International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:1, 2010

Statistical Models of Network Traffic

Barath Kumar, Oliver Niggemann and Juergen Jasperneite (*Senior Member, IEEE*)

Abstract—Model-based approaches have been applied successfully to a wide range of tasks such as specification, simulation, testing, and diagnosis. But one bottleneck often prevents the introduction of these ideas: Manual modeling is a non-trivial, time-consuming task.

Automatically deriving models by observing and analyzing running systems is one possible way to amend this bottleneck. To derive a model automatically, some a-priori knowledge about the model structure–i.e. about the system–must exist. Such a model formalism would be used as follows: (*i*) By observing the network traffic, a model of the long-term system behavior could be generated automatically, (*ii*) Test vectors can be generated from the model, (*iii*) While the system is running, the model could be used to diagnose non-normal system behavior.

The main contribution of this paper is the introduction of a model formalism called 'probabilistic regression automaton' suitable for the tasks mentioned above.

Keywords—Model-based approach, Probabilistic regression automata, Statistical models and Timed automata.

I. Introduction

In this paper, a model formalism for analyzing long-term network behavior is given and assessed. Whenever such a new formalism is introduced, one key question must be answered first: Is a new modeling formalism really needed or can an existing formalism be applied? For this reason, this paper is structured as follows: First, in this section, the cornerstones of the needed formalism are sketched. E.g. such models must be able to capture state-based, temporal and probabilistic behavior.

Because several existing formalism fit these requirements, section III will give an in-detail overview and comparison of existing formalisms.

The applications addressed in this paper have some important points in common:

- Rather than specifying the system (and software) implementation in all detail, the focus is the description of the statistical behavior of a large set of situations.
- The model need not be able to capture the system behavior in detail. Instead it must only be able to recognize some non-normal system behavior and must be able to serve as a basis for test case generation and quantitative analysis.
- The systems addressed show mainly a state-based behavior. I.e. (*i*) at one point in time the system is in one state, (*ii*) within one state the system's values have a constant (or at least simple) relation to each other and (*iii*) the transitions from one state to another state follows a deterministic, repetitive pattern.

B. Kumar, O. Niggemann and J. Jasperneite are with Institute Industrial IT (inIT), Ostwestfalen-Lippe University of Applied Sciences, 32657 Lemgo, Germany.
E-mail: {barath.kumar, oliver.niggemann, juergen.jasperneite}@hs-owl.de

This paper deals with industrial production facilities. Such facilities show normally a state-based behavior—mainly caused by the large number of digital signals. But most facilities also comprise some additional continuous signals. The requirement of capturing such continuous signals complicates the modeling formalisms significantly: Pure state-based behavior can be captured using finite state machines, often extended by formalisms for expressing time and probabilities. Section III gives an introduction to such formalisms. In order to incorporate continuous signals, section IV introduces a formalisms especially suited for the specific tasks described in this paper.

II. Temporal Models of Network Traffic and Their Application

This paper looks for a formalism that is suited for three very specific algorithmic tasks: Learning a model of network traffic by analyzing networks traces (section II-A), generating test cases from network models (section II-B), and the diagnosis of system failures in section II-C. Learning complex models or even complex finite state machine formalisms is a difficult challenge. Generating test cases from too complex models may lead to a low model coverage. So a simple formalism but still a formalism that is able to capture all significant behavior is needed.

A. Analyzing Network Traffic

The key idea is rather simple: a model M of the system behavior is first learned by observing the running system. M focuses on those system aspects that should be analyzed (e.g. signal timing). Then this abstracted model is analyzed.

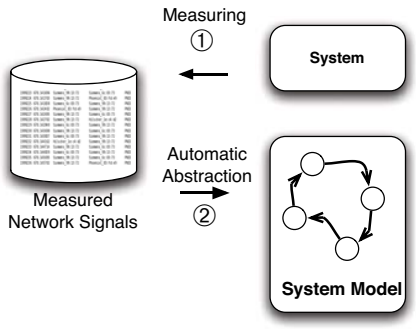

Fig. 1. Learning of system models.

This can also be seen in figure 1: First the network signals of the system are measured. These network traces are then analyzed and a model M is learned.

This application defines some requirements to M's formalism:

1) As mentioned before, M must be able to capture state-based behavior with some continuous signals.
2) When models are learned, often erroneous models are generated. E.g. a model is erroneous if it comprises of states (or situations) that neither allow the automaton to stay in them or to leave them. The best model formalisms do not permit the creation of erroneous models.
3) The timing of signals must be captured.
4) Network traffic on ISO/OSI layer 2-4 (see ITU-T X.200 recommendations) should be captured. E.g. which network message is sent as a reaction to a specific message, or in other words: values of signals in the messages are disregarded in these layers.
5) Network traffic on the application layer should also be captured, or in other words: values of signals in the messages must also be learned—this is of course a very difficult task which can only be solved if some a-priori information about the system is known.

B. Generation of Test Cases

From a top-down perspective test vector generation is rather simple: Based on a given model M of a system S, test cases (i.e. system stimuli) are generated. These test cases, if applied to S, take S into a large number of significant states or situations. Often a criterion f_c measuring the number of reached system states–called the coverage criterion–is used to express the quality of the test vector generation (see also [25], [20]).

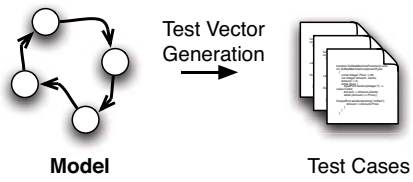

Fig. 2. Test vector generation from models.

Figure 2 shows a test vector generation process: Test cases or test vectors are generated from M in a way that maximizes f_c.

A typical application scenario is system migration: parts of an existing system are replaced or modified. After the modification, the system should show the same performance as before. For this, a model M of the previous system behavior is used to generate test cases that check the modified system's performance. As described in section II-A, M might have been learned from system observations.

This leads to some requirements on M:

1) Often thousands of test cases are generated. But M normally comprises significantly more states—often even an infinite number of states. So M should give hints which test cases are important.
2) A test case takes S into one specific state by sending a sequence of stimuli events $E = (e_1, \ldots, e_p)$ to S. So M should allow for an easy generation of E.
3) The timing of the stimuli events often plays a crucial role: Depending on the arrival time of an event $e_i \in E$, S reaches a different state. So M should also express the event timing.

C. Diagnosis Network Failure

The detection of system deterioration and of non-normal system behavior will prove essential for the improvement of system quality and reliability: Production facilities could detect failures earlier and inspections can be planed whenever first sign of deterioration appears.

The key idea is rather simple: Let M be a system model which captures all fault relevant system aspects. Whenever, later on M's behavior shows a discrepancy to the system's current behavior, the user will be alarmed.

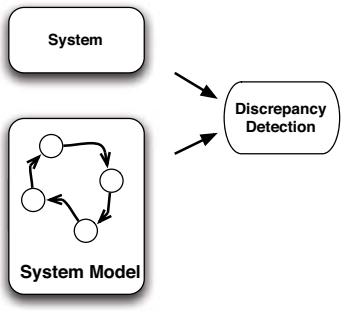

Fig. 3. Detection of system deterioration.

This can also be seen in figure 3: During the runtime of the system, the predictions of the model M are constantly compared to the system behavior and discrepancies are detected.

This application defines some requirements to M's formalism:

1) M must capture the most important fault-relevant system aspects for our domain: (*i*) Signal timing, (*ii*) Signal orders, and (*iii*) Signal values.
2) It must be able to evaluate M in real-time.

III. Timed Automata: State of the Art

Finite automata have gained popularity in the field of system analysis and engineering. Their formal nature allows for an unambiguous communication between engineers and enables computers to use it for e.g. verification, test vector generation, early simulation, etc. These formal models provide a good mechanism to model system behavior and the resulting event sequences. Nevertheless, temporal information cannot be specified using the original formalism. Such finite automata are based on the assumption of 'zero time', i.e. a transition from one state to another takes zero time units. However, for many applications (e.g. real-time embedded systems) timing information are vital.

Inclusion of timing information to finite automata would enable developers to model complex requirements like (*i*) to

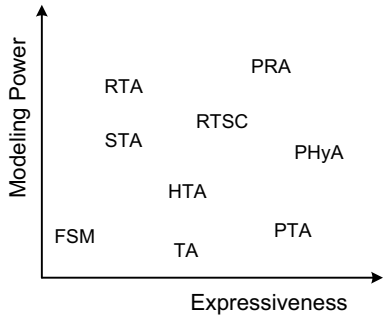

Fig. 4. Potential of Timed automata formalisms.

model transition durations, (*ii*) to verify if a specific state will be reached within a specific period of time, (*iii*) to model scenarios where a specific state shall not be reached after a certain time period, etc.

These requirements lead to timed automata (TA). Timed automata were initially proposed by Alur and Dill in [3]. A timed automaton accepts input alphabets associated with occurrence times $((w_1, t_1), (w_2, t_2), ..., (w_n, t_n))$, where w_i is the input alphabet, t_i is the time at which it occurs and n is the number of inputs.

Such a timed automaton can e.g. later be used for test vector generation and system analysis. This enables engineers to formally verify the model during the early stages of design and development. Early verification of the model helps developers identify design flaws like deadlock or livelock situations, violation of time constraints, etc.

Several timed automata have evolved after the initial proposal in [3], some of these are 'Hierarchical timed automata' (HTA) [8] (section III-B), 'Real-time statecharts' (RTSC) [5], [14] (section III-C), 'Scenario timed automaton' (STA) [13] (section III-D), 'Real-time automata' (RTA) [11], [27] (section III-E), 'Probabilistic timed Automata' (PTA) [19] (section III-F), 'Probabilistic Hybrid Automata' (PHyA) [2], [26] (section III-G), etc.

Here, the automata formalisms are classified based on their contribution to the categories "modeling power" or "expressiveness". 'Modeling power' is the ability of the automaton to be user friendly, i.e. the higher the modeling power, the easier it is for a developer to model a scenario—i.e. the closer is the formalism to the user's mental model. E.g., hierarchical automata can be better handled by a user than large, flat models.

An automaton with high 'expressiveness' can model more complex languages or problems, e.g. including probability information in an automaton will improve the expressiveness of the automaton because engineers can model more complex scenarios such as non-determinism.

Figure 4 shows the expressiveness and modeling power of the various timed automata formalisms. Note — that the graph is not on any particular scale, it is solely provided to give a qualitative impression of the expressiveness and modeling power of the various timed automata formalisms.

A. Basic Timed automata

A timed automaton (TA) [3], [12] $\mathcal{A}$ is a 6 tuple $(S, S_0, F, \Sigma, C, E)$, where S is a finite set of states, $S_0 \in S$ is the initial state, $F \subseteq S$ is a distinguished set of final states/ accepted states, Σ is a finite set of input alphabets, C is a finite set of clocks and E is a set of transition functions of $\mathcal{A}$. A transition from state s to s' over $a \in \Sigma$ is represented with $(s, a, \lambda, \delta, s')$, where λ specifies the clocks to be reset after this transition and δ is the clock constraint over C. It is interpreted as: the automaton changes from s to s' reading the input a, if the current clock value satisfies δ. It should be noted that as soon as this transition takes place the clocks of λ are reset to 0, thus starting to count time with respect to the time of occurrence of this transition.

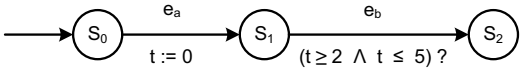

Fig. 5. Timed automaton example.

Figure 5 shows a simple automaton which has an event set $\Sigma = \{a, b\}$. The automaton starts in state S_0 and upon receiving the input letter a the automaton moves from S_0 to S_1, and the clock t is reset. While remaining in state S_1, on receiving the input letter b, the automaton makes a transition from state S_1 to S_2, if the current value of the clock satisfies the clock constraint $(t \geq 2) \wedge (t \leq 5)$.

Pros: TA as compared to finite automata provide means to specify temporal information on transitions.

Cons: TA (*i*) do not allow state invariants and hence the duration for which the automaton stays within a certain state cannot be modeled directly, (*ii*) allow usage of multiple clocks and thus reduces modeling power (refer section IV-B), (*iii*) only support modeling deterministic timed automata, and (*iv*) support only modeling flat structures.

B. Hierarchical Timed Automata

Hierarchical timed automata (HTA) are based on hierarchical state machines. They were proposed by David and Möller in [8] based on the preliminary work done in [9]. Hierarchical state machine are state machines that contain substates within states. Modularity and encapsulation are some concepts of hierarchical structures that scale them well for industrial automation.

Pros: HTA (*i*) support complex hierarchical states, and (*ii*) are suitable for communication system.

Cons: HTA (*i*) do not support complex data and operations, (*ii*) only support modeling deterministic timed automata, and (*iii*) allow multiple clocks.

C. Real-Time State chart

Real-time statecharts (RTSC) [5], [14], [15] are a blend of statecharts and timed automata. It uses real-time constraints expressiveness of timed automata to specify time constructs and restrictions to describe real-time behavior of models.

RTSC like usual statecharts are made up of states and transitions where states are extended with annotations like: worst case execution times for operations to be performed while entering (entry() operations), while exiting (exit() operations), while being in the state (do() operations), clock resets bonded with entry() and exit() operations, the periods associated with do() operations and time invariants. It supports a well-defined real-time semantics based on timed automata which helps overcome the underlying zero-time execution assumption for side-effects/ actions of usual statecharts which is predominantly unrealistic and conflicts with the actual implementation of the high level UML model on the available platform.

Unlike other timed automata, firing a transition in a RTSC consumes time [6], [7]. Hence, transitions are associated with worst case execution times and deadline. In addition to that, priorities can be assigned to transitions, so that mutually exclusive multiple transitions triggered at the same time can be handled.

Pros: RTSC (*i*) support hierarchy and parallelism apart from temporal behavior, (*ii*) allow specification of priorities on transitions, (*iii*) support specifying temporal information both as absolute time and relative time, (*iv*) provide means to specify operations to be executed while entering/exiting a state and also while the automaton stays in a state, and (*v*) contain necessary information for code generation.

Cons: RTSC (*i*) allow multiple clocks, and (*ii*) allow state invariants which can be a threat to temporal correctness of the model (refer section IV-A).

D. Scenario Timed Automata

Scenario timed automaton (STA) [13] was introduced to enable model developers to verify complex real time requirement of system models which consists of several software components or software modules made up of individual timed automaton which are connected together and synchronized to constitute the entire system. Scenario timed automaton is used to formulate query scenarios which consists of the dynamics of the situation to be verified. It supports the concept of global clocks which is very important when it comes to verification of synchronized multiple individual timed automaton. Basic timed automaton on the other hand lacks the concept of global clocks, hence, it proves to be insufficient to verify scenarios that have triggering events and target states located in different timed automaton.

Pros: STA (*i*) allow both local and global clocks, and (*ii*) help verifying complex timing requirements which involves more than one automaton.

Cons: STA (*i*) only support modeling flat structures, (*ii*) allow multiple clocks, and (*iii*) non-deterministic timed automata cannot be modeled.

E. Real-time Automata

Real-time automaton (RTA) [11], [27] contains time constraints expressed as relative time interval with respect to the previous symbol. It has only one clock that represents the time delay between two consecutive events in terms of relative time (unlike other timed automata which have finite number of clocks and clock guards for each transition – for specifying timing constraints) and the guard of the transition represents the constraint on the time delay.

Pros: RTA (*i*) allow only a single clock, and (*ii*) all timing information are specified as relative time.

Cons: RTA (*i*) have very limited expressiveness; as absolute timing information cannot be specified, (*ii*) only support modeling flat structures, and (*iii*) non-deterministic timed automata cannot be modeled.

F. Probabilistic Timed Automata

A timed automaton given its next state and input symbol along with its time of occurrence, is said to be deterministic iff its next state after transition can be uniquely determined. Hence, even multiple transitions starting from the same state with the same label can be deterministic, provided that their clock constraints are mutually exclusive so that only one of the clock constraint can be satisfied at a given point in time.

However, when observing real-world systems, it can be seen that many industrial communication systems [17], [16] have scenarios which cannot be modeled with a deterministic timed automaton. Hence, there is a need to address non deterministic scenarios.

Probabilistic timed automaton (PTA) [19] extends timed automaton with discrete probabilistic information on transitions. Hence, the transition of the automaton from one state to another is now governed by a probability distribution. The inclusion of probabilistic information increases the expressiveness of the automaton to a considerable extent, as it allows engineers to model non deterministic models, where the choice of the transition path is based on the probability distribution. Further, it also incorporates invariant conditions to specify upper bounds on the time within which certain probabilistic choices have to be made. Such automata can be seen as a special type of Markov processes (see [21], [22]). Markov processes have been introduced in 1907 by A. A. Markov and are an established formalism to describe probabilistic, dependent stochastic experiments.

Pros: PTA (*i*) allow modeling non-determinism using probabilistic information, and (*ii*) have high expressiveness.

Cons: PTA (*i*) allow defining multiple clocks, (*ii*) only flat structures can be modeled, and (*iii*) have limited modeling power (as shown in figure 4).

G. Probabilistic Hybrid Automata (PHyA)

Standard automata are used to model (and therefore to predict) variables with only discrete values, continuous variables cannot be modeled well. But many real-world applications comprise both: discrete and continuous variables. To model such hybrid systems, R. Alur introduced in 1995 hybrid automata (PHyA, see [2], [26]). These automata extend Alur's timed automata (see section III-A) by adding differential equations to states. I.e. as long as a system remains in a state, the behavior of variables is described by a set of differential equation. Usually ordinary differential equations (ODEs) such as $\dot{x} = f(x,t)$ (x is a variable, t is the time) are used, so gradients of variables over time are specified. In many

cases hybrid automata are restricted to linear hybrid systems which only support constant gradients; this subclass of hybrid systems still allows for the application of some formal model verification procedures.

Pros: PHyA (*i*) allow modeling non-determinism, and (*ii*) allow for the modeling of continuous variables.

Cons: PHyA (*i*) work with multiple clocks, (*ii*) support only flat structures, (*iii*) allow state invariants, and (*iv*) can not be verified formally in the general case.

Please note, a detailed survey of the above discussed formalisms is provided in [18]. Further, Table I summarizes the various automata discussed in this paper.

IV. Probabilistic Regression Automaton

Three main points render most of the formalisms from section III unsuitable for the tasks from section II:

1) The formalisms do not prevent the modeling of erroneous situations, i.e. of situations where it is neither permissible to remain in a state nor to leave it. This poses a problem for machine learning algorithms. Details are given in section IV-A.
2) When models are learned automatically from samples, often periodic processes and events are encountered. This means that absolute time values are often irrelevant and relative time information should be used instead. This point is discussed in section IV-B.
3) For all formalism from above (with the exception of hybrid automata), it holds true that within one state, system values must not change. For systems with only discrete values, i.e. boolean signals, this does not pose a problem. But with such approaches, systems comprising continuous or analogue signals can only be modeled using a very large number of states—whereat the states approximate the analogue signal characteristics. The formalism introduced here incorporates ideas from hybrid automata (see section III-G)) but allows for arbitrary classification functions within states. This improves the learnability of the automata and renders the automata to be better suited for capturing statistical system behavior while Alur's hybrid automata focus very much on verification scenarios. Details are given in section IV-C.

To overcome these shortcomings, the so-called Probabilistic Regression Automata (PRA) is introduced in section IV-D.

A. Temporal Correctness

Timed automata use state invariants to control the timing of the stimuli, i.e. when the system is in a certain state and waits for an input alphabet, the state invariants decide how long the automaton can wait in that particular state.

For e.g., figure 6 shows an automaton with state invariants, where $(t \geq 2) \wedge (t \leq 5)$ and $(t = 10)$ are the clock constraints of the transitions and $(t \leq 10)$ is the state invariant. Upon reaching state S_0, the clock t is reset to zero and the system can either (a) remain in the current state S_0—letting time to elapse, or (b) make a discrete transition to the next state, if an expected input letter arrives and the clock constraints are satisfied by the current value of the clock. Alternatively, at any point in time, if remaining in the current state and letting time to advance would violate the state invariant $(t \leq 10)$ then the system must make a transition to the next state; upon arrival of the expected input letter (provided that the transition constraints are met by the current clock value).

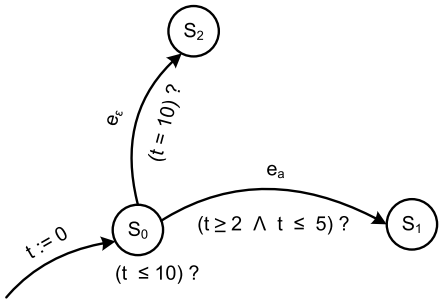

Fig. 6. Timed automata with state invariants.

Thus, the concept of state invariant is quiet useful to control the duration for which the automaton stays in a certain state. However, state invariants can also be a threat to the temporal correctness of the automata. For e.g., by changing the state constraints in figure 6 from $(t \leq 10)$ to $(t \geq 20) \wedge (t < 100)$ an erroneous situation is created: the automaton is neither allowed to stay in the state S_0 nor can it leave the state. Thus, it can be seen that even minor flaws in the model or improper usage of state invariants can lead to temporal inconsistencies.

One of the main targets of the proposed timed automata is its "learnability". Since state invariants can be a threat to the temporal consistency of the automaton and may complicate the learning process, it was decided not to introduce state invariants in the proposed PRA formalism.

B. Annotation of Time

Almost all timed automata discussed in section III use several independent clocks to specify timing constraints. The usage of more than one clock[1] causes several practical modeling problem: (*i*) The model developer has to keep track of the various clocks along with the states in which they were reset, so that the developer can specify the timing constraint for the current transition correctly. (*ii*) The modeling of periodic scenarios and events is often easier using relative time annotations.

Figure 7 shows a system made up of 3 individual automaton (namely automaton 'A', 'B' and 'C'). These 3 automata work in parallel to constitute the behavior of the system, thus the timing behavior of the entire system is characterized by the hand in hand operation of the involved clocks; namely, clock 'z' of automaton A, clock 'x' of automaton B and 'y' of automaton C. Automaton 'A' communicates with automaton 'B' with event e_a and with automaton 'C' using the event e_c. Similarly, automata 'B' and 'C' communicate with each other using the event e_b.

[1]Please note, that time for those set of clocks of course increases homogeneously. But because of the different time offset for the clocks, user often experience modeling difficulties.

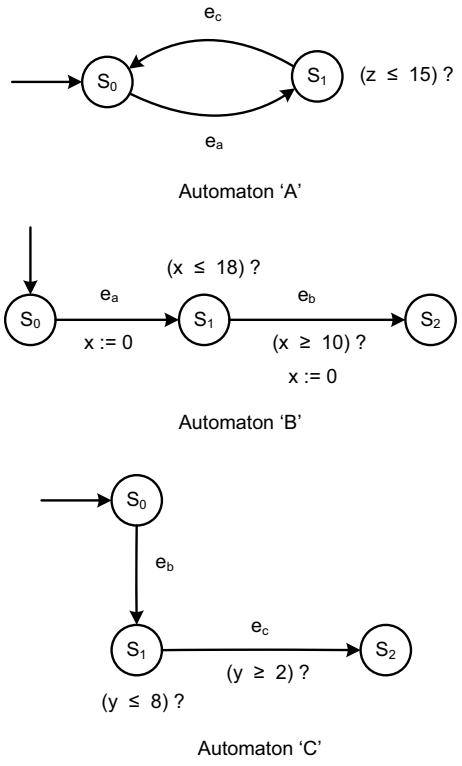

Fig. 7. A system with automata working in parallel.

These three automata when observed separately, their timing information does not seem to pose any threat to the working of the system. However, when they have to be combined functionally to model the behavior of the entire system, the system would run into a erroneous situation, as the timing information of the clocks 'x' and 'y' do not go hand in hand (please check the clock constraints / guard condition of automaton 'B' and the state invariant of automaton 'C').

To avoid such a scenario, the model developer should have kept track of the clock 'x' and its timing information before specifying the timing constraints of automaton 'C'. This problem of multiple clocks not working hand-in-hand can lead to time consuming - tedious modeling problems when it comes to larger systems. Thus, the usage of multiple clocks reduces the user friendliness or the modeling power of the formalism.

In addition to the above mentioned problem, several research works [1], [3], [10] show that the complexity of the verification problem of timed automata is exponential on the number of clocks used in the model. Thus posing an important obstacle to the development of verification algorithms which are efficient in practice [10].

Considering the above mentioned factors, it was decided to introduce a single global clock in the proposed PRA formalism. Further, both time spent in states and timing constraints are modeled as relative timing only; each timing information relates to the last event. Please note, that this relative timing formalism does not allow for the modeling of more complex scenarios; it only makes modeling easier and less error prone.

C. Support for Continuous Systems

Let's first review the problem: Figure 8 shows a typical situation: while most signals in automation systems have discrete values, some signals show analogue characteristics; e.g. y depends on x. Please note, that in many cases x will be time.

Conventional automata as described in section III (except hybrid automata) offer only one way to model such continuous signals: the signal value range is split into several intervals–often automatically–and several corresponding states are created; this is depicted in the upper right corner of figure 8. Such a solution creates many rather "dummy" states and increases the model size dramatically.

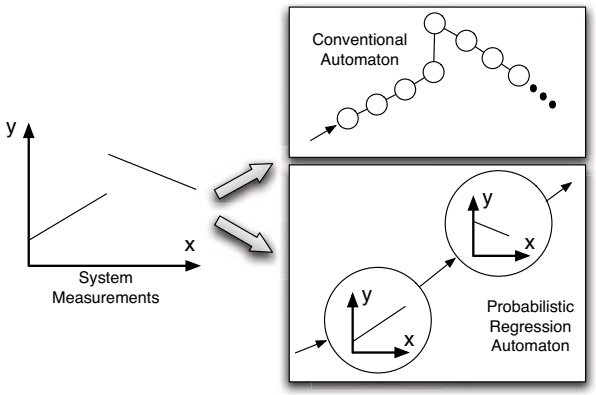

Fig. 8. An example for the application of a probabilistic regression automation to the modeling of non-discrete systems.

The PRA introduced in this paper solves this problem differently: areas with "harmless" signal characteristics are identified, e.g. in figure 8, linear pieces of the function $y(x)$. For each "harmless" area one state is created; within each state the function $y(x)$ is specified. Initially $y(x)$ is learned from the measurements, e.g. the linear functions in figure 8 can be learned using linear regression. This solution is shown in the lower right corner of figure 8: only two states are needed anymore.

Please note, that this approach works also for discrete systems: In that case a function $y(x) = c, c \in \mathbf{R}$ is learned.

D. Probabilistic Regression Automata

The motivation behind PRA is to develop a statistical automaton with temporal information which can be (*i*) learned automatically and (*ii*) can be used for test vector generation (see also section II). PRA should enable engineers to formally verify their models apart from formal description. The factors discussed in section II establish certain key features for the proposed PRA formalism. They are as follows:

- PRA allows only one clock and this clock is a global clock.
- By (*i*) annotating delay time interval at the transitions, by (*ii*) using only relative timing information for durations, and by (*iii*) not using state invariants, only temporal correct scenarios can be modeled.

- Probabilistic information is supported so that non-deterministic automata can be modeled.
- By adding functions for computing signal values to states, continuous variables are supported.

Definition: Probabilistic Regression Automaton (PRA)
A PRA is a tuple $A = (S, S_0, F, \Sigma, E)$, where

- S is a finite set of states,
- $S_0 \in S$ is the initial state,
- $F \subseteq S$ is a distinguished set of final states/ accepted states,
- Σ is the alphabet. Elements from Σ trigger transition. The alphabet is constructed as follows: $\Sigma = \{v \rho c\}$ with $\rho = \{<, \leq, =, \geq, >\}$, $c \in \mathbf{N}$ and $v \in V$. V denotes the set of network signals.
- $E \subseteq S \times \Sigma \times S$ gives the set of transitions.
- Delay interval $\delta : E \rightarrow \mathbf{N} \times \mathbf{N}$ models the time spent in a state before the transition takes place. δ always refers to the time spent since the last event occurred. It is expressed as $[\delta_{low}, \delta_{up}]$; which expresses that the execution of the transition has to be finished at least by δ_{up} and at the earliest by δ_{low}.
- Probabilities $p : E \rightarrow \mathbf{R}$ assigns a probability to edges. For each state $s_1 \in S$ $\sum_{(e=(s_1,s_2)\in E)} p(e) = 1$ holds.
- Functions $f_{s \in S} : V \cup \{t'\} \rightarrow V$. I.e. f_s is the function computing values for all networks signals in V. For this computation, all other networks signal values and t' can be used. Because f is normally learned from the initial network measurements, linear functions or decisions trees are often used to define f. t' denotes the time spent since entering the state.

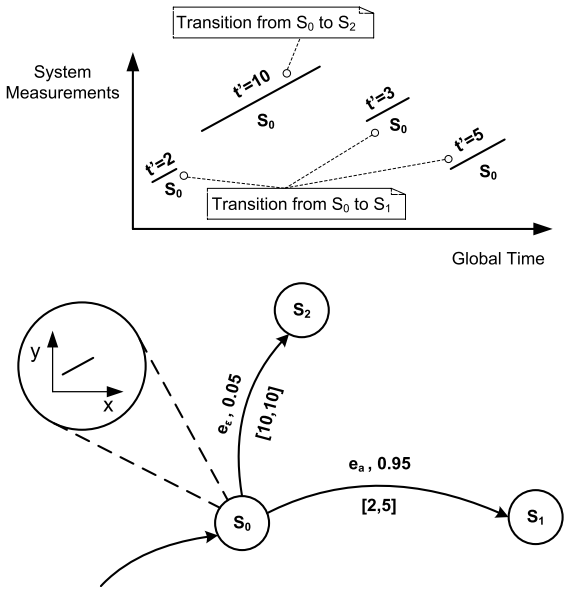

Fig. 9. Automaton based on PRA formalism.

A transition $e \in E$ in this automaton is given by the tuple $\langle s, a, s' \rangle$, where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger event. Additional information about the transitions are provided by the functions p (probability of the transition) and δ (time delay interval). Thus a transition $\langle s, a, s' \rangle$ is interpreted as follows: whenever the automaton is in state s, upon reading the input event $a \in \Sigma$, it will move to state s'. p is then the probability that a transition is taken, i.e. it denotes the percentage of cases (providing a sufficiently number of cases) in which a transition is used.

For e.g. figure 9 shows an automaton modeled using PRA formalism. The graph shown in the upper half of the figure shows the system under consideration. It can be noticed that the time spent in state S_0 before reaching state S_1 ranges between 2 and 5 time units. This information is depicted in the automaton as delay interval $[2, 5]$ on the transition between S_0 and S_1. Similarly, the graph also shows that the time required by the automaton to move from state S_0 to S_2 on receiving the input letter, is 10 time units. This duration is shown in the automaton as $[10, 10]$ as δ_{low} and δ_{up} are equal. Further, the percentage values 0.95 and 0.05 on each edge gives the probability of the respective transitions.

Figure 9 also shows how the duration for which the system stays in a certain state can be controlled even without using state invariants and in turn avoid potential threat to the temporal correctness of the automaton. Since all timing information are specified as relative time t' w.r.t their time of arrival at a certain state; it eliminates chances of modeling erroneous automata with temporal errors.

As mentioned earlier, Table I summarizes the characteristics of the various automata discussed in this paper.

V. Empirical Results

As a first step, this new formalism was applied to the analysis and learning of the network behavior of one specific industrial production research facility. Some first and preliminary results will be described shortly in the rest of this section.

Fig. 10. Pick and Place Robot research facility.

The system under study consists of a pick and place robot, a conveyor belt and different workpieces to be sorted as shown in figure 10. The whole communication is based on the real-time Ethernet standard PROFINET [23], [24]. The robot system consists of 34 sensors, 28 actuators, 4 Input/Output devices, 1 programmable logic units and deals with 72 network

TABLE I
TIMED AUTOMATA COMPARISON

Timed automata	Supported Features								
	Flat structures	Hierarchical structures	No. of Clocks	Absolute time	Relative time	State Invariants	Time Model	Non-determinism	Value changes within a state
Basic Timed Automata	✓	—	multiple	✓	—	—	dense	—	—
Hierarchical Timed Automata	✓	✓	multiple	✓	—	✓	dense	—	—
Real-Time State chart	✓	✓	multiple	✓	✓	✓	dense	✓	—
Scenario Timed Automata	✓	—	multiple	✓	—	✓	dense	—	—
Real-time Automata	✓	—	single	—	✓	—	dense	—	—
Probabilistic Timed Automata	✓	—	multiple	✓	—	✓	dense	✓	—
Probabilistic Hybrid Automata	✓	—	multiple	✓	partially supported	✓	dense	✓	✓
Probabilistic Regression Automata	✓	—	single	—	✓	—	dense	✓	✓

signals. The workpieces are transported by the conveyor belt to their end position; where they are picked up by the robot-arm and placed at the beginning of the belt. Furthermore, the pieces are equipped with an RFID Tag containing their sort ID and a counting variable. The variable is incremented whenever the work piece passes the RFID reader and will be sorted out if the variable is equal to its sort ID.

The learning algorithm itself is not the main subject of this paper but generally speaking an extension of algorithms such as described in [4], [28] is used. The general idea is shown in figure 11: First the network traffic, i.e. all messages, of the robot is captured and stored as a "pcap" file. A small utility reads this file and extracts the hardware topology and communication matrix, i.e. the information which bus device is sending which signals. So the general system structure is generated automatically. This is possible because the PROFINET protocol comprises all necessary information.

Then the values for these signals are extracted and stored as signal values over time. This information is the input for the PRA learning algorithm—this algorithm is implemented using the "R" tool.

For the learning process, a network trace containing 2

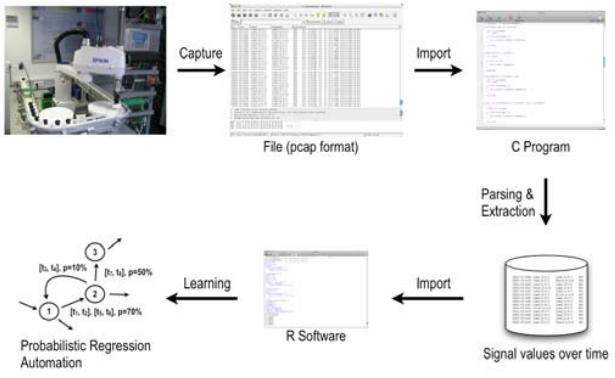

Fig. 11. The general procedure for learning PRA automata.

million messages has been used. The behavior of discrete variables has been learned with an error (on the training set) of $< 5\%$. The average error for continuous signals is (on the learning set) $< 9\%$. The time difference between two subsequent signal has been learned with an error of $< 13\%$. All in all the learned PRA contains 124 states whereat each state is accessed on an average of 14 times. I.e. each state could describe more than 1000 messages.

Especially the learning of the signal timing has proven to be much easier using the formalism introduced here then with traditional formalisms. Furthermore, continuous signals can be introduced in the learning process—which is a big advantage of our model over traditional formalisms. Of course, these are only first and early results—but results indicating a significant potential of the Probabilistic Regression Automaton.

The learned model has then been used to analyze the network behavior:

- Worst-case timing has been checked for signal transmissions. For this, the "sending" state and the "receiving" states are identified (manually) and the longest path in the automata (according to the execution time) is computed.
- Since the PRA are a special type of Markov chains, the probability for reaching error states can be computed. Again, error states have been identified manually.
- By using the probabilities and the execution times for all paths between two states, average transmission times have been computed. Compared to usual worst-case analysis, these results have proven to be very informative.

Using the techniques from above, a thorough analysis of the timing behavior of the robot facility has been done. This analysis can e.g. be used as a basis for future system modifications.

VI. SUMMARY AND OUTLOOK

Timed automata are a powerful means for modeling and verifying real-time system behavior. However, when it comes to, (*i*) learning models and (*ii*) test vector generation the existing formalisms seem to lack simplicity. They often pose

difficult challenges due to their complexity. Some of these challenges were discussed in sections II and IV. To address these challenges, this paper has presented the so-called probabilistic regression automata which has a simpler formalism but a formalism powerful enough to solve the above mentioned problems. PRA formalism enables developers to use a simple formalism (which supports only a single clock, relative timing information and does not allow state invariants) for learning models of network traffic and to describe statistical behavior of a large set of scenarios (statistical models) which can later be used for test vector generation and quantitative analysis. Further, the automata also includes regression functions which allow modeling value changes within a state–so reducing the number of necessary states.

Future work includes the algorithmic objectives of learning network traces for constructing statistical models and test case generation from statistical network models. Furthermore, the quality of the test vector generation has to be verified using a coverage criterion (e.g. all transition-based, all transition-pair-based, all state-based, all data-flow-based, etc.). Since each criterion has its own advantages and disadvantages, an appropriate criterion which suits this domain has to be chosen.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, vol. 126:183–235, 1994.

[4] M. M. F. Bugalho and A. L. Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38(9):1457–1467, 2005.

[5] S. Burmester. Generierung von java real-time code für zeitbehaftete uml modelle. Master's thesis, University of Paderborn, Department of Computer Science, Paderborn, Germany, September 2002.

[6] S. Burmester and H. Giese. The fujaba real-time statechart plugin. In H. Giese and A. Zündorf, editors, *Proc. of the first International Fujaba Days 2003, Kassel, Germany*, pages 1–8. University of Paderborn, October 2003.

[7] S. Burmester, H. Giese, M. Hirsch, D. Schilling, and M. Tichy. The fujaba real-time tool suite: Model-driven development of safety-critical, real-time systems. In *Proc. of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA*, pages 670–671. ACM Press, May 2005.

[8] A. David and M. O. Möller. From huppaal to uppaal: A translation from hierarchical timed automata toflat timed automata. Technical report, BRICS, BRICS Report Series RS-01-11, March 2001.

[9] A. David and W. Yi. Hierarchical timed automata. Contacts: adavid@DoCS.uu.se, yi@DoCS.uu.se, 2000.

[10] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 73–81, Washington, DC, USA, 1996. IEEE Computer Society.

[11] C. Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, Vol. 6, Issue 1:3 – 23, January 2001.

[12] J. S. Dong, P. Hao, and et.al. Timed automata patterns. *IEEE Transactions on Software Engineering*, Vol 34(No. 6), December 2008.

[13] M. Gehrke, P. Nawratil, O. Niggemann, and W. S. M. Hirsch. Scenario-based verification of automotive software systems. In H. Giese, B. Rumpe, and B. Schätz, editors, *Proc. of the Dagstuhl-Workshop: Model-Based Development of Embedded Systems(MBEES), 9.-13.1.2005, Schloss Dagstuhl, Germany*, pages 35–42, 2006.

[14] H. Giese and S. Burmester. Real-time statechart semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Germany, June 2003.

[15] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the compositional verification of real-time uml designs. In *Proc. of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-11)*, pages 38–47. ACM Press, September 2003.

[16] B. Kumar, J. Imtiaz, and J. Jasperneite. Applicability of uml marte's schedulability sub-package for engineering industrial real-time protocols. In *2nd Junior Researcher Workshop on Real-Time Computing (JRWRTC 2008) (in conjunction with the 16th International Conference on Real-Time and Network Systems (RTNS 2008))*, Rennes, France, Oct 2008.

[17] B. Kumar and J. Jasperneite. Industrial communication protocol engineering using uml 2.0: a case study. In *7th International Workshop on Factory Communication Systems (WFCS 2008)*, Dresden, Germany, May 2008.

[18] B. Kumar, O. Niggemann, and J. Jasperneite. Timed automata for modeling network traffic. In *Machine Learning in Real-Time Applications (MLRTA 09) (in conjunction with 32nd Annual Conference on Artificial Intelligence (KI 2009))*, Paderborn, Germany, September 2009.

[19] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[20] P. Mosterman, J. Ghidella, and E. O'Brien. Model coverage as a quality measure and teaching tool for embedded control system design. In *Frontiers in education conference - global engineering: knowledge without borders, opportunities without passports, 2007. FIE '07. 37th annual*, pages T3J–1–T3J–6, Oct. 2007.

[21] J. R. Norris. *Markov Chains*. Cambridge University Press, 1997.

[22] P. Bremaud. *Markov Chains - Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer Verlag, 1999.

[23] PNO. Profinet specification iec 61158-5-10 (v2.1), 2007.

[24] PNO. Profinet specification iec 61158-6-10 (v2.1), 2007.

[25] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. One evaluation of model-based testing and its automation. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 392–401, New York, NY, USA, 2005. ACM.

[26] J. Sproston. Decidable model checking of probabilistic hybrid automata. In *FTRTFT*, pages 31–45, 2000.

[27] S. Verwer, M. de Weerdt, and C. Witteveen. An algorithm for learning real-time automata. In M. van Someren, S. Katrenko, and P. Adriaans, editors, *Proc. of the Sixteenth Annual Machine Learning Conference of Belgium andthe Netherlands (Benelearn)*, pages 128–135, 2007.

[28] S. E. Verwer, M. M. de Weerdt, and C. Witteveen. Efficiently learning simple timed automata. In W. Bridewell, T. Calders, A. K. de Medeiros, S. Kramer, M. Pechenizkiy, and L. Todorovski, editors, *Induction of Process Models*, pages 61–68. University of Antwerp, 2008. Workshop at ECML PKDD.