On the Learning of Timing Behavior for Anomaly Detection in Cyber-Physical Production Systems

Alexander Maier[1] and **Oliver Niggemann**[1,2] and **Jens Eickmeyer**[1]
[1]Fraunhofer Application Center Industrial Automation IOSB-INA
e-mail: {alexander.maier, jens.eickmeyer}@iosb-ina.fraunhofer.de
[2] inIT - Institute Industrial IT
e-mail: oliver.niggemann@hs-owl.de

Abstract

Model-based anomaly detection approaches by now have established themselves in the field of engineering sciences. Algorithms from the field of artificial intelligence and machine learning are used to identify a model automatically based on observations. Many algorithms have been developed to manage different tasks such as monitoring and diagnosis. However, the usage of the factor of time in modeling formalisms has not yet been duly investigated, though many systems are dependent on time.

In this paper, we evaluate the requirements of the factor of time on the modeling formalisms and the suitability for automatic identification. Based on these features, which classify the timing modeling formalisms, we classify the formalisms concerning their suitability for automatic identification and the use of the identified models for the diagnosis in Cyber-Physical Production Systems (CPPS). We argue the reasons for choosing timed automata for this task and propose a new timing learning method, which differs from existing approaches and we proof the enhanced calculation runtime. The presentation of a use case in a real plant set up completes this paper.

1 Introduction

Many learning algorithms have been developed for the identification of behavior models of CPPS, e.g. [1], [2], [3]. However, most of the learning algorithms do not include timing information, not least because the modeling formalisms do not consider timing information.

Indeed, technical systems mostly depend on time, e.g. the filling of a bottle or the moving of a part on a conveyor belt. Therefore, many applications (such as the anomaly detection) require a model with timing information. Some faults only can be detected using timing information (especially degradation faults, e.g. a worn conveyor belt runs slower).

In this paper, we use the term "Cyber-Physical Systems (CPS)" for "systems that associate (real) objects and processes with information processing (virtual) objects and processes through open, partly global, anytime interconnected information networks". Further, a CPPS is a CPS in the context of an industrial production environment.

In this paper, we give a taxonomy of modeling formalisms. These formalisms are evaluated according to specific features. The taxonomy is then used to evaluate whether the models can be identified automatically and used for anomaly detection.

Based on this evaluation, we present a timing learning method, which is used to learn the timing behavior as timed automaton. In contrast to other approaches, we use the underlying timing distribution function to differentiate between transitions with equal events which belong to different processes.

By calculating the computation runtime, we prove that our approach runs faster than other existing methods for timed automaton learning.

The presented learning method is used in an exemplary plant setup to demonstrate the suitability for anomaly detection in CPPS.

The paper is organized as follows: In Section 2 we evaluate some timing learning features and give a taxonomy of how these features are met by three categories of timing modeling formalisms, namely *(i)* Dynamic system models, *(ii)* Operational formalisms and *(iii)* Descriptive formalisms. In Section 3, we argue why we use timed automata as formalism, point out some challenges in timed automaton learning and present our timing learning approach. Further, we prove formally the enhancement of the calculation runtime of our approach. Section 4 completes the contribution with the presentation of a use case in a real plant. Finally in Section 5, we conclude this paper with a short discussion and give an outlook to future work.

2 Classification of Timing Learning Features and Algorithms

The modeling of time for computation purpose is a widely researched area (e.g. in [4], [5] and [6]). Many formalisms have been created to model different aspects of timing behavior. In this paper, some aspects are analyzed which have to be considered when choosing an appropriate timing modeling formalism. Based on this analysis, some modeling formalisms are evaluated according to their capabilities to model the timing behavior. One of those formalisms is chosen that is well suited for the anomaly detection in CPPS.

To keep the application domain in mind, a special focus is on modeling and identification of the timing behavior of CPPS. Additionally, the suitability of the modeling formalisms according to automatic learnability from observations only and the suitability for anomaly detection is evaluated.

2.1 Evaluation of Timing Modeling Features

Before choosing an appropriate timing modeling formalism some key issues have to be considered, which are listed below. Some of that and additional features are given in [5] where the authors provide a comprehensive analysis on timing modeling features and corresponding modeling formalisms is given.

Discrete or dense time domain

The separation of formalisms concerning the usage of discrete and dense time domains is a first natural categorization. Discrete time models comprise a set of isolated points, whereas dense time means that in a dense set, ordered by "<", for every 2 points t_1 and t_2 with $t_1 < t_2$ there is always a third point t_3 in between, such that $t_1 < t_3 < t_2$.

Explicit or implicit modeling of time

Another major distinctive feature is the possibility of implicit and explicit modeling of time. Model formalisms with explicit time allow the modeling of concrete time values for some specific event, e.g. "if the sensor is activated, start the conveyor belt within two seconds". Implicit modeling of time only gives information about the time duration as a whole.

One clock or many clocks

Furthermore, time model formalisms can be differentiated according to their number of used clocks. When dealing with independent modules within a system, the question arises whether to use one or many clocks. The usage of many clocks leads to the need of clock synchronization in the simulation step, whereas the usage of one clock only requires a transformation from an n-clock model to a 1-clock model.

Concurrency and composition

Most real systems are too complex to model them in one overall model. The behavior has to be divided into several subsystems, so that the overall model is a composition of its sub-models. For finite state machines, the number of states reduces enormously if the system is decomposed into subsystems. This is also referred to as modularization.

The decomposition is a less mature process. Difficulties can arise in the synchronization step. Mostly, the separated models of subsystems have equal or identical properties. Furthermore, the time bases can be different between the modules, discrete or continuous, or the time base is implicit for one module and explicit for another.

Single-mode and multiple-modes

The distinction between models, which can only cope with single-modes and models that additionally can deal with multiple-modes, goes a step deeper than concurrency and decomposition. A system may, at some point in time, abruptly change its behavior. In technical systems, this happens for reasons such as shifting a gear or stopping a conveyor belt. All state based models (e.g. statecharts, Petri nets or finite state machines) are able to describe multiple-mode systems, where equation based formalisms (e.g. ordinary differential equation) can only describe the behavior of single-mode systems.

Linear- and branching-time models

A difference can also be made between linear and branching time models [7]. Linear-time formalisms are interpreted over linear sequences of states. Each description refers to (a set of) linear behaviors, where the future behavior from a given state at a given time is always identical. Branching-time formalisms are interpreted over trees of states. That means, in contrast to linear-time models, the future behavior of a given state at a given time can follow different behavior according to the tree.

A linear behavior can be regarded as a special case of a tree. Conversely, a tree can be treated as a set of linear behaviors that share common prefixes (i.e., that are prefix-closed); this notion is captured formally by the notion of fusion closure [8]. Thus, linear and branching models can be put on a common ground and compared.

2.2 Taxonomy of Timing Modeling Formalisms

Mainly, the timing modeling formalisms can be subdivided into three categories: *(i)* Dynamic system models, *(ii)* Operational formalisms and *(iii)* Descriptive formalisms:

Dynamic system models

In various engineering disciplines (like mechanical or electrical) and especially in control engineering, the so-called *state-space representation* is a common way to model the timing behavior of technical systems [9].

Three key elements are essential for the state-based representation: The vector $\mathbf{x}$ with the state variables, the vector $\mathbf{u}$ with the input variables and the vector $\mathbf{y}$ with the output variables. All these values explicitly depend on the time at which they are evaluated (usually represented as $\mathbf{x}(t)$, $\mathbf{u}(t)$, and $\mathbf{y}(t)$), however, the timing information is not explicitly described in the form as *"the filling of the bottle takes five second"* i.e. it uses implicit timing.

The main advantage of dynamical system models is that very detailed physical models can be created using established mathematical methods. But this also can turn into a disadvantage. For many purposes, the models are too detailed, i.e. they are unsuitable for high-level description, since some expert knowledge is required to read and understand the models. As proposed in [10], dynamical systems can be used for the diagnosis of distributed systems.

Various methods exist to identify dynamic system models. These methods are grouped under the term model identification (sometimes the term "system identification" is also used), although, the model is not identified completely, but a structure model is presumed and the identification methods only determine the parameters. So, still some expert knowledge is necessary and manual work has to be done. In [6], Isermann describes some methods, e.g by means of parameter estimation. The states itself are not identified.

Dynamic system models also can be used for fault detection (e.g. [11]). The model-based fault detection uses the inputs $\mathbf{u}$ and the outputs $\mathbf{y}$ to generate residuals $\mathbf{r}$, the parameter estimates $\mathbf{\Phi}$ or state estimates $\mathbf{x}$, that are called features. A comparison of these features with the nominal values (normal behavior) detects changes of features, which lead to analytical symptoms s. The symptoms are then used to determine the faults.

Despite their suitability for the modeling of timing behavior, dynamic system models can hardly be learned automatically based on observations only, since the structure of the model has be given and mostly only the parameters are identified.

Operational Formalisms

Operational formalisms further can be subdivided into *(i)* synchronous state machines and *(ii)* asynchronous abstract machines:

Synchronous state machines:

A large variety of synchronous state machines exists: finite state machine, statecharts, timed automaton, hybrid automaton, Büchi automaton, Muller automaton, and others (see [12]). Here, we confine our self to finite state machines and timed automata, the timing extension of finite state machines.

The main strength and the reason for the wide usage of finite state machines is their accessibility for humans and their simplicity. Often, processes or timing behavior are described by a sequence of events. In fact, technical systems are often programmed in state machines, e.g. using the standardized programming language from IEC 61131. Therefore, modeling the timing behavior of such technical systems, in the sense of finite state machines or timed automata, is consequential.

Some algorithms already exist to identify timed automata from observations (e.g. in [13] , [14], [15], [16], [1]). Most automata identification algorithms are based on the state merging method. The basic procedure is illustrated in Figure 1. It works as follows:

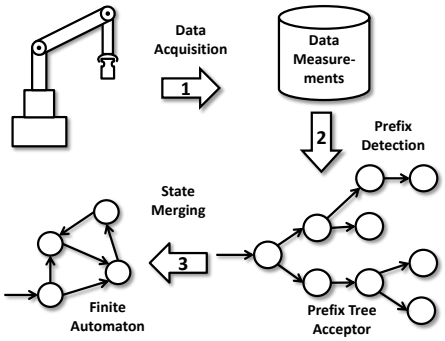

Figure 1: The principle of offline automaton learning algorithms using the state merging approach.

First, in step (1), the data is acquired from the system and stored into a database. In step (2), the observations are used to create a prefix tree acceptor (PTA) in a dense form, whereas equal prefixes are stored only once. Then, in step (3), in an iterative manner all pairs of states are checked for compatibility. If a compatible pair of states is found, the states are merged. In [13], additionally a transition splitting operation is introduced, which is executed when the resulting subtrees are different enough. The result is a finite automaton the generalizes the observed behavior in an appropriate way.

Finite state machines can also be used for fault detection and diagnosis (e.g. in [17], [18], [19]). Depending on the used formalism, different errors can be detected: wrong event sequence, improper event, timing deviation and error in continuous signals.

Asynchronous abstract machines:

Beside the finite state machines, which work synchronously, there exist formalisms that work asynchronously, called the asynchronous abstract machines. The most popular formalism in this group is Petri nets.

Petri nets are named according to Carl Adam Petri, who initially developed this modeling formalism [20]. A variety of Petri nets exists [21]. The most common type is place/transition-nets. It basically consists of states and transitions. Places store tokens and hand them over to the transitions. If all incoming places hold at least one token, a transition is enabled. An enabled transition will be fired. After firing the transition, tokens from incoming transitions are moved to outgoing transitions.

Petri nets also have been extended to handle timing information. Merlin and Farber proposed the first Timed Petri net in [22]. Each transition is extended with the minimum and maximum firing time, where the minimum firing time can be 0 and the maximum can be ∞. A comprehensive survey on several timed extensions to Petri nets can be found in [23] and [24].

Furthermore, several approaches exist to identify Petri nets from sampled data. However, some requirements are put on the language to be identified or some assumptions are made, e.g. in [25], Petri nets are identified from knowledge of their language, where it is assumed that the set of transitions and the number of places is known. Only the net structure and the initial marking are identified.

Petri nets in general are suited for fault detection (e.g. in [26] or [2]). The different types of Petri nets (mainly condition/event-systems, place/transition-nets and high-level Petri nets) have different time and space complexity.

Descriptive Formalisms

As the name suggests, descriptive formalisms describe the model using a natural language, mostly based on mathematical logic [27]. Such formalisms are especially suited if some conditions have to be described.

Example 1. *If it is raining or if it was raining in the last two hours, then the street is wet.*

Similar rules can also be created for the prediction of output signals (actuators) based on the inputs (sensors) in a CPPS.

As already shown in Example 1, the conditions can also contain time information.

There exist different types of descriptive formalisms, e.g. first order logics, temporal logics, explicit-time logics or algebraic formalisms. Further details can be found in the literature, e.g. [27].

Some algorithms exist to identify descriptive models. For the prediction of the behavior of CPPS, a timed decision tree can be learned for instance. Examples for such learning algorithms are ID3 [28], the C4.5 algorithm as extension of the ID3 algorithm [29] or a generic algorithm for building a decision tree by Console [3].

Note that the rule can not always be interpreted backwards. Using Example 1, a reason for the wet street could be that somebody has washed his car on the street. Therefore, descriptive formalisms have a limited suitability for anomaly detection. The usage of descriptive formalisms for anomaly detection puts additional requirements on the rules, they have to be more concrete. Using the given example, it can be modified as follows:

Example 2. *The street is wet if and only if it is raining or it was raining in the last two hours.*

This rule allows a backward interpretation, if the reason for the wet street is unknown. However, the meaning of the

rule has now changed. Additionally, these kind of rules is hardly identifiable from observations only.

Comparison of Modeling Formalisms

Table 1 shows how the mentioned timing modeling features are met by the corresponding modeling formalisms.

It can be seen that operational and descriptive formalisms allow a similar level of timing modeling, while dynamic system models differ in nearly all features. In contrast to the other formalisms, dynamic system models use a dense time domain, only allow implicit modeling of, time, use one clock only and can model linear time models.

Please note the different possibilities to handle concurrent behavior. Petri nets are the first choice for this task. Using tokens, concurrent behavior can be modeled in one model. Timed automata and hidden Markov models (HMM) are able to decompose the behavior in several subsystems.

3 Automaton Learning

The decision of which formalism to use is based on several factors. These can differ based on the individual use case. Here, we consider the models to be used for learning and diagnosis of CPPS.

Despite there exist several algorithms for the identification of timed behavior, it can be seen in Table 2 that the usage of timed automata is a good choice.

- **Understandability:** In contrast to many other automatically identified models, the identified finite state machines can be better understood by third persons. They can be verified by experts.
- **Wide usage:** Finite state machines are widely used, e.g. for modeling or programming.
- **Learnability:** Finite state machines are suitable for automatic learning. The goal is to use as few expert knowledge as possible.
- **Diagnosability:** Finite state machines are suitable for fault detection. This applies for both, manually created and automatically identified finite state machines.
- **Suitability for verification:** The identified finite state machines can be used for automatic verification.
- **Modification:** The identified finite state machine can be manually modified and adapted after learning. This can also be done automatically.

3.1 Challenges in Automaton Learning

Some algorithms have already been introduced for the identification of timed automata, see Section 2.2. However, there are still some challenges in learning timed automata. This applies in particular to the time factor.

- **Identification of states and events:** The timing behavior includes not only the time stamps for some observations, but also some states and transitions with timed events in between. Many learning algorithms (especially for learning of Markov chains) assume the states and transitions as given and only learn the transition probabilities. Here, the structure (states and events) is not given but has to be identified from observations.
- **Timing representation method:** Additionally, an appropriate timing representation method has to be chosen, which is able to correctly describe the technical processes. At the beginning of Section 3.2 we review some state of the art timing representation methods and propose our solution.
- **Relative or absolute time base:** The time base is also a very important issue. The base can be either absolute e.g. referred to the beginning of a production cycle or relative to the last event.
- **Number of clocks:** Technical systems may be programmed using a certain number of clocks. These have to be identified or the behavior has to be expressed using only one clock.

  Timed automata allow both, one and many clocks. However, in [13] Verwer showed that 1-clock timed automata and n-clock timed automata are language-equivalent, but in contrast to n-clock timed automata, 1-clock timed automata can be identified efficiently.
- **Event splitting:** When do events with different timing belong to the same event, or do they describe different events? As can be seen in Figure 2, the events can be split based on the timing, which is based on the container size: The robot needs more time to move the big container compared to the small one, this is captured in the given probability distribution function over time. More formally: The event's timing distribution function can comprise several modes that have to be identified.

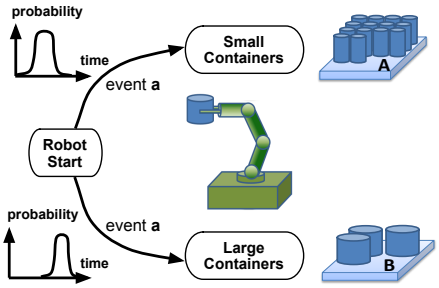

Figure 2: The timing behavior changes based on the container size.

- **Event splitting or timing preprocessing:** Continuing from the previous point, additionally the question arises that whether the modes are identified during the learning process itself or whether a preprocessing can be used to identify multiple modes and use this information in the learning process, avoiding the additional splitting operation.

3.2 Timed Automaton Learning Algorithm

Several algorithms have been introduced to learn an automaton based on observations of the normal behavior only. While most automaton identification algorithms do not consider time (e.g. MDI [30] and Alergia [31]), recently only few algorithms have been introduced that identify a Timed Automaton. RTI+ [13] and BUTLA/HyBUTLA [16] learn in an offline manner, i.e. first the data is acquired and stored and then the automaton is learned. However, for the case that observations cannot be stored, an online learning algorithm is desirable, which includes each observed event online, without a preprocessing. OTALA [1] is an extension of BUTLA and learns a timed automaton in an online manner.

Table 1: Taxonomy of the timing modeling features and how they are satisfied by the corresponding modeling formalisms.

	operational Formalisms			descriptive Formalisms	Dynamic system models
	Timed Automata	HMM	Petri nets	e.g. Rule-based system	e.g. state space representation
Discrete or dense time domain	**discrete**	discrete	discrete	discrete	dense
Explicit or implicit modeling of time	**explicit**	explicit	explicit	explicit	implicit
One clock or many clocks	**one/many**	one/many	one/many	one/many	one
Concurrency and composition	**++**	++	+++	+	+
Single-mode and multiple-modes	**single/ multiple**	single/ multiple	single/ multiple	single	single
Linear- and branching-time models	**linear/ branching**	linear/ branching	linear/ branching	linear/ branching	linear

Table 2: Satisfiability of the mentioned properties by different timing modeling formalisms.

	Timed Automata	HMM	Petri nets	Rule-based system	State space representation
Understandability	**+++**	++	++	+++	+
Wide usage	**+++**	+++	++	++	++
Learnability	**+++**	++	+	+++	+
Diagnosability	**+++**	++	++	++	++
Suitability for verification	**+++**	+++	++	++	+
Modification	**+++**	++	++	+++	+

A crucial issue for the modeling formalism of timed systems is the representation of the timing information. Usually, timed automata use a single clock only and therefore a relative time base is required, where a relative time stamp represents the passed time from entering until leaving a state. The timing information is annotated in the transition next to an event. The usual way is to use intervals recording the minimum and maximum observed time values for a specific event [13], [14], [15], [1].

RTI+, the first algorithm for the identification of timed automata [13], included a transition splitting operation in addition to the merging operation. The timing in the transitions is represented with histograms using bins and uniform distribution [13]. During the state merging procedure, it is also checked, whether a transition can be split. A transition is split when the resulting subtrees are different enough. However, the splitting operation is associated with a high calculation time, since depending on the bin size, all possible splits have to be calculated. The disadvantage of this approach is that the bin size has to be set manually by experts. Further, it does not take the underlying distribution into account.

In contrast to other existing algorithms for the identification of timed automata, our proposed identification algorithm BUTLA [16] uses probability density functions over time (PDFs) to express the timing behavior. Unlike other approaches, we base our decision on the timing information itself, not on the subtree resemblance.

The identification algorithm BUTLA follows the methodology from Figure 1. Additionally, instead of the splitting operation, a preprocessing step is introduced, which identifies the timing behavior and captures different behavior patterns as shown in Figure 2.

Timing preprocessing

The timing of events is analyzed in a preprocessing step. The relative time values of each event are collected in a histogram. It is decided whether the timing behavior is subdivided into multiple modes based on this histogram and the resulting probability density distribution over time. In case of multiple modes, an event is separated according to the number of modes in the PDF such that each event consists of only one mode. For instance, an event e_i with 2 modes is separated into $e_{i,1}, e_{i,2}$, as can be seen in Figure 3.

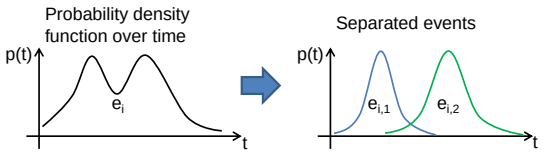

Figure 3: An event with a multi-mode timing behavior is separated into its modes.

For the detection of multiple modes in events, three methods have been evaluated:

- *Kernel density estimation:* This version is straight forward by estimating the density of the distribution function and subdividing at local minimums. It is optimized for efficient computation time. Nevertheless it delivers useful results.
- *ExpectationŰmaximization (EM) - algorithm:* This method is well-known from the state of the art. It per-

forms well, but the number of mixed distribution functions has to be known or determined subsequently by trying all values and take the best fitting.

- *Variational Bayesian inference:* This version has the weakest performance but delivers the best results. The number of overlapping distribution function is calculated in an iterative manner.

Due to the high computation effort of the EM-algorithm and Variational Bayesian inference, we chose to use the kernel density estimation for the timing preprocessing in BUTLA. The determination of the timing modes using the kernel density estimation works as follows:

First, for each event e, all timing values $t_1, t_2, ..., t_k$ are collected and stored in a list $\{e, \{t_1, t_2, ..., t_k\}\}, k \in \mathbf{N}$ is the number of collected timing values for one event. Then, the PDFs are calculated using the kernel density estimation method for each event. Density estimation methods use a set of observations to find the subjacent density function. Given a vector $\mathbf{t}$ with the time values of the observations, the underlying density distribution for a time value t can be estimated as

$$f(t) = \frac{1}{N} \sum_{i=1}^{N} k(\mathbf{t}_i; t) \tag{1}$$

where $N \in \mathbf{N}$ is the number of time values in the vector of observations and $k(\mathbf{t}_i; t)$ is a non negative kernel function

$$\int_{-\infty}^{\infty} k(\mathbf{t}; t)dt = 1. \tag{2}$$

As underlying probability distribution, we use the Gaussian distribution, which is defined as:

$$G(\mu, \sigma^2, t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{3}$$

where σ^2 is the bandwidth (smoothing factor), μ the mean value and t is the time value, for which the probability is calculated.

The choice of the bandwidth is important for the correctness of the results and it is the subject of research in different publications (e.g. [32]). In the case of identifying the normal behavior of production plants, it is useful not to use a fixed value for smoothing factor but to keep it variable. Here, the variable smoothing factor is 5% of the current value. This results in the greater variance for greater time values and smaller variance for smaller time values. Therefore, the density is estimated as:

$$f(t) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi} \cdot 0.05\mathbf{t}_i} e^{-\frac{(x-t)^2}{2 \cdot 0.05\mathbf{t}_i}}. \tag{4}$$

In the next step the local minimums in the calculated PDF are localized. One mode is assumed to be between the local minimums.

Finally, referring to the original data (discrete time values) and based on the assumption of normally distributed data, the needed statistic parameters (mean μ and standard deviation σ) are calculated. This is done for each mode: between the minimum value, all local minimums and the maximum value.

Using this preprocessing of the timing information, the time-consuming splitting operation during the state merging procedure is not necessary, since the transitions are already split according to the identified timing modes.

3.3 Analysis of the Timing Preprocessing

Figure 2 illustrates that a state can be a starting point for different processes: When the robot is started, it depends on the size of the containers that which of the sub-trees is taken for the further process, based on the time that is needed to move the container. Different possibilities exist to identify the different timing behavior of the sub-trees.

The algorithm RTI+ uses a splitting operation, which calculates a p-value for all possible splits and its sub-trees. If the lowest p-value of one split is less than 0.05, the transition is split.

Figure 4 illustrates the problem of the splitting operation. The main drawback of using the splitting operation is that it requires additional computation time. First, all possible splits have to be evaluated. Based on the number of observations, these can be a huge amount. And after finding the best splitting point based on the smallest p-value, the transition has to be split. Here, for all postfixes of the corresponding transition, it has to be decided that which path to follow. Since all these paths are mixed in the previous states, the information that which path follows which states, based on the original data, has to be stored somehow. This leads to a huge memory consumption. To avoid this high memory consumption, RTI+ renews the prefix tree acceptor beginning with the corresponding state after each splitting operation. However, this is still time and space consuming.

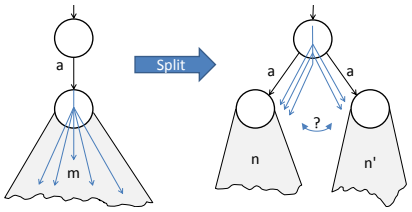

Figure 4: The problem of the splitting operation.

Proposition 1. *The time complexity of calculating and performing a splitting operation is $\mathcal{O}(m^2 \cdot n^2)$, where m is the number of input samples and n is the number of states in the PTA.*

Proof. For each transition (in worst case there are $n - 1$ transitions in the PTA, if it is a linked list of states with only one input sample or all input samples follow the path), the p-value has to be calculated (which has to be done for each input sample using the certain transition). Therefore, the complexity for calculating the p-values is $\mathcal{O}(m \cdot n)$.

One splitting operation itself also needs time in $\mathcal{O}(m \cdot n)$ for the creation of the PTA with m input samples, where each can have n states.

In the worst case, if each transition has to be split, the complexity is in $\mathcal{O}(m^2 \cdot n^2)$. □

BUTLA firstly uses a preprocessing of timing values to avoid this splitting operation. This version is based on the assumption that events with the same changing signals but different timing behavior describe different behavior.

In the preprocessing step, events with multiple timing modes are identified. These modes are used for the creation

of the prefix tree. Events with the same symbol but arising from different timing modes are handled as different events and lead to different states in the prefix tree. In the identification phase, these events are also handled as different. Using this preprocessing step, the splitting process can be omitted. This leads to a computation speed increase.

Proposition 2. *The time complexity of calculating the timing modes in a preprocessing step is in $\mathcal{O}(n)$, where n is the number of observed events.*

Proof. Since this is during the preprocessing step and the PTA does not exist so far, the worst case is not dependent on the PTA structure, but only on the number of incoming events and the number of symbols.

First, the time stamps for each symbol in the alphabet $a \in \Sigma$ have to be collected. This takes time $\mathcal{O}(n)$.

Then for each $a \in \Sigma$, the probability density distribution over time has to be calculated. For this, Equation 4 is computed. Note that all events are not considered for a single symbol $a \in \Sigma$, but only those that belong to this symbol a. All computations together need time $\mathcal{O}(n)$. Additionally the local minimums have to identified, which is also done in $\mathcal{O}(n)$.

All these steps are performed subsequently and therefore the overall time complexity for the preprocessing step is $\mathcal{O}(n)$.

□

Using the preprocessing step, the computation time can be reduced compared to the splitting version. While the splitting version runs in polynomial time, we could reduce this additional timing computation to linear time using the preprocessing step.

4 Learning Automata Results

As mentioned before, the goal of the identified automata is the usage for anomaly detection. An exemplary plant at the institute has been used for experimental results. Figure 5 shows a part of the Lemgo Model Factory and the identified models of two modules.

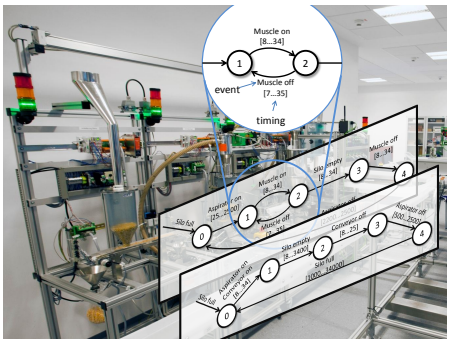

Figure 5: Example plant with identified models for two modules.

During the anomaly detection phase, the running plant's timing behavior is compared to the prognosis of the automaton. A timing anomaly is signaled whenever a measured timing is outside the timing interval in the learned timed automaton. Here, the interval is defined as $[\mu - k \cdot \sigma, \mu + k \cdot \sigma], k \in \mathbf{R}_+$ where μ is the mean value of the corresponding original observations' timings and σ is the standard deviation.

In a first experiment, the Lemgo Model Factory (see Figure 5) is used. A frequently occurring error for example is the wear of a conveyor belt which leads to a decrease in the system's throughput. 12 production cycles are used to identify a normal behavior model. The PTA comprises 6221 states. BUTLA reduces this to 13 states—this corresponds to a compression rate of 99.79%.

To verify the model learning algorithm with a high amount of data, in a second experiment, data is generated artificially using the modified Reber grammar (extended with timing information). 1000 samples are generated to learn the model, then 2000 test samples are created where 1000 comprise timing errors. From the initial 5377 states in the PTA, a model with 6 states is learned.

Table 3 shows the error rates for the anomaly detection applied to both data sets using different factors k in the timing intervals.

Table 3: Experimental results using real and artificial data.

	k =1	k =2	k =3	k =4
false negative rate (%) - LMF	2	5.3	12.8	30
false positive rate (%) - LMF	12	4.2	2	0
false negative rate (%) - Reber	0	1.3	7.5	21
false positive rate (%) - Reber	9	3.1	1.1	0

The experimental results in Table 3 show that the false positive rate could be reduced by enlarging the time bounds. But at the same time, the false negative rate rose. The application of the enlargement of the time requires a trade off between false positive and false negative rate. This has to be done separately for each application.

5 Conclusion

In this paper we analyzed the possibilities of learning the timing behavior for anomaly detection in CPPS. First, we gave a taxonomy of timing modeling formalisms. Based on this taxonomy we analyzed whether the models can be identified automatically and whether they are suitable for anomaly detection.

Timed automata are often the first choice for the modeling of timed behavior of CPPS, especially for the modeling of sequential timed behavior.

Due to the intuitive interpretation, timed automata are well-suited to model the timing behavior. In our proposed learning method, we used probability density distribution functions over time for the timing representation. In a preprocessing step multiple modes in single transitions are identified, this enables the omission of the time consuming splitting operation.

We proved the runtime enhancement formally and gave some experimental results which prove the practicability of timed automata for automatic identification and for anomaly detection.

References

[1] A. Maier. Online passive learning of timed automata for cyber-physical production systems. In *The 12th IEEE International Conference on Industrial Informatics (INDIN 2014)*. Porto Alegre, Brazil, Jul 2014.

[2] M.M. Mansour, M. Wahab, and W.M. Soliman. Petri nets for fault diagnosis of large power generation station. *Ain Shams Engineering Journal*, 4(4):831 – 842, 2013.

[3] L. Console, C. Picardi, and D.T. Dupré. Temporal decision trees: Model-based diagnosis of dynamic systems on-board. *CoRR*, abs/1106.5268, 2011.

[4] A. Maier. *Identification of Timed Behavior Models for Diagnosis in Production Systems*. PhD thesis, University of Paderborn, 2015.

[5] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi. Modeling time in computing: A taxonomy and a comparative survey. *ACM Comput. Surv.*, 42(2):6:1–6:59, March 2010.

[6] R. Isermann and M. Münchhof. *Identification of Dynamic Systems: An Introduction with Applications*. Advanced textbooks in control and signal processing. Springer, 2010.

[7] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 1–22, London, UK, 2001. Springer-Verlag.

[8] R. Alur and T. A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *In Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society Press, 1992.

[9] H. Khalil. *Nonlinear Systems*. Prentice Hall, January 2002.

[10] S. Indra. Decentralized Diagnosis with Isolation on Request for Spacecraft. In Astorga Zaragoza, editor, *Proceedings of the 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 283–288, August 2012.

[11] R. Isermann. Model-based fault detection and diagnosis - status and applications. In *16th IFAC Symposium on Automatic Control in Aerospace*, St. Petersbug, Russia, 2004.

[12] W. Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[13] S. Verwer. *Efficient Identification of Timed Automata: Theory and Practice*. PhD thesis, Delft University of Technology, 2010.

[14] M. Roth, J. Lesage, and L. Litz. Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. In *American Control Conference (ACC), 2010*, pages 2601–2606, June 2010.

[15] M. Roth, S. Schneider, J.-J. Lesage, and L. Litz. Fault detection and isolation in manufacturing systems with an identified discrete event model. *Int. J. Systems Science*, 43(10):1826–1841, 2012.

[16] O. Niggemann, B. Stein, A. Vodenčarević, A. Maier, and H. Kleine Büning. Learning behavior models for hybrid timed systems. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1083–1090, Toronto, Ontario, Canada, 2012.

[17] S. Tripakis. Fault diagnosis for timed automata. In Werner Damm and Ernst-Rüdiger Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 205–224. Springer, 2002.

[18] P. Supavatanakul, C. Falkenberg, and J. J. Lunze. Identification of timed discrete-event models for diagnosis, 2003.

[19] Z. Simeu-Abazi, M. Di Mascolo, and M. Knotek. Diagnosis of discrete event systems using timed automata. In *International Conference on cost effective automation in Networked Product Development and Manufacturing*, Monterrey, Mexico, 2007.

[20] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[21] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[22] P. M. Merlin and D. J. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, Sep 1976.

[23] A. Cerone. *A Net-based Approach for Specifying Real-time Systems*. Serie TD. Ed. ETS, 1993.

[24] A. Cerone and A. Maggiolo-Schettini. Time-based expressivity of time petri nets for system specification. *Theoretical Computer Science*, 216(1 - 2):1 – 53, 1999.

[25] M.P. Cabasino, A. Giua, and C. Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007.

[26] P. Nazemzadeh, A. Dideban, and M. Zareiee. Fault modeling in discrete event systems using petri nets. *ACM Trans. Embed. Comput. Syst.*, 12(1):12:1–12:19, January 2013.

[27] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[28] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[30] Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. of the 17th International Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, 2000.

[31] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *GRAMMATICAL INFERENCE AND APPLICATIONS*, pages 139–152. Springer-Verlag, 1994.

[32] Z. I. Botev, J. F. Grotowski, and D. P. Kroese. Kernel density estimation via diffusion. *Annals of Statistics*, 38(5):2916–2957, 2010.