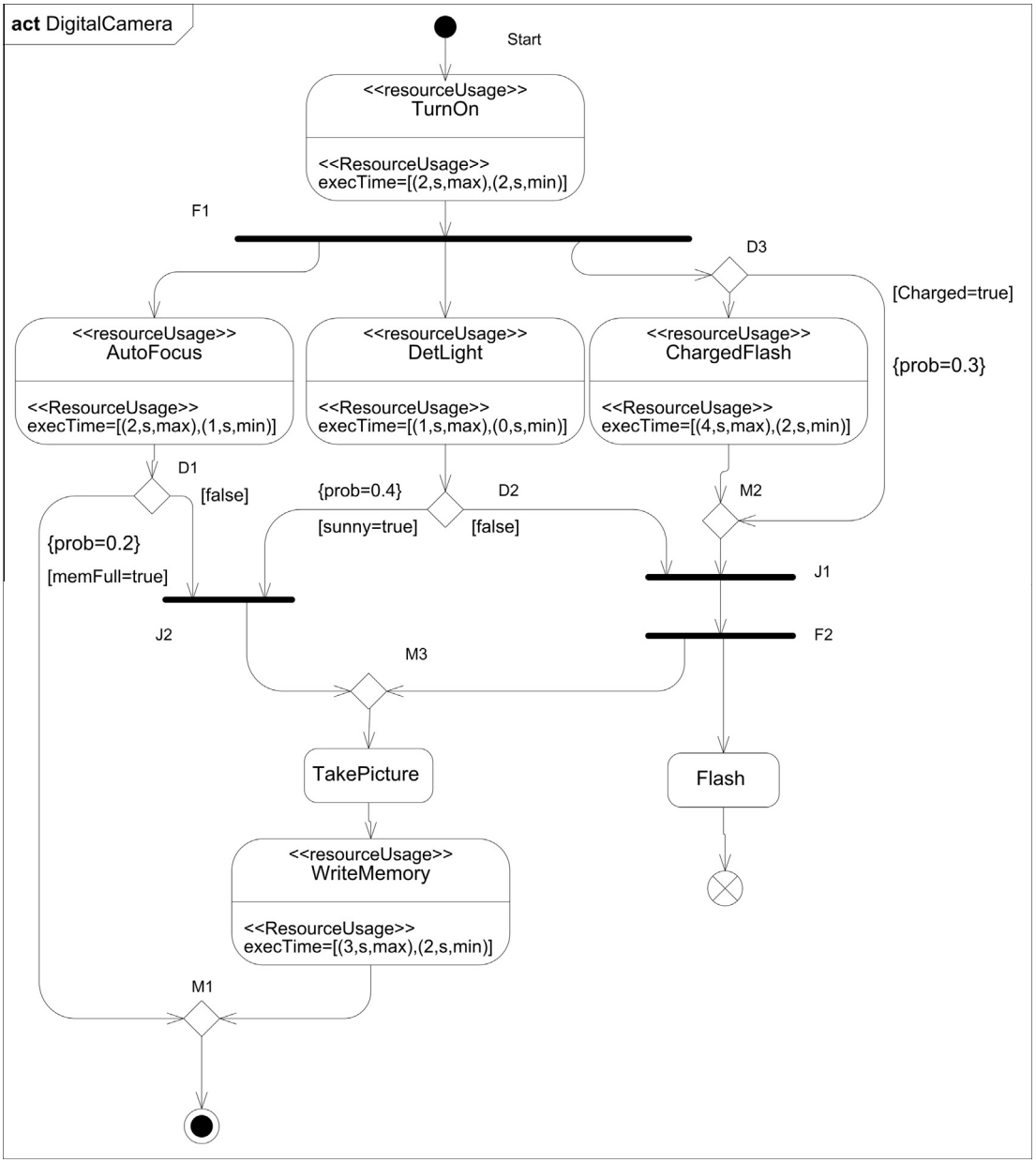


Fig. 4. Digital camera activity diagram design.

and capable of being taken; otherwise, the action is disabled. Conditions which depend on clock values are called clock constraints which is used to limit the amount of time that may be spent in a location. The following definition prescribes how constraints over clocks are to be defined.

Definition 1 (*Clock Constraint*). A clock constraint over set X of clocks is formed according to the grammar
$g ::= x < c \mid x \leqslant c \mid x > c \mid x \geqslant c \mid g \wedge g$

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 an Probabilistic Automata with clocks variable. 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. Probabilistic Timed Automata (PTA). PTA is a tuple $M = \langle \bar{s}, S, X, Act, Inv, Prob, L \rangle$, where:

- $\bar{s}$ is an initial state, such that $\bar{s} \in S$,
- 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, α) where g is a the clock constraint of the PTA, α is an action. The intuitive interpretation of $s \xrightarrow[p]{g,\alpha} s'$ 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 α 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 is a pair $(s, \vartheta) \in \mathbb{R}_{\geqslant 0}$ such that for a set X of clocks, ϑ is a clock valuation with $\vartheta : X \to \mathbb{R}_{\geqslant 0}$, assigning to each clock $x \in X$ its current value $\vartheta(x)$. In any location (s, ϑ), either a certain amount of time $t_s \in \mathbb{R}_{\geqslant 0}$ 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 α can only be chosen if it is enabled i.e. when clock constraint g holds, $\vartheta(t_s) \models g$. Once an enabled action α 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 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 $s \xrightarrow[p]{g,\alpha} s'$ $t_s \in \mathbb{R}$
  (b) $\vartheta + t_s \models g$
  (c) $\vartheta' = (v + t_{s'})[X := 0]$
  (d) $\vartheta + t_s \models Inv(s)$
- Delay transition: $(s, \vartheta) \xrightarrow{d} (s, \vartheta + d), d \in \mathbb{R}$
  (e) if $\vartheta + d \models Inv(s)$

For a transition that corresponds to (a) traversing a transition $s \xrightarrow[p]{g,\alpha} s'$ in PTA it must hold that (b) ϑ 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 ($\vartheta + t_s \models Inv(s)$). 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 $\to R^{\geqslant 0}$ is a function assigning to each state the rate at which rewards are accumulated as time passes in that location and r_{Act}: $S \times Act \to R^{\geqslant 0}$ is a function assigning the reward of executing each action in each state.

Definition 3 (*Parallel composition of PTAs*). The parallel composition of PTAs $M_1 = (\bar{s}_1, S_1, X_1, Act_1, inv_1, Prob_1, L_1)$, $M_2 = (\bar{s}_2, S_2, X_2, Act_2, inv_2, Prob_2, L_2)$ is the PTA $M = M_1||M_2 = (\bar{s}_1 \times \bar{s}_2, S_1 \times S_2, X_1 \cup X_2, Act_1 \cup Act_2, inv$, Prob,L(s1) ∪ L(s2)): where $Prob(S1 \times S2, Act1 \cup Act2)$ is the set of transitions, such that one of the following requirements is met.

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

The PRISM tool supports three PTA model checking techniques: Digital Clocks (Kwiatkowska, Norman, Parker, & Sproston, 2004) and abstraction refinement using stochastic games (Kwiatkowska, Norman, & Parker, 2009) where stochastic games supports the reward operator and the Digital clocks supports the probabilistic operator. The structure of our temporal logic is expressed by the following BNF grammar:

3.3. Property specification for PTAs

The syntax of our logic is given by the following grammar:

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

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 ψ being true always satisfies the bound $\bowtie$ p, p ∈ [0, 1]. $R^r_{\bowtie q}[\rho]$ means that the expected value of reward function ρ on reward structure r meets the bound $\bowtie$ q, q ∈ $\mathbb{Q}$. "$\bowtie$" ∈ <, ⩽, >, ⩾. "∧" represents the conjunction operator and "¬" is the negation operator. Two paths formulas are included bound until $\varphi 1 \cup \varphi 2$ and time-bound until $\varphi 1 \cup^{\leqslant k} \varphi 2$. 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, $C^{\leqslant k}$ refers to the total reward accumulated up until time point k, and F φ to the total reward accumulated until a state satisfying φ is reached, e.g:

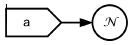
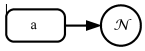
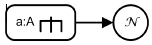
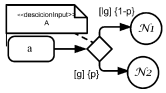
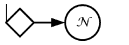
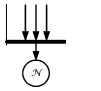
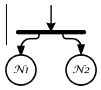
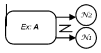
- $R^{time}_{max} = ?$[F success]: what is the expected reward accumulated before the system successfully terminates?
- Pmin = ?[true $\cup^{\leqslant 100}$ Complete]: the minimum probability of the system eventually completing its execution successfully after 100 time units?

Let P is PTA and [[P]] = $(S, \bar{s}, \mathbb{T}$, Act, lab, $Step_P)$ its semantics where S is a set of states, $\bar{s}$ 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 (Norman et al., 2013) is denoted by "$\models$" and defined as follows where (s, ϑ) 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,
- $s, \vartheta \models \varphi_1 \wedge \varphi_2 \iff s, \vartheta \models \varphi_1 \wedge s, \vartheta \models \varphi_2$,
- $s, \vartheta \models \neg\varphi \iff s, \vartheta \not\models \varphi$,
- $s, \vartheta \models P_{\bowtie p}[\psi] \iff Prob^{\sigma}_{[[P]]}(\pi \in Path_{s,\vartheta} | \pi \models \psi) \bowtie p$, for all $\sigma \in Adv_{[[P]]}$,
- $s, \vartheta \models R^r_{\bowtie q}[\rho] \iff Exp^{\sigma}_{[[P]]}(rew(r, \rho)) \bowtie q$, for all $\sigma \in Adv_{[[P]]}$.

Where for any finite path π of [[P]]:

Table 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
	$l : Ex(\mathcal{A}, \mathcal{N}_1, \mathcal{N}_2)$	InterruptibleActivityRegion invokes a sub behavior that can be interrupted

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

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:

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

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

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

4. SysML activity diagram formalization

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

4.1. SysML activity diagrams syntax

Based on textual specification of SysML (OMG, 2012) we formalize the SysML activity diagram by developing its Calculus: Timing Activity Calculus (TAC). In Table 2, each SysML artifact is represented and described formally by its related TAC term. The TAC terms in Fig. 5 extends NuAC defined by Debbabi et al. (2010a) and enhanced by Ouchani et al. (2014b) adding the time and data. In this calculus we distinguish between two syntactic concepts: Marked and Unmarked terms. A marked terms are activity diagrams with tokens. The unmarked terms corresponds 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 $l; l_0; l_1, \ldots$ 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 decisions, respectively. For the workflow observation on SysML activity diagrams, we use structural operational semantics (Milner, 1999 & Norman et al., Norman, Palamidessi, Parker, & Wu, 2007) 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 α 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 in Table 3.

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 4.

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

Fig. 5. Syntax of Timing Activity Calculus (TAC).

Table 3
Operational semantic rules of the TAC calculus.

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

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 $\mathcal{N}$.

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

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 $\mathcal{N}$

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

EXP-1 $\frac{\mathcal{A}[l':a?v \rightarrow \mathcal{N}_1] \xrightarrow{l'}_1 |\mathcal{A}|}{l:\overline{EX(\mathcal{A},\mathcal{N}_1,\mathcal{N}_2)}^n \xrightarrow{l'}_1 l:\overline{EX(|\mathcal{A}|,p,g,\mathcal{N}_1,\mathcal{N}_2)}^n}$ EXP-1 derivation rule shows the termination of a behavior sub actions with a transition to the exception node $\mathcal{N}_1$

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

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

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 $\mathcal{N}$

BEH-3 $\frac{\mathcal{A} \xrightarrow{t,\alpha}_p \mathcal{A}'}{l:\overline{a \uparrow \mathcal{A}^n} \rightarrowtail \mathcal{N} \xrightarrow{t,\alpha}_p l:\overline{\bar{a} \uparrow \mathcal{A}^m} \rightarrowtail \mathcal{N}}$ 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

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

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

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

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

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

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

DEC-4 $\frac{\mathcal{A}[\overline{l':\odot}] \xrightarrow{l'}_1 |\mathcal{A}|}{l:\overline{D(\mathcal{A},p,g,\mathcal{N}_1,\mathcal{N}_2)}^n \xrightarrow{g,l'}_p l:\overline{D(|\mathcal{A}|,p,g,\mathcal{N}_1,\mathcal{N}_2)}^n}$ 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

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

MRG-1 $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : M(x,y)}^n \xrightarrow{l} l : \overline{\mathcal{N} \rightarrowtail l' : M(\overline{x},y)}^n$ 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

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 $\mathcal{N}$

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

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

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

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

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

JOIN-4 $\frac{\mathcal{N} \xrightarrow{t,\alpha} \mathcal{N}'}{l:\overline{J(x,y) \rightarrowtail \mathcal{N}}^n \xrightarrow{t,\alpha} l:\overline{J(x,y) \rightarrowtail \mathcal{N}'}^n}$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

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

FFIN $\mathcal{A}[l : \overline{\otimes}] \xrightarrow{l} \mathcal{A}[l : \otimes]$ 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

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

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

Definition 4. *(TAC-PTA).* A Probabilistic Timed Automata of TAC term A is the tuple $M_{\mathcal{A}} = \langle \bar{s}, S, X, Act, Inv, Prob, L \rangle$, where:

- $\bar{s}$ is an initial state, such that $L(\bar{s}) = \left\{ l : \bar{i} \rightarrowtail \mathcal{N} \right\}$,
- S is a finite set of states reachable from $\bar{s}$, such that, $S = \left\{ s_{i:0 \leqslant i \leqslant n} | L(s_i) = \left\{ \overline{\mathcal{N}} \right\} \right\}$
- 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 μ where:
  - For each $S' \subseteq S$ and $t \in X$ such that $S' = \left\{ si : 0 \leqslant i \leqslant n : s \xrightarrow[pi]{t,\alpha} s_i \right\}$, 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: $[[\mathcal{L}]] = \{true, false\}$.

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 (Ouchani, Mohamed, & Debbabi, 2013). 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:

$$[act]guard \rightarrow p_1 : u_1 .. + p_n : u_n$$

Where *act* is an (optional) action labeling the command, the **guard** 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 $(V'_j = val_j)\&\ldots\&(V'_k = val_k)$ 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 5.

Definition 5. A PRISM command is a tuple $c = \langle a, g, u \rangle$.

- "act" is an action label.
- "guard" 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 $u = \{(v, eval(v)) : v \in V_l\}$.

The set of commands are associated with modules that are parts of a system and it definition is given in Definition 6.

Definition 6. A PRISM module is tuple $M = \langle V_l, I_l, Inv, C \rangle$, where:

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

To describe the composition between different modules, PRISM use CSP communication sequential process operators (Hoare, 1985) such as Synchronization, Interleaving, Parallel Interface, Hiding and Renaming. Definition 7 provides a formal definition of PRISM system.

Definition 7. A PRISM system is tuple $P = \langle V, I, exp, M, CSPexp \rangle$, 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.

5.1. PRISM syntax

The syntax of PRISM PTA is defined by the BNF grammar presented in Fig. 6. To clarify the syntax we define the following:

- $[min \ldots max]$ 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 $||, ||\ |, |[a, b, \ldots]|, /\{a, b, \ldots\}$, and $\{a \leftarrow b, c \leftarrow d, \ldots\}$.

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 μ is a distribution over S such that $\mu(s, \vartheta + v_t) = \{|0 \leqslant p_i \leqslant 1, v \in V, s'(v) = eval(V)|\}$.

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

Fig. 6. The syntax of PRISM Probabilistic Timed Automata.

Table 4
PRISM-PTA operational semantic rules.

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 $v_i \in V_i$

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

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

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

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

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

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

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

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

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

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

$$
\begin{array}{l}
\Gamma : \mathcal{A} \longrightarrow \mathcal{P} \\
\Gamma(\mathcal{A}) = \forall n \in \mathcal{A} \ , \ L(n = (i)) = true \ , \ L(n \neq (i)) = false \\
\\
l : i \rightarrowtail \mathcal{N} \Longrightarrow \\
\quad \textbf{in} \\
\qquad \{[l]l \longrightarrow (l' = false)\&(L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N})) \\
\quad \textbf{end} \\
l : M(x,y) \rightarrowtail \mathcal{N} \Longrightarrow \\
\quad \textbf{in} \\
\qquad \{[l_x]l_x \longrightarrow (l'_x = false)\&(L(\mathcal{N})' = true);\} \cup \left\{[l_y]l_y \longrightarrow (l'_y = false)\&(L(\mathcal{N})' = true);\right\} \cup \Gamma(L(\mathcal{N})) \\
\quad \textbf{end} \\
l : J(x,y) \rightarrowtail \mathcal{N} \Longrightarrow \\
\quad \textbf{in} \\
\qquad \left\{[l]l_x \wedge l_y \longrightarrow (l'_y = false)\&(l'_x = false)\&(L(\mathcal{N})' = true);\right\} \cup \Gamma(L(\mathcal{N})) \\
\quad \textbf{end} \\
l : F(\mathcal{N}_1, \mathcal{N}_2) \Longrightarrow \\
\quad \textbf{in} \\
\qquad \{[l]l \longrightarrow (l' = false)\&(L(\mathcal{N}_1)' = true)\&(L(\mathcal{N}_2)' = true);\} \cup \Gamma(L(\mathcal{N}_1)) \cup \Gamma(L(\mathcal{N}_2)) \\
\quad \textbf{end} \\
l : D(\mathcal{A}, p, g, \mathcal{N}_1, \mathcal{N}_2) \Longrightarrow \\
\quad \textbf{in} \\
\qquad \left\{[l]l \rightarrow p : (l' = false)\&(l_g)' = true) + 1 - p : (l' = false)\&(l_{\neg g})' = true);\right\} \cup \Gamma(L(\mathcal{N}_1)) \cup \Gamma(L(\mathcal{N}_2)) \\
\qquad \cup \left\{[l_g]l_g \wedge g \rightarrow (l'_g = false)\&(L(\mathcal{N}_1)' = true)\right\} \cup \left\{[l_{\neg g}]l_{\neg g} \wedge \neg g \rightarrow (l'_{\neg g} = false)\&(L(\mathcal{N}_2)' = true)\right\} \\
\quad \textbf{end} \\
\\
l : a \uparrow \mathcal{A}_i \overset{\alpha}{\rightarrowtail} \mathcal{N}, case(\alpha)\ of \\
\\
\quad b(y),\ for\ z \in \mathcal{A}_i \Longrightarrow \\
\qquad \textbf{in} \\
\qquad\quad \{[l]l \longrightarrow (l' = false);\} \cup \{[L(E(\mathcal{A}_i))]E(\mathcal{A}_i) \longrightarrow (l' = false)\&(y' = z)\&(L(\mathcal{N})' = true);\} \cup \Gamma'(A_i) \cup \Gamma(L(\mathcal{N})) \\
\qquad \textbf{end} \\
\quad otherwise \\
\qquad \textbf{in} \\
\qquad\quad \{[l]l \longrightarrow (l' = false);\} \cup \{[L(E(\mathcal{A}_i))]E(\mathcal{A}_i) \longrightarrow (l' = false)\&(L(\mathcal{N})' = true);\} \cup \Gamma'(\mathcal{A}_i) \\
\qquad \textbf{end} \\
\\
l : \odot \Longrightarrow \\
\qquad \textbf{in} \\
\qquad\quad [l] \rightarrow \&_{l \in \mathcal{L}}(l' = false); \\
\qquad \textbf{end} \\
\\
l : \otimes \Longrightarrow \\
\qquad \textbf{in} \\
\qquad\quad [l] \rightarrow (l' = false); \\
\qquad \textbf{end} \\
\\
l : a \overset{t>c,\alpha}{\rightarrowtail} \mathcal{N},\ case(\alpha)\ of \\
\qquad b(y),\ for\ z \in a, \Longrightarrow \\
\qquad\ \textbf{in} \\
\qquad\quad \{[l_{bz}]l\&(t > c) \longrightarrow (l' = false)\&(y' = z)\&(L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N})) \\
\qquad\ \textbf{end} \\
\quad otherwise \\
\qquad\ \textbf{in} \\
\quad \{[l_\alpha]l\&(t > c) \longrightarrow (l'_\alpha = false)\&(L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N})) \\
\qquad\ \textbf{end} \\
\\
\Gamma' : \mathcal{A} \rightarrow \mathcal{P} \\
\Gamma'(\mathcal{A}_i) = \forall m \in \mathcal{A} \ , \ L(m) = false \\
l : i \rightarrowtail \mathcal{N} \Longrightarrow \\
\quad \textbf{in} \\
\qquad \{[l]l \longrightarrow (L(S(A_i))' = true);\}\{[L(S(A_i))]L(S(A_i)) \longrightarrow (L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N})) \\
\quad \textbf{end} \\
\quad l : \odot \Longrightarrow \\
\quad \textbf{in} \\
\qquad \{[L(E(\mathcal{A}_i))]L(E(\mathcal{A}_i)) \longrightarrow (L(E(\mathcal{A}_i))' = false);\} \\
\quad \textbf{end} \\
\quad\ otherwise\ \ \Gamma(\mathcal{A})
\end{array}
$$

Listing 1. Generating PRISM Commands Function-Part1.

Definition 1 stipulates the formal definition of PRISM Probabilistic Timed Automata denoted by M_P. The states of M_P take the form $\langle V1, \ldots, Vn, eval\rangle$. The stepwise behavior of M_P is described by the operational semantic rules provided in Table 4.

Definition 8. *(PRISM-PTA).* A Probabilistic Timed Automata of PRISM program p is the tuple $M_p = \langle s_i, \mathrm{S}, \mathrm{X}, Act, Inv, Prob, \mathrm{L}\rangle$, 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, $S = \{s_{i:0\leqslant i\leqslant n} | L(s_i) \in \{AP\}\}$,
- 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 : $\mathrm{S} \times Act \rightarrow Dist(2^x \times S)$ is a probabilistic transition function assigning for each $s \in S$ and $a \in Act$ a probabilistic distribution μ where: For each s ∈ S, $v \in V$ is a PRISM variable, $a \in Act$ and $v_t \in V$ is a clock variable such that $s(v, v_t) \xrightarrow{a} \mu_{(v,\vartheta+v_t)}$, $\vartheta + v_t \models Inv(v)$ and $s(v, v_t) \models g$.
- L : S → 2^{AP} is a labeling function that assigns for each state a set of valuated propositions.

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 algorithms output produces PRISM synchronized modules.

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
```

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 list vNode of visited nodes (line 12) if it is not already visited (line 10). *PrismCode* is constructed by calling the function Γ and ϕ. 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 Γ presented in Listing 1 and Listing 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 26) produces two commands (line 30), and it calls the function Γ' (line 57). The first command in (line 30) synchronizes with the first command in (line 61) produced by the function Γ' in the action l from the diagram $\mathcal{A}$. Similarly, the second command in (line 30) synchronizes with the command of line 65 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 Γ' is similar to the function Γ except for the initial and the final nodes as shown in (line 52) and (line 56), respectively. The region behavior calls the sub actions within the region in the iterative manner Listing 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 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 (line 7). The region is interrupted Listing 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 (Line 38). The ϕ function Listing 2 (line 49–54) 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.

7. The transformation soundness

Our aim is to prove the soundness of the transformation algorithm Γ 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. 7, proving the soundness of Γ algorithm is to find the adequate relation $\mathcal{R}$ between $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$.

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 (Ben-Menachem, 2010; Segala, 1995) in Definitions 9 and 10. This relation is based on the probabilistic equivalence relation $\mathcal{R}$ defined in Definition 8 where $\delta/\mathcal{R}$ denotes the quotient space of δ with respect to $\mathcal{R}$ and $\equiv_{\mathcal{R}}$ is the lifting of $\mathcal{R}$ to a probabilistic space.

Definition 8. (The equivalence $\equiv_{\mathcal{R}}$). If $\mathcal{R}$ is an equivalence on δ, then the induced equivalence $\equiv_{\mathcal{R}}$ on $\mathrm{Dist}(\delta \times 2^X)$ is given by: $\mu_{\equiv_{\mathcal{R}}}\mu'$ iff $\mu(\delta, \vartheta + d) \equiv_{\mathcal{R}} \mu(\delta, \vartheta + d')$.

Definition 9 (*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 $s_1 \xrightarrow{d,\alpha} \mu(s_1, \vartheta + d)$ there is a transition $s_2 \xrightarrow{d',\alpha} \mu(s_2, \vartheta + d')$, 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.

$\Gamma' : \mathcal{A} \rightarrow \mathcal{P}$

$l : R \uparrow \mathcal{A}_j \overset{\alpha}{\rightarrowtail} \mathcal{N}, case(\alpha)\ of$

$b(y_j),\ for\ z_{j1}, z_{j2}, ...z_{jsize} \in \mathcal{A}_j \Longrightarrow$
in
$\{[l]l \longrightarrow (l' = false);\} \cup \left\{[L(E(\mathcal{A}_j))]E(\mathcal{A}_j) \longrightarrow (l' = false)\&_{j1..jsize}(y'_j = z_j)\&(L(\mathcal{N})' = true);\right\} \cup \Gamma''(A_j) \cup \Gamma(L(\mathcal{N}))$
end

$l : Ex(\mathcal{A}_k, \mathcal{N}_1, \mathcal{N}_2) \Longrightarrow$
in
$\{[l]l \longrightarrow (l' = false);\} \cup \{[L(E(\mathcal{A}_{k1}))]E(\mathcal{A}_{k1}) \longrightarrow (l' = false)\&(L(\mathcal{N}_1)' = true);\} \cup \Gamma'''(A_k) \cup \Gamma(L(\mathcal{N}_1))$
$\cup \{[L(E(\mathcal{A}_{k2}))]E(\mathcal{A}_{k2}) \longrightarrow (l' = false)\&(L(\mathcal{N}_2)' = true);\} \cup \Gamma(L(\mathcal{N}_2))$
end

$\Gamma'' : \mathcal{A} \rightarrow \mathcal{P}$
$\Gamma''(\mathcal{A}_j) = \forall m \in \mathcal{A}\ ,\ L(m) = false$
$l : a \rightarrowtail \mathcal{N} \Longrightarrow$
in
$\left\{[l]l \longrightarrow (L(S(A_j))' = true);\right\} \left\{[L(S(A_j))]L(S(A_j)) \longrightarrow (L(N)' = true);\right\} \cup \Gamma(L(\mathcal{N}))$
end

$l : a \rightarrowtail E(\mathcal{A}_j) \Longrightarrow$
in
$\left\{[l]l \longrightarrow (l' = false)\&(L(E(\mathcal{A}_j))' = true);\right\}$
$\left\{[L(E(\mathcal{A}_j))]L(E(\mathcal{A}_j)) \longrightarrow (L(E(\mathcal{A}_j))' = false);\right\}$
end

$\Gamma''' : \mathcal{A} \rightarrow \mathcal{P}$
$\Gamma'''(\mathcal{A}_k) = \forall m \in \mathcal{A}\ ,\ L(m) = false$
$l : a \rightarrowtail \mathcal{N} \Longrightarrow$
in
$\{[l]l \longrightarrow (L(S(A_k))' = true);\} \{[L(S(A_k))]L(S(A_k)) \longrightarrow (L(N)' = true);\} \cup \Gamma(L(\mathcal{N}))$
end

$l : a?v \rightarrowtail \mathcal{N} \Longrightarrow$
in
$\{[l]L(S(A_k))|....|L(E(\mathcal{A}_k) \longrightarrow (L(S(\mathcal{A}_k))' = false)\&....\&(L(E(\mathcal{A}_{k1}))' = false)\&(L(E(\mathcal{A}_{k2}))' = true);\}$
$\cup \{[L(E(\mathcal{A}_{k2})]L(E(\mathcal{A}_{k2}) \longrightarrow (L(E(\mathcal{A}_{k2}))' = false);\}$
end

$l : a \rightarrowtail E(\mathcal{A}_{k1}) \Longrightarrow$
in
$\{[l]l \longrightarrow (l' = false)\&(L(E(\mathcal{A}_{k1}))' = true);\}$
$\{[L(E(\mathcal{A}_{k1}))]L(E(\mathcal{A}_{k1})) \longrightarrow (L(E(\mathcal{A}_{k1}))' = false);\}$
end

$\phi : \mathcal{A} \rightarrow \mathcal{P}$
$\phi(\mathcal{A}) = \forall m \in \mathcal{A}\ ,\ t_l$ clock variable $,\ c_l$ integer constant
$l : a \rightarrowtail \mathcal{N} \Longrightarrow$
in
$\{(l = true) \Longrightarrow (t_l <= c_l)\&\phi(\mathcal{N})\}$
end

Listing 2. Generating PRISM Commands Function-Part2.

Definition 10 (*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 11 (*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 Γ is sound means showing the existence of a timed probabilistic bisimulation between M_A and M_P.

Lemma 1 (*Soundness*). *The mapping algorithm* Γ *is sound, i.e.* $M_{\mathcal{A}} \sim M_{\mathcal{P}}$.

Proof. 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:

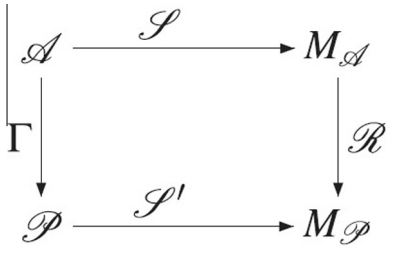


Fig. 7. The transformation soundness.

1. $L(s_1) = l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{i, a\} \Rightarrow \exists s_1 \xrightarrow{d,\alpha}_1 s'_1$, $L(s'_1) = l : x \rightarrowtail \overline{\mathcal{N}}$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle L(x), \neg L(\mathcal{N})\rangle$ then $\exists s_2 \xrightarrow[1]{d',\alpha} s'_2$ where $L(s'_2) = \langle \neg L(x), L(\mathcal{N})\rangle$.
2. $L(s_1) = l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{a!v, a?v\} \Rightarrow \exists s_1 \xrightarrow[1]{\alpha} s'_1$, $L(s'_1) = l : x \rightarrowtail \overline{\mathcal{N}}$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle L(x), \neg L(\mathcal{N})\rangle$ then $\exists s_2 \xrightarrow[1]{\alpha} s'_2$ 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 $\exists s_1 \xrightarrow[1]{g_1,\alpha} s'_1$, $L(s'_1) = l : \overline{D(g_1, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. 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 $\exists s_2 \xrightarrow[1]{g_1,\alpha} s'_2$ 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 $\exists s_1 \xrightarrow[1]{\alpha} s'_1$, $L(s'_1) = l : \odot$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l \rangle$ then $\exists s_2 \xrightarrow[1]{\alpha} s'_2$ where $\forall l_i \in \mathcal{L} : L(s'_2) = \langle \neg l_i \rangle$.
5. $L(s_1) = l : \overline{\otimes}$ then $\exists s_1 \xrightarrow[1]{\alpha} s'_1$, $L(s'_1) = l : \otimes$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l \rangle$ then $\exists s_2 \xrightarrow[1]{\alpha} s'_2$ 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 Γ 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 ϕ, then: $(M_{\mathcal{A}} \models \phi) \Longleftrightarrow (M_{\mathcal{P}} \models \phi)$.*

Proof. 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 $\zeta \in \left\{U, U^{\leqslant k}, I^{=k}, C^{\leqslant k}, F, P_{\bowtie p}, R_{\bowtie q}\right\}$, we find that $(M_{\mathcal{A}} \models \zeta) \Longleftrightarrow (M_{\mathcal{P}} \models \zeta)$ which means: $(M_{\mathcal{A}} \models \phi) \Longleftrightarrow (M_{\mathcal{P}} \models \phi)$. □

8. Implementation and experimental results

In this section, we apply our verification framework on Digital camera case study (Debbabi et al., 2010a). The related SysML activity diagrams are modeled on Topcased2 then mapped into Prism code via our Java implementation. Listing 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:

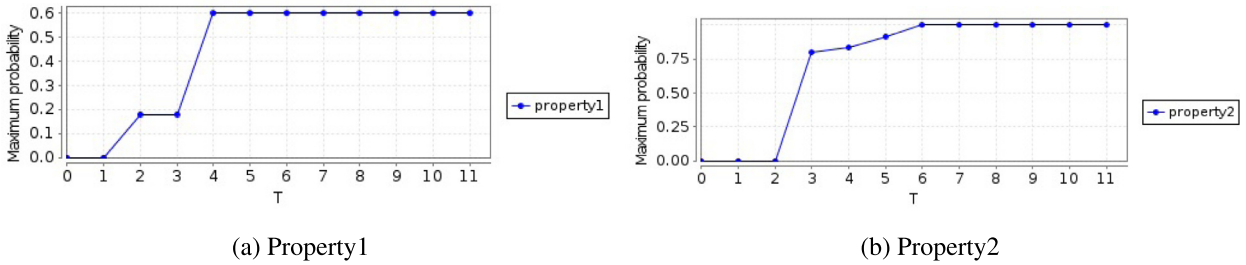
1. The maximum probability value that the *TakePicture* action should not be activated if either the memory is full memFull = true or the AutoFocus action is still ongoing. T is a constant value referring to the time bound

$$Pmax = ?[F^{\leqslant T}(memFull|AutoFocus)\&TakePicture] \quad (1)$$

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

```
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 3. Digital camera PRISM code fragment.



(a) Property1 (b) Property2

Fig. 8. The verification of PCTL properties on the digital camera.

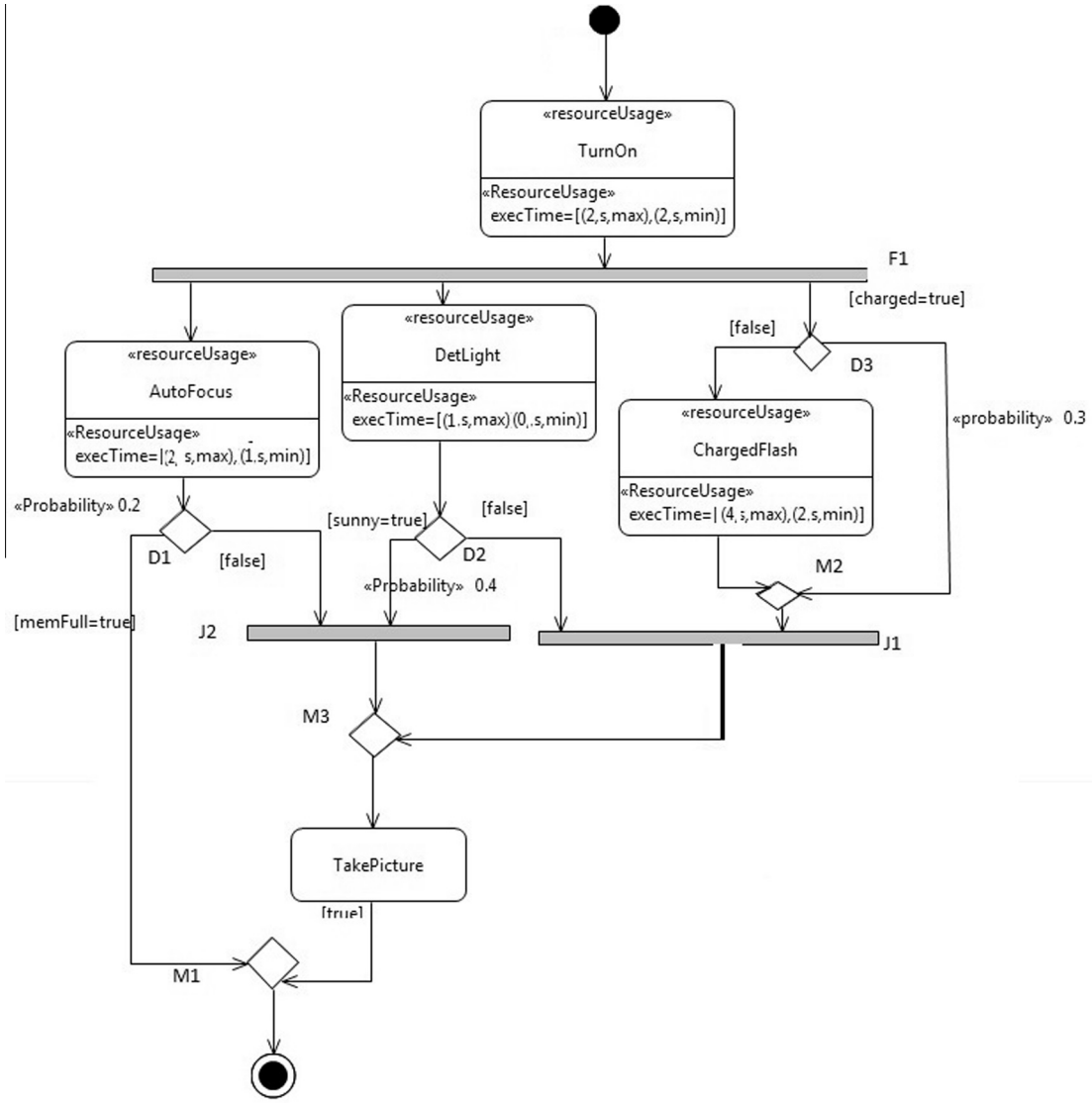


Fig. 9. The abstract SysML activity diagram for Property 4.

$$Pmax = ?[F^{\leqslant T} Final] \quad (2)$$

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

$$R\{\text{“time”}\}min = ?[F(TurnOff)] \quad (3)$$

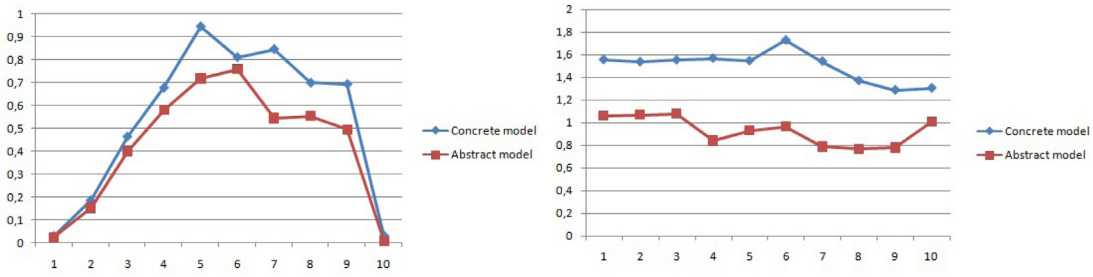
The verification results of the above two properties are done using an i7 CPU 2.67 GHz with 8.0 GB of RAM and shown in Fig. 8. For different values of time T, Fig. 8(a) shows that the verification result for Property 1 converges to 0.6 after 4 time units. Fig. 8(b) shows that the verification result for Property 2 converges to 0.916 after 6 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.

Now, we want to verify the abstraction effects over SysML activity diagram described above Fig. 4 to cope with state explosion problem when we check the property

$$Pmax = ?[F^{\leqslant T} TakePicture] \quad (4)$$

Table 5
Verification results for Property 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



(a) Verification Time (b) Construction Time

Fig. 10. The abstraction effects on digital camera activity diagram.

For this purpose we apply the algorithm and rules defined by Ouchani, Mohamed, and Debbabi (2014a). 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. 9 and Table 5 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. 10. The number of states and transitions for the concrete model are 9364 states and 23,653 transitions, respectively and 4580 states, 12,221 transitions for the abstracted model. As conclusion, we notice that the abstraction rules preserves the results while verification and construction time are optimized.

9. Conclusion

In this paper, we presented a formal verification approach of systems modeled by using SysML activity diagram. 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. (2014), which use NuSMV (Cimatti et al., 1999) 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. (2014b) 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. (2015), which use C programs as component behavior but 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.

The research contribution of this work 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 have shown the effectiveness of our approach in realistic case study: digital Camera where time and probability are evaluated. In addition, we evaluated the time construction and time checking when the size of diagram is reduced according to PCTL properties.

The practical advantages of the proposed approach in the context of predicting the behavior of systems, consists on providing key decision for errors minimization and product cost reducing via probabilistic model checking. The limitations of the proposed approach relate to the problem of state explosion, we explain in our work how to reduce the number of state by removing just the negligible states that are not time-constrained.

The presented work can be extended in the following four directions. First, we want to reduce the size of the model generated after mapping the SysML activity diagrams into PRISM code that affects memory and time especially for the timed states. Second, we intend to extend our approach to support more constraints that affect the entire system behavior such as energy. Third, we want to study the problem of partitioning using model checking in case of HW/SW CoDesign starting from the component based

specification. Finally, we intend to translate the system modeled using Component based specification with activity diagram as behavior model to specific language such as VHDL/C/C++/NesC.

References

Abdulkhaleq, A., & Wagner, S. (2014). A software safety verification method based on system-theoretic process analysis. In A. Bondavalli, A. Ceccarelli, & F. Ortmeier (Eds.), *Computer safety, reliability, and security. Lecture notes in computer science* (Vol. 8696, pp. 401–412). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-10557-4_44. ISBN 978-3-319-10556-7.

Andrade, E., Maciel, P., Callou, G., & Nogueira, B. (2009). A methodology for mapping SysML activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In *Third international conference on digital society, 2009. ICDS '09* (pp. 266–271). http://dx.doi.org/10.1109/ICDS.2009.19.

Baier, C., & Katoen, J. P. (2008). *Principles of model checking (representation and mind series)*. The MIT Press. ISBN 026202649X, 9780262026499.

Ball, T., Bounimova, E., Kumar, R., & Levin, V. (2010). Slam2: Static driver verification with under 4% false alarms. In *Proceedings of the 2010 conference on formal methods in computer-aided design, FMCAD '10, 2010. FMCAD Inc* (pp. 35–42). Austin, TX. <http://dl.acm.org/citation.cfm?id=1998496.1998508>.

Behrmann, Gerd, David, Alexandre, & Larsen, KimG. (2004). A tutorial on uppaal. In Marco Bernardo & Flavio Corradini (Eds.), *Formal methods for the design of real-time systems. Lecture notes in computer science* (Vol. 3185, pp. 200–236). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-30080-9_7. ISBN 978-3-540-23068-7.

Ben-Menachem, M. (2010). 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, p. 300. *SIGSOFT Software Engineering Notes, 35*(4), 34–35, July 2010. ISSN: 0163-5948. http://dx.doi.org/10.1145/1811226.1811243. URL http://doi.acm.org/10.1145/1811226.1811243.

Bertot, Yves, & Castéran, Pierre (2004). *Interactive theorem proving and program development. Coq'Art: The calculus of inductive constructions*. Springer<http://www.labri.fr/perso/casteran/CoqArt/index.html>.

Carlson, J., Hkansson, J., & Pettersson, P. Saveccm: An analysable component model for real-time systems. Electronic Notes in Theoretical Computer Science, 160(0):127 – 140, 2006. ISSN 1571-0661. doi: http://dx.doi.org/10.1016/j.entcs.2006.05.019. URL http://www.sciencedirect.com/science/article/pii/S1571066106003811. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005) Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005).

Carneiro, E., Maciel, P., Callou, G., Tavares, E., & Nogueira, B. (2008). Mapping SysML state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints. In *International conference on advances in electronics and micro-electronics, 2008. ENICS '08* (pp. 1–6). http://dx.doi.org/10.1109/ENICS.2008.19.

Cafe, D. C., dos Santos F. V., Hardebolle, C., Jacquet, C., & Boulanger, F. (2013). Multi-paradigm semantics for simulating SysML models using SystemC-AMS. In *2013 Forum on specification design languages (FDL)* (pp. 1–8).

Cimatti, A., Clarke, E. M., Giunchiglia, F., & Roveri, M. (1999). Nusmv: A new symbolic model verifier. In *Proceedings of the 11th international conference on computer aided verification, CAV '99* (pp. 495–499). London, UK, UK: Springer-Verlag. <http://dl.acm.org/citation.cfm?id=647768.733923>, ISBN 3-540-66202-2.

Clarke, E. M. (2008). The birth of model checking. In O. Grumberg & H. Veith (Eds.), *25 Years of model checking. Lecture notes in computer science* (Vol. 5000, pp. 1–26). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-69850-0_1. ISBN 978-3-540-69849-4.

Debbabi, M., Hassane, F., Jarraya, Y., Soeanu, A., & Alawneh, L. (2010a). *Verification and validation in systems engineering: Assessing UML/SysML design models* (1st edition). New York, NY, USA: Springer-Verlag New York, Inc.. ISBN 3642152279, 9783642152276.

Debbabi, M., Hassane, F., Jarraya, Y., Soeanu, A., & Alawneh, L. (2010b). Probabilistic model checking of SysML activity diagrams. In *Verification and validation in systems engineering* (pp. 153–166). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-15228-3_9. ISBN 978-3-642-15227-6.

Doligalski, M., & Adamski, M., (2013). UML state machine implementation in FPGA devices by means of dual model and verilog. In *2013 11th IEEE international conference on industrial informatics (INDIN)* (pp. 177–184). http://dx.doi.org/10.1109/INDIN.2013.6622878.

Feiler, P. H. (2010). Model-based validation of safety-critical embedded systems. In *Aerospace conference, 2010* (pp. 1–10). IEEE. http://dx.doi.org/10.1109/AERO.2010.5446809.

Fredlund, LarsAAke, & Svensson, Hans (2007). Mcerlang: A model checker for a distributed functional programming language. *SIGPLAN Notices, 42*(9), 125–136. http://dx.doi.org/10.1145/1291220.1291171<http://doi.acm.org/10.1145/1291220.1291171>. ISSN 0362-1340.

Friedenthal, Sanford, Moore, Alan, & Steiner, Rick. (2008). *A practical guide to SysML: Systems modeling language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.. ISBN 0123743796, 9780080558363, 9780123743794.

George, R., & Samuel, P. (2012). Improving design quality by automatic verification of activity diagram syntax. In *2012 12th International conference on intelligent systems design and applications (ISDA)* (pp. 303–308). http://dx.doi.org/10.1109/ISDA.2012.6416555.

Grobelna, Iwona (2013). *Formal verification of logic controller specification by means of model checking. Lecture notes in control and computer science*. Springer International Publishing.

Grobelna, I., Grobelny, M., & Adamski, M. (2014). Model checking of UML activity diagrams in logic controllers design. In W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, & J. Kacprzyk (Eds.), *Proceedings of the ninth international conference on dependability and complex systems DepCoS-RELCOMEX. June 30–July 4, 2014. Advances in intelligent systems and computing* (Vol. 286, pp. 233–242). Springer International Publishing. ISBN 978-3-319-07012-4.

Hoare, C. A. R. (1985). *Communicating sequential processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.. ISBN 0-13-153271-5.

Hoque, K. A., Mohamed, O. A., Savaria, Y., & Thibeault, C. (2014). Early analysis of soft error effects for aerospace applications using probabilistic model checking. In C. Artho & P. Csaba-lveczky (Eds.), *Formal techniques for safety-critical systems. Communications in computer and information science* (Vol. 419, pp. 54–70). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-05416-2_5. ISBN 978-3-319-05415-5.

Huang, X., Sun, Q., Li, J., & Zhang, T. (2013). MDE-based verification of SysML state machine diagram by uppaal. In Y. Yuan, X. Wu, & Y. Lu (Eds.), *Trustworthy computing and services. Communications in computer and information science* (Vol. 320, pp. 490–497). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-35795-4_62. ISBN 978-3-642-35794-7.

Jacobs, J., & Simpson, A. (2013). Towards a process algebra framework for supporting behavioural consistency and requirements traceability in SysML. In L. Groves & J. Sun (Eds.), *Formal methods and software engineering. Lecture notes in computer science* (Vol. 8144, pp. 265–280). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-41202-8_18. ISBN 978-3-642-41201-1.

Jacobs, J., & Simpson, A. (2015). A formal model of SysML blocks using CSP for assured systems engineering. In C. Artho & P. C. veczky (Eds.), *Formal techniques for safety-critical systems. Communications in computer and information science* (Vol. 476, pp. 127–141). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-17581-2_9. ISBN 978-3-319-17580-5.

Jarraya, Y., Soeanu, A., Debbabi, M., & Hassaine, F. (2007). Automatic verification and performance analysis of time-constrained SysML activity diagrams. In *14th Annual IEEE international conference and workshops on the engineering of computer-based systems, 2007. ECBS '07* (pp. 515–522). http://dx.doi.org/10.1109/ECBS.2007.22.

Kaliappan, P. S., Koenig, H., & Kaliappan, V. K. (2008). Designing and verifying communication protocols using model driven architecture and spin model checker. In *2008 International conference on computer science and software engineering* (Vol. 2, pp. 227–230). http://dx.doi.org/10.1109/CSSE.2008.976.

Katoen, J. P., Zapreev, I. S., Hahn, E. M., Hermanns, H., & Jansen, D. N. (2011). The Ins and Outs of the probabilistic model checker MRMC. *Performance Evaluation, 68*(2), 90–104. http://dx.doi.org/10.1016/j.peva.2010.04.001. ISSN 0166-5316.

kerholm, M., Carlson, J., Fredriksson, J., Hansson, H., Håkansson, J., Möller, A., et al. (2007). The save approach to component-based development of vehicular systems. *Journal of Systems and Software, 80*(5), 655–667. http://dx.doi.org/10.1016/j.jss.2006.08.016. ISSN 0164-1212.

Knorreck, D., Apvrille, L., & de Saqui-Sannes, P. (2011). Tepe: A SysML language for time-constrained property modeling and formal verification. *SIGSOFT Software Engineering Notes, 36*(1), 1–8. http://dx.doi.org/10.1145/1921532.1921556<http://doi.acm.org/10.1145/1921532.1921556>. ISSN 0163-5948.

Kwiatkowska, M., Norman, G., & Parker, D. (2009). Stochastic games for verification of probabilistic timed automata. In J. Ouaknine & F. W. Vaandrager (Eds.), *Formal modeling and analysis of timed systems. Lecture notes in computer science* (Vol. 5813, pp. 212–227). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-04368-0_17. ISBN 978-3-642-04367-3.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). Prism 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan & S. Qadeer (Eds.), *Computer aided verification. Lecture notes in computer science* (Vol. 6806, pp. 585–591). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-22110-1_47. ISBN 978-3-642-22109-5.

Kwiatkowska, M., Norman, G., Parker, D., & Sproston, J. (2004). Performance analysis of probabilistic timed automata using digital clocks. In K. Larsen & P. Niebert (Eds.), *Formal modeling and analysis of timed systems. Lecture notes in computer science* (Vol. 2791, pp. 105–120). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-40903-8_9. ISBN 978-3-540-21671-1.

Lasnier, G., Zalila, B., Pautet, L., & Hugues, J. (2009). Ocarina: An environment for AADL models analysis and automatic code generation for high integrity applications. In F. Kordon & Y. Kermarrec (Eds.), *Reliable software technologies Ada-Europe 2009. Lecture notes in computer science* (Vol. 5570, pp. 237–250). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-01924-1_17. ISBN 978-3-642-01923-4.

McMillan, K. L., (1992). Symbolic model checking: An approach to the state explosion problem (PhD thesis). Pittsburgh, PA, USA. UMI Order No. GAX92-24209.

Liu, S., Liu, Y., Sun, J., Zheng, M., Wadhwa, B., & Dong, J. S. (2013). Usmmc: A self-contained model checker for UML state machines. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering, ESEC/FSE 2013* (pp. 623–626). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2491411.2494595<http://doi.acm.org/10.1145/2491411.2494595>. ISBN 978-1-4503-2237-9.

Mallet, F., & de Simone, R. (2008). 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* (pp. 43:1–43:8). Brussels, Belgium, Belgium: ICST<http://dl.acm.org/citation.cfm?id=1416222.1416271>. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.

Marinescu, R., Kaijser, H., Mikucionis, M., Seceleanu, C., Lonn, H., & David, A. (2015). Analyzing industrial architectural models by simulation and model-checking. In C. Artho & P. C. Iveczky (Eds.), *Formal techniques for safety-critical systems. Communications in computer and information science* (Vol. 476, pp. 189–205). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-17581-2_13. ISBN 978-3-319-17580-5.

Milner, Robin (1999). *Communicating and mobile systems: The &Pgr;-calculus*. New York, NY, USA: Cambridge University Press. ISBN 0-521-65869-1.

Noll, T. (2015). Safety, dependability and performance analysis of aerospace systems. In C. Artho & P. Csaba-lveczky (Eds.), *Formal techniques for safety-critical systems. Communications in computer and information science* (Vol. 476, pp. 17–31). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-17581-2_2. ISBN 978-3-319-17580-5.

Norman, G., Palamidessi, C., Parker, D., & Wu, P. (2007). Model checking the probabilistic π-calculus. In *Proc. 4th international conference on quantitative evaluation of systems (QEST'07)* (pp. 169–178). IEEE Computer Society.

Norman, G., Parker, D., & Sproston, J. (2013). Model checking for probabilistic timed automata. *Formal Methods in System Design, 43*(2), 164–190. http://dx.doi.org/10.1007/s10703-012-0177-x. ISSN 0925-9856.

OMG unified modeling language: Superstructure 2.1.2. object management group. O.M. Group (Ed.), 2007.

OMG systems modeling language (Object management group SysML). O.M. Group (Ed.), 2012.

Ouchani, S., Jarraya, Y., & Mohamed, O. A. (2011). Model-based systems security quantification. In *2011 Ninth annual international conference on privacy, security and trust (PST)* (pp. 142–149). http://dx.doi.org/10.1109/PST.2011.5971976.

Ouchani, S., Mohamed, O. A, & Debbabi, M. (2013). A probabilistic verification framework of SysML activity diagrams. In *2013 IEEE 12th international conference on intelligent software methodologies, tools and techniques (SoMeT)* (pp. 165–170). http://dx.doi.org/10.1109/SoMeT.2013.6645657.

Ouchani, S., Mohamed, O. A., & Debbabi, M. (2014a). A property-based abstraction framework for SysML activity diagrams. *Knowledge-Based Systems, 56*, 328–343. http://dx.doi.org/10.1016/j.knosys.2013.11.016. ISSN 0950-7051.

Ouchani, S., Mohamed, O. A., & Debbabi, M. (2014b). A formal verification framework for SysML activity diagrams. *Expert Systems with Applications, 41*(6), 2713–2728. http://dx.doi.org/10.1016/j.eswa.2013.10.064<http://www.sciencedirect.com/science/article/pii/S0957417413008968>. ISSN 0957-4174.

Pientka, B. (2007). Proof pearl: The power of higher-order encodings in the logical framework LF. In K. Schneider & J. Brandt (Eds.), *Theorem proving in higher order logics. Lecture notes in computer science* (Vol. 4732, pp. 246–261). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-74591-4_19. ISBN 978-3-540-74590-7.

Qureshi, T. N., Chen, D., Lönn, H., & Törngren, M. (2011). From EAST-ADL to AUTOSAR software architecture: A mapping scheme. In *Proceedings of the 5th European conference on software architecture, ECSA'11* (pp. 328–335). Berlin, Heidelberg: Springer-Verlag<http://dl.acm.org/citation.cfm?id=2041790.2041834>. ISBN 978-3-642-23797-3.

Rajlich, V. (2014). Software evolution and maintenance. In *Proceedings of the on future of software engineering, FOSE 2014* (pp. 133–144). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2593882.2593893. ISBN 978-1-4503-2865-4.

Rodriguez, R. J., Fredlund, L., Herranz, A., & Marino, J. (2014). Execution and verification of uml state machines with erlang. In D. Giannakopoulou & G. Salan (Eds.), *Software engineering and formal methods. Lecture notes in computer science* (Vol. 8702, pp. 284–289). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-10431-7_22. ISBN 978-3-319-10430-0.

Segala, R. (1995). A compositional trace-based semantics for probabilistic automata. In I. Lee & S. A. Smolka (Eds.), *CONCUR '95: Concurrency theory. Lecture notes in computer science* (Vol. 962, pp. 234–248). Berlin Heidelberg: Springer. http://dx.doi.org/10.1007/3-540-60218-6_17. ISBN 978-3-540-60218-7.

Singhoff, F., Legrand, J., Nana, L., & Marcé, L. (2004). Cheddar: A flexible real time scheduling framework. *Ada Letters, XXIV*(4), 1–8. http://dx.doi.org/10.1145/1046191.1032298<http://doi.acm.org/10.1145/1046191.1032298>. ISSN 1094-3641.

US Dept of Commerce. (2010). Free nist software tool boosts detection of software bugs. http://2010-2014.commerce.gov/blog/2010/11/09/free-nist-software-tool-boosts-detection-software-bugs.

Yan, G., Zhu, X. Y., Yan, R., & Li, G. (2014). Formal throughput and response time analysis of marte models. In S. Merz & J. Pang (Eds.), *Formal methods and software engineering. Lecture notes in computer science* (vol. 8829, pp. 430–445). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-11737-9_28. ISBN 978-3-319-11736-2.