Towards Adaptive Scheduling of Maintenance for Cyber-Physical Systems

Alexis Linard(✉) and Marcos L.P. Bueno(✉)

Institute for Computing and Information Sciences, Radboud University Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
{A.Linard,M.Bueno}@cs.ru.nl

Abstract. Scheduling and control of Cyber-Physical Systems (CPS) are becoming increasingly complex, requiring the development of new techniques that can effectively lead to their advancement. This is also the case for failure detection and scheduling component replacements. The large number of factors that influence how failures occur during operation of a CPS may result in maintenance policies that are time-monitoring based, which can lead to suboptimal scheduling of maintenance. This paper investigates how to improve maintenance scheduling of such complex embedded systems, by means of monitoring in real-time the critical components and dynamically adjusting the optimal time between maintenance actions. The proposed technique relies on machine learning classification models in order to classify component failure cases vs. non-failure cases, and on real-time updating of the maintenance policy of the sub-system in question. The results obtained from the domain of printers show that a model that is responsive to the environmental changes can enable consumable savings, while keeping the same product quality, and thus be relevant for industrial purposes.

Keywords: Model-based scheduling · Predictive maintenance · Machine learning · Cyber-physical systems

1 Introduction

Due to the growing complexity of Cyber-Physical Systems [11,19], many techniques have been proposed to improve failure detection and scheduling component replacement [17,21]. Indeed, new needs in terms of reliability and safety have appeared with the new applications of such systems. That is the reason why leading-edge technology manufacturers seek to design more robust and reliable systems [12]. Currently, a major issue in many industrial settings is how to correlate failure occurrences and the maintenance actions performed in order to prevent breakdowns. Modeling the failure behavior of many components in advance is an intricate task. Indeed, we claim that maintenance actions are frequently scheduled with fixed intervals that are suboptimal and implemented to the detriment of productivity and efficiency.

T. Margaria and B. Steffen (Eds.): ISoLA 2016, Part I, LNCS 9952, pp. 134–150, 2016.
DOI: 10.1007/978-3-319-47166-2_9

Exclusively relying on experts to build models that can describe the behavior of machines has been recently recognized as a limiting feature [14,23]. Therefore, the use of machine learning techniques to construct such models has been investigated [3,10,16,18]. However, using such techniques in order to update the maintenance scheduling of a CPS in real time has so far not been explored. The main difficulties in this case relate to finding an appropriate predictive model, and then defining a procedure for updating the timing conditions. Predictive models can be used instead of costly sensors intended to provide information about the state of the machine at any moment, which is an additional reason why we introduce machine learning techniques to maintenance scheduling. The ultimate goal would be to develop embedded systems capable of dynamically scheduling their own maintenance. To that end, we aim to define a procedure for updating in real time when maintenance should be performed. Our work is based on an experiment carried out in partnership with industry, specifically in the domain of printers. In addition, we consider the scope of *automatic* maintenance, where the intervention of human beings is no longer needed.

In this study, we investigate to what extent machine learning can help to improve fixed maintenance scheduling of complex embedded systems. The contributions of this paper are as follows. First, we propose using machine learning techniques, in which the embedded system learns to distinguish between failure vs. non-failure cases using data related to critical components of the CPS. This can be done by monitoring critical components in real time. Next, we propose an algorithm to dynamically adjust the timing of maintenance actions. This algorithm uses timed automata [2], which is the formalism used to model the maintenance policy. Indeed, timed automata can provide an intuitive representation of the maintenance policy and its real-time update is proceeded by using information on the overall printer at any moment. Thus, our main contributions are (a) the use of decisions from a data-driven model to dynamically schedule maintenance, and (b) the use of timed automata to formally describe and analyze the proposed algorithm. Naturally, we only select relevant features to determine if the printer is working properly – that is to say, if we can schedule the maintenance actions later – or not. To that end, we consider a set of realistic, industry-based scenarios and simulations to provide evidence that a reduced amount of maintenance can be done while achieving similar product quality [8,9]. The considered scenarios have been implemented in *Uppaal* [4], which is another relevant practical-oriented contribution of this paper.

The remainder of this paper is organized as follows. In Sect. 2 we present the industrial problem that motivated our approach. In Sect. 3, we define the key concepts associated with model-based scheduling and classification techniques used to separate the failure from the non-failure cases of a Cyber-Physical System, as well as discuss the related literature. In Sect. 4 we explain our approach to updating when to trigger maintenance and the experiments done, using a model-checking tool and data about large-scale printers. Finally, we discuss the results.

2 Case Study

Large-scale printers are cyber-physical systems made of a large number of complex components, the interaction of which is often challenging to understand. Among their main components are the printheads, which are composed of thousands of printing nozzles. These are designed to jet ink on paper according to specifications concerning, for example, jetting velocity and direction. During the operation of these industrial printers, nozzles can behave inadequately with respect to the demanded task, e.g. by jetting incorrect amounts of ink or jetting in an incorrect direction. If that is the case, a nozzle is considered to be failing.

Failing nozzles can be repaired by performing one or more maintenance actions, including for example different types of cleaning actions. The maintenance actions are executed automatically, e.g. the printer cleans its own nozzles. In this context, determining the appropriate moment to execute each nozzle-related maintenance action is crucial to achieving a proper balance of conflicting objectives, such as productivity, machine lifetime, and final product quality. However, the number of individual nozzles, their physical architecture, the substantial number of variables that can potentially be correlated to them and the definition of printing quality, make particularly difficult to manually construct models that express all the potentially relevant correlations among these variables and nozzles. Ultimately, this creates a challenge when designing maintenance policies, since they are intended to be either too conservative or tolerant, otherwise one or more of the mentioned requirements could be seriously degraded.

A solution that is sometimes used to construct a policy consists of performing a large set of tests involving different usages of the CPS, aiming to derive time-based maintenance policies. These policies are based on monitoring, hence they do not suffer so much from the drawbacks of fixed-time-based strategies [17]. However, these policies rely on time counters that are not directly related to the state of machine parts, e.g. the time elapsed since the last finished printing task and the period that the printer stands idle. This implies that the only failure behaviors that can be explicitly captured by these rules are those that were previously seen during the tests, usually only accounting for a fraction of all possible machine statuses. Hence, unseen failure behavior cannot be handled properly by such maintenance strategies, which is likely since these machines can be used with a wide range of parameter combinations. In other words, such policies are prone to perform blindly on at least some situations, which can lead to too many or too few maintenance actions.

Nowadays, industrial printers record large amounts of data about the state of their components over time, so a data-driven approach seems feasible. A data-driven approach can provide evidence that helps to decide on whether the current time parameters are adequate or not, given the current state of the CPS. In order to represent time parameters and system states, a state machine appears to be an appropriate formalism. In this study we show how to dynamically update the parameters of a maintenance scheduler, which is based on timed state machines. This formalism is used in order to capture the intuition that the CPS

moves between different states during its operation, in a time-dependent manner. A significant advantage of combining a statistical approach and state machines is that the real-time updating of timed parameters requires no human intervention, since correlations between potentially relevant variables can be learned algorithmically. In addition, the involvement of a failure behavior model is substantiated in our case, since there is no sensor available to directly provide, at any moment, information on the state of the nozzles. That is the reason why such a model could provide the desired outcome whenever required.

3 Background

In this section, we present the different concepts on which our definition of an adaptive scheduler is based. First, we describe how to model and verify maintenance schedulers. Then, we discuss the machine learning techniques used in our paper and to what extent the need for them is relevant for real-time updating of scheduling. Finally, we discuss related literature.

3.1 Modeling Maintenance Strategies

State machines are abstract machines with a wide range of applications such as in process modeling, software checking and pattern matching. They are composed of a set of states and transitions between states. They can be used in our industrial case study, that is to say modeling maintenance actions schedules of CPS, since it is possible to gather as many states as different maintenance actions plus one or more states representing when no maintenance action (MA) is being performed in the CPS. Transitions between the states would take place when a given maintenance is started and completed [1]. As shown in Fig. 1, the maintenance policy of a system can be represented intuitively by means of a specific type of state machine: a timed automaton (TA).

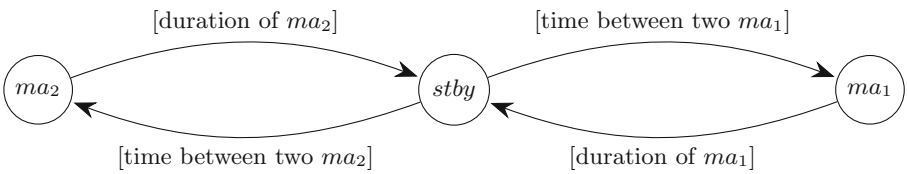


Fig. 1. Maintenance policy of a component represented by a simplified TA.

A timed automaton [2] is a finite-state machine extended with a finite set of real-valued clocks constraints. During the execution of a timed automaton, clock values increase at the same speed. A timed automaton has also clock guards, that enable or disable transitions and constrain the possible behaviors of the automaton. Furthermore, when a transition occurs, clocks can be reset.

The particular TA presented in Fig. 1 is a way of modeling a given maintenance strategy. In our case, we assume that maintenance actions are executed sequentially. In order to establish that the derived maintenance strategy achieves an optimal trade-off, many techniques exist, including model-checking of real-time systems [9]. We used the tool *Uppaal* [4] to evaluate time-monitoring-based maintenance strategies. More details about how this tool was used are presented in Sect. 4.2.

3.2 Classification Techniques

This study relies on the use of classification techniques [13], also known as supervised learning, belonging to the field of Machine Learning. These techniques consist of learning models from data. They are designed to classify instances into a set of possible classes, according to the values of the attributes of each instance. To that end, we used Weka [29], a suite of implemented learning algorithms. Among the possible classifiers that can be used for the classification task, we considered in particular: bayesian networks, naives Bayes classifiers, decision trees, random forests and neural networks.

The main reason for the choice of the classifiers listed above is related to our case of study. Indeed, most of them can be considered as *white box* classifiers, in the sense that it is easy to interpret their provided outcomes. This is particularly important in the context of industrial cases, because such a readable model can be more insightful than a *black box* oracle.

In order to learn a classifier from log data, labeled data from each possible class is needed. In the case of learning the failure behavior of the CPS of interest (i.e. nozzles of large-scale printers), we assume that nozzles are either failing or not failing. Thus, each instance is composed of a number of features corresponding to relevant machine components and the class feature indicating the occurrence of a failure or not.

actual→ / predicted↓	f	!f
f	A	B
!f	C	D

$$P_f = \frac{A}{A+C} \quad R_f = \frac{A}{A+B} \quad P_{!f} = \frac{B}{B+D} \quad R_{!f} = \frac{B}{C+D}$$

Fig. 2. Confusion matrix and evaluation criteria for the classes *failure* (denoted by f) and *non-failure* (denoted by !f).

In machine learning, an important goal when learning classifiers is that of properly generalizing to new data. To meet this goal, overfitted models should be detected, since they tend to perform very well on the data used during learning. This issue is often dealt by using the n fold cross-validation evaluation. Cross-validation with n folds consists of first dividing the dataset into n disjoint sets, then learning the model on the $n-1$ first folds and evaluating the learned model

on the last fold. Then, learning is done on folds 2 to n and evaluation is done on fold 1, and so on. The part of the data used to learn a model is usually called training data, while the part used to evaluate it is usually called test data. On each fold, a classifier is typically evaluated by comparing the predicted class provided by the classifier and the original class, as seen on each instance from the test set. The results are placed in the confusion matrix shown in Fig. 2, and from this matrix the metrics precision P and recall R are computed. After processing all the folds, the mean of precision and recall is taken, corresponding to the final result of cross-validation.

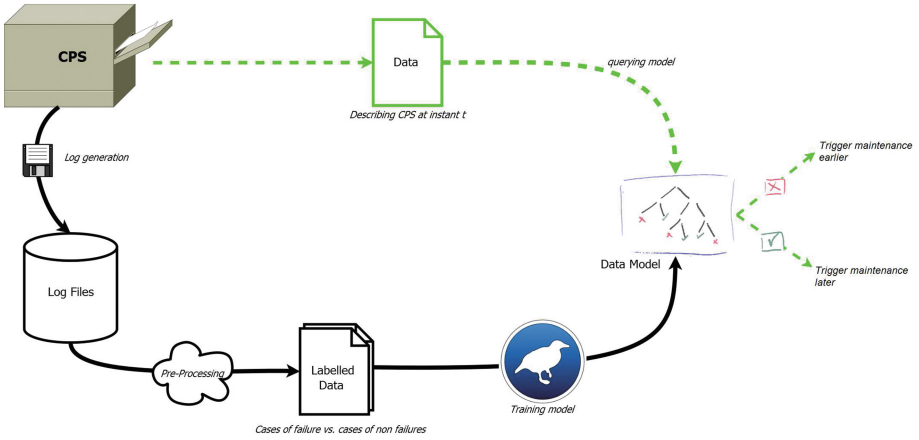


Fig. 3. The process of collecting log data for a CPS component, learning classifier models, and using the outcomes as input to reschedule maintenance.

Once a classifier has been trained, it is possible to use it in order to classify new unlabeled instances. The process is as follows: in real-time, new data about the same features used to train the model are recorded and represent the state of the CPS at instant t. Such state of the system corresponds to an instance to be classified by the model. Once the outcome is provided, a decision is taken by the real-time updating method controlling the schedule of maintenance actions. The overall process of learning and using the classifier in our case is described in Fig. 3.

3.3 Related Work

Extensive research has been done in the field of timed automata [2], including algorithmic and computability aspects [5,7] and model-checking oriented research [9]. To a lesser extent, research has been developed in the context of passive learning of TA from data [28], and learning sub-classes of TA known as event recording automata [15]. In the context of CPS, real-time online learning of TA has been investigated [22], however not related to predictive maintenance.

With regard to predictive maintenance, timed automata were applied to model the operating modes in the case of duration of tasks in manufacture scenario [27]. Statistical approaches to model the failure behavior of CPS have already been considered [20,24], but none of them combined the use of TA to model maintenance actions together with machine learning models.

4 Proposed Method and Experiments

In this section, we describe our method and the results of our experiments. Indeed, our hypothesis is that less maintenance can be done, and fewer or as many failures can occur. Hereinafter, we present the protocol observed and the data used in order to dynamically schedule nozzle-related maintenance[1].

4.1 Learning Nozzle Failure Behavior

The first step towards dynamic scheduling of nozzle maintenance actions is to build a failure behavior model describing how the nozzles of printers are likely to fail. In order to do so, we rely on all the logs of a set of 8 printers of the year 2015. The main advantage of the logs we have at our disposal is that a lot of metrics and nozzle-related factors are monitored continuously. Moreover, we also dealt with labeled data with data representing the nozzles at moments when they were considered as failing or not. The failing measure is the conjunction of several metrics related to the print quality. We stress again the key role of a failure behavior model, since labeled data are costly to obtain: the model reproduces the outcome provided by printing *test pages*, a process that inevitably leads to a loss of productivity on the overall printer. We thus trained the corresponding model by selecting relevant features. To this end, we benefited from the expertise of engineers related to the field, who indicated to us the possible relevant features.

In Table 1, we present the results achieved for failure detection (f). We state the results in terms of precision and recall. Precision (P) represents the proportion of failure cases as correctly classified. Recall (R) reflects the proportion of caught failure cases among the cases of failures and non failures. We also present the results achieved for the other class (non-failure – !f). Both results are important since both outcomes are used by our scheduler i.e. to advance or postpone maintenance. The classifiers have been trained with 117k instances to classify, the same features, and evaluated with a cross-validation of 10 folds. The set was divided into 10 % of instances belonging to the *failure* class and 90 % to the *non-failure* class.

In our experiments, we consider the results above as good enough to be considered as reliable. Moreover, we trained several classifiers (among others, decision trees, naive Bayes classifiers, neural networks, etc.), and the decision tree always performed the best. A decision tree is an interesting way of modeling the failure behavior of a component thanks to its high understandability. As a

[1] Experimental data available upon request.

Table 1. Quality of the best classification models trained with the data.

Classification algorithm	f		!f	
	P	*R*	*P*	*R*
Decision tree	0.788	0.631	0.951	0.977
Random forest	0.626	0.579	0.943	0.953
Bayesian network	0.683	0.809	0.973	0.949
Naive Bayes	0.329	0.424	0.918	0.882
Multilayer perceptron	0.652	0.224	0.903	0.984

consequence, we consider that we can safely schedule nozzle-related maintenance actions using the outcomes of the built Decision Tree.

In this case, we have used the J48 implementation of the C4.5 algorithm to learn it [25]. This algorithm builds a model from the training set using information entropy. It iteratively builds nodes choosing the attribute that best splits the current sample into subsets, using information gain. The attribute having the greatest information gain is chosen to make the decision, that is to say to be used as the next decision node. Of course, once a subset is only composed of instances belonging to the same class, no further node is created, but a leaf instead, standing for the final class of all the instances belonging to the subset.

The fact that a decision tree can be implemented easily by a succession of *if* conditions is another reason to consider it as premium model for industrial purposes. The lower quality of the results for the class *failure* is due to the low number of instances belonging to this class.

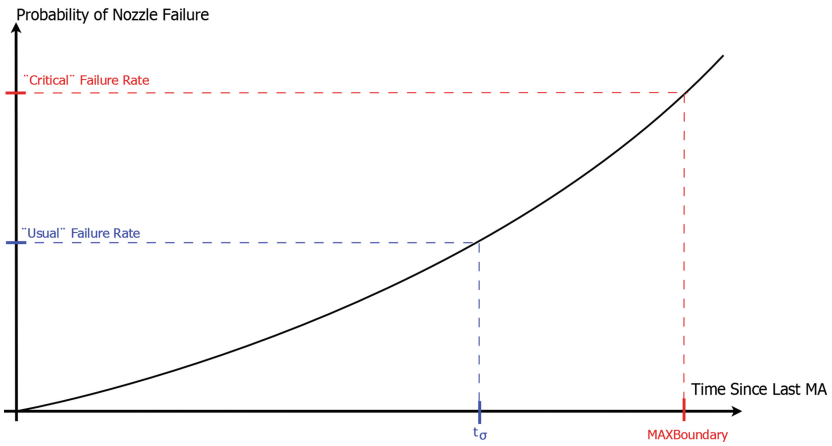


Fig. 4. Nozzle failure rate in function of the time since last MA.

4.2 Scheduling Nozzle-Related Maintenance Actions

The idea of dynamic scheduling of maintenance actions is to rely on the outcome of a predictor (component failure behavior model that classifies the system at instant t into two classes, failure or not failure) to put forward or backward the moment when to trigger it.

Algorithm 1. Dynamic Scheduling of a Maintenance Action

- q : query made periodically
- y : the reducing factor for the timing of the MA
- z : the increasing factor for the timing of the MA
- t_σ : the timing where the MA is usually triggered
- $currentBoundary$: the current timing when the MA will be triggered
- $MAXBoundary$: the maximum acceptable time to wait until triggering the MA

```
1: for each q do
2:     prediction ← failureBehaviorModelQuery()
3:     if prediction = failure then                    ▷ Advance the MA schedule
4:         currentBoundary ← currentBoundary × y%
5:     else                                            ▷ Postpone the MA schedule
6:         currentBoundary ← currentBoundary × z%
7:     if currentBoundary is reached then
8:         triggerMaintenance()
9:         currentBoundary ← t_σ OR (currentBoundary + t_σ)/2
```

As presented in Algorithm 1, we first of all consider the actual maintenance policy of the printer. Our idea is to query the classification model built previously in order to get the outcome desired. In case a failure is detected by our classification model, then the timing for all the maintenance actions triggering is decreased with a given rate y. Otherwise, if no failure is detected in the few moments before maintenance actions are triggered, then the timed boundary to enable a maintenance to occur is postponed with a given rate z. Assuming the use of a Decision Tree as classifier, we can notice here that the parameters y and z standing for how much to increase or decrease the maintenance clock guards can be function of the *confidence factor* provided with each outcome performed by the classifier. This confidence factor is based on the error rate of each leaf in the tree, hence it consists in a metric to assess how a provided outcome can be considered as reliable or not.

Finally, we distinguish two possibilities in our algorithm concerning how to reset the timed boundaries once the related maintenance has been performed. The first one consists in resetting the clocks guards to their initial values, e.g. defined in the original TA inferred from the specifications. The second one consists in resetting the future timed boundary by computing the average between the last used limit and the actual moment when the maintenance has been launched. In such a way, we assume that after several runs of the algorithm, the value computed will tend to the ideal time to wait between two maintenance actions. Both options are presented in Sect. 4.3 and the benefits between those two variants are compared.

4.3 Simulations Using Uppaal-SMC

In order to make simulations and evaluate the benefits of our dynamic scheduler, we used the *Uppaal-SMC* [8] modeling tool. *Uppaal* is a toolbox for verification of real-time systems represented by one or more timed automata extended with integer variables, data types and channel synchronization. The relevance of using *Uppaal-SMC* in our case lies in the possibility of running simulations of the system specified as a set of probabilistic timed automata.

One of the challenges of this work and its practical case study is to validate the failure behavior model built using machine learning techniques, and to use it in order to schedule maintenance actions. Of course, we had to find a way of integrating such a model in a tool like *Uppaal*, in order to benefit from the integration of the representation of the maintenance strategy (gathered from specifications of the component, and using TA), the decision tree providing insight on the failure behavior of the nozzles (learned prior to its implementation in *Uppaal* following description in Sect. 4.1), and the function that dynamically updates the triggering of maintenance (presented in Sect. 4.2).

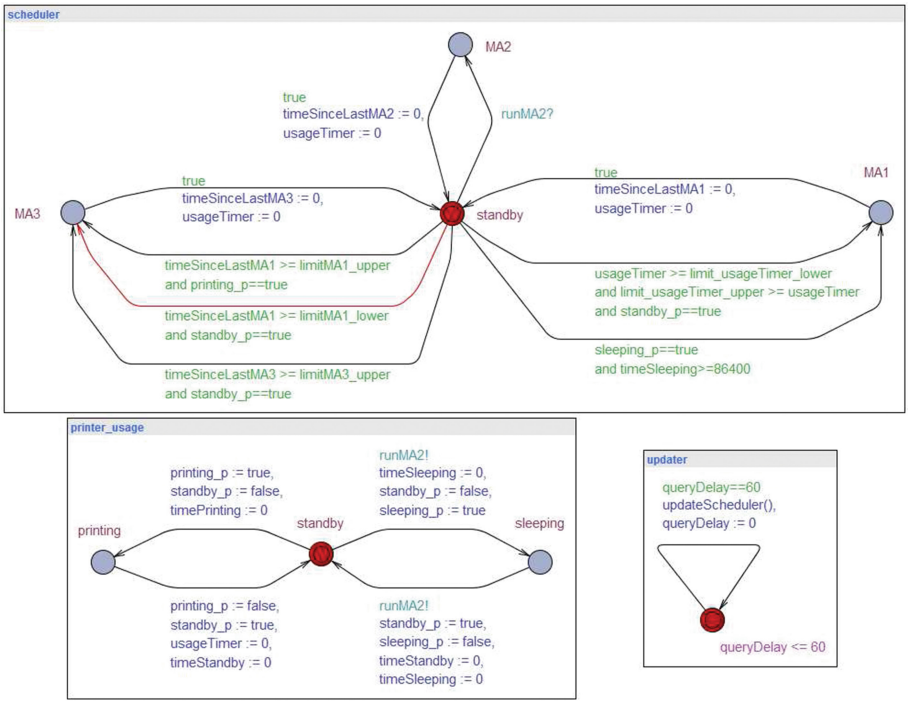


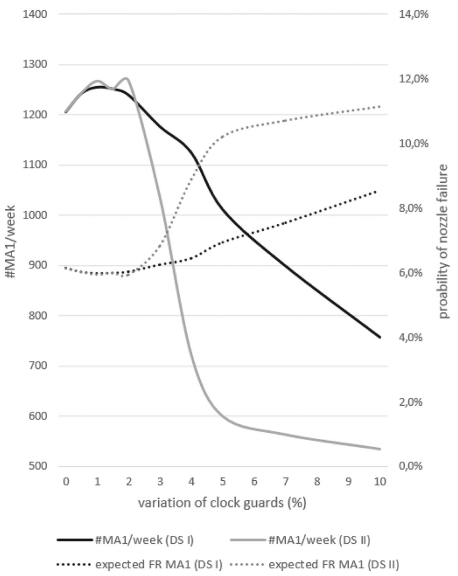
Fig. 5. Designing components by state-machines in *Uppaal*.

As shown in Fig. 5, the whole system is composed of 3 components, all of them modeled using state-machines:

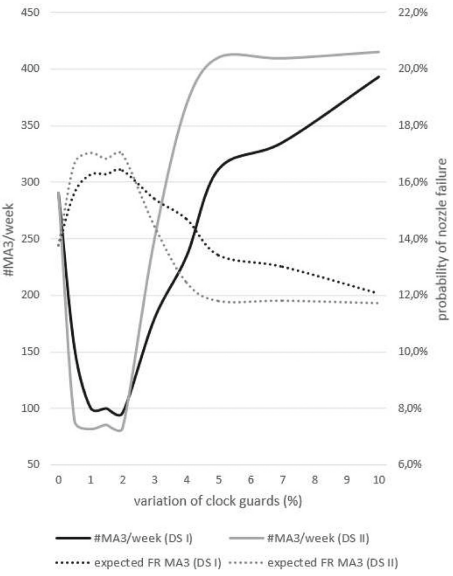
1. The *printer usage*: this models how the printer is currently being used. It is composed of 3 states, printing, standby (e.g. waiting for a print job) and sleeping (for long periods out of use). It is important to model the printer usage since the way maintenance actions are scheduled depends on the usage of the printer. Indeed, print jobs will never be interrupted to perform maintenance. It is similar when the printer is in sleeping mode, since less maintenance is required when the printer is not in use.
2. The *scheduler*: this models the maintenance actions and under what conditions they are performed. In our case, we focuses especially on 3 nozzle-related maintenance actions. Inside the specifications of the scheduler, an inner function is defined reproducing the outcome of the decision tree as well as the values set to the refreshed timed conditions (see Algorithm 1). The *scheduler* reflects the specifications of the nozzles under which maintenance is supposed to be performed.
3. The *updater*: this calls – with a given frequency – the function refreshing the *scheduler* from the outcome of the classification model. This function consists of a call to the decision tree implemented in *Uppaal* as an embedded function, itself consisting of a succession of *if* conditions representing the overall structure of the nozzle failure behavior. Moreover, this function also updates the timed conditions for the triggering of maintenance actions (such as `limitMA1_upper`, `limitMA1_lower`, `limitMA3_upper`, `limit_usageTimer_lower` and `limit_usageTimer_upper` in Fig. 5).

4.4 Results

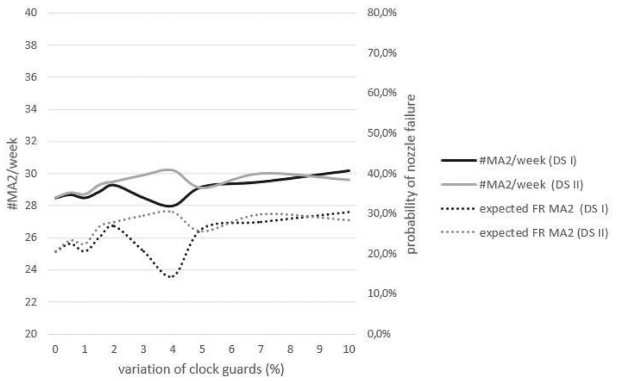
Experimental Parameters. In order to measure the benefits of our dynamic scheduler compared to a static scheduler, we relied on several metrics. First, by using our designed models in *Uppaal* and making simulations, we were able to compute how many times each state has been visited e.g. how many maintenance actions have been triggered. That is the reason why we made 10 runs with a virtual duration of 600 ks (approximately 1 week) for each configuration we wanted to test. From those runs, we were able to find out the behavior of our scheduler in the long term, since the average of the runs is computed over 10 weeks of simulation. We were finally able to compare our results with a *static* scheduler (no call to classification model) and a *dynamic* scheduler. In the case of the dynamic scheduler, we distinguish 3 cases: the first (*Dynamic Scheduler I*) resets, once maintenance is triggered, the values of the timed conditions to their initial values e.g. defined in the specifications ($currentBoundary \leftarrow t_\sigma$). The second (*Dynamic Scheduler II*) aims, after a maintenance action has been performed, to average the timed conditions between the past boundary and when the MA has really been done ($currentBoundary \leftarrow (currentBoundary + t_\sigma)/2$). The last one (*Dynamic Scheduler III*) computes the parameters y and z as a function of the confidence factor provided by the classification model.



(a) Results achieved for MA_1



(b) Results achieved for MA_3



(c) Results achieved for MA_2

Fig. 6. Results concerning *Dynamic Scheduler I* and *II*. The value corresponding to a variation of clock guards of 0 % stands for the *Static Scheduler*.

In order to run experiments with *Uppaal*, several parameters have been set, such as the refreshing frequency (60 s), the variation of how much to postpone or advance maintenance actions (from 0.5 % to 10 %), the usage of the printer (the printer goes to sleeping mode every 8 hours and during at least 2 hours; after being printing during more than 2 min, then, the printer can stop printing and go into standby mode; after being without printing during more than 1 min, then, the printer can start printing again) and the nozzle behavior (a nozzle is likely to fail every 20 min). We can also mention maintenance-related additional information and settings:

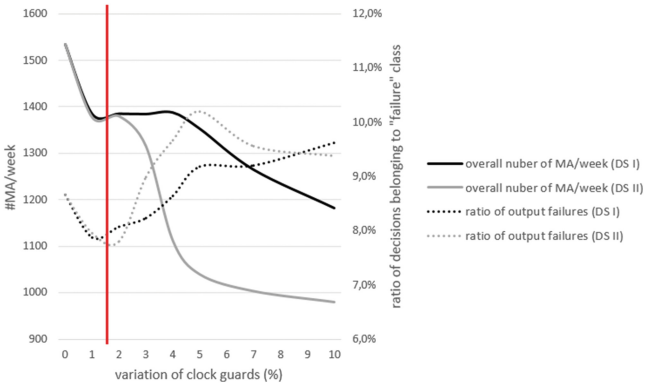


Fig. 7. Overall MA performed and ratio of decisions classified as *failure*. The value corresponding to a variation of clock guards of 0 % stands for the *Static Scheduler*. The vertical line shows a possible optimal variation of the clocks guards. (Color figure online)

- Maintenance action #1 (MA_1): according to the usage of the printer, usually triggered every 40 s – 3.5 h. Maximum acceptable threshold set: 5 h.
- Maintenance action #2 (MA_2): usually triggered when the system is going to and coming back from sleeping mode.
- Maintenance action #3 (MA_3): according to the usage of the printer, usually triggered every 15 min – 3.5 h. Maximum acceptable threshold set: 5 h.

Results. We give a graphical representation of the number of the achieved maintenance actions performed per week in function of the variation of the degree of clock increase/decrease, as well as the expected nozzle failure rate in Fig. 6a to c (for *Dynamic Scheduler I* and *II*). The value corresponding to a variation of clock guards of 0 % stands for the static scheduler, since parameters y and z equaling 0 means no change of the clock guards. We also give the results of the *Static Scheduler* and the *Dynamic Scheduler III* in Table 2. The results are stated for each type of MA by number of MA performed ($\#MA$) and the expected failure rate when the maintenance is performed (FR_{MA}). The expected failure rate is computed from the distribution shown in Fig. 4, since the information providing the nozzle failure rate as a function of the time since last maintenance has been computed from the logs. We also present for each scheduler the proportion of cases classified as failures by our failure behavior model ($\mathcal{C}_f$ in Table 2 and ratio of output failures in Fig. 7).

Discussion. From the results achieved, we can see that in some settings, the number of maintenance actions performed can decrease. Nonetheless, while the number of maintenance actions is decreasing, the *expected* nozzle failure rate slightly increases, yet still within acceptable values. We can refer to the first

Table 2. Number of maintenance actions triggered using *Dynanic Scheduler III*.

Scheduler	$\#MA_1$	FR_{MA_1}	$\#MA_2$	FR_{MA_2}	$\#MA_3$	FR_{MA_3}	$\mathcal{C}_f$
Static	1205.6	6.1 %	28.5	20.5 %	290.3	13.8 %	8.7 %
Dynamic III	806.6	8.2 %	29.1	25.6 %	378.1	12.3 %	9.5 %

dynamic scheduler and to MA_3 with an optimal decrease of three times as few maintenance actions performed. The counterpart is an increase of the expected failure rate by 2 points. We can also see that in some cases, especially MA_2, our scheduler has no influence on the triggering of this specific type of maintenance. Indeed, MA_2 is an example of usage-based maintenance, whereas our method only modifies timed-based maintenance. According to the results achieved by the *Dynamic Scheduler III* e.g. advancing or postponing the clock guards using the confidence factor of the Decision Tree, we can see that less MA_1 is performed but the expected failure rate when MA_1 is triggered is increased by 2 points, whereas MA_3 is done more often. Furthermore, according to the number of cases detected as failure cases by our classifier as well as the number of actions performed (whatever the type), we can see that in some cases, less maintenance is done and fewer failures are detected by our model, in particular when using a variation of how much to postpone or advance maintenance actions of 1 or 2 %. This is shown in Fig. 7 by a red line. This result proves our assumption stating that we can at the same time save maintenance and improve the print quality. Finally, for some settings and independently of the scheduler, our dynamic schedulers perform more actions than a static scheduler.

5 Conclusion

In this paper, we describe a new method for dynamic maintenance scheduling. We expect that our method can be generalized to other domains. The major novelty we bring, to the best of our knowledge, is a method involving machine learning techniques by using the decision of a classifier in order to put forward or postpone when maintenance should be triggered, i.e. how to update scheduling of automatic periodic maintenance defined by a TA.

According to our results, we can conclude that our dynamic scheduler reduces the number of actions performed in different settings. We also note that the price to pay in order to do less maintenance is a slight increase of the expected failure rate. Thus, depending on how critical the component is, our technique can reduce maintenance costs for a negligible increase in the risk of breakdown. In our case, it entailed an insignificant loss of print quality. Moreover, in some settings, we could reduce the failure rate as well as the number of maintenance performed. We also believe that our technique can be particularly interesting in the case of the unavailability of sensors that provide direct information about component failures. The strength of an embedded decision model is its availability at any time. Furthermore, we expect that, applied to other real systems, our technique could achieve similar results to those found in our simulation.

With regard to further work, we think that within the scope of our case study, we can extend the current experiment not only to nozzles but also to other related components, or at least components that share related maintenance actions. Furthermore, in future, we will look into the use of fault trees [26]. Indeed, we believe that a fault tree pattern can be used to model interactions between several components, and how a failure can propagate from one component to others. We also hope that extending such a technique to the use of real-time automata can enhance the schedulability and control of CPS. We can also enhance the scheduler by taking into account the timing occurrences of anomalies [21]. Finally, we could orientate the choice of the machine learning techniques used towards stream mining tools and algorithms [6], which would additionally offer the possibility of updating the failure behavior model the more new labeled data are available. Then, it could be possible to deal with unseen events or combination of parameters, and keep an accurate model throughout the life of the CPS.

Acknowledgments. Thanks to Lou Somers and Patrick Vestjens for providing industrial datasets as well as required expertise related to the case of study. This research is supported by the Dutch Technology Foundation STW under the Robust CPS program (project 12693).

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theor. Comput. Sci. **354**(2), 272–300 (2006)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Arab, A., Ismail, N., Lee, L.S.: Maintenance scheduling incorporating dynamics of production system and real-time information from workstations. J. Intell. Manuf. **24**(4), 695–705 (2013)
4. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III: Verification and Control. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996)
5. Bengtsson, J.E., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)
6. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
7. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Inf. Process. Lett. **98**(5), 188–194 (2006)
8. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: statistical model checking for priced timed automata. arXiv preprint arXiv:1207.1272 (2012)
9. Burns, A.: How to verify a safe real-time system: the application of model checking and timed automata to the production cell case study. Real-Time Syst. **24**(2), 135–151 (2003)

10. Butler, K.L.: An expert system based framework for an incipient failure detection and predictive maintenance system. In: Proceeding of the International Conference on Intelligent Systems Applications to Power Systems, Orlando, Florida, USA, pp. 321–326 (1996)
11. Cardenas, A.A., Amin, S., Sastry, S.: Secure control: towards survivable cyber-physical systems. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, pp. 495–500 (2008)
12. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012)
13. Flach, P.: Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press, Cambridge (2012)
14. Fumeo, E., Oneto, L., Anguita, D.: Condition based maintenance in railway transportation systems based on big data streaming analysis. Procedia Comput. Sci. **53**, 437–446 (2015)
15. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 379–395. Springer, Heidelberg (2004)
16. Gross, P., Boulanger, A., Arias, M., Waltz, D.L., Long, P.M., Lawson, C., Anderson, R., Koenig, M., Mastrocinque, M., Fairechio, W., et al.: Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In: Proceedings of the 21st National Conference on Artificial Intelligence, Boston, Massachusetts, USA, vol. 21, pp. 1705–1711 (2006)
17. Hashemian, H.M., Bean, W.C.: State-of-the-art predictive maintenance techniques. IEEE Trans. Instrum. Meas. **60**(10), 3480–3492 (2011)
18. Kaiser, K.A., Gebraeel, N.Z.: Predictive maintenance management using sensor-based degradation models. IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. **39**(4), 840–849 (2009)
19. Lee, E.A.: Cyber physical systems: design challenges. In: Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, Orlando, Florida, USA, pp. 363–369 (2008)
20. Li, H., Parikh, D., He, Q., Qian, B., Li, Z., Fang, D., Hampapur, A.: Improving rail network velocity: a machine learning approach to predictive maintenance. Transp. Res. Part C: Emerg. Technol. **45**, 17–26 (2014)
21. Maier, A., Niggemann, O., Eickmeyer, J.: On the learning of timing behavior for anomaly detection in cyber-physical production systems. In: Proceedings of the 26th International Workshop on Principles of Diagnosis, Paris, France, pp. 217–224 (2015)
22. Maier, A.: Online passive learning of timed automata for cyber-physical production systems. In: Proceedings of the 12th IEEE International Conference on Industrial Informatics, Porto Alegre, Brazil, pp. 60–66 (2014)
23. Niggemann, O., Biswas, G., Kinnebrew, J.S., Khorasgani, H., Volgmann, S., Bunte, A.: Data-driven monitoring of cyber-physical systems leveraging on big data and the internet-of-things for diagnosis and control. In: Proceedings of the 26th International Workshop on Principles of Diagnosis, Paris, France, pp. 185–192 (2015)
24. Nowaczyk, S., Prytz, R., Rögnvaldsson, T., Byttner, S.: Towards a machine learning algorithm for predicting truck compressor failures using logged vehicle data. In: Proceedings of the 12th Scandinavian Conference on Artificial Intelligence, Aalborg, Denmark, pp. 205–214 (2013)
25. Quinlan, J.R.: C4.5: Programs for Machine Learning. Elsevier, Amsterdam (2014)
26. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15**, 29–62 (2015)

27. Simeu-Abazi, Z., Bouredji, Z.: Monitoring and predictive maintenance: modeling and analyse of fault latency. Comput. Ind. **57**(6), 504–515 (2006)
28. Verwer, S., Weerdt, M., Witteveen, C.: Efficiently identifying deterministic real-time automata from labeled data. Mach. Learn. **86**(3), 295–333 (2011)
29. Witten, I.H., Frank, E., Trigg, L.E., Hall, M.A., Holmes, G., Cunningham, S.J.: Weka: practical machine learning tools and techniques with Java implementations (1999)