Abstract: In the paper the idea of timed automata simulator is introduced. The proposed simulator is an extension of the UPPAAL tool simulator. Instead of choosing transition to be performed randomly or manually, the simulator calculates transition times according to transition guard and state invariant for each transition. A probability density function, which is needed for such calculation, is formally added to each edge of the automaton. Described procedure can reveal realistic behavior of the system modeled by the automaton. Within the paper the concept of a state-space tracing is shown on a case-study.

Key-Words: Timed automata, Probabilistic simulation, Simulator, UPPAAL PRO, UPPAAL extension

1 Introduction
System simulation is in present the very basic requirement during system modeling. The simulation is the first step to prove behavior the system during and after modeling. Although it is not possible to surely reveal incorrect behavior by simulation (unlike by system verification), it is simple method to get the idea about the modeled system.

UPPAAL is a tool box for modeling, simulation and verification of real-time systems developed jointly by Uppsala University and Aalborg University [1]. UPPAAL is appropriate for modeling systems that can be described as a collection of stochastic timed automata. UPPAAL consists of three main parts: a description language, a simulator and a model checker. The description language is non-deterministic guarded command language with data types (currently only integer and clock data types are implemented). The simulator and the model-checker are designed for interactive and automated analysis of system behavior. The simulator enables examination of possible dynamic executions of a system during modeling stages and thus provides an inexpensive mean of fault detection prior to verification by the model-checker. During system simulation in UPPAAL is in each state calculated set of feasible transitions to future states. There are two possible ways of simulation. The user can either affect choosing of some particular transition or simulator could be left to choose one of the possible transitions randomly.

In this paper possible extension of UPPAAL simulator is presented. Instead of random choosing from the feasible transitions set we propose implementation of the algorithm which from known transitions properties and present clocks values generates randomly the times when transitions should be performed and chooses the fastest transition. The simulator then generates the sample trace in the system state space. We start with basic definition of timed automata syntax and semantics to clarify our notion. Further we briefly describe logic structure of the simulator and system data hierarchy. In the third section we show the example describing the basic problems.

2 Present state of simulation
At present there exist many tools for modeling systems using nets of state automata. At least two should be specifically named. The SPIN tool, using the Promela language, is designed mainly for system verification [2]. However the tool allows random, guided or interactive simulation of the modeled system as well. KRONOS is a tool developed to verify real-time systems modeled by timed automata [3]. Several tiny tools for simulating state automata should not be omitted, but these tools concerns only about deterministic non-timed state automata, e.g. büchi automata. Inspired by this lack of research we decided to develop the system based on the UPPAAL tool.

Several UPPAAL modifications were developed; among them there are two most interesting to meet the target. First of them, UPPAAL PRO, allows user to define purely probabilistic transitions as well as standard timed ones [7]. The problem is solved by adding the possibility to insert completely new transition type to the system.
templates. The probability of such a transition to be taken is defined as its weight. At present, these weights are not included to the UPPAAL PRO simulator algorithm. During the simulation in the each step the set of enabled transition is created. The transition to be performed is chosen non-deterministically from this set by the manner that all transitions have the same probability.

Another modification - UPPAAL TIGA, intended for timed games solving, implements the different concept of the simulator [8]. Instead of working with states in a symbolic way, it adds specific transition times to them. Therefore states are reduced only to specific locations independently of span of their feasibility times. During the simulation the user can choose not only transition to be taken but also specific time when the transition is taken. When the transition is performed, this time is added to all clocks and simulation can proceed.

For our purpose the concrete simulator concept as well as the ability to define probabilistic transitions is beneficial. In the following the shortened definition of the stochastic timed automata syntax and semantics is clarified. Moreover a syntax extension needs to be elaborated. For each traditional timed transition there has to be added an extra piece of information defining the probability density function, according to which the transition time till be determined. For the basic step we take into account basic types of probability density functions like the uniform or exponential. However, special kinds of probability density functions may be needed in reality. Besides, a concept of pure probabilistic transition might be needed as well. The reason for such assumption is described in detail in the section 4.

The paper is segmented as follows. In the section 3 a definition of stochastic timed automata syntax and semantics is shown. Moreover a proposed extension is defined. In the section 4 a basic simulation algorithm is described as both a pseudo-code with an explanation or as a simulation scheme. In the section 5 a case study of assumed behavior of the simulator is presented.

3 Basic definitions

3.1 Transition system, timed state automaton

Transition system is a tuple \((S, s_0, \Sigma, A)\), where \(S\) is a set of automaton states, \(\Sigma\) is finite set of labels, \(\Sigma \subseteq S \times A \times S\) is set of transitions and \(s_0\) is initial state \([4]\). For a transition \((q, a, q') \in \Sigma\) we write \(q \xrightarrow{a} q'\). that means system can change its state from \(q\) to \(q'\) on event \(a\). Automaton starts in the initial state \(s_0 \in S\).

Let \(X = \{x_1, x_2, ..., x_n\}, x_i \in \mathbb{R}, \forall i \in \{1, 2, ..., m\}\) be the set of clocks and let \(\Psi = \{\psi_1, \psi_2, ..., \psi_k\}\) be the set of clocks constraints over clock set \(X\) \([4]\). The clock constraint is a conjunction of atomic constraints. Atomic constraints compare clock values with time constants.

Clock constraint \(\psi\) can be defined by the grammar (1).

\[
\psi ::= x < c \mid x - y < c \mid c < x \mid x < y \mid \psi_1 \land \psi_2 \mid \neg \psi
\]

where \(x, y \in X, c \in \mathbb{Z}, \forall c \in \{<, \leq\}\)

(1)

Particular terms, originally related by conjunction are separated by commas.

A clock interpretation \(v \in V\) for a set \(X\) of clocks assigns real values to each clock. We say that clock interpretation \(v\) satisfies \(\psi\) and we write \(\psi \models v\), if \(\psi(v)\) is satisfied (acquire true value).

For \(\delta \in \mathbb{R}\): \(v + \delta\) denotes the clock interpretation for \(X\) which maps every clock to the value \(v(x) + \delta\).

For \(Z \subseteq X\): \(v(Z := 0)\) denotes the clock interpretation for \(X\) which assigns all clocks from \(Z\) the value 0. This action is equivalent to reset of the clocks. Clock constraints are evaluated over clock valuations. An clock interpretation \(v\) is said to satisfy the clock constraints \(\psi \in \Psi\), denoted \(v \models \psi\), if

\[
\begin{align*}
&v \models x < c & &\text{iff } v(x) < c \\
&v \models x - y < c & &\text{iff } v(x) - v(y) < c \\
&v \models \psi \land \psi' & &\text{iff } v \models \psi \text{ and } v \models \psi' \\
&v \models \neg \psi & &\text{iff } v \not\models \psi
\end{align*}
\]

(2)

Let \(G : G\{g_1, g_2, ..., g_n\}, \forall j = (1, n) : g_j = \Gamma(E(j))\) be the set of transition clocks constraints (guard) and function \(\Gamma : E \rightarrow \psi\), that maps transitions to clock constraints set.

Let \(I : I\{i_1, i_2, ..., i_m\}, \forall j = (1, n) : i_j = \Phi(S(j))\) be the set of clocks constraints (invariant) and function \(\Phi : S \rightarrow \psi\), that maps states to clock constraints set. In case of the invariant definition is the above mentioned rule restricted to (3).

\[
\psi := x < c \mid x - y < c | \psi_1 \land \psi_2 | \neg \psi
\]

(3)

Let \(R : R\{R_1, R_2, ..., R_n\}, \forall i = (1, j) : R_i = \{E(i) \rightarrow 2^X\}\) be a set of clocks to reset. For \(R_i \subseteq X\) and \(v, v' \in V\) let us define function \(v(R_0 = 0)\) as follows:

\[
\forall \chi \in R_i : v'(\chi) = 0, \forall \chi \notin R_i : v'(\chi) = v(\chi), \text{ where } v'(\chi) \text{ is value of } \chi \text{ in next state.}
\]

Timed transition relation is then \(E \subseteq S \times \Sigma \times G \times R \times S\).

A switch \((s, a, g, r, s')\) represents transition from state \(s\) to state \(s'\) on symbol \(a\). \(g\) is clock constraint over \(X\).
that specifies when the transition is enabled and \( r \) is set of clocks to be reset with this transition.

Timed automaton is a tuple \((S,s_0,E,A,X,G,R,I)\).

### 3.2 Extended timed automaton

For our purpose we need to extend the timed automaton definition by a probabilistic part. We propose to define the probability density function for each possible transition. Let \( F \) be a set of probability density functions over the time domain and \( E \) the probabilistic timed transition relation \( E \subseteq S \times \Sigma \times G \times F \times R \times S \). We can write \( s \xrightarrow{a,g,f,r,s'} \in E \).

The extended timed automaton is then a tuple \((S,s_0,E,F,A,X,G,R,I)\). Each transition can be extended by the probability density function. In case of two transitions sharing one label \( a \), i.e., in case of transition synchronization by labels \( \text{act!} \) and \( \text{act?} \), the probability density function is allowed to be defined only on one of these transitions, typically on the \( \text{act!} \) (receiving) side.

### 3.3 Timed automaton semantics [5, 6]

The state of timed automaton is a pair \( \langle s, v \rangle \) such that \( s \) is a location of automaton and \( v \) is a clock valuation that satisfies the invariant \( I(s) \). State \( \langle s, v \rangle \) is an initial state if \( s \) is an initial location of automaton and \( \forall x \in X : v(x) = 0 \). There are two types of transitions – transition due to elapse of time and transition due to location change. These two transitions are introduced below. Let us define relations

\[
\xrightarrow{a} \subseteq (S \times \mathbb{R}^m) \times (S \times \mathbb{R}^m)
\]

and

\[
\xrightarrow{1} \subseteq (S \times \mathbb{R}^m) \times (S \times \mathbb{R}^m), \text{ where } a \in A, \delta \in R, \text{ for describing these two states.}
\]

- State can change due to elapse of the time (4). Rule defines that system is allowed to idle in the location \( s \) for \( I \) time units if within this time no deadline force the execution of the transition.

\[
\forall t' \leq t : (v + t') = I(s) \frac{\langle s, v \rangle}{\xrightarrow{1} \langle s, v + t \rangle} (4)
\]

- State can change due to the location change (5). Rule defines that system is allowed to switch to location \( s' \) if there is transition from current state \( \frac{s}{\epsilon} \rightarrow s' \) and guard condition along this transition is satisfied.

\[
\epsilon \in E \wedge G(\epsilon) = v \wedge v' = v[R(\epsilon) = 0] \frac{\langle s, v \rangle}{\xrightarrow{a} \langle s', v' \rangle} (5)
\]

Automaton is thus a transition system \( S \times \mathbb{R}^m \) with label set \( \Sigma \cup \mathbb{R} \). For example state space for the timed automaton is \( \{s_0,s_1,s_2,s_3\} \times \mathbb{R}^2 \) and label set is \( \{a,b\} \cup \mathbb{R} \). Clocks \( x_1 \) are reset each time automaton reaches states whereas clocks \( x_2 \) are never reset. Sample path through state space could be for example:

\[
(s_0,0.0) \xrightarrow{1} (s_0,1.5,1.5) \xrightarrow{1} (s_0,0.1,5) \xrightarrow{1} (s_0,0.8,2.3) \xrightarrow{1} (s_2,0.2,3) \xrightarrow{1} \text{etc}.
\]

### 4 Simulation

#### 4.1 The simulation algorithm

The following pseudo-code introduces the simulation algorithm. Particular functions explanation is provided below.

```plaintext
1 while (not (stop)) {
2   Enabled ← Succesors (CurrentState)
3   for each \( \epsilon = \langle a, g, r, f \rangle \) in Enabled do {
4      create \( \Delta = g \otimes I(\text{CurrentState}) \)
5      create \( H(t) : \Delta \rightarrow \int f(t) \)
6      Value = Random \( H^{-1} \)
7      if Value = min \( \text{Value} \) (ResultSet) then
8         ResultSet ← \( \epsilon, \text{Value} \)
9      else Value < min \( \text{Value} \) (ResultSet) then
10         ResultSet = \( \epsilon, \text{Value} \)
11     }
12     if |ResultSet| = 1 then {
13        CurrentState = TakeTransition \( \epsilon, \text{Value} \)
14     else
15        \( \Theta(w) : \sum \omega_j \rightarrow \mathbb{N} \)
16        \( i = \text{Random} (\Theta^{-1}) \)
17        CurrentState = TakeTransition \( \epsilon, \text{Value} \)
18    }
19}
```

The whole simulation runs inside a cycle. At the beginning the UPPAAL core is called #2 and all present state successors are therefore found. At the end the UPPAAL core is called and informed about the transition to be taken #13 or #17. The function #4 creates an interval between lower bound of the specific transition guard \( g \) and upper bound of the current state invariant \( I \). The mapping function #5 calculates the conditional distribution function from the density function \( f(t) \). This function is mapped to the domain \( \Delta \). Consequently the new function is named \( H(t) \). The mapping is shown in
the figure 1a. The first one describes mapping of probability density and distribution functions in case of uniform distribution. Here no additional parameter is needed – function parameters are calculated according to the range of the function domain as 
\[ \frac{dy}{dx} = \frac{1}{\max(\Delta) - \min(\Delta)}. \]
Note that this type of distribution can be used only in case of bounded function domain. Probability density and distribution functions, for the case that the exponential distribution is used, are shown in the figure 1b. Due to the memoryless property of this distribution, the shape of the function is not changed after mapping. Obviously the exponential distribution can be used only in case of unbounded interval.

Random (bounds) functions #6 and #16 generate the random value according to the uniform distribution inside bounds defined by the function parameter. The syntax of lines #6 or #16 shows that the random value may be generated by the inverse function method. Functions \( \min_{\text{value}} \left( \text{set of } \langle \text{transition, time} \rangle \right) \) at lines #7 and #9 return couple \( \langle \text{transition, time} \rangle \) with the least value of time among all elements in the set.

The Mapping function \( \Theta(w) : \sum w_i \rightarrow \mathbb{Z} \) at line #15 is used in case that more than one transition needs to be performed in one time instant. The input to the function is the set of values of probability density functions of all involved transitions \( w_i = f_i(k) \) in the time when these transitions need to be performed. The function thus allows simple decision among several probabilistic transitions. The formal definition of this function is written in (6).

\[ \forall w_i, k = \left( \sum_{j=0}^{i} w_j \right) : \Theta(k) = i \quad (6) \]

Each weight function value \( (w_1, w_2, etc...) \) is taken and mapped to the domain of the final function with constant value \( (1, 2, etc...) \). The occurrence of the deadlock is covered by the highest value of \( k \) in this function. The decision process then consists only of generating the random value with uniform distribution within the domain and calculating the value of the function \( \Theta \).

**4.2 The simulation scheme**

The simulation scheme shown below describes necessary dependences between specific simulator function blocks. The scheme is closely related to the simulation algorithm shown in the section 4.1. The scheme shows the process of passing information between specific functions. Functions that communicate with UPPAL core are shaded. Dashed lines describe renewing default values during initialization prior running a new simulation round.

![Fig. 2 – the simulation scheme](image)

**4.3 Implementation to the UPPAL tool**

The described simulation algorithm is intended to be an extension of the UPPAL simulator and for this reason is being implemented directly as a part of it. The intention is to enhance the graphical user interface (GUI) of the simulator tab so it shows specific transitions properties and proposed transition times as well as detailed statistics.
5 Case study

5.1 Automaton without weights

Let us suppose two automata as depicted in the Figure 3. The initial state of this system is the couple \([INIT1, INIT2]\). The system can choose its state either to \([INIT1, GUESS2]\) or to \([INIT2, GUESS1]\).

In the following paragraphs one of possible runs of this system is described. At first let us choose the transition to the state \([GUESS1, INIT2]\) at the global time 0.5 time units. Notice that due to the \(INIT1\) state invariant condition, the system cannot stay in the init state longer than 2 time units. For the second transition \([GUESS1, GUESS2]\) can be chosen.

![Fig. 3 – System containing two state automata](image)

<table>
<thead>
<tr>
<th>Transition</th>
<th>Before transition</th>
<th>After transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT1→GUESS1</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>INIT2→GUESS2</td>
<td>1.5</td>
<td>2</td>
</tr>
</tbody>
</table>

The system reached the state \([GUESS1, GUESS2]\) and now three possible transitions can be taken. The clocks vector has now the value \(<1.5, 2>\). Due to the state invariant, the transition to the \([DN1, GUESS2]\) can be taken within the time interval \((1.5, 2.5)\) time units. In the similar way we can deduce the others time intervals. On the following picture the present system state and graphical representation of time intervals are shown.

![Fig. 4 – State space of the system with 2 clocks](image)

For the second choice the distribution is of the exponential type, thus \(f(t) = \exp(-\lambda \cdot t)\). After mapping, the resulting function is \(H(t) = 1 - \exp(-\lambda (t - 1.5))\) within the domain \(t \in (1.5, 4)\). Particular clocks values can be calculated in the similar fashion it was shown previously. Repeating this procedure for all possible transitions leads to evaluation of all conditional distribution functions. In the next step random values according to these distribution functions has to be found. This can be done either by the inverse function method in case one can found such a inverse function or by any other known method. These random values describe transitions times and the transition with the least computed value is the one that has to be performed.

5.2 Weights concept

It can happen two or more transitions could be performed at one time instant. This is obviously not the situation from the section 5.1 (the probability that two

The function \(H(t)\) for first choice \((GUESS1\rightarrow DN1)\) can be calculated according to the mapping \(H(t) : \Delta \rightarrow \int f(t)\), thus if an original probability density function has an constant value in the whole function domain (which depends on the width of its domain), the resulting function is \(H(t) = t\) within the domain \(t \in (0, 1)\). Notice that \(t\) means the global time and particular clocks values can be calculated in this case as \(x = t + 3\) and \(y = t + 3.5\).
transitions occurs at the same time is equal to zero and this should be taken into an account). However, suppose system of one timed automaton as shown in the Figure 5. Both transitions from the state WAIT must occur at the time equal 4 units. If no specific condition is added, one possible solution can be simple decision between these two transitions, both with the probability 0.5. Moreover, this is the exact case of pure probabilistic transitions, which can be defined in the UPPAAL PRO tool. Some additional information has to added to determine specific transitions probabilities. As shown on the Figure 5 there are two numbers – one at each transition – describing these probabilities. When the system reaches the state WAIT and 4 time units pass, the decision process is started and transition \( \text{WAIT} \rightarrow \text{UP} \) respectively \( \text{WAIT} \rightarrow \text{DOWN} \) is taken with the probability 0.25 respectively 0.7. Reader can immediately see that there is chance for deadlock to occur with probability 0.05.

From the practical point of view, mentioned weights can be defined as part of the transition’s probability density function as follows:

\[
    f_{\text{WAIT} \rightarrow \text{UP}}(t) = \begin{cases} 
        0.25 & t = 4 \\
        0 & \text{otherwise}
    \end{cases},
\]

respectively

\[
    f_{\text{WAIT} \rightarrow \text{UP}}(t) = \begin{cases} 
        0.7 & t = 4 \\
        0 & \text{otherwise}
    \end{cases}.
\]

The mapping function shown at section 4.1 (pseudo-code #15) creates the weight function (7).

\[
    \Theta(k) = \begin{cases} 
        0 & k \in [0, 0.25) \\
        1 & k \in [0.25, 0.95) \\
        2 & k \in [0.95, 1]
    \end{cases} \quad (7)
\]

6 Conclusion

In this paper a proposed extension of the UPPAAL tool has been introduced. The main contribution of this extension is the implementation of the automatic probabilistic simulator. Instead of tracing the state space randomly, the firing transition can be calculated from specific state and transition properties. To specify the transition completely, the extension of the stochastic timed automaton was proposed. For each unique transition label a probability density function is defined. During simulation, this function is mapped to the interval, where the transition is enabled, and then an exact firing time can be determined. Among all enabled transition the winner is that one with the minimal firing time.

With the proposed language extension the definition of stochastic timed automata as well as the Markov Processes or Markov Chains is possible. During the simulation the most interesting information can be served to the user through the graphical user interface, thus the simulation can become more transparent. In the automatic mode the simulator can stochastically determine the steady-state probabilities as well as time spent in specific states or number of visits by the multiple run of the same simulation.

7 Acknowledgement

This work has been supported in part by Ministry of Education, Youth and Sports of the Czech Republic (Research Intent MSM0021630503 New Trends in Microelectronic Systems and Nanotechnologies (MIKROSYN), Grant Agency of the Czech Republic (102/09/H081 SYNERGY – Mobile Sensoric Systems and Network) and by Brno University of Technology. Without kind support of the above-mentioned agencies and institutions the presented research and development would not be possible.

References: