

P_UPPAAL

A tool for capturing the probabilistic behaviour of UPPAAL models

G. Díaz, D. Cazorla, F. Cuartero and V. Valero
University of Castilla-La Mancha, Departamento de Informática de Albacete
Avd. de España s/n, 02071 Albacete, Spain.

Abstract

In this article we present a new tool P_UPPAAL that allows the system modeling, simulation and verification. The modeling and simulation processes are driven through a graphical interface and the verification process is realized by RAPTURE. This tool, P_UPPAAL, introduces the probabilistic behaviour to UPPAAL models. To add this behaviour, the tool imports models from UPPAAL and translates them into a graphical model. The graphical model is validated through the simulator. The last step is to translate to a textual format which is used by RAPTURE engine. RAPTURE makes model check of quantitative reachability properties on Markov decision processes. The technique is based on the analysis performed on an abstraction of the model under analysis and a refinement process performed on this abstraction.

Keywords: Model Checking, Probabilistic Real-Time Systems, Probabilistic Transition System and Markov Decision Processes.

1 Introduction

We treat about the suite tool P_UPPAAL. We show the P_UPPAAL functionality (system modeling, simulation and verification) and its internal behaviour. To see the P_UPPAAL functionality, we describe its graphical interface. And to see the P_UPPAAL internal behaviour, we show the simulation and the verification process.

The suite tool P_UPPAAL allows us the system modeling, simulation and verification. The simulation process is required for validating the system modeling. The modeling and simulation processes are led through graphical interfaz and the verification process is realized by RAPTURE [6, 4]. This tool, P_UPPAAL, introduces the probabilistic behaviour to the UPPAAL models. To add this behaviour, the tool imports models from UPPAAL and translates them into a graphical model. The

graphical model is validated through the simulator. The last step is to translate to a textual format which is used by RAPTURE engine.

The Rapture engine provides the strategies for model checking quantitative reachability properties of Markov decision processes [8] by successive refinements. The properties are analyzed on abstractions rather than directly on the given model. Such abstractions are expected to be significantly smaller than the original model, and may safely refute or accept the required property. Otherwise, the abstraction is refined and the process repeated.

Model checking of finite state systems allows settlement of qualitative properties such as “the system will never reach an erroneous situation”. However, it is often vital that additional quantitative properties are established in order for the system to be considered correct. Such properties include real-time requirements such as “a desired state will be reached within 105 seconds” and probabilistic properties of the type “a desired state will be reached with probability at least 99%”.

Thus, this kind of Model checking allows us to check the probabilistic behaviour into UPPAAL models. Notice that, P_UPPAAL does not require UPPAAL to work. Because it allows modeling the system using the editor provided and after that, it export the system modeled to a UPPAAL model. Furthermore, it might validate the system modeled and verify quantitative reachability properties within the system modeled. Thus, P_UPPAAL is a suite of tools.

2 P_UPPAAL Architecture and Functionality

The architecture of P_UPPAAL, shown in figure 1, is divided in three main parts, i.e. the Editor, the Simulator and the Verifier. The Editor allows us to model the system studied. The system is modeled as a probability transition system (PTS for short). The probability behavior of the system is captured by a variable in the transitions. The Simulator allows us to validate the system. Through the simulation, you may decide if the system modeled holds the behaviour expected. The verifier checks the quantitative reachability properties defined on it. It translates the system modeled into textual format and, after that, it connects with RAPTURE tool which delivers the verdict. Therefore, if the system holds the property defined, the verifier receives from RAPTURE the answer yes, otherwise, it receives the answer not.

3 Importing Models from UPPAAL

The import process is based in the following idea: “The UPPAAL and P_UPPAAL models are derived from Labeled Transitions System”. It allow us to abstract the

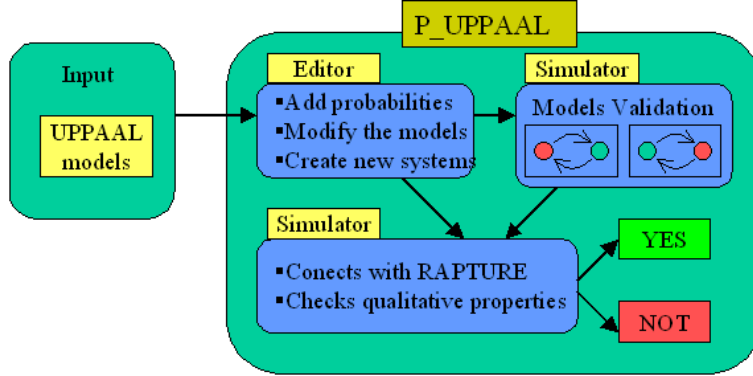


Figure 1: Architecture of the RAPTURE tool

modeling phase. Thus, we only import from the UPPAAL models the states, the transitions, the variables and channels, the synchronization, the guards; i.e, We import all the model except the timing characteristics (invariants and clocks).

The figure 2, shows a simple example of the original model from UPPAAL and the imported model in P_UPPAAL respectively. As you can see in the figure, the import model does not have the clock i which could be seen at the UPPAAL model.

You may found the *import option* at the P_UPPAAL menu that opens a dialog to search the UPPAAL model for import process.

4 Modeling Systems

P_UPPAAL uses the Probabilistic Transitions Systems for modeling the systems. It allows us to add probabilities to the models.

4.1 Probabilistic Transitions Systems

As is pointed out in [4], Probabilistic transition systems (PTS for short) generalize the well-known transition systems with probabilistic information. In a PTS, a transition does not lead to a single state but to a probability space whose sample space is a set of states. The model we define is widely used (see e.g. [7]) and is also known as Markov decision processes [8].

Let $Distr(\Omega)$ denote the set of all probability distributions over the sample space Ω .

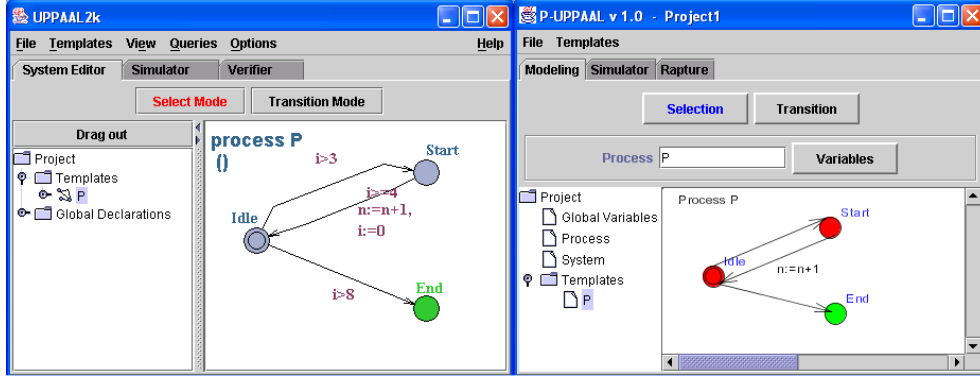


Figure 2: An example of UPPAAL model and the P-UPPAAL imported model, respectively.

Definition 1. Probabilistic Transition Systems. A probabilistic transition system (PTS for short) is a structure $T = (S, \rightarrow)$ where S is a set of states, and $\rightarrow \subseteq X \times \text{Distr}(S)$ is the transition relation. We write $s \rightarrow \pi$ for $(s, \pi) \in \rightarrow$. A PTS is said to be a fully probabilistic transition systems (FPTS for short) if $s \rightarrow \pi \wedge s \rightarrow \rho \Rightarrow \pi = \rho$. A rooted PTS (resp. FPTS) (T, s_0) is PTS (resp. FPTS) equipped with an initial state $s_0 \in S$. A PTS may be equipped with a proposition assignment $p: S \rightarrow \mathcal{P}(AP)$, where AP is a finite set of atoms and $\mathcal{P}(AP)$ the set of propositional formula on AP . We define $\models \subseteq X \times \mathcal{P}(AP)$ by $s \models g$ iff $p(s) \Rightarrow g$ is a tautology.

We write $s \rightarrow$ whenever there is a π such that $s \rightarrow \pi$; otherwise, we write $s \nrightarrow$. We let $\text{sup}(\pi) = \{s' | \pi(s') > 0\}$. We call s a sink state if $s \nrightarrow$.

Let $T = (S, \rightarrow)$ be a PTS. A simple path in T is a finite sequence of states $\sigma = s_0 s_1 s_2 \dots s_n$, where for each $0 \leq i < n$ there exists $\pi_i \in \text{Distr}(S)$ such that $s_i \rightarrow \pi_i$ and $\pi_i(s_{i+1}) > 0$. Let $\sigma(i)$ denote the state in the i -th position. Let $|\sigma|$ be the length of σ and let $\text{first}(\sigma) = \sigma(1)$ and $\text{last}(\sigma) = \sigma(|\sigma|)$. A simple path starting from $s \in S$ is a simple path σ with $\sigma(1) = s$. A state t is reachable from another state s in T if there is a simple path in T with $s = \text{first}(\sigma)$ and $t = \text{last}(\sigma)$.

A full path in T is a sequence of states σ being either a simple path with $\text{last}(\sigma) \nrightarrow$, or a infinite sequence. We denote by $s - \text{paths}(T)$ and $f - \text{paths}(T)$ the sets of simple paths and fullpaths in T starting from s . Let $\text{reach}(T, s)$ denote the set of all states reachable from s in T .

We now define a probability measure on the full paths of a FPTS F . For any simple path $\sigma \in s - \text{paths}(F)$, define $\sigma \uparrow = \{\pi \in f - \text{paths}(F) | \sigma \leq \pi\}$ where $\leq$ is the classical prefix order on sequences. Let $\mathcal{F}(F)$ be the smallest σ -fied on

$f - paths(F)$ which contains $\sigma_{\uparrow}$ for each $\sigma \in s - paths(F)$. Let $\mathcal{F}(F)$ such that for any $\sigma = s_0 s_1 \dots s_n \in s - paths(F)$ such that for any $\sigma = s_0 s_1 \dots s_n \in s - paths(F)$ such that $s_1 \rightarrow_F \pi_i$ for all $i, 0 \leq i < n$:

$$P_{F,s}(\sigma_{\uparrow}) \triangleq \text{if } (s = s_0) \text{ then } \pi_0(s_1) \cdot \pi_1(s_2) \cdot \dots \cdot \pi_{n-1}(s_n) \text{ else } 0$$

We will write $P_{F,s}(\sigma)$ to denote $P_{F,s}(\sigma_{\uparrow})$. Intuitively, $P_{F,s}(\sigma)$ is the probability of σ in F starting from s .

Any given PTS T defines a set of probabilistic executions, each one obtained by iteratively scheduling one of the possible post-state distributions from each pre-state, starting from a given state $s_0 \in S$. Notice that the same state s of T may occur more than once during a probabilistic execution and each time a different distribution from s may be scheduled. In order to distinguish such occurrences we include in all states s of a probabilistic execution the past history of s which is the unique path leading from the start state to s . Thus, a probabilistic execution essentially defines a finite or infinite *tree*.

Definition 2. Probabilistic Execution *A probabilistic execution of a PTS $T = (S, \rightarrow_T)$ is a FPTS $F = (s - paths(T), \rightarrow_F)$ such that $(q \rightarrow_F \rho) \Leftrightarrow (\exists \pi : last(q) \rightarrow_T \pi \wedge \forall s \in S : \rho(qs) = \pi(s))$*

We denote by $execs(T, s_0)$ the set of all probabilistic execution of T rooted in s_0 .

4.2 Graphical interface for modeling the system

P_UPPAAL provides three tabs: Editor, Simulator y RAPTURE. The first tab, the Editor, is a graphical interface to draw the model. This graphical interface allows us to describe and draw the probabilistic transition system.

The Editor is divided in two main panels:

- *The left panel* allows us to describe the processes, the variables, the system and the templates, that will be part of the model. In **variables**, you can define integral variables and channels that provide the synchronization between the processes. In **process**, you can define the process from the different templates defined before. Finally, in **system**, you can describe the processes that run in parallel.
- *The right panel* allows us to draw the template as a PTS. Thus, we can draw states (initial or not) and transitions and its guards, synchronizations, assignments and probabilities.

The editor provides two modes to work, the selection mode and the transition mode. *The selection mode* allows us to create new states or erase it. Furthermore, it allows us to edit the state and transition properties; i.e, it edits the name of the state and it checks if the state is initial, and, it edits the transitions properties: guards, assignments, synchronizations between other processes through the channels and probabilities. *The mode transition* allows us to draw the relations that connect the states with transitions, but we have to change to the selection mode for add the transitions properties seen before.

In the figure 2 , we can see the tab Editor selected with a simple model. This model has been only designed with one process. First, it has three states: Idle, Start and End.

5 Systems Simulator

The P_UPPAAL tab **Simulator** provides a way to validate the system under study. We can check if the system behaviour is correct, i.e., if the system holds all the design requirements that we have collected previously.

To check the model syntax, P_UPPAAL uses a parser in JAVA. The simulation algorithm used by P_UPPAAL is as follows:

1. Checks the guards.
2. Store the enabled transitions, if enabled transitions = $\emptyset \Rightarrow$ Deadlock (The simulation end)
3. Choose the transition or transitions (if there are synchronization).
4. Execute transition $\Rightarrow$
 - Make the assignments.
 - Make the synchronization.
5. Return to the beginning.

As you can see in figure 6, the simulator panel is divided in four different panels: enabled transitions, simulator trace, variables and the system. The panel “Enabled Transitions” collects the transitions that are enabled now. The panel “Simulation Trace” shows the trace history, i.e., It shows the transitions that have been simulated before. The panel “Variables” shows all the systems variables and its current value. The last panel shows the running simulation.

Before running a simulation, you have to save the system modeled. After that, you can start the simulation. The first step is to choose the transition to run, it is made from the panel “Enabled transition”. Then you push Next button and the generated trace is stored in the trace panel.

6 The Verification

The verification process is realized through the RAPTURE tool. To connect with RAPTURE, P_UPPAAL provides the RAPTURE tab. This tab is divided in two parts. At the top, you can write the initial and final state, and the probability to reach the final state from the initial state and, at the bottom, you can show the RAPTURE output (figure 7).

6.1 RAPTURE

The context systems of RAPTURE are described in terms of Markov decision process [8], also called probabilistic transition systems (PTS) or probabilistic automata. This model allows to combine probabilistic and non-deterministic steps providing a natural extension to traditional non-deterministic models. The choice of this model is partly due to the fact that it is closed under parallel composition (which facilitates modeling and compositional reasoning), but primarily because PTSs are amenable to abstractions.

RAPTURE is focused on a restricted class of reachability properties. These properties allow to specify that the probability of reaching a particular condition ϕ_f from any reachable state satisfying a given initial condition ϕ_i is smaller (or greater) than a given probability p regardless of how non-deterministic choices of the model are resolved.

RAPTURE is based on automatic abstraction and refinement [4]. The basic idea is to use abstractions in order to reduce the high cost of the numerical analysis involved in computing the minimum and maximum reachability probabilities for PTSs. The abstractions considered are obtained via successive refinements, starting from an initial coarse partitioning of the state space derived from the property under study. For a given refinement the property is checked on the induced abstract model, hopefully settling the property. However, the verdict may be inconclusive, when threshold probability p happens to be between the calculated minimum and the maximum abstract probabilities. In this case, the abstraction is further refined and the property checked again. This process is successively repeated until either the property is settled, or no further refinement is possible.

6.2 Computing Extremum Probabilities

For a given rooted PTS (T, s_0) we are interested in the extremum probabilities of reaching some final condition from a given initial condition. For any given formula $\phi \in \mathcal{P}(AP)$ we define the set of all minimal simple paths of T that end in a state satisfying condition ϕ as:

$$\Sigma_\phi^T \triangleq \{\sigma \in s - \text{paths}(T) \mid \text{last}(\sigma) \models_T \wedge \forall i, 0 < i < |\sigma| : \sigma(i) \models_T \neg \phi\}$$

By recording history information in states, the above set characterizes uniquely a set of simple paths of probabilistic executions of T . We also use Σ_ϕ^T to denote this alternative characterization. It should be clear from the context which alternative is used. We omit T in the notation whenever clear from context.

Definition 3. Extremum Probabilities *The minimum and maximum probabilities of reaching a final condition ϕ_f from an initial condition ϕ_i in a rooted PTS (T, s_0) equipped with a proposition assignment are defined respectively by*

$$P_{T, s_0}^{inf}(\phi_i, \phi_f) \triangleq \inf\{P_{F, s_0}(\Sigma_{\phi_f}) \mid s \in \text{reach}(T, S_0) \wedge s \models \phi_i \wedge (F, s) \in \text{execs}(T, s)\} \quad (1)$$

$$P_{T, s_0}^{sup}(\phi_i, \phi_f) \triangleq \sup\{P_{F, s_0}(\Sigma_{\phi_f}) \mid s \in \text{reach}(T, S_0) \wedge s \models \phi_i \wedge (F, s) \in \text{execs}(T, s)\} \quad (2)$$

We talk of an extremum probability to refer to either the infimum or supremum probability.

We use the shorthand $P^{inf}(s)$ and $P^{sup}(s)$ for $P_{T, s_0}^{inf}(s = s_0, \phi_f)$ and $P_{T, s_0}^{sup}(s = s_0, \phi_f)$ respectively. We denote by I and F the sets of states satisfying ϕ_i and ϕ_f , respectively. Our aim is to efficiently compute $P^{inf}(I) \triangleq \inf_{s \in I} P^{inf}(s)$ and $P^{sup}(F) \triangleq \sup_{s \in F} P^{sup}(s)$.

The equations 1 and 2 of definition 3 define extremum probabilities, but do not provide an effective way of computing them. However, it is well known [1, 2] that P^{inf} and P^{sup} can be characterized as the least fixpoints of operators $F^{inf}, F^{sup} = (S \rightarrow [0, 1]) \rightarrow (S \rightarrow [0, 1])$ defined as follows. If $s \in F$ then $F^{inf}(f)(s) F^{sup}(f)(s) = 1$. If $s \notin F$ then

$$F^{inf}(f)(s) = \min_{s \rightarrow \pi} \sum_{s' \in S} \pi(s') \cdot f(s') \text{ and } F^{sup}(f)(s) = \max_{s \rightarrow \pi} \sum_{s' \in S} \pi(s') \cdot f(s') \quad (3)$$

Based on the above equations, two methods have been explored to compute $P^{inf}(s)$ and $P^{sup}(s)$. One can either compute the least fixpoints by iterative methods, or the equations can be transformed into a linear optimization problem that can be solved using classical techniques of linear programming. RAPTURE implements the linear programming method.

We use a standard precomputation of certain sets of system states in order to simplify the system before applying linear programming techniques. These sets are: the set of all reachable states $Reach$, and for each $p \in \{0, 1\}$ the set of states having infimum (resp. supremum) probability p of reaching ϕf . These latter sets of states are denoted $P_{=0}^{inf}$, $P_{=0}^{sup}$, $P_{=1}^{inf}$, and $P_{=1}^{sup}$ respectively. All of the above sets can be computed using discrete fixpoint analysis [5] on a boolean abstraction of the system.

Based on the above precomputations our linear programming problems for computing P^{inf} and P^{sup} become as follows:

maximize P^{inf} under the constraints

$$\begin{cases} P^{inf} \leq P^{inf}(s) & s \in I \\ P^{inf}(s) = 0, & s \in P_{=0}^{sup} \\ P^{inf}(s) = 1, & s \in (F \cup P_{=1}^{inf}) \\ P^{inf}(s) = \sum_{s' \in S} \cdot P^{sup}(s'), & s \rightarrow \pi, s \in S \setminus (P_{=0}^{sup} \cup P_{=1}^{inf} \cup F) \end{cases} \quad (4)$$

minimize P^{sup} under the constraints

$$\begin{cases} P^{sup} \geq P^{sup}(s) & s \in I \\ P^{sup}(s) = 0, & s \in P_{=0}^{sup} \\ P^{sup}(s) = 1, & s \in (F \cup P_{=1}^{sup}) \\ P^{sup}(s) = \sum_{s' \in S} \cdot P^{sup}(s'), & s \rightarrow \pi, s \in S \setminus (P_{=0}^{sup} \cup P_{=1}^{sup} \cup F) \end{cases} \quad (5)$$

7 An example

Communication through unreliable media Let us describe a small example. We will model a client and a server which communicate through two unreliable channels (a channel for each way). The client has to send a sequence of M different requests to the server. The server treats the request and send an acknowledgement. If there is a failure (we suppose that the client can detect it), the client can retry

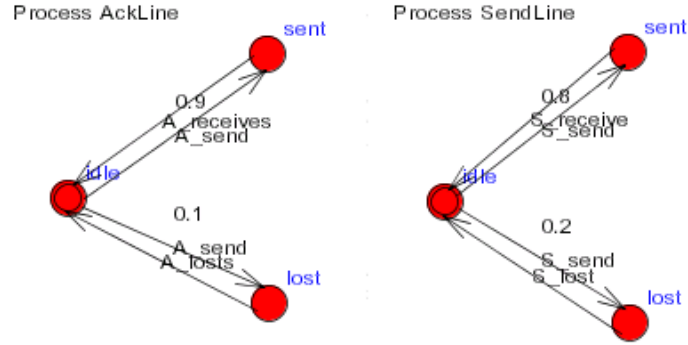


Figure 3: The two communication media: AckLine & SendLine

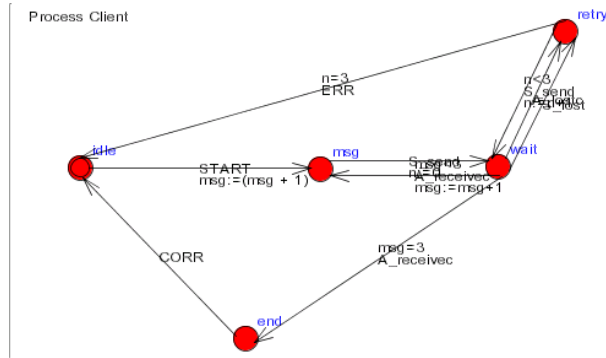


Figure 4: The client process

twice to send the request. On completion, the sender emit the message CORR, otherwise the message ERR will be emitted.

Global Declarations We first declare the set of channels:

channels: S_send, S_receive, S_lost, A_receive, A_receivec, A_lost, Alostc, START, CORR, ERR;

The two communication media: We define the SendLine and AckLine processes (see figure 3). The field synchronizations allows us to define the set of channels on which the process synchronize.

The Client process: The more complex process is the Client process that you can see in the figure 4. This process owns two local variables. Their type is `uint(2)`, which means: 2-bits unsigned integers.

The Server process: Last, we have the (dummy) Server process that is shown

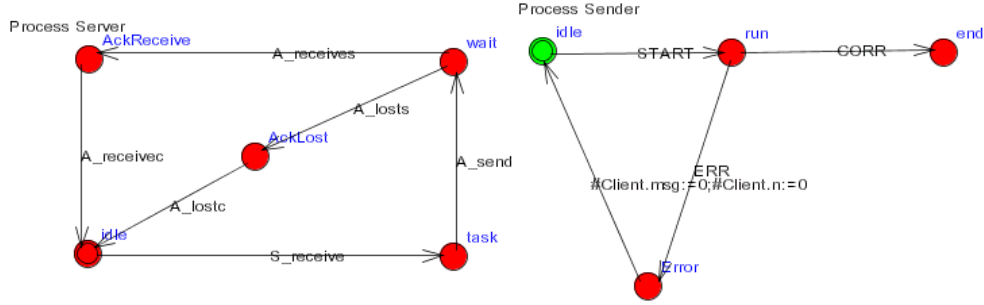


Figure 5: The server process and the sender process, respectively.

in the figure 5.

The Simulation: See figure 6.

The property and the verification: To check is the probability to send correctly one message. Let's add the process sender into the system (see the figure 5). To specify the property we set initial to “#Sender.idle” and final to “#Sender.end”

The output provides by RAPTURE can be seen in the figure 7. The property is false, because the $P^{sup} \leq 0.5$ on the stabilized partition of the system. As every member of this partition are bisimulation equivalence classes, we get that $P^{sup} = 0.975639513753$ on the concrete system.

8 Conclusions

UPPAAL and P_UPPAAL provide us two ways for studying system correctness. These ways are different from traditional bisimulation and testing. Both of them are based in the same idea, i.e, “*Verify the system*”. One of them treats the temporal behaviour and the other one treats the probabilistic behaviour respectively.

Thus, the *UPPAAL* tool treats system temporal behaviour. To make it, UPPAAL models are represented as networks of timed automata that allows us to introduce the time through the variables named clocks. Furthermore, the timed automata provide us the invariants and the guards that implements temporal requirements.

On the other hand, *P_UPPAAL* tool treats system probabilistic behaviour. As UPPAAL uses timed automata, P_UPPAAL uses probabilistic transitions systems (PTS) to represent the model. And while UPPAAL uses invariants and guards, P_UPPAAL adds a new transition field called probability that represents the prob-

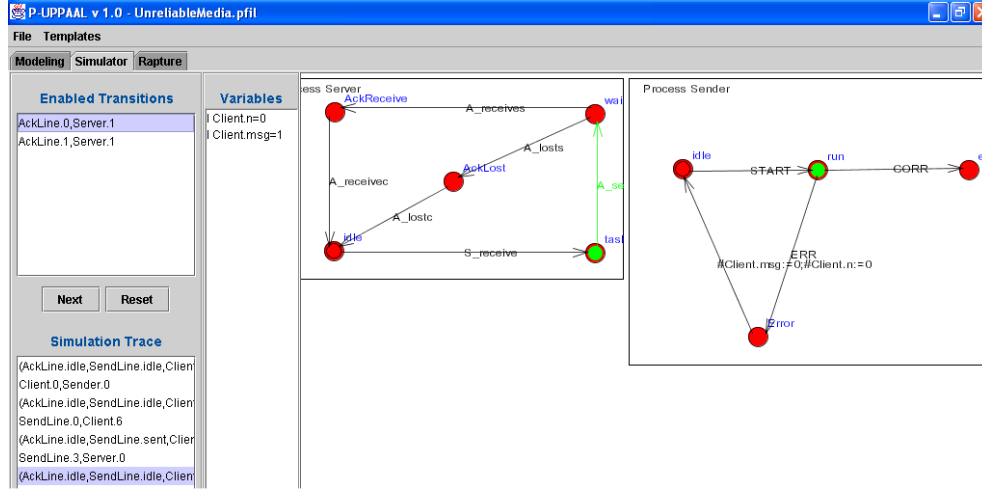


Figure 6: The system simulation

ability to make this transition.

Both of them, UPPAAL and P-UPPAAL provide a way to validate the system. In both, you can simulate the systems under study.

Finally, both UPPAAL and P-UPPAAL can verify the system. UPPAAL uses a symbolic model checking based on reachability analysis by constraint solving that verify safety properties for example. However, P-UPPAAL, through RAPTURE, uses a reachability analysis based on probabilistic simulations over an abstraction of the system that reduces the cost of the numerical analysis. This analysis calculates the extremum probabilities (minimum and maximum) to reach a final condition from a reachable state that holds a initial condition.

In short, P-UPPAAL shows a different way to automatic verification for other kind of properties.

Other main characteristic of P-UPPAAL is that P-UPPAAL is able to import UPPAAL models. It is a important goal in this work, because it allows us to capture the probabilistic behaviour of Real Time Systems. It means that P-UPPAAL can connect with UPPAAL to expand UPPAAL features.

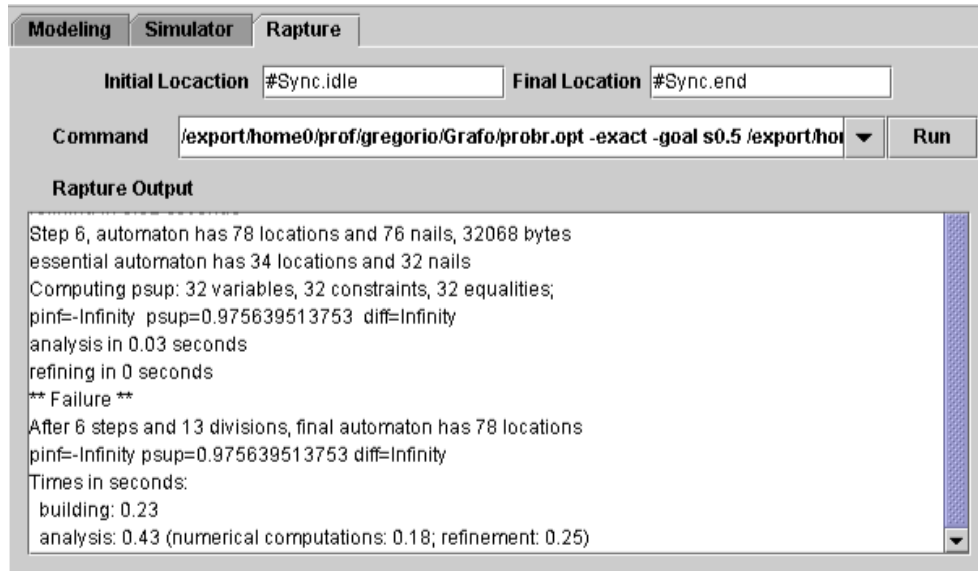


Figure 7: The verifier verdict

9 Future Research

Although, P_UPPAAL is in a high degree of development, there are a several points that have to be discussed into detail. For example, P_UPPAAL models can be exported to UPPAAL models. This idea is similar to the developed import process. Thus, we might use the same principle: “PTS and Timed Automate are derived from labeled transition systems”. Furthermore, if we extend this idea, we may generate a label transition system that may be exported to all its derived classes.

Thus, because of the high degree of development, it is in time to prove the system. To prove the system, we must introduce use cases. We are interested in implementing the video standard MPEG-2 and Ghost Packet algorithm.

The reason for choosing MPEG-2 is due the fact that this protocol has been widely studied, as pointed out in [9, 10]. The basis of MPEG-2 is that it addresses the combining of one or more elementary streams of video and audio, as well as, other data into single or multiple streams which are suitable for storage or transmission.

Ghost Packet algorithm is a new starvation and deadlock free routing algorithm [3].

Much work is still left, but an other step has been made towards capturing system behaviours and its verification.

References

- [1] Christel Baier, *On algorithmic verification methods for probabilistic systems*, Ph.D. thesis, Faculty for Mathematics and Informatics, University of Mannheim, 1998.
- [2] A. Bianco and L. de Alfaro, *Model checking of probabilistic and nondeterministic and non-deterministic systems*, LNCS, no. 1026, 1995.
- [3] M. C. Carrión, G. Díaz, and B. Caminero, *Performance issues of deterministic and adaptive ghost-packet routers*, Proceeding of the 2001 International Conference on Parallel Processing (Valencia, Spain), IEEE Computer Society, 3–7 September 2001, pp. 33–40.
- [4] Pedro R. D’Argenio, Bertrand Jeannet, Henrik E. Jensen, and Kim G. Larsen, *Reachability analysis of probabilistic systems by successive refinements*, PAPM-PROBM 2001 (Aachen (Germany)), LNCS, vol. 2165, September 2001.
- [5] Luca de Alfaro, *Formal verification of probabilistic systems*, Phd thesis, Stanford University, 1997.
- [6] Bertrand Jeannet, Pedro R. D’Argenio, and Kim G. Larsen, *RAPTURE: A tool for verifying markov decision processes*, Tools Day, International Conference on Concurrency Theory, CONCUR’02 (Brno (Czech Republic)), August 2002, Technical Report, Faculty of Informatics at Masaryk University Brno.
- [7] Bengt Jonsson, Kim G. Larsen, and Wang Yi, *Probabilistic process algebra*, E-HPA, 2001, pp. 685–710.
- [8] M.L. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*, Willey series in probability and mathematical statistics, Jhon Wiley and Sons, 1994.
- [9] Valentín Valero, Fernando L. Pelayo, Fernando Cuartero, and Diego Cazorla, *Specification and Analysis of the MPEG-2 Encoder with Timed-Arc Petri Nets*, Electronic Notes in Theoretical Computer Science **66** (2002), no. 2.
- [10] Valentín Valero, Fernando L. Pelayo, Fernando Cuartero, and Diego Cazorla, *Analysis of the MPEG-2 Encoder Algorithm with Timed-Arc Petri Nets*, Actas de las X Jornadas de Concurrency (Jaca, Spain), Editorial Kronos, June 2002, pp. 71–86.