PROBABILISTIC HYBRID AUTOMATA WITH VARIABLE STEP WIDTH APPLIED TO THE ANAYLSIS OF NETWORKED AUTOMATION SYSTEMS

Jürgen Greifeneder and Georg Frey

Electrical and Computer Engineering Department
University of Kaiserslautern, Germany
Erwin-Schrödinger-Str. 12, 67663 Kaiserslautern
e-mail: greifeneder@eit.uni-kl.de

Abstract: Probabilistic Timed Automata (PTA) have successfully been applied to discuss problems specific to the field of Networked Automation Systems (NAS). This paper shows the transition from PTA towards Probabilistic Hybrid Automata (PHA) by introducing an event triggered variable time version of PTAs. The main idea is to increase accuracy and decrease state space of the underlying Discrete Time Markov Chains (DTMC) which are input for those probabilistic model checking algorithms for which these models should be used. For a better illustration of the advancement, the approach is applied to a typical example from the field of NAS. *Copyright © 2006 IFAC*

Keywords: Time varying systems, Probabilistic Models, Quality, Delay analysis, Discrete-time models, Markov models, Networks, Modeling, Automata.

1. INTRODUCTION

In the last decade, the field of automation has been interspersed with apparently simple components borrowed from other technical fields, particularly those dealing with distributed systems. One of the driving forces thereby can be found in the rapid development of Ethernet or TCP/IP-based technologies resulting in a collapse of prices for the corresponding hard- and software. Apparently to that, the miniaturization went on as well as distribution of components and the need to communicate (not at least triggered by a still growing complexity of automation control algorithms). The hence composed systems aggregate control (i.e. automation) systems and network technologies. This fusion is called Networked Automation Systems (NAS) and represents a structure consisting of a number of programmable logic controllers (PLCs), several sensors and actuators, the network itself, and various input-output-Network-Devices.

The use of open communication platforms entails an increased and not previously predictable data flow, which may induce delays not present in classical structures. Furthermore, while the use of wireless networks (e.g. WLAN, Zigbee) offers a great opportunity in terms of variability and miniaturization, it adds a potential risk of data loss (loss of data in a NAS being defined as the non-arrival of data within a given time frame). All in all, this leads to a system structure which contains delay times, hard time bounds, stochastic distributions, deterministic choice connecting physical and computation processes.

There already is some work done to tackle the challenges yielded by NAS (Dingle et al., 2002 ; Marsal et al., 2005). However, these approaches are based on simulation, which implies that there is no guarantee for a full account of all possible evolutions. Yet, such a guarantee is essential for the analysis of NAS. Additionally, there are several formal approaches, but those are unfortunately unable to deal with the system structures described above or are restricted to producing worst case analysis, which can only lead to infeasible demands on the system's hard- and software. Therefore, a new modeling approach is proposed which can deal with time, stochastic distributions and probabilistic choice, namely probabilistic model

checking (PMC). When PMC is used for a NAS analysis, several advantages as well as disadvantages have to be considered. One of the advantages is that PMC is certain to cover all possible evolutions of a system instead of only a subset as is the case with simulation and testing methods. The disadvantages are best described by the price to be paid in terms of modeling abstractions, assumptions, and computational limits.

For this work, the considered systems are modeled on a discrete version of Probabilistic Timed Automata (PTA) and projected to Discrete Time Markov Chains (DTMC) by the use of an event triggered variable time step (which will be introduced in the next section). These DTMCs are integrated for the sake of the probabilistic model checking algorithms. The results presented within this work are calculated using PRISM, a model checker from the University of Birmingham (Kwiatkowska et al., 2002). For further illustration of design and consequences, a small, but typical NAS-example is given in section 3. Section 4 furnishes an in-depth explanation of some details of the time concept introduced in section 2; a short discussion follows in section 5. Finally, the concluding section 6 gives an outlook on further work.

2. MODELING OF TIME

Since physical processes are concerned, the correct functioning of the control system depends crucially upon real-time considerations. For that reason Alur and Dill developed the theory of timed automata (Alur and Dill, 1994). Therein, they used a dense-time representation, where times are represented by real numbers with an arbitrary accuracy tolerance up to which two numbers are considered as equal. However, using this approach it is still necessary to integrate different times to result in the same characteristic. This is accomplished through the use of clock regions or region graphs (actually the same thing). The method of digital clocks suggested by (Henzinger et al., 1992) could be used as an alternative to clock regions for a successful discretisation. Yet, the question arises why the discretisation is not carried out earlier in the process. This idea leads to a discrete model, which has each state attached with time information built of discrete and monotonically increasing (mostly integer) numbers. The Analysis of a timed system with discrete time creates certain phenomena, such as the problem of finding an appropriate time step and a set of initial states which would at least be able to cope with constant time drifts among non-synchronized subsystems (Greifeneder and Frey, 2006b). While in systems where signal changes are considered to occur synchronously with a clock signal (e.g. digital circuits) the problem of finding an appropriate time step is inherent, it still provides a potential source for state explosion. When it comes to continuous time physical interfaces (sensors), there is neither a lower nor an upper limit to this decision: if, on the one hand, a time step which is shorter than necessary to exactly model the fastest system change is used, the size of the model has the tendency to increase exponentially; if, on the other hand, a time step which is longer than necessary is applied, information loss will occur. One of the most important advantages of this modeling approach is that it can be transformed easily into an ordinary formal language via the addition of a new event to the set: a synchronize tick.

Note: It is not necessary to explicitly add the absolute clock information to each state as this information can be regenerated through a count of the number of states on a given path. A discrete time model usually requires a strong monotony of the integers representing the time, but in this case, the use of a fictitious time which only needs to be non-decreasing eases this restriction.

The time delay between two events is measured as the sum of all time steps elapsed between the two of them, that is: the point of time of each event is the same as its position in a series of states leading from an initial to a final state (this is to be called a path). To do the discretisation, it is impossible to state precisely certain about delays. On the other hand, the exact occurrence time of most of the signals is of lower interest, especially if any digital device forces the incidence towards a discrete time axis. Furthermore, it is irrelevant for most purposes whether an event A (e.g. the change of a sensors' value) occurred before or after another independent event B (e.g. the arrival of a package at a switch). In these cases, it really is an advantage (in terms of an enormous decrease in state space and calculation time) that only the fact that an event occurred within the last time step is recorded instead of the exact point of time within this time step. The best interpretation of this approach is to say that events occur in the specified order at real value times, but only the (integer) readings of the time value are recorded. Hence, by the way, the system model eliminates the occurrence of non-deterministic choice. All in all, this leads to the claim that the maximum length of a time step must be shorter than the time passing in between two consecutive significant events (a significant event is an event which has an influence on other processes in the system). From thereon, a modeling approach, which has already been successfully used for several case studies (Greifeneder and Frey, 2006a; 2006b), was derived. Yet, all of them suffered from the state explosion problem which raised exponentially to the chosen time accuracy. Therefore, the originally fixed time steps are this time replaced by an event triggered time step. This fuses the advantages of the authors' previous discrete time approach and the work done by (Alur et al., 1991). As in a DTMC the consecutive path only depends on the actual state (that is: the momentary state relies solely on the previous one), the information about the next time step's length must be determined before this state transition is executed. This is the real challenge to be taken up and changes the modeling task in a quite fundamental way (as will be demonstrated in chapter 4).

3. INTRODUCTION OF AN EXAMPLE

3.1. System description

The system under observation (cf. Fig. 1) consists of a transport unit, which is driven by a speed controlled engine, a drilling unit, and two inductive sensors. While the transport unit knows its speed, but is unable to determine its own position, the two inductive sensors – indicated with s_A and s_B – can only detect when a position is reached.

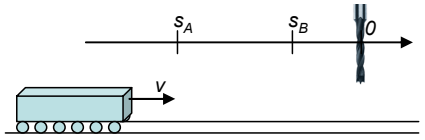

Fig. 1. Schematic representation of the controlled process used for the case study. The aim of the control is to reach a given drilling position based on the position of the sensors s_A and s_B.

The value to be optimized is the position where the drill puts the hole. The system is to be understood as NAS as illustrated in Fig. 2 (this, of course, is a very small example, set up only to demonstrate the capacities). The two sensors are connected to an IO-card, which itself is connected to the network. The network is assumed to be wireless TCP/IP-Ethernet based. Package loss is not considered; for a discussion on that see e.g. (Greifeneder and Frey, 2006a). The IO-access of the transport unit and the PLC-IO are also connected to the network. The PLC is connected to its IO-card via a shared memory (that is, incoming data will be available for the PLC as soon as it has arrived). The drilling process itself is not included in the model.

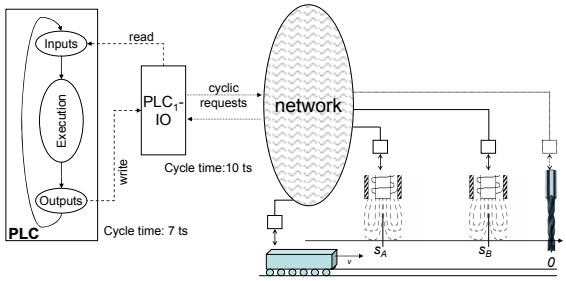

Fig. 2: Architecture of the Networked Automation System used in the case study.

When the object passes the sensor at s_A, this piece of information is carried by the network to the PLC-I/O card. From there, the value is written into the PLCs input buffer via a backplane-bus. Inside the PLC, the information is processed, which results in a new setting of an output variable used to slow down the motor from normal speed to a slow mode. The new output value is transferred via the PLC I/O-card and the network to the transport unit. The latter will execute the command as soon as the signal arrives. When the object passes the sensor at s_B, the same procedure (as described above for sensor s_A) will be accomplished except that this time the transport unit will decelerate from the actual speed (which might still be larger than the slow mode speed, depending on when the 'start decelerate for slow mode' signal arrived) until it stops. After this state is reached, a stop signal is passed through the network to the PLC, and after being processed from there, the drilling command is issued. Obviously, by moving the positions of the sensors s_A and s_B, the accuracy of the drilling position can be optimized, as well as the time needed for this processing task.

3.2. Abstractions and Assumptions

The sensors are assumed to be exact within one length unit and the cyclic sensor-reading behavior is neglected. Both would add another delay distribution, which should be accounted for in a real system. However, those effects will not add a new phenomenon to this discussion (except another increase in calculation time). The network itself is assumed to be TCP/IP-based wireless Ethernet containing an access point which acts as a router. Depending on other traffic conditions (e.g. signals from other sensors or to other actuators), the routing time varies in between one (90%) and two (10%) time steps. This corresponds to measurements at a lab system which finds most of the packages in between one and two milliseconds (i.e. one time step can be assumed to equal one millisecond). It is assumed that there is no other disturbance or delay within the network. For a discussion on consequences of failures in the network on the delay time see e.g. (Greifeneder and Frey, 2005). The PLC uses a cycle period of 7 time steps, while its I/O card cycle uses 10 time steps. The speed is assumed to be 24 length units per time step (lu/ts) in the normal mode and 4 lu/ts in the slow mode. The first deceleration is assumed to be 2 length units per time step square and the second one to be 4 lu/ts². These numbers are of an academic nature and designed to guarantee that all the state variables will represent an integer value after each time step. The definition for a length unit to be e.g. 0.01 mm with the given time step of 1ms results in reasonable values for speed and deceleration.

The moment in which the object enters the system is of stochastic nature. Since the (minimum) time step is fixed, the initial value of the position x has to inherit this stochastic character from the time: equally distributed over all possible initial values, i.e. with the initial (normal) velocity of 24 lu/ts the first value after entering the system ranges over 24 neighboring values. While this does not affect the medium end position, it has an effect on deviation and spread. It is more important for the result that PLC, PLC-I/O, engine and sensors are not synchronized, and therefore all possible permutations have to be considered by initial states (again, equally distributed). Furthermore, it is assumed that the relative time drift in between the modules is less than one time step over the total time period (i.e. that it is less than 1%). For the following discussion, these different initial states will not be considered, since they are merely important for the correctness of the results, but not for the way in which the new approach is built.

The influence of the NAS on the velocity-trajectory is shown in Fig. 3 for a single value-setting: s_A=175, s_B=5. While the lower curve shows a typical evolution for the case that the sensors' signals act directly on the actuators, the upper curve shows the influence of the delay caused by the NAS on the positioning problem. First, both curves are identical until the first sensor is passed. Then, the system without NAS starts to decelerate, while the system with NAS still runs on high velocity until the signal is passed through the network (in this very example this takes 17 time steps). Afterwards, the system using NAS also decelerates down to v_{slow}.

This paper focuses on the implementation of the event based approach and shows the differences towards the formerly used fixed step approach. For a more detailed discussion on the example, its behavior and the corresponding PMC-modules see (Greifeneder and Frey, 2006c).

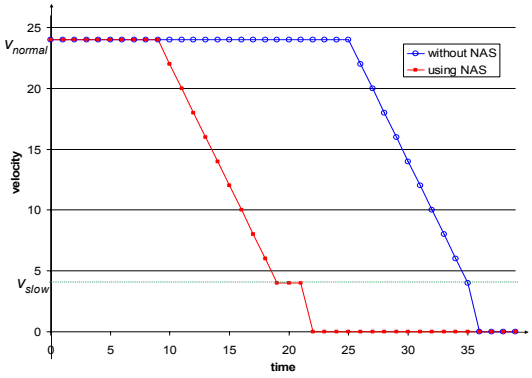

Fig. 3: Illustration of the velocities of a positioning process with direct sensor-actuator coupling in contrast to a NAS-system.

4. DETERMINING THE NEXT TIME STEP'S LENGTH

Some components in NAS exist more than once, and the model of the system has to be scalable in an easy way. Therefore, it makes sense to model the different components separately and connect them afterwards to build the system. This also offers the opportunity (or introduces the curse) for each module to determine its next maximal time step's length separately from all the other modules. As the information about the overall length of the next time step must be available to determine the next state, the conjunction of all the modules' maximal time steps (each of them determined in the actual time step) must be done in between the current one and the next step. Obviously, this could be done by the use of an intermediate step, but hence the state space would be increased by an exponent of two without any benefit for the system. Fortunately, PRISM offers a min-function which returns the minimal value out of a set of values delivered to the function. This function is used to determine the next step's length (dt) within the transition guards itself:

$$dt_{i+1} = \min_j (fdt_i^j) \qquad (1)$$

fdt_i^j thereby indicates the maximally possible time step forecasted by the j^{th} module on the basis of the state at the i^{th} time step. In principle, this forecast is produced as the determination of the time necessary to reach a state which would trigger a transition within the j^{th} module.

Let us have a closer look at the previously introduced example. Five modules are necessary to describe it completely: The sensors, the engine, the network (for reasons of complexity reduction separated in three parts, see below), the PLC, and the associated PLC-IO. As PRISM makes all (local) variables visible to all other modules, all modules know about all variables in the system (that is, the exact systems state). In the following, the different modules will be introduced.

4.1. The sensors' module

The sensors' module determines the current position of the object and sends a signal each time that one of the two sensors is passed as shown in Fig. 4. The event dt associated with the diagonal line on the transition symbolizes the fact that this transition is fired synchronously with all the other modules. Note: In each module there is always one and only one active transition. For a more detailed introduction to the automaton representation see (Greifeneder and Frey, 2006a).

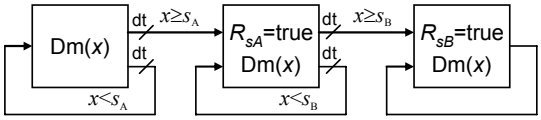

Fig. 4: Position determination automaton (sensors' module).

$Dm(x)$ symbolizes the determination of the current x-value (position). The dotted variables stand for the velocity and acceleration at the i^{th} time step.

$$Dm(x_{i+1}) = x_i + dt \cdot \dot{x}_i + \frac{1}{2} \cdot dt^2 \cdot \ddot{x}_i \qquad (2)$$

If the automaton is in the first state, the maximum time which may elapse before a state change occurs is the time which is necessary to reach the s_A-position given the current velocity:

$$fdt_i^{Sensor} = \frac{s_A - x_i}{\dot{x}_i} \qquad (3)$$

The same equation is used for the state located in the middle position of Fig. 4 if the actual acceleration equals 0 (using s_B instead of s_A). If the acceleration does not equal 0, the corresponding equation can be deduced from equation (2):

$$fdt_i^{Sensor} = \frac{-\dot{x}_i \pm \sqrt{\dot{x}_i^2 - 2 \cdot \ddot{x}_i \cdot (x_i - s_B)}}{\ddot{x}_i} \qquad (4)$$

If the right state of the automaton is reached, fdt_i^{Sensor} will be set to its maximum value (in PRISM all variables must be defined by type and range). As a large range induces a large state space, it is wise to chose the maximum value equaling the maximum result possibly occurring from equation (3) respectively (4).

Finally, there is an exception to the above equations: If a sensor is activated (R_{sA} or R_{sB} set to true), this information must be handed over to the network immediately (i.e. maximum one time step).

4.2. PLC and PLC-IO

Both the PLC and its IO module are abstracted from ring-counters. Since the counter variables (ct) as well as the maximum counter variables (CtMax) are known through the system both counters can be implemented as shown in Fig. 5.

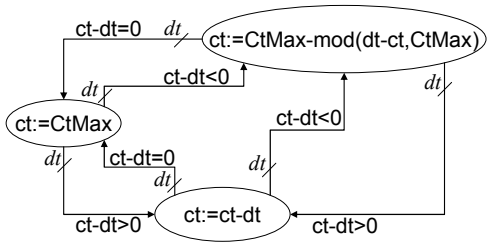

Fig. 5: State transition representation of a ring counter (PLC- and PLC-IO-module).

4.3. The networks

As already mentioned, the network is split up into three modules. The first one works in the following way: it waits until R_{sA}=*true* is received from the position determination module (for reasons of simplification, the time for sending a package from the IO to the network is assumed to equal one time step). As long as this signal is not received, this module simply does nothing. In analogy with the sensors' module, the fdt_i^{Net1} variable should be set to its maximum value. The maximum time delay occurring in the network is the sending time of two time steps. Yet, this – is not a suitable value to be used as a maximum value, for it will be smaller than all other values in most of the cases. Therefore, the network modules are provided with two variables. One, the already known one ($fdt_i^{Net_k}$) and second a binary variable ($ExNet_k$) which indicates whether or not the first value has to be considered by the minimum function given in equation (1). $fdt_i^{Net_k}$ in equation (1) therefore is replaced by:

$$\max(fdt_i^{Net_k}, 500 \cdot ExNet_k) \qquad (5)$$

500 is just an arbitrary number. As soon as the R_{sA}=*true* signal is received, $ExNet_1$ is set to 0 and $fdt_i^{Net_1}$ is set to the sending time, given by the queue length (1 or 2 time steps). As soon as $fdt_i^{Net_1}$ minus *dt* equals 0, $ExNet_1$ is set back to 1 and the automaton waits for R_{sB}=*true*. Then, the same procedure is repeated. Afterwards this first of the three network automaton modules has no special task any more.

The remaining two modules are used to implement a signal tracking task; for a more detailed discussion on this cf. (Greifeneder and Frey, 2006b). To facilitate this concept, the downlink from the PLC-IO to the engine is modeled separately for each of the two signals. Hence, both of them have the same functionality although they are activated by different sensor signals (s_A versus s_B). The task to be observed is given as follows: Wait until the signal got received from the uplink network. Then wait until the signal got read by the PLC. Wait until the PLC processed the signal and finally wait until the PLC-IO has sent the signal to the network. Next, the same functionality is implemented as already discussed for the uplink module. The first waiting procedure already got discussed. The commands to wait for the PLC to read, process and afterwards write out the signal are implemented by the assignment of the actual value (minus *dt*) of the PLC-counter to $fdt_i^{Net_k}$ plus the PLC's cycle time minus one (and discount *dt* as long as $fdt_i^{Net_k} - dt \le 0$). In the case of waiting period for the PLC-IOs cycle, the assumption is made that the signal will be send when the counter equals zero. Therefore $fdt_i^{Net_k}$ is simply set to the current value of the corresponding counter.

4.4. The engine

The engine-module is probably the most complex one. Therefore, it will be introduced in two steps. First, the physical behavior will be discussed as visualized in Fig. 6. *S1* and *S2* represent the events of the reception of the information from the PLC that the first (respectively the second) sensor has been passed. Unless this is not the case, the automaton remains in the initial state (bottom left). If *S2* is received, the automaton changes towards the top mid state. However, this should not happen in normal operation before the two remaining bottom states are passed. From bottom left to bottom mid *S1* must be received. From there to bottom right, the velocity has been decelerated until the slow mode is reached. If the velocity equals zero, the top right state will be entered.

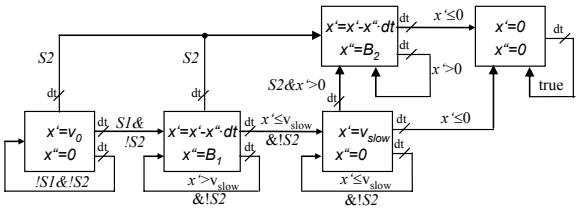

Fig. 6: Automaton of the engine.

The corresponding maximum step time can be determined as follows. In the first step, it is set to the maximum of the defined variable's range as it only has to wait. In the two middle states (bottom and top), it is defined as the actual velocity (old velocity plus old acceleration times *dt*) divided by the actual acceleration (which equals B_1 in the bottom mid and B_2 in the top mid state):

$$fdt_i^{Engine} = \frac{\dot{x}_i + \ddot{x}_i \cdot dt_i}{-\ddot{x}_{i+1}} \qquad (6)$$

If one of the two remaining states is entered for the first time, the change in the acceleration will lead to higher velocities in further steps. As this information

is important it should be passed on immediately – or within one time step. If one of theses states is re-entered, the maximum time step can be set to its maximum value. Note: in general, there should be such an information transfer step for each state re-entered, but in this work, it is only necessary for the deceleration to be set back to zero instead of to a higher deceleration, as there is no module in the system which would suffer from not receiving this information.

5. DISCUSSION

Fig. 7 shows one possible trajectory of the above example. While the solid line with the little rhombi represents the calculation points for a constant time step of one (summed up, there are 54 of them), the big circles represent the calculation points created if a variable step width is used. In fact, the number of points to be determined could thus be reduced from 54 to 16.

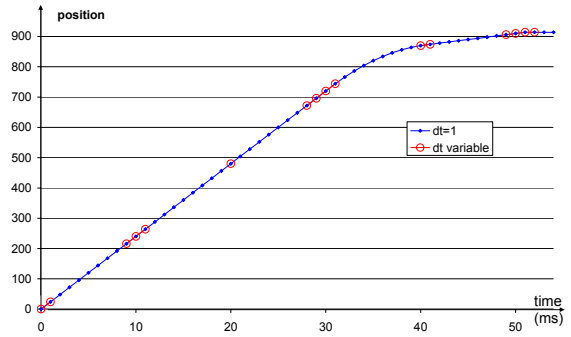

Fig. 7: Position trajectory of the transport unit over time for one single initial state. Calculation is done for a constant step size of 0.01ms (solid line) and for a variable step size (circles).

The proposed approach not only reduces the state space, but also the time needed for model checking. Unfortunately, while the time needed for model checking can be reduced, the time needed by PRISM to build the matrix of reachable states increases by a factor of 80 in comparison with the use of a fixed time step width. Therefore, the next aim should be to analyze the coding of PRISM and improve the effectiveness of this operation.

In Fig. 8 the distribution of the drilling position is given (s_A=*699*, s_B=*470*). The value of *6‰* at position *40* means that the probability of a stop in between 39 and *40* length units is determined to be *6‰*. The probability for values outside a range of ±1mm (100 length units) can be determined towards *7%*, outside of *±1.5mm* towards 0.21*‰*. Similar values are found for the standard deviation. The medium position is *-0.09mm*. The medium time needed to pass the last *centimetre* is determined to *49.035ms*.

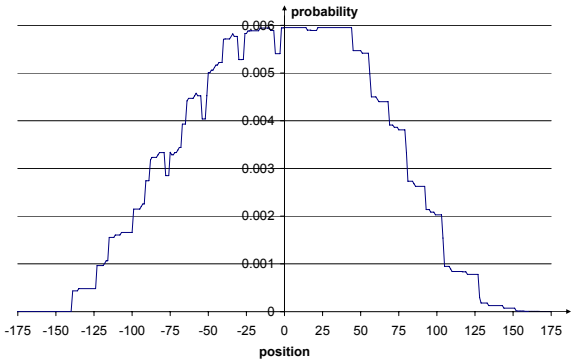

Fig. 8: Distribution of the final position (step width of 1).

6. CONCLUSION

In this paper a new approach for probabilistic model checking of NAS which uses variable (event triggered) Discrete Time Markov Chains was introduced. The next step for this project will be the transition towards dense time and a dense representation of physical variables. Finally, it would be interesting to implement an estimation algorithm able to fully use the next state for the estimation instead of an extrapolation from the actual state.

REFERENCES

Alur R., C. Courcoubetis, and D. Dill (1991). "Model-checking for probabilistic real-time systems". *ICALP'91*, LNCS, vol **510**, pp. 1–100, Springer.

Alur R. and D. Dill (1994). "A theory of timed automata". *Theoretical Computer Science*, 126(**2**), pp. 183–235.

Dingle, N.J., P.G. Harrison, and W.J. Knottenbelt (2002). "Response Time Densities in Generalised Stochastic Petri Net Models" *Proc. 3rd* WOSP, Italy.

Greifeneder, J. and G. Frey (2005). Probabilistic Delay Time Analysis in Networked Automation Systems. In: *Proc. of the 10th IEEE ETFA 2005, Catania*, Italy, Vol. **1**, pp. 1065–1068.

Greifeneder J. and G. Frey (2006a). "Dependability analysis of networked automation systems by probabilistic delay time analysis", *Proc. of 12th IFAC incom*, pp. 269-274.

Greifeneder J. and G. Frey (2006b). "Determination of Delay Times in Failure Afflicted Networked Automation Systems using Probabilistic Model Checking". *Proc. of 6th IEEE WFCS,* Torino, Italy, pp. 263-272.

Greifeneder J. and G. Frey (2006c) "Optimizing Quality of Control in Networked Automation Systems using Probabilistic Models". *Proc. of 11th IEEE ETFA*, Prague, Czech Republic.

Henzinger, T., X. Nicollin, J. Sifakis, and S. Yovine (1992). "What good are digital clocks?" in Proc. *ICALP'92, LNCS*, vol. **623**, p. 545–558, Springer.

Kwiatkowska M., G. Norman, D. Parker (2002). „PRISM: Probabilistic symbolic model checker". Proc. *TOOLS'02, LNCS*, vol. **2324**: 200–204. Springer, 2002.

Marsal, G., D. Witsch, B. Denis, J.-M. Faure, G. Frey (2005): "Evaluation of Real-Time Capabilities of Ethernet-based Automation Systems using Formal Verification and Simulation". *Proc. of 1ère RJCITR'05*, Nancy, France, pp. 27–30.