Electronic Communications of the EASST
Volume 70 (2014)

Proceedings of the 14th International Workshop on Automated Verification of Critical Systems (AVoCS 2014)

Performance Analysis of Distributed and Asynchronous Systems using Probabilistic Timed Actors

Ali Jafari , Ehsan Khamespanah , Marjan Sirjani , and Holger Hermanns

15 pages

Guest Editors: Marieke Huisman, Jaco van de Pol
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Performance Analysis of Distributed and Asynchronous Systems using Probabilistic Timed Actors

Ali Jafari [1], Ehsan Khamespanah [21], Marjan Sirjani [1], and Holger Hermanns [3]

[1] Reykjavik University, School of Computer Science and CRESS
[2] University of Tehran, School of ECE
[3] University of Saarland, School of Computer Science

Abstract: Many real-time distributed applications exhibit probabilistic and non-deterministic behaviors. In this paper, we introduce Probabilistic Timed Rebeca (PTRebeca) as an actor-based language for modeling probabilistic distributed real-time systems with asynchronous message passing. We propose the semantics of PTRebeca model in Timed Markov Decision Process (TMDP), the integral semantics of probabilistic timed automaton (PTA) with one digital clock. To analyze PTRebeca models, we develop a tool set to automatically generate a TMDP model from a PTRebeca model in the form of the input language of PRISM model checker. We use PRISM for performance analysis of PTRebeca models against expected reachability and probabilistic reachability properties. We show the applicability of our approach using a few case studies and experimental results.

Keywords: Probabilistic real-time system, probabilistic Timed Automata, Timed Markov Decision Process, Probabilistic Timed Rebeca, Model checking, Performance analysis

1 Introduction

Our society more and more relies on computing systems that are distributed, consisting of concurrently executing components which communicate asynchronously over networks. Modeling and analyzing such systems is a nontrivial and intricate task, owed to their complex behavior. There is thus a need for modeling languages that match well with computational models and are supported by tools to analyze performance and dependability aspects of such systems.

The actor model was proposed for modeling distributed and asynchronous systems, advocating that software systems are built by composing concurrent objects. Actors are distributed, autonomous objects that interact by asynchronous messages. Building on an event-driven and message-based foundation, actors provide scalability and also less error-prone concurrent models. With the growth of cloud computing, web services, networks of embedded computers, and multicore architectures, programming using the actor model has become increasingly relevant.

Popular actor programming languages and frameworks include Erlang and the Scala / Akka family. Many projects in industry, e.g. at Google (like DART) and Microsoft (like

Asynchronous Agents Library), have explored the actor model. Large applications such as Twitter's message queuing, image processing in MS Visual Studio 2010, as well as the Vendetta game engine have been designed on the basis of this model.

Rebeca [SM01, SMSB04] is an actor-based modeling language designed to enable formal verification of actor models and hence bridge the gap between formal methods and software engineering. Using Rebeca we can deploy a model-driven development approach with formal basis. Rebeca is supported by formal verification tools and techniques which are based on the formal semantics of the language [SJ11]. An extension of Rebeca [ACI+11] has been proposed to provide the ability of modeling and verification of distributed systems with real-time constraints. In this context, Floating Time Transition System (FTTS) were introduced to significantly reduce the state space generated for model checking of Timed Rebeca (TRebeca) models [KSS+14]. Deadlock freedom and schedulability analysis of TRebeca models can be performed using FTTS.

Since its introduction, TRebeca has been used in different areas. One example is in analyzing different routing algorithms and scheduling policies in NoC (Network on Chip) designs [SMMS13, SMS13]. Another example is schedulability analysis of distributed real-time sensor network applications [MKSA13], more specifically a real-time continuous sensing application for structural health monitoring in [LMS13]. An ongoing project is on evaluating different dispatching policies in clouds where we have priorities and deadlines in Mapreduce clusters, based on the work in [GCR+09]. In analyzing all the above mentioned applications, we observed the need for modeling probabilistic behavior. In an earlier work, pRebeca is proposed as an extension of Rebeca to model probabilistic systems [VK12], but pRebeca does not support the time features.

In this paper, we propose Probabilistic Timed Rebeca (PTRebeca) which benefits from modeling features of TRebeca and pRebeca, combining the syntax of pRebeca and TRebeca languages. This aims at enhancing our modeling ability in order to cover more properties, by performance evaluation of probabilistic real-time actors. Although the syntax of PTRebeca is a combination of TRebeca and pRebeca, their semantics and supporting tools are not applicable for PTRebeca. Consequently, we propose a semantics to support timing, probabilistic, and non-deterministic features. To the best of our knowledge, PTRebeca is the first actor-based language which supports time, probability, and non-determinism in modeling distributed systems with asynchronous message passing.

We propose PTRebeca on the basis of a study of different distributed and asynchronous applications, studied to identify what is needed for modeling and analysis of those applications, relative to different probabilistic and timed probabilistic models (discrete, continuous, stochastic) proposed in the literature. In PTRebeca, time is discrete, and discrete probability distributions are used. Using probabilistic and non-deterministic assignments, the computation outcomes and network delays can become probabilistic or non-deterministic. For performance evaluation of PTRebeca models we employ probabilistic model checking, as a single computational techniques for both functional verification and performance evaluation. The benefits of combining performance evaluation with functional verification is elaborated upon in [BHHK10].

The main contributions of this paper are as follows:

- Modeling: PTRebeca supports the modeling of non-deterministic and probabilistic behaviors which is widely required in distributed asynchronous real-time systems.
- Semantics: We propose Timed Markov Decision Process (TMDP) as semantics of PTRebeca. TMDP can be regarded as the discrete time semantics of probabilistic timed automata (PTA) [KNPS06].
- Analysis: We harvest probabilistic model checking algorithms developed for PTA and MDP for the analysis of probabilistic timed properties. For the analysis, we use PRISM [HKNP06] as a back-end model checker, so as to also support expected reachability and probabilistic reachability analysis for PTRebeca models.
- Implementation: We present a tool developed to generate the TMDP of PTRebeca models automatically. The generated TMDP is in the form of an XML file. The XML file is converted to the input language of PRISM.
- Case Studies: We present a ticket servicing and a sensor network application example to demonstrate the feasibility of the approach.

Advantages of PTRebeca in digital time semantics. In PTRebeca, time is discrete, and when generating state space, time can be modeled using a single integer-valued variable. The state space of a PTRebeca model is finite whenever the model represents a recurrent behavior. Additionally, the time-shift equivalency approach [KSS+14] can be used to bound state space when time progresses, and efficient model checking algorithms developed for untimed systems can be applied to integer-timed models [KNPS06, HH09].

Using this approach, PTRebeca models can be verified against properties specified in PCTL [KNSS02]. The model checker PRISM supports this and two other performance measures: expected reachability and probabilistic reachability properties [KNPS06]. This analysis trajectory can be applied to PTRebeca models because the TMDP semantics is equivalent to a diagonal-free and closed PTA. A closed PTA does not contain strict inequalities, a diagonal-free PTA does not compare values of different clocks, which is assured by construction since the sematics has a single clock.

The TMDP semantics unfolds parallel composition of PTRebeca components. There is an alternative approach for performance analysis of PTRebeca models where each component of the PTRebeca model is converted to a PTA. The parallel composition of PTA (of all components) represents the behavior of the PTRebeca model. This approach is explained in [JKS], and PRISM can be used for performance analysis of the PTRebeca model through model checking the resulting PTA. The apparent benefit of avoiding the state space explosion often caused by interleaving parallel composition does not manifest itself in the PTRebeca setting: in [JKS], we demonstrate that the state space generated via the TMDP semantics is much smaller than the state space generated from the parallel composition approach. Therefore we advocate the TMDP semantics as the basis for our mapping to PRISM and for performance analysis of PTRebeca models.

2 Probabilistic Timed Rebeca

In this section, we introduce Probabilistic Timed Rebeca (PTRebeca). We first present Rebeca [SM01, SMSB04], and then we show its extension with timing features to build TRebeca [ACI+11]. Finally we discuss how probability and time are added to Rebeca to build PTRebeca, enabling the modeling of probabilistic timed behaviors. The syntax of PTRebeca is presented in Figure 1. We model a simple ticket service example to explain the modeling features of PTRebeca.

Rebeca. Rebeca is an actor-based modelling language with formal semantics that is supported by model checking tools. A Rebeca model consists of the definition of reactive classes and the instantiation part which is called main. The main part defines instances of reactive classes, called *rebecs*. The behavior of the instances of a reactive class is determined by its message servers. The internal state of a reactive class is represented by the valuation of its state variables.

In Rebeca, computation is event-driven, where messages can be seen as events. Each rebec takes a message from its message queue and executes the corresponding message server. Execution of a message server body takes place atomically (non-preemptively). Communication takes place by asynchronous message passing, which is non-blocking for both sender and receiver. The sender rebec sends a message to the receiver rebec and continues its work. The message is put in the message queue of the receiver. The message stays in the queue until the receiver takes and serves it. Although in theory we define no boundary for the queue length, in the supporting tools we always have a queue length that is defined by the user. The operational semantics of Rebeca is introduced in [SMSB04], to which we refer for more details. The syntax of Rebeca is represented in Figure 1.

Timed Rebeca. TRebeca was introduced as an extension of the Rebeca language to model real-time reactive systems. Just as with Rebeca, the formal semantics of TRebeca is defined using Structural Operational Semantics (SOS) [ACI+11]. In a TRebeca model, each rebec has its own local time, which can be considered as synchronized distributed clocks. Methods are executed atomically, but passing of time can be modeled while executing a method. Instead of a message queue for each rebec, there now exists a bag containing sent messages together with timing information, which are used to process the message in the intended order in time. Different timing primitives are added to Rebeca syntax to cover a variety of timing features that a modeler might need to address in a message-based, asynchronous and distributed setting. These timing primitives are *delay*, *deadline* and *after*, and detailed below. The syntax of timing primitives is shown in Figure 1.

Delay: *delay(t)* increases the value of the local time of the respective rebec by the amount of t.

Deadline: *r.m() deadline(t)*, after t units of time the message m of rebec r is not valid any more and is to be purged from the bag.

$$
\begin{aligned}
\mathit{Model} &::= \mathit{Class}^{*}\ \mathit{Main} \\
\mathit{Main} &::= \textbf{main}\ \{\ \mathit{InstanceDcl}^{*}\ \} \\
\mathit{InstanceDcl} &::= \mathit{className}\ \mathit{rebecName}(\langle \mathit{rebecName} \rangle^{*}) : (\langle \mathit{literal} \rangle^{*}); \\
\mathit{Class} &::= \textbf{reactiveclass}\ \mathit{className}\ \{\ \mathit{KnownRebecs}\ \mathit{Vars}\ \mathit{MsgSrv}^{*}\ \} \\
\mathit{KnownRebecs} &::= \textbf{knownrebecs}\ \{\ \mathit{VarDcl}^{*}\ \} \\
\mathit{Vars} &::= \textbf{statevars}\ \{\ \mathit{VarDcl}^{*}\ \} \\
\mathit{VarDcl} &::= \mathit{type}\ \langle v \rangle^{+}; \\
\mathit{MsgSrv} &::= \textbf{msgsrv}\ \mathit{methodName}(\langle \mathit{type}\ v \rangle^{*})\ \{\ \mathit{Stmt}^{*}\ \} \\
\mathit{Stmt} &::= v = e;\ |\ v = ?(e\langle , e \rangle^{+});\ |\ \mathit{Call};\ |\ \mathit{if}\ (e)\ \{\ \mathit{Stmt}^{*}\ \}\ [\mathit{else}\ \{\ \mathit{Stmt}^{*}\ \}] \\
\mathit{Call} &::= \mathit{rebecName.methodName}(\langle e \rangle^{*})
\end{aligned}
$$

(a) Abstract Syntax of Rebeca

$$
\begin{aligned}
\mathit{Stmt} &::= v = e;\ |\ v = ?(e\langle , e \rangle^{+});\ |\ \mathit{Call};\ |\ \mathit{if}\ (e)\ \{\ \mathit{Stmt}^{*}\ \}\ [\mathit{else}\ \{\ \mathit{Stmt}^{*}\ \}]\ |\ \textbf{delay}(v); \\
\mathit{Call} &::= \mathit{rebecName.MethodName}(\langle e \rangle^{*})\ [\textbf{after}(v)]\ [\textbf{deadline}(v)]
\end{aligned}
$$

(b) Changes in the syntax of Rebeca to build TRebeca

$$
\begin{aligned}
\mathit{Stmt} &::= v = e;\ |\ v = ?(e\langle , e \rangle^{+});\ |\ \mathit{Call};\ |\ \mathit{if}\ (e)\ \{\ \mathit{Stmt}^{*}\ \}\ [\mathit{else}\ \{\ \mathit{Stmt}^{*}\ \}]\ |\ \mathit{delay}(v); \\
&\quad |\ \mathbf{v} = ?(\mathbf{e_p} : \mathbf{e}\langle , \mathbf{e_p} : \mathbf{e} \rangle^{+});
\end{aligned}
$$

(c) Changes in the syntax of TRebeca to build PTRebeca

Figure 1: (a) Abstract syntax of Rebeca. Angle brackets ⟨...⟩ are used as meta parenthesis, superscript + for repetition at least once, superscript ∗ for repetition zero or more times, whereas using ⟨...⟩ with repetition denotes a comma separated list. Brackets [...] indicates that the text within the brackets is optional. The symbol ? shows non-deterministic choice. Identifiers *className*, *rebecName*, *methodName*, *v*, *literal*, and *type* denote class name, rebec name, method name, variable, literal, and type, respectively; and *e* denotes an (arithmetic, boolean or nondetermistic choice) expression.
(b) Changes for Timed Rebeca. The timing primitives are added to *Stmt* and *Call* statements. The value of variable *v* in timing primitives is a natural number.
(c) Changes for Probabilistic Timed Rebeca. The probabilistic assignment is added to *Stmt*. The expression e_{p_i} denotes an expression which returns probability. The symbol ? shows either non-deterministic assignment or probabilistic assignment.

After: *r.m() after(t)*, the message cannot be taken from the bag before *t* time units have passed.

Upon sending, a message is put in the message bag at the receiver together with its associated time tag and *deadline* tag. The time tag of a message is the value of local time

of the sender when the message was sent, unless the message is augmented with an *after* primitive. In this case the value of the argument of *after* is added to the value of local time of the sender to build the time tag.

2.1 Probabilistic Timed Rebeca

PTRebeca language supports the modeling and verification of real-time systems with probabilistic behaviors. The syntax of PTRebeca is a combination of pRebeca and TRebeca. We propose the appropriate semantics for PTRebeca to be able to model and verify probabilistic properties. In Figure 1, we show the extension made to the syntax of TRebeca to build PTRebeca. In a probabilistic assignment, a value is assigned to the variable with the specified probability. In probabilistic assignment, $e_{p_1} \ldots e_{p_n}$ are real values between 0 and 1, and sum up to 1. Notably, by using probabilistic assignment, the value of the timing constructs (delay, after, and deadline) can also become probabilistic.

Different probabilistic behaviors can be modeled using PTRebeca, depending on the system under study. We present a simple ticket service system in Figure 2 to illustrate how PTRebeca can be applied. Each entity in the system is mapped to an actor in PTRebeca model. The ticket service model includes a customer, a ticket service, and an agent. The customer *c* sends a ticket request by sending the message *sendRequest()* to the agent *a* (line 27). The agent forwards the request to the ticket service *ts* by sending the message *requestTicket()* (line 17). The message *requestTicket()* has a deadline which is set non-deterministically (line 16). The ticket service issues a ticket and replies to the agent request by sending the message *sendTicket()* (line 6). The agent sends the message *getTicket* to the customer to complete the issuing process (line 20). The customer sends a new request after 10 or 30 units of time with probabilities of 0.25 or 0.75, respectively (lines 29-32).

```
1  reactiveclass
       TicketService(4){
2    knownrebecs {Agent a;}
3    TicketService (){ }
4    msgsrv requestTicket() {
5      delay(3);
6      a.sendTicket();
7    }
8  }
9  reactiveclass Agent(4){
10   knownrebecs {
11     TicketService ts;
12     Customer c;
13   }
14   Agent(){ }
15   msgsrv sendRequest() {
16     int a = ?(4,5);
17     ts.requestTicket()
           deadline(a);
18   }
19   msgsrv sendTicket() {
20     c.getTicket();
21   }
22 }
23 reactiveclass Customer(4) {
24   knownrebecs {Agent a;}
25   Customer() {self.try();}
26   msgsrv try() {
27     a.sendRequest();
28   }
29   msgsrv getTicket() {
30     int b =
           ?(0.75:30,0.25:10);
31     self.try() after(b);
32   }
33 }
34 main {
35   Agent a(ts, c):();
36   TicketService ts(a):();
37   Customer c(a):();
38 }
```

Figure 2: The model of the ticket service system.

3 Semantics of Probabilistic Timed Rebeca

In this section, we define the Timed Markov Decision Process (TMDP) semantics of a PTRebeca model. Formally, a TMDP is defined as follows [JLS07].

Definition 1 (Timed Markov Decision Process) *A timed Markov decision process is a tuple of* $(TMDP)T = (S, s_0, Act, \rightarrow, L)$ *includes the following components:*

- *A set of states* S *with an initial state* $s_0 \in S$,
- *A set of actions* Act,
- *A timed probabilistic, non-deterministic transition relation* $\rightarrow \subseteq S \times Act \times \mathbb{N} \times Dist(S)$ *such that, for each state* $s \in S$, *there exists at least one tuple* $(s, -, -, -) \in \rightarrow$,
- *A labelling function* $L : S \rightarrow 2^{AP}$, *where* AP *is the set of atomic propositions.* □

The transitions in a TMDP are performed in two steps: given that the current state is s, the first step is a non-deterministic selection of $(s, act, d, v) \in \rightarrow$, where act denotes a possible action and d specifies the duration of the transition; in the second step, a probabilistic transition to state s' is made with probability $v(s')$. Function $v \in Dist(S)$ denotes a discrete probability distribution.

In the following, we define some concepts for PTRebeca models before turning to the TMDP semantics of PTRebeca.

Definition 2 (Probabilistic Timed Rebeca Model) *A Probabilistic Timed Rebeca model* $\mathcal{M}$ *is the set of rebecs which are concurrently executing.* □

A computation of Probabilistic Timed Rebeca model $\mathcal{M}$ takes place by execution of all rebecs defined in the model according to the SOS-semantics in [ACI+11]. For a Probabilistic Timed Rebeca model $\mathcal{M}$, the function $O(\mathcal{M})$ returns all rebecs in the model $\mathcal{M}$.

Definition 3 (State of a PTRebeca model in TMDP) *A state of a PTRebeca model* $\mathcal{M}$ *is a tuple* $s = \left(\prod_{r_i \in O(\mathcal{M})} (state(r_i) \times pc \times rt)\right) \times \mathbb{T}$, *where* $state(r_i)$ *is the state of rebec* r_i, $\mathbb{T} \in \mathbb{N}$ *is the current time of state,* $pc \in \mathbb{N}$ *is the program counter of rebec* r_i, *and* $rt \in \mathbb{N}$ *is the resuming time of rebec* r_i. □

Each rebec of $\mathcal{M}$ has a state which consists of the values of its state variables, its local time, and its message bag. Functions $\text{sv}(s, r_i)$, $\text{bag}(s, r_i)$, and $\text{now}(s, r_i)$ return the state variable valuation function, the content of message bag, and the local time of rebec r_i in state s, respectively. In TMDP semantics of a PTRebeca model, the local times of rebecs have the same value. We define function $now(s)$ to access the time in state s.

The rebec program counter, pc of rebec r_i specifies the statement to be executed, and function $pc(s, r_i)$ returns the value of program counter of rebec r_i in state s. The rebec resuming time, rt of rebec r_i determines the time when the statement of the message

server of rebec r_i, pointed to by pc, is executed. Function $rt(s,r_i)$ returns the value of resuming time of rebec r_i in state s.

In the initial state, the local time of all rebecs are set to zero, and the constructor of all rebecs are executed to initialize state variables and queues content. Initially, for all rebecs the value of program counter and the value of resuming time are supposed to be *null*.

Definition 4 (The Content of a Message Bag) *A tuple* tmsg = (msgsig, arrival, deadline) *is a message where* msgsig *is the message content,* arrival *is the arrival time of the message, and* deadline *is the deadline of the message. The arrival time of the message is computed based on the local time of the sender and the value of "after" of send message statement. The deadline of the message is also computed based on the local time of the sender.* □

For tmsg ∈ bag(s,r_i), the functions sig(tmsg), ar(tmsg), and dl(tmsg) return the msgsig, arrival, and deadline of the message tmsg, respectively. The message content *msgsig* consists of the message name, the sender, the receiver, and its actual parameters and is shown as "sender → receiver.msgname(parameters)".

Definition 5 (Possible Messages) *The set of messages* $Tmsg = \{tmsg \mid \forall r_i, r_j \in O(M), \forall ar, dl \in \mathbb{N}, tmsg = (r_i \rightarrow r_j.msgname(), ar, dl)\}$ *is the set of all possible messages which can be sent by any rebec* r_i *to another rebec* r_j *at any arrival time ar and deadline dl.* □

Definition 6 (Rebec Enabled Messages) *Enabled messages of a rebec are messages whose arrival time is less than the time of state s:* $em(s,r_i) = \{\text{tmsg} \in \text{bag}(\text{s},\text{r}_\text{i}) \mid \text{ar(tmsg)} \leq \text{now(s)}\}$. □

Definition 7 (TMDP semantics of a PTRebeca model) *A TMDP of PTRebeca model* $\mathcal{M}$ *is a tuple* $(S, s_0, Act, \rightarrow, L)$*, where:*

- S *is the set of* states *according to Definition 3,*
- $s_0 \in S$ *is the* initial *state,*
- Act *is a set of* $Tmsg \cup \{\tau\} \cup \mathbb{T}$*, where Tmsg is the set of all possible messages which can be sent by any rebec to its known rebecs,* τ *is an internal action and* $\mathbb{T} \in \mathbb{N}$ *is the progress of time.*
- $\rightarrow \subseteq S \times Act \times \mathbb{N} \times Dist(S)$ *is the transition relation, where* $(s, act, d, \nu) \in \rightarrow$ *if and only if one of the following conditions hold for* s*.*

  1. ***(Taking a message for execution)*** *If in state* s*, there exists* $r_i \in O(\mathcal{M})$ *such that* $pc(s,r_i) = null$ *and* $em(s,r_i) \neq \emptyset$*: The execution of* $tmsg \in em(s,r_i)$ *results in* s' *with probability* $\nu(s') = 1$ *and d=0. In this case act is equal to tmsg, tmsg is extracted from the message bag of the rebec* r_i*,* $pc(s,r_i)$ *is set to the first statement of message server tmsg, and* $rt(s,r_i)$ *is set to* $now(s)$*.*

2. ***(Internal action τ)*** *If in state s, there exists $r_i \in O(\mathcal{M})$ such that $pc(s,r_i) \neq null$ and $rt(s,r_i) = now(s)$: The statement of message server of r_i specified by $pc(s,r_i)$ is executed and one of the following cases may occur based on the statement execution:*
   (a) *The statement is an ordinary statement: the execution of statement may change the value of some state variables of the rebec r_i or induce sending a message to a rebec. Then, $pc(s,r_i)$ is increased by one, the act is τ, d=0, and the execution of τ results in s' with probability $\nu(s') = 1$.*
   (b) *The statement is a non-deterministic assignment: the execution of non-deterministic assignment $a =?(v_1,...,v_n)$ results in n different transitions from s to states $s'_1,s'_2,...,s'_n$, where $a = v_i$ in state s'_i. For each transition, the act is τ, d=0, and the execution of τ results in s'_i $(1 \leq i \leq n)$ with probability $\nu(s'_i) = 1$.*
   (c) *The statement is a probabilistic assignment: the execution of probabilistic assignment $a =?(p_1 : v_1,...,p_n : v_n)$ results in a transition from s to states $s'_1,s'_2,...,s'_n$, where $a = v_i$ in state s'_i. The act is τ, d=0, and the execution of τ results in s'_i $(1 \leq i \leq n)$ with probability $\nu(s'_i) = p_i$.*
   (d) *The statement is a delay statement with parameter $t \in \mathbb{N}$: the execution of the delay statement does not change $pc(s,r_i)$ (because the execution of delay statement is not yet complete), and $rt(s,r_i)$ is set to $now(s)+t$. (Note: the value of $pc(s,r_i)$ will change to the next statement after completing the execution of the delay which can be seen in item 3.) The act is τ, d=0, and the execution of τ results in s' with probability $\nu(s') = 1$.*

   When the last statement of the message server of r_i is executed, the $pc(s,r_i)$ is set to null.
3. ***(Progress of time)*** *If in state s, none of the aforementioned conditions in items 1 and 2 hold: this means $\nexists r_i \in O(\mathcal{M}), ((pc(s,r_i) = null \wedge em(s,r_i) \neq \emptyset) \vee (pc(s,r_i) \neq null \wedge rt(s,r_i) = now(s)))$. In this case, $now(s)$ is increased by the minimum amount of $t_1 \in \mathbb{N}$ such that one of the aforementioned conditions becomes true. If $pc(s,r_i) \neq null$ and $rt(s,r_i) = now(s)$ (the current value of $pc(s,r_i)$ points at a delay statement), $pc(s,r_i)$ is increased by one. The act is set to time, $d = t_1$, and the execution of action time results in s' with probability $\nu(s') = 1$.*

– *A labelling function $L : S \rightarrow 2^{AP}$.*

When more than one transition is enabled in state s, a non-deterministic selection is made. □

4 Analysis of Probabilistic Timed Rebeca and Experimental Results

We have developed a tool set [Reb] in order to generate the TMDP semantics from PTRebeca models. This TMDP semantics can be exported to PRISM as a single MDP module

with one integer-valued variable modeling the passage of time. In [JKS], we show the PRISM code generated for the ticket service example presented in Figure 2. Using a dedicated time action and the ability of assigning rewards to transitions in PRISM, we can analyze expected-time reachability and time-bounded probabilistic reachability properties.

In PTRebeca models, the capacity of message bags is bounded. The number of states in PTRebeca model can be finitely represented when the system shows recurrent behavior. We also use the time-shift equivalence approach proposed in [KSS+14] to avoid state space explosion otherwise induced by time progress. In this approach, two TMDP states s and t (in the sense of Definition 3) are time-shift equivalent if the values of all variables except timing variables, i.e. local time, arrival time, deadline, in states s and t are identical. Therefore, the two states can be identified by shifting time.

The PRISM modeling language is a state-based language while PTRebeca language benefits from high-level data structures and constructs which arguably makes modeling easier. PRISM models are thus closer to the underlying probabilistic models and therefore we bridge to PRISM on the semantics level.

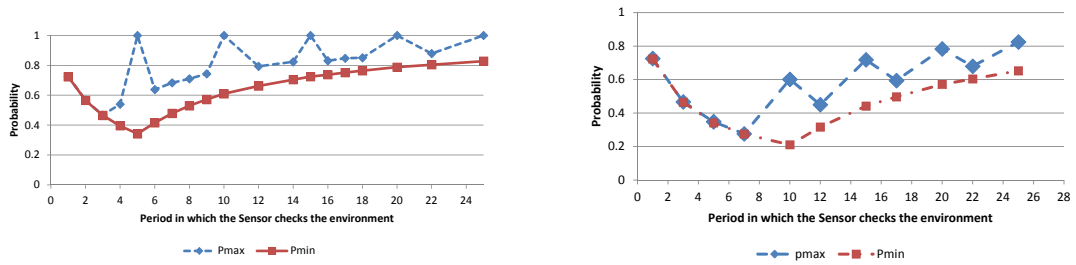
4.1 Experimental Results

In this section, we present an example demonstrating the applicability of the proposed approach for performance analysis of asynchronous systems. We also examine different versions of the ticket service model shown in Figure 2, which is detailed in [JKS].

Sensor Network. The sensor network model is shown in [JKS]. There is a lab environment in which the toxic level changes periodically after an amount of time, specified by the value of variable *changingPeriod*. The environment is safe at first. The toxic level is changed to a dangerous level with a probability of 0.02. If the toxic level of the environment reaches a dangerous level, the scientist, working inside the lab, will die after a specified amount of time, specified by the value of variable *scientistDeadline*. One sensor in the lab environment is measuring the toxic level of the lab. The measured information is sent periodically by the sensor to the administrator. A sensor may fail to report the measurement to the administrator with a probability of 0.01. If the toxic level reported by the sensor reaches a predefined dangerous level, the administrator sends a message to the scientist, who is assumed to be working in the environment, to inform him to leave the lab and go to a safe place. The administrator waits for a while for an acknowledgement from the scientist. If the acknowledgement is not received by the admin on time, the admin orders a rescue team to get hold of the scientist.

In this model the value of probabilities are small enough to let the model converge to the optimum value; and consequently show the real behavior. Since the model includes non-deterministic behaviors, the model checker computes the maximum and the minimum probabilities over all paths in the generated state space.

Figure 3(a) shows the maximum and the minimum probabilities of the scientist death when the value of variable *checkingPeriod* changes. If the sensor checks the environment with a high frequency, i.e. the value of variable *checkingPeriod* is low, the probability



(a) The value of variable *scientistDeadline* is 10.

(b) The value of variable *scientistDeadline* is 12.

Figure 3: The maximum and minimum probabilities that the scientist eventually dies, when the sensor frequency changes.

of sensor failure will increase, resulting in high probability of the scientist death. For example, when the sensor checks the environment once every units of time, the environment is checked five times before the first change in the environment. Therefore, the cost of the sensor use and consequently the probability of sensor failure increases. When the sensor frequency is low, the environment changes cannot be detected on time; resulting in a high probability of the scientist death. There is an optimum value for the variable *checkingPeriod*, i.e. sensor frequency, which is five according to the obtained results reported in Figure 3(a).

As the results show, at times 5, 10, 15, 20, and 25, the maximum probability of the scientist death equals one. At these times because of concurrency between time related behaviors in the system, there is a scenario in which the dangerous level is reported too late to the administrator and the scientist will die. At these times, the execution sequence of the following messages is important and causes the special behavior: (1) checking the sensor value by the administrator (it is repeated periodically after 5 units of time), (2) changing the toxic level of the environment to a dangerous level (period is 5 units of time), (3) checking the environment by the sensor (Figure 3(a) shows the probability of the scientist death for different value of this period), and (4) sending a message *die* to the scientist (after 10 units of time) when the environment is dangerous.

This experiment shows that the exceptional timing behavior can be revealed by probabilistic performance evaluation. It is not possible to find this special behavior by the tools and techniques developed for TRebeca language at this moment.

In Figure 3(b), the value of variable *scientistDeadline* equals 12; the scientist has more time to be saved before being killed by the toxic environment. The maximum probability of the scientist death is not equal to one at times 5, 10, 15, 20, and 25, but because of concurrency between time related behaviors, there is a scenario in which the dangerous level is reported too late and consequently the maximum probability of the scientist death increases. There is an optimum value for the variable *checkingPeriod*, i.e. sensor frequency, which is ten in this experiment.

5 Related Work

PRISM. PRISM is a well-established and powerful model checker with a state-based input language. An input model of PRISM is composed of a number of modules which can share variables and interact with each other. PRISM is well equipped with theories and reduction techniques [HKNP06], but lacks high-level programming constructs like loops, and primary data structures like arrays, which makes modeling hard.

In contrast, PTRebeca provides high-level object-based programming features and asynchronous message passing, which makes modeling easier. So, in modeling we benefit from capabilities of PTRebeca, and in analysis we use the capabilities of the PRISM model checker.

Modest. Modest [HHHK13] is a high-level and convenient language for describing stochastic timed and hybrid systems. It supports loop constructs, structs and arrays, exception handling, and other advanced programming constructs. For the probabilistic timed fragment of Modest, model checking can be performed using a digital time semantics [HH09] or by a direct mapping to timed automata. Both approaches use PRISM as a backend model checker.

In contrast to Modest, PTRebeca supports object-based programming features, and follows the asynchronous message passing paradigm of actors, while Modest relies on synchronous message passing. Otherwise, the spirit, especially with respect to the analysis via PRISM, is similar.

ProbMela. ProbMela is a probabilistic version of Promela [Hol97]. The operational semantics of ProbMela is defined as an MDP [BCG04]. In [CB06], ProbMela is used as input language for the MDP model checker LiQuor which provides qualitative and quantitative analysis of LTL properties. There is also a mapping from ProbMela to the PRISM language, which makes probabilistic analysis possible [CBGP08].

PTRebeca is an event-driven and actor-based language whereas ProbMela is process-based. Both languages are asynchonous in spirit. We proposed a semantics of PTRebeca as TMDP (or PTA with digital clocks), enabling the analysis of timing and probabilistic behaviors of asynchronous systems.

PMaude. PMaude extends standard rewriting theories of Maude with probability [AMS06]. There is an actor extension of probabilistic rewriting theories for PMaude which removes non-determinism. A statistical technique is provided to analyze quantitative aspects of systems using discrete-event simulation. In comparison with PMaude, modeling asynchronous systems is more straightforward in PTRebeca language as it is an actor-based language. Also PTRebeca supports non-determinism in the model and there is no need to resolve it by assuming distribution on different choices of non-determinism. It is because of the probabilistic model checking facilities which are provided by PRISM.

Summary. In PMaude, probability distribution functions (rates and stochastic functions) are provided for modeling probabilistic behaviors. Also, PMaude implements stochastic continuous-time. In ProbMela, probabilities are drawn from discrete probability distributions, and passage of time can be modeled using a timer process. Modest enables a direct high-level modelling of PTA and more complex models. In all aforementioned languages, non-deterministic behavior can be modeled. In analysis, PMaude resolves non-determinism, and uses statistical model checking to verify properties which results in inaccurate results. In the analysis of ProbMela and Modest, non-determinism is not resolved. Modest also provides the option of a digital clock semantics, which, just like we do here, is handed over to PRISM for model checking.

Our focus in designing PTRebeca has been on ease of modeling and efficiency of analysis mainly for asynchronous applications. To this end, we use discrete time model and discrete probability distributions. These decisions showed to be effective in modeling different applications that we have targeted. Moreover, resolving non-determinism by a discrete probability distribution generates inaccurate estimations, so, we avoided that by choosing TMDP as the semantics of PTRebeca. We were able to formalize the advance of time in our model using a single integer-valued variable. The language design of PTRebeca and its analysis approach is closest to the Modest approach, apart from the latter not being object-oriented and not being asynchronous by design.

6 Conclusion

In this paper we introduced the syntax and semantics of Probabilistic Timed Rebeca (PTRebeca) for modeling and verification of probabilistic real-time actor systems. As the model of time in PTRebeca is discrete, we decided to use discrete-time TMDP with an integer-valued time variable for the semantics of PTRebeca. PTRebeca models can thus be analyzed against PCTL, expected reachability, and probabilistic reachability properties. Our proposed approach is implemented as a part of Afra toolset [Reb]. PRISM is used as a back-end model checker for analyzing PTRebeca models. This is similar to the approach put forward for Modest, and we are therefore considering a direct connection between PTRebeca and Modest.

In addition to the benefits of using TMDP semantics for analysis of PTRebeca models, our technique is based on the actor model of computation where the interaction is solely based on asynchronous message passing between the components. Hence, the proposed semantics is general enough to be applied to similar computation models where there is message-driven communication and autonomous objects as units of concurrency, and there exists discrete probabilistic behaviors in the model such as agent-based systems.

Acknowledgement

The work on this paper was supported by the project “Timed Asynchronous Reactive Objects in Distributed Systems: TARO” (nr. 110020021) of the Icelandic Research Fund.

Bibliography

[ACI+11] L. Aceto, M. Cimini, A. Ingólfsdóttir, A. H. Reynisson, S. H. Sigurdarson, M. Sirjani. Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca. In *FOCLASA'11*. Pp. 1–19. 2011.

[AMS06] G. Agha, J. Meseguer, K. Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *Electronic Notes in Theoretical Computer Science (ENTCS)* 153(2):213–239, May 2006.

[BCG04] C. Baier, F. Ciesinski, M. Groesser. PROBMELA: a modeling language for communicating probabilistic processes. 2004.

[BHHK10] C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen. Performance evaluation and model checking join forces. *Commun. ACM* 53(9):76–85, Sept. 2010.

[CB06] F. Ciesinski, C. Baier. LiQuor: A tool for Qualitative and Quantitative Linear Time analysis of Reactive Systems. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*. Pp. 131–132. IEEE CS Press, 2006.

[CBGP08] F. Ciesinski, C. Baier, M. Groesser, D. Parker. Generating Compact MTBDD-Representations from Probmela Specifications. In *Proceedings of the 15th international workshop on Model Checking Software*. SPIN '08, pp. 60–76. 2008.

[GCR+09] I. Gupta, B. Cho, M. R. Rahman, T. Chajed, C. L. Abad, N. Roberts, P. Lin. Natjam: Eviction Policies For Supporting Priorities and Deadlines in Mapreduce Clusters. 2009.

[HH09] A. Hartmanns, H. Hermanns. A Modest Approach to Checking Probabilistic Timed Automata. In *QEST*. Pp. 187–196. IEEE Computer Society, 2009.

[HHHK13] E. M. Hahn, A. Hartmanns, H. Hermanns, J.-P. Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design* 43(2):191–232, 2013.

[HKNP06] A. Hinton, M. Z. Kwiatkowska, G. Norman, D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proceedings of 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*. Lecture Notes in Computer Science, pp. 441–444. Springer-Verlag, 2006.

[Hol97] G. J. Holzmann. The Model Checker SPIN. *Software Engineering* 23(5):279–295, 1997.

[JKS] A. Jafari, E. Khamespanah, M. Sirjani. Performance Analysis of Distributed and Asynchronous Systems using Probabilistic Timed Actors (technical report). http://rebeca.cs.ru.is/files/Documents/PTR2PTA.pdf.

[JLS07] M. Jurdziński, F. Laroussinie, J. Sproston. Model checking probabilistic timed automata with one or two clocks. In *Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems*. TACAS'07, pp. 170–184. 2007.

[KNPS06] M. Kwiatkowska, G. Norman, D. Parker, J. Sproston. Performance Analysis of Probabilistic Timed Automata using Digital Clocks. *Formal Methods in System Design* 29:33–78, 2006.

[KNSS02] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.* 282(1):101–150, June 2002.

[KSS+14] E. Khamespanah, Z. Sabahi Kaviani, M. Sirjani, R. Khosravi, M.-J. Izadi. Timed Rebeca Schedulability and Deadlock Freedom Analysis Using Bounded Floating-Time Transition System. In *Journal of Science of Computer Programming*. 2014.

[LMS13] L. Linderman, K. Mechitov, B. F. Spencer. TinyOS-based Real-Time Wireless Data Acquisition Framework for Structural Health Monitoring and Control. *Structural Control and Health Monitoring* 20(6):10071020, June 2013.

[MKSA13] K. A. Mechitov, E. Khamespanah, M. Sirjani, G. Agha. A Model Checking Approach for Schedulability Analysis of Distributed Real-Time Sensor Network Applications. In *Submitted for Publication*. 2013.

[Reb] Rebeca. Rebeca Homepage. http://www.rebeca-lang.org.

[SJ11] M. Sirjani, M. M. Jaghoori. Ten Years of Analyzing Actors: Rebeca Experience. In *Formal Modeling: Actors, Open Systems, Biological Systems*. Pp. 20–56. 2011.

[SM01] M. Sirjani, A. Movaghar. An Actor-Based Model for Formal Modelling of Reactive Systems: Rebeca. Technical report CS-TR-80-01, Tehran, Iran, 2001.

[SMMS13] Z. Sharifi, M. Mosaffa, S. Mohammadi, M. Sirjani. Functional and Performance Analysis of Network-on-Chips Using Actor-based Modeling and Formal Verification. In proceedings of AVOCS'13. 2013.

[SMS13] Z. Sharifi, S. Mohammadi, M. Sirjani. Comparison of NoC Routing Algorithms Using Formal Methods. In proceedings of PDPTA'13. 2013.

[SMSB04] M. Sirjani, A. Movaghar, A. Shali, F. de Boer. Modeling and Verification of Reactive Systems using Rebeca. *Fundamenta Informatica* 63(4):385–410, Dec. 2004.

[VK12] M. Varshosaz, R. Khosravi. Modeling and verification of probabilistic actor systems using prebeca. In *Proceedings of the 14th international conference on Formal Engineering Methods*. ICFEM'12, pp. 135–150. 2012.