1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07)
ENS Cachan, France - June 13-15, 2007

PROBABILISTIC TIMED AUTOMATA FOR MODELING NETWORKED AUTOMATION SYSTEMS

Jürgen Greifeneder and Georg Frey

University of Kaiserslautern; Erwin-Schrödinger-Str. 12; 67663 Kaiserslautern; Germany
{greifeneder|frey}@eit.uni-kl.de

Abstract: For the formal analysis of dependability of Networked Automation Systems (NAS) it is necessary to model the whole system first. Due to the probabilistic and distributed nature of the problem, a modular automata based approach is preferable for modeling and analysis. In this paper a formal automata definition for the specific needs of modeling NAS is given including continuous density distributions. For this model a discretization procedure as well as a transformation of the resulting discrete model into the input language of the probabilistic model checking software PRISM is presented. Finally an example is given, to illustrate the approach. The results of the automata based analysis are compared to measurements. *Copyright © 2007 IFAC*

Keywords: Networked Automation Systems, Formal Verification, Modeling, Automata

1. INTRODUCTION

Modern automation systems consist of one or more controllers, several sensors as well as actuators and a network, connecting all these components using network-IOs, switches, ... (Fig. 1). This configuration – commonly called Networked Automation Systems (NAS) – has several advantages, like sharing resources (sensors, cables), the opportunity of cross accessibility and the possibility to spatially separate controllers from one another (on the one hand) as well as from the process (on the other hand). Using Ethernet leads to even further advantages like steadily decreasing hardware prices, enduring improvements in quality and amount of the offered technologies and the possibility to negotiate barriers in communication systems.

Networking, together with the decentralized and cycle based computation integrates the areas of control, communication and computation (C^3-Systems, e.g. [Report, 2000]) towards a system structure, which exhibits the superposition of constant and cyclic delays. Components' stochastic failure rates lead to further delays. A model representation to be used for those systems therefore must be able to represent times, stochastic distributions and deterministic sequences. Time thereby is twice important: First, it is used as an indispensable input variable and second, it is – besides of the pure functional analysis – the most important aspect for analyzing dependability. To address the latter one, a method must be developed, which can process the NAS' model structures and return time- as well as probability based answers.

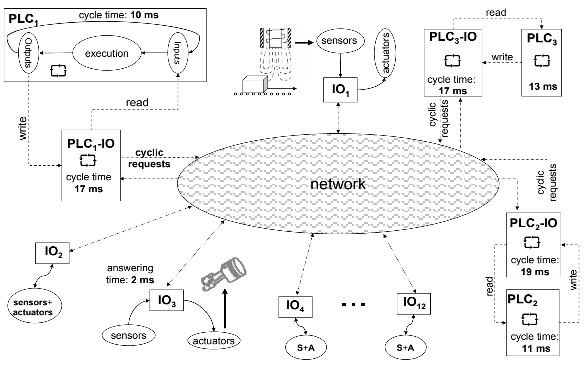
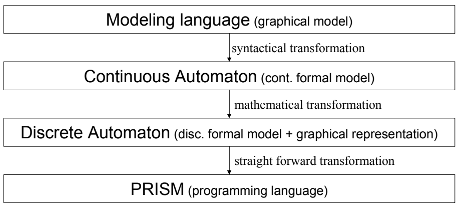


Fig. 1: Networked Automation System (NAS)

The rest of the paper is structured as follows. In the next section a new type of probabilistic timed automata (PTA), created for the special needs in NAS, is introduced. For this model a discretization procedure as well as a transformation of the resulting discrete model into the input language of the probabilistic model checking (PMC, [Katoen, 2006]) software PRISM [Kwiatkowska et al., 2002] is presented. In section 3 an example is given, to illustrate the approach. The results of the PMC based analysis are compared to measurements. The last section gives a summary of the paper.

2. MODELING APPROACH

In order to find a model representation which covers all possible behaviors of a NAS, it is important to first define a description language, which reflects the engineers structural thinking. In a second step (cf. section 2.2) this will be transformed towards an automaton model related to PTA. Due to computation and memory limitations, this model can not be used directly, but must be discretized, to be used in Probabilistic Model Checking. This is done in the sections 2.3 and 2.4. Section 2.5 presents two examples for defining the initial state. Finally the transformation into the used PMC-language PRISM is given in section 2.6. This design process is illustrated in Fig. 2.


Modeling language (graphical model)
syntactical transformation
Continuous Automaton (cont. formal model)
mathematical transformation
Discrete Automaton (disc. formal model + graphical representation)
straight forward transformation
PRISM (programming language)
Fig. 2: Design Process

2.1. *Description Language*

In NAS the activation of a transition between two distinct system states can be given in four basic ways (cf. Fig. 3):

(1) A fixed time interval after which's elapsing the transition is activated, e.g. a fixed processing time. Each state is allowed to have one and only one outbound transition.

(2) A density distribution over time d(x), which gives the probability for activation at a given time, e.g. the transmission time for passing the network. As in the first case, there is only one outbound transition allowed: It does not make sense to have two density functions active at the same time. Note: A condition of type (1) can be easily written as $d(x)=\delta(t_e)$, where δ means the Kronecker-Symbol (Dirac-Impulse) and t_e represents the activation time. As it is not very convenient for formulating conditions, this replacement is done later in the transformation step, automatically.

(3) The activation is coupled to the occurrence of a predefined situation (this can be understood as event, even it is written as the occurrence of a condition based on states), e.g. an IO sends a request or the PLC reads/writes values. There are no two conditions active at the same time

(4) The activation occurs immediately but the choice of path is of stochastic nature, e.g. the transmission will fail with a probability of 1%. The sum over all probabilities obviously must add up to one. Contrary to the types (1) to (3), the corresponding state is of transient nature, i.e. no time passes while being in this state. Therefore transition type (4) could be attached to each transition directly (cf. Fig. 5).

Despite for type (4) there is no need to have an activated transition at any time, but there must be at least one transition, which may become active. Besides the possibility to arrange those basic types arbitrarily in series, it is possible to use a mix of transitions of type (3) together with type (2), respectively (1) (the mix of (1) and (2) is not allowed; same for any arbitrary mix together with (4)).

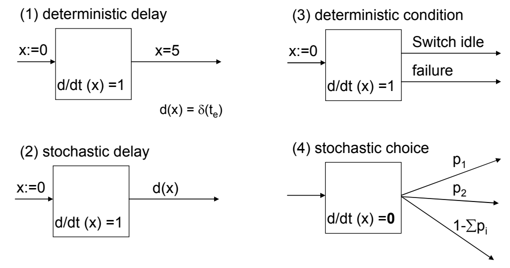


Fig. 3: Basic transition types

As an example for an possible mix of transitions the automaton on the left hand side of Fig. 4 shows a machine, which relocates from idle to working when the "start"-button is pressed. The working process takes 5 ms and can be interrupted by pressing the "stop"-button. The latter transition therefore is labeled by a disjunction of transition type (1): t=5 and transition type (3): stop.

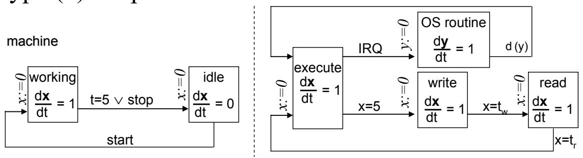


Fig. 4: Examples for states with mixed transition types

The automaton on the right hand side of Fig. 4 shows a processor which first reads in some values, what takes t_r ms, executes them, what takes 5 ms, and finally writes out the results (t_w ms). While the processor is in the execution state, an interrupt IRQ from the Operating System may arrive, which interrupts the execution and starts to run the corresponding OS routine. The time needed to finish this routine is given by a distribution over time y. After finishing, the process will be continued where it got interrupted (clock x was not changed by the OS routine).

The most general transition type therefore may assess (I) a condition which is disjuncted with a density distribution and (II) a probability vector (evaluating to a set of successive states). Note: This resulting automaton (cf. Fig. 5) is not able to choose in between different state-conditions and activate a corresponding density function!

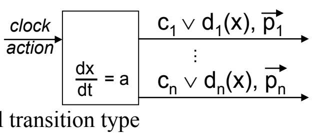


Fig. 5: General transition type

2.2. *Continuous Automaton*

To formalize the graphical model, a continuous automaton is used given by the following notation

$$\mathbf{A = (Z, Prob(Z(t{=}0)), X, X^0, X^{act}, X^R, D(X), L)}$$

and defined over the set of sub-automata A_i (i=1, .., n) given by

$$\mathbf{A_i = (Z_i, Prob(Z_i(t{=}0)), X_i, X_i^0, X_i^{act}, X_i^R, W_i, V_i, C_i(V_i), D_i(X_i), L_i)}$$

Z_i is the set of states, $\mathrm{Prob}(Z_i(t=0))$: $Z_i \rightarrow [0,1]$ the initial state distribution, X_i is the set of clock variables, X_i^0: $X_i(t=0) \rightarrow \mathbb{R}^+_0$ the set of initial clock values, X_i^{act}: $Z_i \rightarrow 2^{Xi}$ the clocks' activation function and X_i^R the set of clocks' reset function. $W_i \subseteq Z_i$ is the output alphabet and V_i the input alphabet given by:

$$V_i \subseteq \bigcup_{\substack{j=1 \\ j \neq i}}^{n} W_j \tag{1}$$

Using this definition, the sub-automaton can react on the occurrence of a predefined situation (deterministic condition as discussed in section 2.1, case (3)) in the complete system. This dependency (extended by $\varnothing$) is described by the set of Boolean functions named $C_i(V_i)$ over the input alphabet. $D_i(X_i)$ is the set of probability density functions, which is based on the set of clocks $D_i(X_i)$: $d_i(x) \rightarrow \mathbb{R}^+_0$; and necessary to model the transition types (1) and (2), respectively. Finally, the transition relation is given by L_i: $Z_i \times D_i(X_i) \times C_i(V_i) \times Z_i \rightarrow [0,1]$ and represents the description of the general transition type shown in Fig. 5, where the probability is derived from attaching a transition of type (4). As the validities of $C_i(V_i)$ and $D_i(X_i)$ can be understood as independent stochastic events, $D_i(X_i) \times C_i(V_i)$ builds a single probability space. Prove: Consider a pure stochastic delay distribution $d_k(x)$, cf. Fig. 3. Then the following must hold:

$$\int_{t=0}^{\infty} d_k(x)dx = 1 \tag{2}$$

Consequently, for a combination of C (c_{k1}, ..., c_{km}) and D (d_{k1}, ..., d_{km}), the following must hold:

$$\sum_{j=1}^{m} \left(\int_{t=0}^{\infty} \left\{ c_{kj} \vee d_{kj}(x) \right\} dx \right) = 1 \tag{3}$$

As $D_i(X_i) \times C_i(V_i)$ is assumed to build a single probability space, it must be possible to convert the notation $c_{kj} \vee d_{kj}(x)$ into a density function $d_{kj}^*(x)$ which satisfies equation (2). For that, please recall that if any of the conditions c_{kj} is activated, the corresponding transition is defined to be fired immediately. The conditions c_{k1}, ..., c_{km} become true at time $t_{e,k1}$ ... $t_{e,km}$, $t_{e,kj} \in [0,\infty)$; that is: each condition eventually becomes true. The minimum over $t_{e,k1}$ ... $t_{e,km}$ shall be called t_e. Then the following definition for $d_{kj}^*(x)$ results (qed.):

$$d_{kj}^*(x) = \begin{cases} d_{kj}(x) & \text{for } x < t_{e,k} \\ 0 & \text{for } x \geq t_{e,k} \text{ and } t_{e,kj} \neq t_{e,k} \\ \left\{ 1 - \sum_{o=1}^{m} \left(\int_{t=0}^{t_{e,k}} d_{ko}(t)dt \right) \right\} \cdot \delta(t_{e,k}) & \text{for } x \geq t_{e,k} \text{ and } t_{e,kj} = t_{e,k} \end{cases} \tag{4}$$

Obviously: as long as none of the conditions is satisfied ($x < t_{e,k} = \min(t_{e,k1}, ..., t_{e,km})$) the activation probability for the jth transition of state k (i.e., the probability that this transition got activated) can be determined to:

$$p_{kj}(x) = \int_{\xi=0}^{x} d_{kj}(\xi)d\xi \tag{5}$$

Note: This continuous automaton is more expressive than necessary in terms of the description language proposed in section 2.1.

2.3. Discrete Automaton

The discrete automaton to be used describes a system, controlled by an external clock impulse with pulse length Δt. The formal notation is the following:

$$A=\{A_i\}, \text{ i=1...n, with}$$

$$\mathbf{A_i = \{Z_{pi}, Z_{ti}, z_{pi}^0, T_i, V_i, W_i, X_i, x_i^0, f_i, c_i, p_i\}}$$

The product automaton A has feedback automaton structure, i.e. each sub-automaton A_i may know the actual state of the complete automaton. Note: While not mentioned explicitly, the index i, indicating the ith sub-automaton, is neglected in the rest of this subsection for reasons of readability. The different elements of each of the sub-automata are given as follows:

Z_p	set of permanent (i.t.s.o non-transient) states
Z_t	set of transient states with $Z_t \cap Z_p = \varnothing$
z_p^0	inital state distribution function. Mapping associating each of the permanent states with a probability of being the initial state (active at t=0). z_p^0: $z \in Z_p \rightarrow [0,1]$ with $\sum_{z \in Zp} (z_p^0) = 1$.
T	set of transitions $T \subseteq ((Z_p \times Z_t) \cap (Z_t \times Z_p))$. T contains all possible transitions between states where only transitions between elements of the different state sets are allowed (bipartite graph).
V	set of Boolean input variables, cf. equation (1).
W	set of Boolean output variables with $W_i \subseteq Z_{pi}$
X	set of local integer clocks
x^0	mapping that assigns an initial value to each local clock. x^0: $x \in X \rightarrow \mathbb{N}_0$, $\forall x \in X$
f	clock modification function that assigns to each transition from a transient state to a permanent one a vector scaling for each clock whether it is incremented by one (*inc*), reset (*res*) or not changed at all (-). f: $t \in T \cap (Z_t \times Z_p) \rightarrow (inc, res, -)^{\vert X \vert}$
c	transition condition. Mapping c: $t \in T \cap (Z_p \times Z_t) \rightarrow C$ with C being the set of all Boolean functions of the form $c = c_l \vee c_g$ where c_l is a Boolean function over the set of input variables V and c_g is a Boolean function over the set of clock guards, where a clock guard is a relation of the form $x \Diamond a$ with $x \in X$, $a \in \mathbb{N}^+$ and $\Diamond \in \{=, \leq, <\}$.
p	transition probability. Mapping p: $t \in T \cap (Z_p \times Z_t) \rightarrow [0,1]$.

Fig. 6 shows the structure of the discrete automaton for one specific state $z_k \in Z$. Transient states (normally shown as shaded squares) with only one outgoing transition (p=1) are omitted from the graphical representation. The clock modification is then noted at the remaining arc.

For the probabilities associated with the transitions from a specific transient state to all the successive permanent ones, the following condition must hold:

$$\sum_{l} p_{kj,l} = 1 \;\; \forall k, j \tag{6}$$

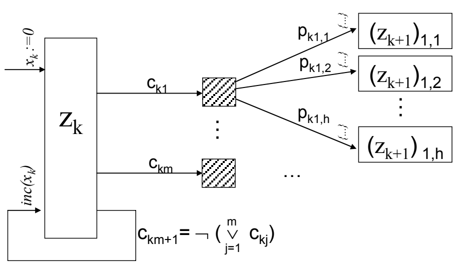


Fig. 6: Discrete Automaton

For the conditions at one state $z_k \in Z$ the following must hold:

1. There are no two conditions active at the same time:

$$\forall_{l,j\in[1..m];l\neq j} c_{kl} \wedge c_{kj} = false \tag{7}$$

2. The disjunction of all conditions must cover the total state space:

$$\bigvee_{\forall j\in[1..m+1]} c_{kj} = true \tag{8}$$

The latter condition was derived from the following consideration: If there were no condition active at one time, the automaton would just increment the counter when the clock impulse arrives. For the implementation, however, it is easier to fire a transition synchronously with each clock impulse. That's why one more transition, looping back on the state, is introduced. The corresponding condition $c_{k(m+1)}$ is thereby defined as the complement over all other c_{kj}s:

$$c_{km+1} = \neg\left(\bigvee_{j=1}^{m} c_{kj}\right) \tag{9}$$

Together, this leads to the following semantic: If A_i is in a permanent state $z \in Z_{Pi}$, it waits for the sync-pulse at Δt; now, all outgoing transition conditions are evaluated (exactly one is true). This leads to a transient state. From this state one of the following permanent states is selected according to the probabilities p. Finally, the clocks are modified with reaching the permanent state.

Note: David Parker, one of PRISM's developers gave in his PhD [Parker, 2002] the following sub-automata: $A_i=\{S_i, S0_i, P_i\}$, using S_i as the set of state of the ith sub-automaton, $S0_i$ for the set of initial states and P_i being a subset of the automaton's transition matrix $P: S \times S \rightarrow [0,1]$, which assigns a probability value to all the transitions, based on a mix of conditions and probability values defined also in his work. It is easy to transform this automaton structure into the one introduced here. The discrete structure presented here is a way for describing the link between an intuitive engineering model and the PRISM implementation. Parker's structure on the other hand is much more useful, if one and the same structure is used to describe different kinds of Markov models.

2.4. From continuous to discrete model

After the discretization, the model does no longer represent the exact occurrence time of an event, but only the fact that an event has occurred within the last time step. Thereby the correct order of two events occurring in the same time step is neglected (in the sense of masked) by the model structure. This reduces the size of the automaton drastically. Note: [Alur and Dill, 1994] are introducing digital clocks. However, the point in time of the events (and therefore the distance in between several events) is still a dense time and will be projected to the discrete axis afterwards, while the presented model uses discrete times directly.

Transforming from continuous to discrete, it is important to know that not all the elements are defined in the same way. This is because the discrete automaton was created to best fit the PRISM coding language, while the continuous automaton was created to best describe the behavior of a NAS. The nice thing is that $Z_{pi}=Z_i$, $W_i=W_i$, $V_i=V_i$, $z_{pi}^0=Prob(Z_i(t=0))$ and $X_i=X_i$ (despite the fact that the clocks in X_i are defined over $\mathbb{R}_0^+$ and the ones in X_i over $\mathbb{N}_0$).

The reason for the differences in the remaining sets and functions is that the continuous model separates strictly in between conditions and times, while the PRISM structure does not. From there it follows that $C_i(V_i)$, $D_i(X_i)$ and L_i of the continuous model have to be evaluated to create the discrete T_i, c_i, p_i and in this run also the set of transient states Z_{ti}. For each transition of each state in Z_i the following has to be done:

1. Initialize the counter variable for the state.
2. If $d_{kj}(x)\equiv 0$, then the discrete condition c_{kj} equals the continuous one (while the corresponding probability $p_{kj}=1$).
3. Otherwise for all discrete time steps, where the integral of $d_{kj}^*(x)$ over the corresponding time frame $q \cdot \Delta t$

$$p_{kj}(x=q) = \begin{cases} \int_{q\cdot\Delta t}^{(q+1)\cdot\Delta t-\varepsilon} d_{kj}^*(t)\,\mathrm{dt} & \text{for } q \le 2 \\ \int_{q\cdot\Delta t}^{(q+1)\cdot\Delta t-\varepsilon} d_{kj}^*(t)\,\mathrm{dt} \cdot \dfrac{1}{\prod_{o=1}^{q-1} p_{kj}(x=o)} & \text{for } q>2 \end{cases} \tag{10}$$

   is larger than zero, a transition from this permanent to a transient state is build, which's discrete condition c equals the continuous condition disjuncted with the counter variable's value: $c=c_l \vee (x=q)$. The transient state has two outgoing transitions as shown in Fig. 7: The first one connects the transient state with the permanent one given by the continuous transition. The probability of this transition equals $p_{kj}(q)$. As the probability sum of all outbounding transitions of an transient state must add up to one, the second transition connects from the transient state back to the original state and increases the counter.

4. If the value given by L_i (for the transition from this state to another state), is not 1 (i.e. the transition is taken only with a probability p<1), then the values of $p_{kj}(q)$ have to be multiplied with this probability.

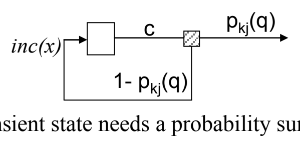


Fig. 7: Each transient state needs a probability sum of 1

Then, for each state the following has to be done:

1. If there are two transitions labeled with the same condition, they have to be merged. Note: This should not be necessary!
2. A self loop around the state must be initiated, which is activated, if and only if no other transition is activated. The clock modification for this transition is set to “inc(x)”.
3. The clock modifications of all other transitions descending from this state are set according to X_i^R and X_i^{act}.

Finally, the initial state distribution together with the set of initial clock values has to be transformed into the discrete initial conditions. In principle, this is just a mapping from the continuous to the discrete set, despite the fact that the time in the discrete case is no longer continuous and therefore the corresponding density values for each (state, counter) can be received by integrating over the (state, time) of the continuous set for each time step’s interval (which indeed transforms the density function to a discrete set of probability values).

2.5. The initial state density function

The creation of the initial state density function will be discussed on the basis of two typical NAS-examples, namely a PLC and a switch failure.

A PLC has three states: read, execute and write. The forth and fifth state, waiting for the cycle time to elapse and “operating system”, are added to the execution part, as for analyzing times, only the read and write states are of interest. For the initial state, the assumption is used that all possible times are equally likely. Let’s take t_{cyc} for the cycle time, t_w for the write time and t_r for the read time. Then, the probability density that the clock x equals a specific time value in the beginning equals $1/t_{cyc}$. The probability that the PLC is currently writing, therefore is t_w/t_{cyc}; t_r/t_{cyc} for reading and $(1-t_r-t_w)/t_{cyc}$ for the execution. Fig. 8 shows the continuous and its corresponding discrete automaton. For reasons of feasibility, the assumption was made that $t_w=t_r=\Delta t$. Defining t_d towards the next integer number from t_{cyc} divided by Δt, then, the (initial) probability for the discrete counter variable to represent a given value is $1/t_d$. The probability of being in the read (or in the write) state therefore is $1/t_d$, the one for the execution state: $1-2/t_d$. The density distribution, for both, a continuous and a discrete time axis, is given in Fig. 10a.

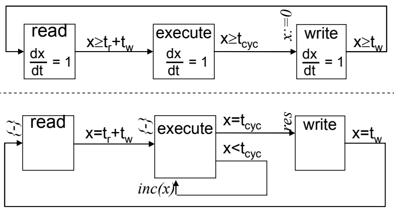


Fig. 8: Continuous and discrete automaton of the PLC module

The switch failure is modeled as follows: A failure might occur with a probability $pF_c(x)$ and will then last for $F_t \cdot \Delta t$. It is possible that another failure occurs immediately afterwards. The corresponding automata for the continuous as well as for the discrete case (Note: pF_d is a constant value) are shown in Fig. 9, while the associated density function for the switch failure is shown in Fig. 10b. The initial probabilities and clock values are determined as follows (Note: All initial clock values are equally likely):

$$\mathrm{Prob}(\mathrm{OK}_{(t=0)}) = z_p^0(OK) = \frac{1-pSwF_d}{1+(F_t-1)\cdot pSwF_d} \tag{11}$$

$$\mathrm{Prob}(\mathrm{failure}_{(t=0)}) = z_p^0(\mathrm{failure}) = \left\{1-\mathrm{Prob}(\mathrm{OK}_{(t=0)})\right\} \tag{12}$$

$$X_i^0 = \left\{x \mid 0 \leq x < F_t, x \in \mathbb{R}\right\} \tag{13}$$

$$x_i^0 = \left\{x \mid 0 \leq x < F_t, x \in \mathbb{N}\right\} \tag{14}$$

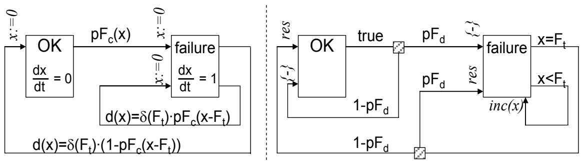


Fig. 9: Continuous and discrete automaton of a failure module

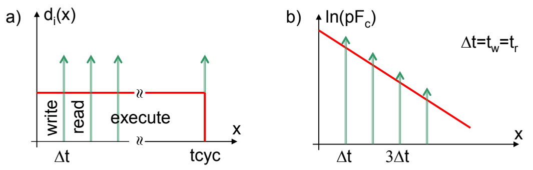


Fig. 10: Probability density functions

2.6. Transformation into PRISM Language

The transformation of the automaton into PRISM language is straight forward: For each sub-automaton a module of the following form is generated:

```
module <modname>
      <statevar>: [<range>] init <initial_s>;
      <countervar>: [<range>] init <initial_c>;
  [Dt]  <ck1>  &  <zk>  →  <pk11>:(<zk+1,1,1>)&(<fk11>)  +
<pk12>:(<zk+1,1,2>)&(<fk12>) +<pk12>: ... ;
  [Dt] <ck2> & <zk> → <pk21>:(<zk+1,2,1>)&(<fk21>) + .... ;
...
  [Dt]  <c(k+1)1>  &  <z(k+1)>  →  <pk11>:(<zk+2,1,1>)  &
 (f (k+1)11) + .... ;
...
endmodule
```

<modname> thereby is the name of the module, <statevar> is the identification variable of the state (there might be several <statevar>s), <initial_s> can be used to assign an initial value to <statevar>, <countervar> is the implementation of the clock x, and <inital_c> the assignment of an initial clock value, [Dt] means that in each of the modules, whose command lines are indicated with [Dt] the conditions (6) and (8) must be fulfilled. <ck1> means the condition c_1 of the state z_k and <zk> is to indicate that this condition is valid, when the automaton is in the state z_k. “→” symbolizes the firing of the transition, <pk11> equals $p_{k1,1}$ (1st probability of the 1st transition of the state z_k), <zk+1,2,1> means the state $(z_{k+1})_{2,1}$ and <fk11> is the clock modification function $f_{k1,1}$.

If the initial state distribution has more than one nonzero element, the initial state must be determined first. This can either be done using the init specification of PRISM (which induces some problems) or by introducing a pre-process to the affected modules that maps a probability to each of the possible states (cf. [Greifeneder and Frey, 2006]).

3. CASE STUDY

The example contains a PLC, its PLC-IO, a wireless LAN, a sensor, an actuator and a field-IO (cf.Fig. 11).

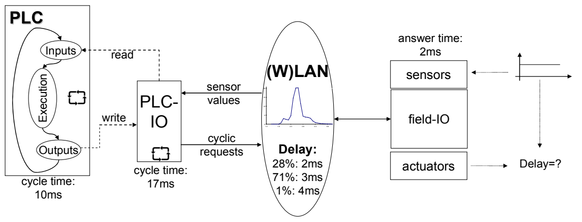


Fig. 11: Structure of the case study

To be determined is the delay between a sensor's activation and the reaction at a corresponding actuator. To do so, the following process must be supervised (cf. Fig. 12): After the input got triggered, some time may pass until the next request from the PLC-IO arrives at the field-IO. Afterwards, the answer is generated and sent back through the network. Then, the second synchronization – this time on the cycle of the PLC – is necessary. After being executed by the PLC, the result must wait for the next PLC-IO send-time (this is not any more a cycle based function, but only a function of time being passed since the first synchronization on the PLC-IO-cycle and the cycle time). Finally, the result must pass the network and be processed by the field-IO / the actuator.

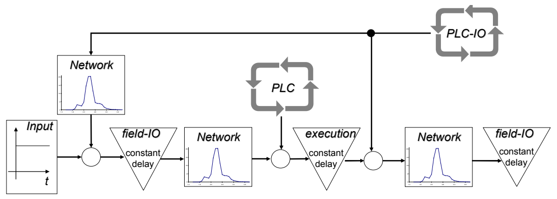


Fig. 12: Signal tracking trajectory

Analyzing the response time of this system leads to the distribution shown in Fig. 13. The value of 4.5% at time 50 ms means that the probability of a response being delayed in between 49 and 50 ms (discretization step width = 1 ms) equals 4.5%. The second line of the graph (rombi) shows the corresponding distribution arisen from measurements (200 values). Note: All measurements in this paper had to be shifted by a time offset of +3 ms. This offset is assumed to be caused by a systematic failure in the measuring method and therefore neglected.

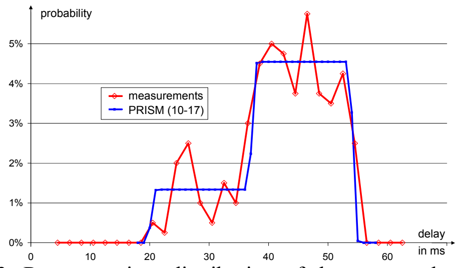


Fig. 13: Response time distribution of the case study (measurements and PMC-results)

In a second experiment, the cycle times of the PLC and its IO got changed, so the PLC had a cycle of 17 ms and the PLC-IO of 10 ms. By doing so, it is guaranteed that the PLC always has new values. Unfortunately, the medium delay rises from 41.9 to 43.4 ms as shown in Fig. 14.

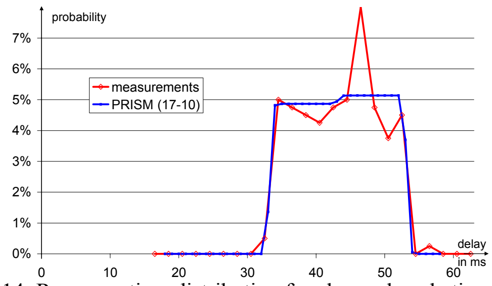


Fig. 14: Response time distribution for changed cycle times

In both experiments, the cyclic superposition did result in the demonstrated distributions. Therefore, a third experiment was done, assuming a really fast PLC (1 ms), coupled with the maximum the authors could do for the (serial interface based) sending rate of the PLC-IO (3 ms). The corresponding distribution is shown in Fig. 15. While the PLC itself only has a constant influence of 1 ms, PLC-IO and network are the two sources of the distribution's shape.

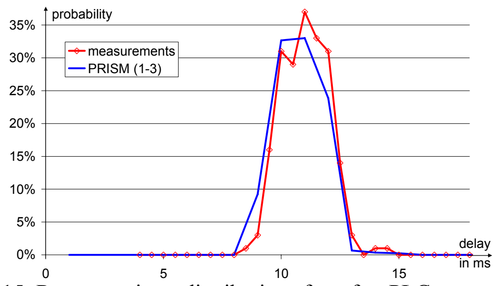


Fig. 15: Response times distributions for a fast PLC

This implies that in the first two cases the network only causes a small part of the delay. However, the network enables architectures with a lot of synchronization delays.

4. SUMMARY

In this paper a formal automaton definition for modeling Networked Automation Systems (NAS) is given, followed by the corresponding discrete automaton and the affiliating transformation. The discrete automaton can easily be re-written in the PRISM coding language, which can be used for probabilistic model checking of NAS. The comparison of results from PMC and measured values has lead to an astonishing compliance, which allows the generalization of the results by adding different not really measurable effects (like stochastic failures) to the model.

REFERENCES

Alur, R. and D. Dill: *A theory of timed automaton*. Theoretical Computer Science, 126(2): 183-235, 1994.

Greifeneder, J. and G. Frey: *Dependability analysis of networked automation systems by probabilistic delay time analysis*. Proc. of IFAC incom, St. Etienne, France, Vol. 1 pp. 269-274, 2006.

Katoen, J-P.: "*Stochastic Model Checking*", in Stochastic Hybrid Systems, ed. C.G. Cassandras and J. Lygeros, Control Engineering Series, Taylor & Francis, Boca Raton, pp. 79–107, 2006.

Kwiatkowska, M., G. Norman and D. Parker: *PRISM: Probabilistic symbolic model checker*. In TOOLS'02, LNCS, vol. 2324, pp. 200–204, Springer, 2002.

Parker, D. *Implementation of Symbolic Model Checking for Probabilistic Systems*. Ph.D., University of Birmingham, 2002.

Report on the *workshop on future and emerging control systems* organized by unit E1, European Commission, 2000.